Randomized Approximation Algorithms:
Facility Location, Phylogenetic Networks, Nash Equilibria

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
Rector Magnificus, prof.dr.ir. C.J. van Duijn, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op maandag 13 oktober 2008 om 16.00 uur

door

Jarosław Byrka

geboren te Wrocław, Polen

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr. K.I. Aardal
en
prof.dr. M.T. de Berg

A catalog record is available from the Eindhoven University of Technology Library
ISBN: 978-90-386-1416-8

Promotor: prof.dr. K.I. Aardal (Delft University of Technology)

2nd Promotor: prof.dr. M.T. de Berg (Eindhoven University of Technology)

Kerncommissie:
prof.dr. D.B. Shmoys (Cornell University, Ithaca, NY, USA)
dr. K. Lorys (Wroclaw University, Poland)
prof.dr. G.J. Woeginger (Eindhoven University of Technology)

The research reported in this thesis has been carried out at the Centrum voor Wiskunde en Informatica, and at the Eindhoven University of Technology. The first three years of the research were supported by the EU Marie Curie Research Training Network ADONET, Contract No MRTN-CT-2003-504438.

Printing: Eindhoven University Press

Acknowledgements

In the last four years, I have enjoyed the opportunity to work with outstanding researchers both in Centrum voor Wiskunde en Informatica (CWI) in Amsterdam, and in the Technische Universiteit Eindhoven (TU/e).

I thank Karen Aardal for giving me the opportunity to work with her and for her great supervision. I am especially grateful for her trust in me as a researcher. She was probably the only person who believed, that our interest in the Facility Location problem will eventually be fruitful. I also appreciate her comments to the manuscripts that I worked on, as well as her advice on the publication process in general. Finally, her support and understanding of my personal situation was invaluable. It was particularly important for me when my son Filip was born. I also thank Mark de Berg for being my second supervisor.

I am grateful to a number of people who invited me to their areas of research. Namely, to Karen Aardal (facility location), Steven Kelk (computational biology), Vangelis Markakis (algorithmic game theory), Marcin Bienkowski (online algorithms), Alexander Wolff (graph drawing). I truly enjoyed this diversity of topics, which I would probably never approach if not my great colleagues. I also thank each of my co-authors, for the precious experience of our collaboration.

It is barely impossible to mention all the people with whom I discussed the scientific content of my thesis. Nonetheless, I recognize my colleagues from both CWI and TU/e, and the referees of my papers. I especially enjoyed the discussions with my wife Kasia, who as a psychologist often had a different view on the meaning and the importance of various aspects of the work I was involved in.

A number of people made an effort to help me with the preparation of the manuscript of this thesis. In particular, I am grateful to Karen who was assisting me throughout the entire writing process. She had to fight with my notorious spelling mistakes (like writing "guaranty" instead of "guarantee"). I thank Kasia for reading most of the text and also Maciek for helping me with the introduction.

Apart from research, I spent hours discussing less relevant issues with, among the others, my roommates at CWI, Nebojsa Gvozdenovic and Hartwig Bosse, and Alexander Wolff and Maciek Modelski at TU/e.

Finally, I would like to thank the people who made it possible for me to spend the last four years working in the places I did. I am grateful for the opportunity to be a member of the PNA1 group at CWI. The first three years of my PhD research were possible due to the generous support of the EU project ADONET Contract

No MRTN-CT-2003-504438. I thank Marc de Berg and the other members of the algorithmic group at TU/e for hosting me during the last year of my PhD study. I also thank Leen Stougie for inviting me to join the EU project ARRIVAL and offering a postdoc position which I currently hold.

Jarosław Byrka

Contents

Chapter 1

Preliminaries

Despite a great effort, researchers are unable to find efficient algorithms for a number of natural computational problems. Typically, it is possible to emphasize the hardness of such problems by proving that they are at least as hard as a number of other problems. In the language of computational complexity it means proving that the problem is *complete* for a certain class of problems.

For optimization problems, we may consider to relax the requirement of the outcome to be optimal and accept an approximate (i.e., close to optimal) solution. For many of the problems that are hard to solve optimally, it is actually possible to efficiently find close to optimal solutions. In this thesis, we study algorithms for computing such approximate solutions.

1.1 Problems, algorithms and computation complexity

The field of computational complexity concerns itself with determining in a comparative way the computational difficulty of problems. The difficulty/complexity of a problem is related to the existence of a good solution method, an algorithm. Let us first briefly discuss some types of computational problems and algorithms.

Decision problems. These problems are probably the most studied in theoretical computer science. A decision problem is defined by two sets, the set of instances I and a subset $Y \subset I$ of, so called, "yes" instances. The problem is: Given an instance $i \in I$, decide whether or not $i \in Y$. As an example of a decision problem, consider the following problem PRIME. In this problem the instance set is the set of positive integers and the set of "yes" instances is the set of prime numbers.

We will mainly be interested in discrete problems, which have countably many instances. Moreover, as we will ignore constant factors in the running times of the algorithms, the problems with finite number of instances are of minor interest to us. To give general statements about classes of problems, we need to assume a certain encoding of instances. A standard assumption is that instances are given as binary strings, and that the size of an instance is the length of such a string. (Equivalently,

we could say, that instances are positive integers and the size of an instance is the logarithm of the number representing the instance). The set of "Yes" instances of a decision problem is often called a *language*, as it is just a set of words over the binary alphabet, and the problem is then called a membership problem of a word in the language.

Algorithms for decision problems. To provide a simple method of testing whether an instance is a "yes" instance for a particular problem, a formal definition of an algorithm is not necessary. Often it suffices to think about algorithms as of instructions, descriptions of methods designed to solve specific problems. However, to give a proof that certain problems do not have algorithms, or do not have algorithms with certain properties, it is essential to formally define the concept of algorithms.

The first important model of an algorithm was proposed by Turing [Tur37], who developed his famous Turing Machine (TM). A TM reads the input data, performs a number of simple operations, possibly prints some output, and may decide to stop at some point. Once a machine stops, we may interpret its output as an answer to the computational question we consider. Particularly useful are the machines that are guaranteed to stop no later than after a certain number of operations, expressed as a function of the input size. Such a TM corresponds to a computational problem it solves, and the number of operations it uses is called its *running time*.

Turing used his machines to prove the existence of, so called, undecidable problems, i.e., problems for which no TM solving the problem exists. In algorithmic computer science we are typically interested in the question whether there exists a TM/algorithm that solves a particular problem, and moreover, among the algorithms solving the problem, we search for an algorithm with a short running time.

For a better control of the running time of an algorithm, instead of TMs, a different computational model may be used. A Random Access Machine (RAM) [RAM] model allows for simple arithmetic operations to be performed in a single unit of time. For a TM to perform any operation on a number x it has to at least read x, which occupies a space in memory that is at least logarithmic in the value of x, whereas in the RAM model it is assumed that the operation time does not depend on the size of the involved numbers. One could argue that the TM model is more realistic, but the difference is not significant in case of the algorithms studied in this thesis. We will mainly distinguish between polynomial and nonpolynomial time algorithms, and it is well known that polynomial time algorithms in one model may easily be turned into polynomial time algorithms in the other model. In this thesis, whenever a more precise running time of an algorithm is provided, it is the running time in the RAM model.

Optimization problems. Whereas for computational complexity the decision problems are easier to handle, for modeling real life problems the language of *optimization problems* is often a more natural choice. In an optimization problem, we are given a description of what makes a solution feasible for a given instance of

the problem. Within the set of the feasible solutions, we are supposed to find one which maximizes/minimizes a given objective function. The set of feasible solutions is typically described in terms of a list of constraints. A solution is feasible if it satisfies all of these constraints, and we say that an instance is feasible if and only if the set of feasible solutions for this instance is not empty.

To use computational complexity results obtained for decision problems in the context of optimization problems, we may proceed as follows. For a given optimization problem we may study a related decision problem (e.g. the feasibility problem, or the problem of existence of a feasible solution with at least a certain value of the objective function). If the obtained decision problem D is difficult, it implies a certain difficulty of the optimization problem as it is more general than D.

The link between the complexity of an optimization problem, and the complexity of related decision problems is not always simple. For example, if we think of the Nash equilibrium problem (discussed in Chapter 4) as a problem of minimizing the incentive of players to deviate, then, by the theorem of Nash, the natural corresponding decision problems become trivial: there always exists a solution (equilibrium) such that no player wants to deviate. Some other related decision problems, e.g., if there exists an equilibrium with at least a certain payoff for each of the players, are NP-complete (see Section 1.1.1 for a definition). Nevertheless, the problem of finding a Nash equilibrium, which is equivalent to the optimization problem we started with, falls into yet another complexity class: it is PPAD-complete (see Chapter 4).

From an algorithm for an optimization problem we expect that the output it provides may be interpreted as one of the feasible solutions, or the information that no feasible solution exists. If the solutions that a particular algorithm returns are always optimal, then we call this algorithm an *exact* algorithm. If an algorithm produces solutions that are close to optimal (but still feasible), then we call it an *approximation* algorithm.

In this thesis, we are mainly interested in problems with the following property. For a given instance of the problem, there is a finite number of feasible solutions. Typically, the number of feasible solutions will be exponential in the size of the instance encoding, and no polynomial time algorithm can examine all of the feasible solutions.

Asymptotic notation. Running times of algorithms are expressed as functions from the size of the problem instance to the number of operations that the algorithm is allowed to perform. These functions map nonnegative integers into nonnegative integers. We need to compare such functions to be able to conclude that one algorithm is faster than the other. As we would like to ignore constant factors in the running time, we will use the following notation (see [CLRS01]).

Let $\mathbb{N}$ be the set of nonnegative integers (also called natural numbers). In this section all the functions are of the type $\mathbb{N} \to \mathbb{N}$. We use $f(n)$, $g(n)$ to denote functions that take parameter n.

Definition 1.1.1

$$\begin{aligned}
O(g(n)) &= \{f(n) : \exists_{c,n_0>0}\ 0 \le f(n) \le cg(n) \textit{ for all } n \ge n_0\},\\
\Omega(g(n)) &= \{f(n) : \exists_{c,n_0>0}\ 0 \le cg(n) \le f(n) \textit{ for all } n \ge n_0\},\\
\Theta(g(n)) &= O(g(n)) \cap \Omega(g(n)),\\
o(g(n)) &= \{f(n) : \forall_{c>0}\exists_{n_0>0}\ 0 \le f(n) < cg(n) \textit{ for all } n \ge n_0\},\\
\omega(g(n)) &= \{f(n) : \forall_{c>0}\exists_{n_0>0}\ 0 \le cg(n) \le f(n) \textit{ for all } n \ge n_0\}.
\end{aligned}$$

For brevity, we will abuse the notation and write $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$, etc. to denote $f(n) \in O(g(n))$, $f(n) \in \Omega(g(n))$, ..., respectively. The intuitive meaning of these symbols is the following. If we write $f(n) = o(g(n))$, $f(n) = O(g(n))$, $f = \Theta(g(n))$, $f(n) = \Omega(g(n))$, or $f(n) = \omega(g(n))$, then for big enough values of n (ignoring constant factors) we have $f(n) < g(n)$, $f(n) \le g(n)$, $f(n) \approx g(n)$, $f(n) \ge g(n)$, or $f(n) > g(n)$, respectively.

1.1.1 P vs. NP

Here we discuss complexity classes, which are sets of problems with similar computational complexity. Whether a problem belongs to a particular complexity class depends on the existence of an appropriate algorithm for this problem.

Consider a function $f : \mathbb{N} \to \mathbb{N}$. Define $DTIME(f(n))$ to be the class of problems for which there exist deterministic algorithms which run in at most $O(f(n))$ time on the input of size n. We use the word *deterministic* to emphasize that we mean algorithms whose computation depends only on the input data, and would always be the same, no matter how many times we run the algorithm. There are, however, different possibilities. One may study *randomized* algorithms that "flip coins" to make certain decisions. We discuss them in Section 1.3.

Another type of algorithms are *nondeterministic* algorithms. Part of their decisions depend on some additional input, e.g., the position of stars in the sky or a hint from a friendly oracle. We say that a nondeterministic algorithm accepts an instance if there exists a hint that makes the algorithm output "yes, it is a yes instance". The additional input (hint) that convinces the algorithm that a given instance is a "yes" instances is called a *certificate*. We define $NTIME(f(n))$ to be the class of problems for which there exist nondeterministic algorithms that run in at most $O(f(n))$ time on the input of size n.

Observe that nondeterministic algorithms are at least as powerful as the deterministic ones. Namely, if there exists a deterministic algorithm for a problem, we may think that it is also a nondeterministic algorithm, which is simply not using any hint (i.e., has no nondeterministic decisions to make). Therefore, we have $NTIME(f(n)) \supseteq DTIME(f(n))$ for any $f(n)$. Is the converse true? Is it true for some specific choice of $f(n)$? To the best of our knowledge, no rigorous mathematical proof that would resolve the above questions has been found.[1]

[1] For the space restricted computations, however, it was shown that nondeterministic linear time algorithms are more powerful that deterministic linear time algorithms [Gur90].

Intuitively, a good hint can make the problem easier. For example, suppose the problem is to decide if there exists in a given set an object with a special property. A nondeterministic algorithm may use a hint on which object to consider and say "yes" if the chosen object is good enough. Observe, that whenever an object with the special property exists, there is a hint that will convince the algorithm. However, if no special object exists, no hint will make the algorithm say "yes". Therefore, the described algorithm accepts exactly when there exists a good object in the set. By contrast, it seems that a deterministic algorithm cannot do any better then just examine all the objects in the set. Intuitive arguments like this, are sufficient to convince most of the researchers that nondeterministic algorithms are strictly more powerful that the deterministic ones. Such a statement, although formally unproven, is often used by the researchers as a conjecture. Before we state this conjecture more formally, let us first restrict our attention to algorithms with limited running time.

For solving problems with a computer we need algorithms whose running times are reasonable. A commonly accepted criterion for the algorithm to be called *efficient* is that its running time is bounded by a polynomial function of the input size. The class of decision problems having efficient deterministic algorithms is called P and is defined as

$$P = \cup_{i \in \mathbb{N}} DTIME(n^i).$$

Another important class of problems is NP. It contains the problems that have polynomial time nondeterministic algorithms and is defined as

$$NP = \cup_{i \in \mathbb{N}} NTIME(n^i).$$

The class NP may also be described as the class of problems, for which there exist compact certificates for the "yes" instances. Whether a certificate really certifies a particular "yes" instance needs to be testable by a deterministic polynomial time algorithm, called *verifier*. For the problem to be in NP, there needs to be a verifier that accepts at least one certificate for each "yes" instance and no certificate for a "no" instance. Note the asymmetry of problems from NP, only the "yes" instances are guaranteed to have certificates. A complementary class called $coNP$ is the class of problems for which the "no" instances are guaranteed to have compact certificates. Unless $NP = coNP$ (which is considered unlikely by the experts in the field), there are, in general, no certificates for "no" instances of the NP problems.

By far the most intriguing open problem in theoretical computer science, and perhaps in modern mathematics, is the question whether P is equal to NP. A great number of researchers have attempted to resolve this question. For some recent efforts to solve this problem see [Woe]. As mentioned above, most of the researchers interested in the field of computational complexity believe that P is a proper subset of NP. In other words, it is believed that for the hardest problems from the class NP there are no polynomial time deterministic algorithms.

The P vs. NP problem has been a central issue since the publication of results by Cook and Levin. Cook [Coo71] found a problem which is essentially as hard as any other problem in the class NP. This problem is called 3-SAT and it asks whether or not there exists an assignment of "true"/"false" values to logical variables in a

given formula[2] which satisfies the formula. Cook proved that any problem from the class NP may be polynomially reduced[3] to the 3-SAT problem. We call a problem *NP-hard* if it has this property, i.e., if it is at least as difficult as any other problem in the class NP. If the studied problem is NP-hard and belongs to NP, then we call it an *NP-complete* problem. Simultaneously Levin [Lev73] studied different NP-complete problems. His results are considered to be essentially equivalent to the findings of Cook, hence the NP-completeness of the 3-SAT problem is often referred to as the Cook and Levin Theorem. Many other problems have been subsequently proven to be NP-complete.

The P vs. NP question can be rephrased to ask whether an NP-complete problem, e.g., 3-SAT, belongs to the class P. Therefore, asking for a polynomial time algorithm for an NP-complete problem is equivalent to asking for a proof of $P = NP$. Hence, when a problem is proven to be NP-complete, we often give up our attempts to find a deterministic polynomial time algorithm to solve it, and conclude that such an algorithm does not exist under the $P \neq NP$ conjecture.

One of the possible paths to follow while dealing with an NP-complete problem is to look for algorithms that solve the problem approximately. Formally, we will not talk about approximation algorithms for decision problems, but rather change them into optimization problems first. This approach allows to separate constraints that need to be satisfied from those that may potentially be given up. The latter constraints may be turned into an objective function, which encodes a certain penalty for violating a constraint. Computing approximate solutions for such optimization problems is the main topic of this thesis.

1.1.2 IP and LP

Many combinatorial optimization problems may easily be expressed in the form of an Integer Program (IP)[4]. An IP is an optimization problem, whose feasible solutions are defined in terms of linear inequalities with an additional requirement that the variables should only take integral values. The objective function is also

[2]In the 3-SAT problem the formula is assumed to be in a special 3-CNF form, i.e., it is assumed to be a conjunction of clauses, each of the clauses is a disjunction of at most three literals, where a single literal is a logical variable or its negation.

[3]A polynomial reduction from problem A to problem B is a function $r : A \to B$ with the following properties. For any instance $a \in A$ the computation of $r(a)$ is possible in time polynomial in the size of a, and therefore the size of $r(a)$ is also polynomial. Moreover, we require that $r(a)$ is a "yes" instance of problem B if and only if a is a "yes" instance of problem A. Suppose there exists a deterministic polynomial time algorithm for the problem B, we may combine this algorithm with reduction r to get a deterministic polynomial time algorithm for problem A. Therefore, such a reduction from A to B may be interpreted as an indication that A is at most as difficult as B.

[4]To be more precise, we could use a name Integer Linear Programs (ILP), since we assume both the objective function and the constraints to be linear. However, we will mainly study the linear problems, and we decided to use the shorter name, IP in place of ILP. The only place in this thesis where nonlinear integer programs are considered is the Section 3.4.5. There we use an integer program with a quadratic objective function, which we call Integer Quadratic Program (IQP).

required to be linear. A standard form of a maximization IP is the following.

$$\max c^T x, \tag{1.1}$$
$$\text{subject to } Ax \leq b, \tag{1.2}$$
$$x \in \mathbb{N}^n \tag{1.3}$$

where x is a vector of n elements, which we call variables, $A \in R^{m \times n}$ and $b \in R^m$ are the given coefficients defining the set of feasible solutions, and $c \in R^n$ is the vector of coefficients defining the objective function.

Since integer programs are optimization problems, not all the complexity results obtained for decision problems are applicable here. In particular, it makes little sense to ask whether an IP problem belongs to the class NP. However, we may encode any NP-complete problem as an integer program. Therefore, we say that the problem of solving IPs is in general NP-hard. We may interpret such an encoding as an argument for the nonexistence of a polynomial time algorithm for solving IPs.

For certain integer programs (e.g., programs encoding instances of a particular optimization problem) the related decision problems (like feasibility) are often NP-complete. In such cases, we also say that these integer programs are NP-hard.

Despite the complexity of integer programming in general, many IP formulations of real life problems may actually be solved. For an extensive coverage of methods for solving IPs we would like to direct an interested reader to [NW88].

Consider the following, related type of problems. If we drop the integrality requirement in an IP, we obtain a Linear Program (LP). An LP asks to optimize a linear function subject to linear constraints, but, in contrast to IP, the variables may take any real values. A standard form of a maximization LP is

$$\max c^T x, \tag{1.4}$$
$$\text{subject to } Ax \leq b, \tag{1.5}$$
$$x \geq 0. \tag{1.6}$$

Since the above LP formulation is a relaxation of the IP formulation (1.1)-(1.3), the value of the optimal solution to the LP formulation yields an upper bound on the value of the optimal solution to the IP program. The difference between these values is called the *integrality gap* of the IP.

Linear programs have many useful features, such as the existence of dual programs with interesting properties. In particular, the LP defined by (1.4)-(1.6) has the following dual program

$$\min b^T y, \tag{1.7}$$
$$\text{subject to } A^T y \geq c, \tag{1.8}$$
$$y \geq 0. \tag{1.9}$$

The dual of the dual program is again the original (primal) program. Consider a feasible solution x to the primal problem and a feasible solution y to the dual problem.

Let $z(x)$ and $z'(y)$ denote the values of these solutions. The Weak Duality Theorem states that $z(x) \leq z'(y)$. The Strong Duality Theorem states that, assuming the existence of optimal solutions to both programs, the values of the optimal solutions are equal.

For a number of algorithms discussed in this thesis, it is essential that linear programs are polynomially solvable. The first polynomial time algorithm for LPs was the *ellipsoid method*. The ellipsoid method is an algorithm for solving convex optimization problems introduced by Naum Z. Shor, Arkady Nemirovsky, and David B. Yudin in 1972. Leonid Khachiyan [Kha79] used the method to prove the polynomial-time solvability of linear programs.

Linear and integer programming problems have been extensively studied. For an overview of the classical results it this field we recommend [Sch86; NW88].

1.2 Approximation algorithms

We will now discuss the main topic of this thesis, *approximation algorithms* for optimization problems. From an approximation algorithm we expect that it computes "reasonable" solutions in polynomial time. As we would like our algorithms to be practical, we prefer algorithms that are really fast (e.g. run in linear time). Nevertheless, from the theoretical point of view, we find it more important that the algorithm we study computes provably good solutions, as long as its running time is polynomial.

We say that an algorithm is a λ-approximation algorithm for a minimization problem A if, given any input $a \in A$, it produces, in polynomial time, a solution that is at most λ times more expensive than an optimal solution to a. By analogy, we say that an algorithm is a λ-approximation algorithm for a maximization problem B if, given any instance $b \in B$, it produces, in polynomial time, a solution that has value at least λ times the value of an optimal solution to b. Observe, that we have $\lambda \geq 1$ in approximation algorithms for minimization problems and $\lambda \leq 1$ for maximization problems. We call λ the approximation guarantee. Depending on the considered algorithm, the parameter λ may either be an absolute constant or it may be expressed as a function of the instance size.

We shall emphasize the worst case flavor of the above definition. For different instances of a problem an approximation algorithm typically gives approximate solutions of different quality. The approximation guarantee of an algorithm for a particular problem is determined by the instances of the problem that are worst for the studied algorithm. It is often the case, that the same approximation algorithm, when considered as an algorithm for the problem restricted to only a subset of instances, may actually be proven to have a much better approximation guarantee.

Well known combinatorial optimization problems such as the Graph Coloring problem, the Independent Set problem, or the Maximum Clique problem are difficult to approximate. For these problems the existence of a $n^{1-\epsilon}$-approximation algorithm

for any positive ϵ would imply a collapse of certain complexity classes[5]. For these difficult problems, researchers still try to estimate the values of optimal solutions to particular instances (see e.g. [Gvo08]).

Some other problems are easier to approximate. For the Set Cover problem, for example, it is possible to compute $\ln(n)$-approximate solutions with a greedy algorithm. Still, constant factor approximation algorithms for this problem are rather unlikely to exist, as shown by Feige [Fei98]. For more discussion about the Set Cover problem and its variants, see Chapter 5.

For a number of problems, constant factor approximation algorithms exist. The class of problems having this property is called APX. Many, otherwise hard to approximate, problems become constant factor approximable under the assumption that the objective function is metric. For an example of a problem with a metric version having constant factor approximation algorithms, see the Facility Location Problem discussed in Chapter 2. Within the class APX, we distinguish the hardest to approximate problems, which are called APX-complete problems. A minimization (maximization) problem is APX-complete if there exists a positive constant ϵ such that the problem of finding $(1+\epsilon)$-approximate ($(1-\epsilon)$-approximate) solutions to the problem is NP-hard.[6]

There is a class of problems for which arbitrarily good approximations are possible. Formally, for such a problem there exists an infinite sequence of approximation algorithms whose approximation ratios get closer to 1. We express this approximation guarantee as $1+\epsilon$, where ϵ tends to 0. Such a sequence of algorithms is called a Polynomial Time Approximation Scheme (PTAS). Observe, that proving a problem APX-complete is equivalent to disproving the existence of a PTAS for the problem, unless $P = NP$. For many problems restricted to only dense[7] instances there exist approximation schemes [AKK99].

Each of the algorithms in a PTAS has running time polynomial in the instance size, but as we choose algorithms with better approximation guarantee, typically, their running time gets longer. Therefore, we express the *running time of PTAS* as a function of both the input size and the approximation guarantee $1+\epsilon$. PTAS with the additional property that the running time is also polynomial in $1/\epsilon$ is called a Fully Polynomial Time Approximation Scheme (FPTAS).

For a detailed presentation of various techniques used in approximation algorithms we recommend [Vaz01]. For a compendium of approximation results for a number of NP-hard problems see [CK].

[5]It would imply $NP = ZPP$ [Has99].

[6]Originally, the class of APX-complete problems was defined to be the problems hardest in APX under PTAS-reductions (also called L-reductions). With the recent results from the PCP theory, we know that finding arbitrarily good approximations for APX-complete problems is actually NP-hard.

[7]For graphical problems, an instance is dense if the underlying graph has a minimum degree $\Omega(n)$, where n is the number of vertices. For constraint satisfaction problems, dense instances are the instances where each variable is involved in a large number of constraints.

FPT algorithms. We will now discuss an alternative approach to NP-hard problems. For certain NP-hard problems there exist natural parameters (like the size or the value of an optimal solution) and there exist algorithms whose running time becomes polynomial if we assume the value of such a parameter to be fixed.

We say that problem A is Fixed-Parameter Tractable (FPT) with respect to parameter k if there exists a deterministic algorithm solving A with running time in $O(p(n) \cdot f(k))$, where $p(n)$ is a polynomial function of n and $f(k)$ is an arbitrary function of k. We may also use a notation $O^*(f(k))$ which neglects the polynomial part of the running time of an FPT algorithm. Note, that an algorithm with running time n^k is not considered to be an FPT algorithm.

We only briefly mention the concept of parametrized complexity as it is related to approximation algorithms. For a textbook on FPT see [DF99]. In this thesis, FPT algorithms are provided for two of the studied problems (see Corollary 3.3.6 and Theorem 3.4.6).

1.3 Randomized algorithms

We have already discussed deterministic and nondeterministic algorithms. We argued that the nondeterministic algorithms are perhaps more powerful, but less realistic to implement on a computer. We will now consider yet another group of algorithms, the *randomized algorithms*.

Recall, that the computation of a nondeterministic algorithm was described as being dependent on some additional input, which we also called a hint. Imagine, that these hints are now given randomly, and as a result, the actual computation is a random process with a random outcome. A randomized algorithm may, hence, be thought of as a nondeterministic algorithm equipped with a source of random bits.

With randomized algorithms for decision problems, we may now talk about the probability that an algorithm will accept a particular instance of the problem. Values like the running time of an algorithm on a particular instance, or the value of a produced solution, in case of an optimization problem, become random variables. We may study distributions of these variables and prove statements about their expected values.

We often distinguish two types of randomized algorithms. An algorithm that always returns the right answer (optimal solution in case of optimization), but whose running time depends on the random choices is called a *Las Vegas* algorithm. An algorithm whose output is random is called a *Monte Carlo* algorithm.

In Monte Carlo algorithms for decision problems, we also distinguish the algorithms that can be wrong only in one type of instances (either only for "yes" instances or only for "no" instances). Algorithms with this property are called *one-sided error* and the remaining algorithms are called *two-sided error*.

In the context of approximation algorithms, we are mostly interested in randomized algorithms that always run in polynomial time and always return feasible solutions (or a proof that an instance is infeasible). The value of the objective

function in the obtained solution is typically a random variable whose distribution we study. We say that an algorithm is a randomized λ-approximation algorithm for an optimization problem A if it runs in polynomial time, and for each instance $a \in A$ the expected value of the produced solution is at most λ times the value of an optimal solution to the instance a.

Each of the randomized approximation algorithms provided in this thesis may be derandomized using the standard *method of conditional expectation*. We present one such derandomization explicitly (see Section 3.2.3) as we find it important to give an accurate running time analysis for this particular algorithm. A general discussion of the conditional expectation method, also called the *probabilistic method* may be found in the book by Motwani and Raghavan [MR95].

A note on determinism. We may question if the computational model of randomized algorithms is any more realistic than the model of nondeterministic algorithms. Philosophers dispute about the (non)determinism of the universe (see, e.g., [Abe76]). If there is no source of pure randomness, we cannot expect to perfectly implement algorithms that greatly depend on randomness.

With the algorithms provided in this thesis, however, the problem with randomness is not an issue, because the presented algorithms have their deterministic counterparts (which we may construct with the method of conditional expectation, as already mentioned above). The main purpose of using the language of randomized algorithms in this context is to simplify the analysis of the discussed algorithms. For example, the algorithms presented in Section 3.2 are essentially producing completely random solutions. If we had presented only a derandomized version of these algorithms, we would make our arguments unnecessarily technical and inaccessible.

1.4 Results in this thesis

In this thesis we provide approximation algorithms for four different problems. Each problem is discussed in a separate chapter.

In Chapter 2, we consider the Facility Location problem, which is the following. We are given a set of possible locations of facilities and a set of clients who require a certain service. The task is to open a number of facilities such that each of the clients is serviced and the total cost of the solution is minimal. Depending on the structure of the cost functions and additional properties required from the solution, we obtain variants of the problem. Section 2.1 contains an overview of the existing algorithms for a number of these variants. It is based on a joint work with Karen Aardal and Mohammad Mahdian [ABM08]. In the following two sections, which are based on a joint work with Karen Aardal, we address the most standard version of the problem, the metric Uncapacitated Facility Location (UFL) problem. The previously best known approximation ratio for the UFL problem was achieved by the 1.52-approximation algorithm of Mahdian, Ye and Zhang. In Section 2.2 we give a lower bound of 1.4943 on the approximation ratio of this algorithm. Our result

was published as [BA07]. Section 2.3 contains the main results of this chapter. Here, we present an optimal bifactor approximation algorithm for the UFL problem. We show, that a combination of the new algorithm with the approximation algorithm of Jain, Mahdian, and Saberi results in a 1.5-approximation algorithm for the UFL problem. We also improve the approximation ratio for the 3-level version of the problem. A preliminary version of these results appeared in [Byr07].

In Chapter 3, we study problems motivated by phylogenetics. Phylogenetics describes connections between all groups of living organisms by ancestor/descendant relationships. The evolutionary history of certain groups of species is modelled with phylogenetic networks. In Sections 3.2 and 3.3, we study problems of constructing phylogenetic trees/networks from data that is given in the form of small trees called triplets. In Sections 3.2, which is based on a joint work with Pawel Gawrychowski, Katharina Huber, and Steven Kelk [BGHK08], we show that, assuming a fixed geometry of a network, random placement of species is consistent with a certain fraction of the input data. As a result, we improve approximation ratios for constructing, so called, level-2 and level-3 networks. In Section 3.3, we speculate if a better than random solution is possible, at least under a certain restriction on the shape of the networks. A number of partial results is presented. Part of them will appear in [BGJ08]. Section 3.4 is dedicated to a related problem of comparing given trees. Rather than computing a distance measure, we propose to draw the given pair of binary trees in such a way, that the structural differences are easy to spot. Formally, we ask for a planar drawing of trees, such that if we connect corresponding leaves of each tree with straight lines, the number of crossings between these lines will be minimal. We give a 2-approximation algorithm for the full-binary case and show that, assuming that the Unique Games Conjecture holds, constant factor approximation for general binary trees in not possible. The section is based on a joint work with Kevin Buchin, Maike Buchin, Martin Nöllenburg, Yoshio Okamoto, Rodrigo Silveira, and Alexander Wolff [BBB$^+$08].

In Chapter 4, we consider the problem of computing approximate Nash equilibria in Noncooperative games. It has been recently proven, that finding an equilibrium is essentially as hard as finding a fixed point of a function. We give an algorithm that, in polynomial time, finds strategies that form a, so called, additive approximate Nash equilibrium. The chapter is based on a joint paper with Hartwig Bosse and Vangelis Markakis [BBM07].

In Chapter 5 we study a generalization of the standard Set Cover problem. The Set Cover problem is to cover a certain set of elements with a minimal number of subsets, chosen from the family of subsets given as an input. In the k-SetCover problem, each element needs to be covered at least k times, but each subset may be used many times. We give a constant factor approximation algorithm for the case where k is at least logarithmic in the number of elements to cover. The chapter is based on a joint work with Richard Beigel and Marcin Bienkowski. An earlier version of the argument was given in [BB05].

Chapter 2

Facility location

2.1 On facility location problems

Facility location problems concern situations where a planner needs to determine the location of facilities intended to serve a given set of clients. The objective is usually to minimize the sum of the cost of opening the facilities and the cost of servicing the clients by the facilities, subject to various constraints, such as the number and the type of clients a facility can serve. There are many variants of the facility location problem, depending on the structure of the cost function and the constraints imposed on the solution. Early references on facility location problems include Kuehn and Hamburger [KH63], Balinski and Wolfe [BW63], Manne [Man64], and Balinski [Bal66]. Review works include Krarup and Pruzan [KP83] and Mirchandani and Francis [MF90].

Perhaps the simplest version of the problem is the *uncapacitated facility location* (UFL) problem. This problem was first formulated by Kuehn and Hamburger [KH63] and has since been studied extensively in the operations research literature, see for instance [Bal66; Sto63; Erl78; Man64; CNW90; Vyg05; KP83; MF90]. It is interesting to notice that the algorithm that is probably one of the most effective ones to solve the UFL problem to optimality is the primal-dual algorithm combined with branch-and-bound due to Erlenkotter [Erl78] dating back to 1978. His primal-dual scheme is similar to techniques used in the modern literature on approximation algorithms.

More recently, extensive research on approximation algorithms for facility location problems has been carried out. Review articles on this topic include Shmoys [Shm00; Shm04] and Vygen [Vyg05]. Besides its theoretical and practical importance, facility location problems provide a showcase of common techniques in the field of approximation algorithms, as many of these techniques such as linear programming rounding, primal-dual methods, and local search have been applied successfully to this family of problems. This section defines several facility location problems, gives a few historical pointers, and lists approximation algorithms. Since the contributions of the author of this thesis are mainly to the UFL problem, the

techniques applied to this problem are discussed in some more detail.

2.1.1 Problem definition

In the UFL problem, we are given a set $\mathcal{F}$ of n_f *facilities* and a set $\mathcal{C}$ of n_c *clients* (also known as *cities*, or *demand points*). For every facility $i \in \mathcal{F}$, a nonnegative number f_i is given as the *opening cost* of i. Furthermore, for every facility $i \in \mathcal{F}$ and client $j \in \mathcal{C}$, we have a *connection cost* c_{ij}. The objective is to open a subset of the facilities and connect each client to an open facility so that the total cost is minimized. Notice that once the set of open facilities is specified, it is optimal to connect each client to the open facility that yields smallest connection cost. Therefore, the objective is to find a set $S \subseteq \mathcal{F}$ that minimizes $\sum_{i \in S} f_i + \sum_{j \in \mathcal{C}} \min_{i \in S}\{c_{ij}\}$. This definition and the definitions of other variants of the facility location problem in this section assume unit demand at each client. It is straightforward to generalize these definitions to the case where each client has a given demand. The UFL problem can be formulated as the following integer program due to Balinski [Bal66]. Let $y_i,\ i \in \mathcal{F}$ be equal to 1 if facility i is open, and equal to 0 otherwise. Let $x_{ij},\ i \in \mathcal{F},\ j \in \mathcal{C}$ be the fraction of client j assigned to facility i.

$$\min \sum_{i \in \mathcal{F}} f_i y_i + \sum_{i \in \mathcal{F}} \sum_{j \in \mathcal{C}} c_{ij} x_{ij} \tag{2.1}$$

$$\text{subject to} \sum_{i \in \mathcal{F}} x_{ij} = 1, \qquad \text{for all} \quad j \in \mathcal{C}, \tag{2.2}$$

$$x_{ij} - y_i \leq 0, \qquad \text{for all} \quad i \in \mathcal{F},\ j \in \mathcal{C}, \tag{2.3}$$

$$x_{ij} \geq 0,\ y_i \in \{0,1\} \qquad \text{for all} \quad i \in \mathcal{F},\ j \in \mathcal{C}. \tag{2.4}$$

Notice that in the uncapacitated case, it is not necessary to require $x_{ij} \in \{0,1\},\ i \in \mathcal{F},\ j \in \mathcal{C}$ if we want each client to be serviced by precisely one facility. We have $0 \leq x_{ij} \leq 1$ by constraints (2.2) and (2.4), and if x_{ij} is not integer, then it is always possible to create an integer solution with the same cost by assigning client j completely to one of the facilities currently servicing j. In the *linear programming (LP) relaxation* of UFL the constraint $y \in \{0,1\}^{n_f}$ is substituted by the constraint $y \in [0,1]^{n_f}$.

Hochbaum [Hoc82] developed an $O(\log n)$-approximation algorithm for UFL. By a straightforward reduction from the Set Cover problem, it can be shown that this cannot be improved unless $NP \subseteq DTIME[n^{O(\log \log n)}]$ due to a result by Feige [Fei98]. However, if the connection costs are restricted to come from distances in a metric space, namely $c_{ij} = c_{ji} \geq 0$ for all $i \in \mathcal{F}, j \in \mathcal{C}$ (nonnegativity and symmetry) and (triangle inequality)

$$c_{ij} + c_{ji'} + c_{i'j'} \geq c_{ij'} \qquad \text{for all } i, i' \in \mathcal{F}, j, j' \in \mathcal{C}, \tag{2.5}$$

then constant approximation guarantees can be obtained. In all results mentioned below, except for the maximization objectives, it is assumed that the costs satisfy these restrictions. If the distances between facilities and clients are Euclidean, then for some location problems approximation schemes have been obtained [ARR98].

The first approximation algorithm for the metric UFL problem with a constant performance guarantee was the 3.16-approximation algorithm by Shmoys, Tardos, and Aardal [STA97]. Currently, the best known approximation guarantee is 1.5 [Byr07], which is one of the main results of this thesis and will be presented in Section 2.3. We first briefly discuss other variants of the facility location problem and then continue with a small survey on approximation results for UFL in Section 2.1.3.

2.1.2 Variants and related problems

Many algorithms have been proposed for location problems. We will give a quick overview of some key results. Some of the algorithms giving the best values of the approximation guarantee γ are based on solving the LP-relaxation by a polynomial algorithm, which can actually be quite time consuming, whereas some authors have suggested fast combinatorial algorithms for facility location problems with less competitive γ-values. We have decided to focus on the algorithms that yield the best approximation guarantees. For more references we refer to the survey papers by Shmoys [Shm00; Shm04] and by Vygen [Vyg05].

max-UFL. A variant of the uncapacitated facility location problem is obtained by considering the objective coefficients c_{ij} as the per unit profit of servicing client j from facility i. The *maximization version* of UFL, **max-UFL** is obtained by maximizing the profit minus the facility opening cost, i.e., $\max \sum_{i\in\mathcal{F}} \sum_{j\in\mathcal{C}} c_{ij}x_{ij} - \sum_{i\in\mathcal{F}} f_i y_i$. This variant was introduced by Cornuéjols, Fisher, and Nemhauser [CFN77], who also proposed the first constant factor approximation algorithm. They showed that opening one facility at a time in a greedy fashion, choosing the facility to open as the one with highest marginal profit, until no facility with positive marginal profit can be found, yields a $(1-1/e) \approx 0.632$-approximation algorithm. The current best approximation factor is 0.828 by Ageev and Sviridenko [AS99].

k-median, k-center problem In this problem, the facility opening cost is removed from the objective function (2.1) to obtain $\min \sum_{i\in\mathcal{F}} \sum_{j\in\mathcal{C}} c_{ij}x_{ij}$, and the constraint that no more than k facilities may be opened, $\sum_{i\in\mathcal{F}} y_i \leq k$, is added. In the *$k$-center problem* the constraint $\sum_{i\in\mathcal{F}} y_i \leq k$ is again included, but the objective function is to minimize the maximum distance used on a link between an open facility and a client.

The first constant factor approximation algorithm for the k-median problem is due to Charikar, Guha, Tardos, and Shmoys [CGTS99]. This LP-rounding algorithm has the approximation ratio of $6\frac{2}{3}$. The currently best know approximation ratio is $3+\epsilon$ achieved by a local search heuristic of Arya, et al. [AGK$^+$01].

The first constant factor approximation algorithm for the k-center problem was given by Hochbaum and Shmoys [HS85], who developed a 2-approximation algorithm. This performance guarantee is the best possible unless $P = NP$.

Capacitated facility location problem. In this problem, a capacity constraint $\sum_{j \in \mathcal{C}} x_{ij} \leq u_i y_i$ is added for all $i \in \mathcal{F}$. Here it is important to distinguish between the *splittable* and the *unsplittable* case, and also between *hard capacities* and *soft capacities*. In the splittable case we have $x \geq 0$, so we allow for a client to be serviced by multiple depots, and in the unsplittable case we require $x \in \{0,1\}^{n_f \times n_c}$. If each facility can be opened at most once (i.e., $y_i \in \{0,1\}$), the capacities are called hard; otherwise, if the problem allows a facility i to be opened any number r of times to serve ru_i clients, (i.e., if y_i's are allowed to be non-negative integers) the capacities are called soft capacities.

For the soft-capacitated problem with equal capacities the first constant factor approximation algorithms are due to Shmoys et al. [STA97] for both the splittable and unsplittable demand cases. Recently, a 2-approximation algorithm for the soft capacitated facility location problem with unsplittable unit demands was proposed by Mahdian et al. [MYZ06]. The integrality gap of the LP relaxation for the problem is also 2. Hence, to improve the approximation guarantee one would have to develop a better lower bound on the optimal solution.

In the hard capacities version it is important to allow for splitting the demands as otherwise even the feasibility problem becomes difficult. Suppose demands are splittable, then we still distinguish between the equal capacity case, where we have $u_i = u$ for all $i \in \mathcal{F}$, and the general case. For the problem with equal capacities, a 5.83-approximation algorithm was given by Chudak and Wiliamson [CW99]. Subsequently, Levi et al. [LSS04] gave a 5-approximation LP-rounding algorithm.

The first constant factor approximation algorithm, with $\gamma = 8.53 + \epsilon$, for general capacities was given by Pál, Tardos, and Wexler [PTW01]. This was later improved by Zhang, Chen, and Ye [ZCY05] who obtained a 5.83-approximation algorithm also for general capacities.

k-level facility location. In this problem, we are given a set $\mathcal{C}$ of clients, k disjoint sets $\mathcal{F}_1, \ldots, \mathcal{F}_k$ of facilities, an opening cost for each facility, and connection costs between clients and facilities. The goal is to connect each client j through a path $i_1, \ldots, i_k$ of open facilities, with $i_\ell \in \mathcal{F}_\ell$. The connection cost for this client is $c_{ji_1} + c_{i_1 i_2} + \cdots + c_{i_{k-1} i_k}$. The goal is to minimize the sum of connection costs and facility opening costs.

The first constant factor approximation algorithm for $k = 2$ is due to Shmoys et al. [STA97], with $\gamma = 3.16$. For general k, the first algorithm, having $\gamma = 3$, was proposed by Aardal, Chudak, and Shmoys [ACS99]. For $k = 2$, Zhang [Zha04] developed a 1.77-approximation algorithm. He also showed that the problem for $k = 3$ and $k = 4$ can be approximated by $\gamma = 2.523$ [1] and $\gamma = 2.81$ respectively. In this thesis we also improve the approximation factor for the 3-level UFL problem. Namely, we give a 2.492-approximation algorithm (see Theorem 2.3.11 in Section 2.3.6).

[1] This value of γ deviates slightly from the value 2.51 given in the paper. The original argument contained a minor calculation error.

Fault tolerant facility location. In this problem (first defined in [JV00]), we are given a connectivity requirement r_j for each client j, which specifies the number of distinct open facilities to which client j should be connected.

Facility location with outliers. In this problem, a number l is given as part of the input, and the solution is only required to connect $n_c - l$ clients to open facilities. This variant and the next one were defined by Charikar et al. [CKMN01].

Prize collecting facility location. In this problem, for each client j, there is a specified penalty p_j. A solution has the option of either connecting j to an open facility, or paying this penalty. The goal is to minimize the sum of the penalties paid, the facility costs, and the connection costs.

k-facility location problem. This problem (defined in [KP90]) is a common generalization of k-median and UFL, where in addition to facility opening costs, an upper bound k on the number of facilities that can be opened is imposed.

Universal facility location. In this problem, the cost of each facility $i \in \mathcal{F}$ is given as a function $f_i : \{0, \ldots, n_c\} \mapsto \mathbb{R}^{\geq 0} \cup \{\infty\}$ with $f_i(0) = 0$, where the value of $f_i(k)$ indicates the cost of opening facility i, when it is used to serve k clients. Formally, a solution to this problem can be represented as a function $\varphi : \mathcal{C} \to \mathcal{F}$ that assigns each client to a facility. The cost of this solution is the sum of the facility cost $\sum_{i \in \mathcal{F}} f_i(|\{j : \varphi(j) = i\}|)$ and the connection cost $\sum_{j \in \mathcal{C}} c_{\varphi(j),j}$. The universal facility location problem (first defined in [HMM03; MP03]) generalizes several variants of the facility location problem such as UFL (where the function $f_i(u)$ is constant for $u > 0$), hard-capacitated facility location (where $f_i(u)$ is a constant for $0 < u \leq u_i$ and is infinity for larger values of u), and soft-capacitated facility location (where $f_i(u)$ is the opening cost of the facility times $\lceil u/\bar{u}_i \rceil$).

There are many other variants of the facility location problem that we will not discuss here. Examples include online facility location [Mey01; Fot03; ABUvH04], multicommodity facility location [RS04], priority facility location [RS04; Mah04], facility location with hierarchical facility costs [ST06], stochastic facility location [RS06; Mah04; GPRS04], connected facility location [SK04], load-balanced facility location [GMM00; KM00; Mah04], concave-cost facility location [HMM03], and capacitated-cable facility location [RS02; Mah04].

2.1.3 Algorithms for UFL

We will now focus on the metric Uncapacitated Facility Location problem (which we call UFL). Many algorithms have been proposed for this problem. They include heuristics, approximation algorithms and exact algorithms. We concentrate on the approximation algorithms and give a quick overview of the important classical results and the current state of knowledge. Eventually, we will conclude this section with

the statement of the author's main contributions, whose technical expositions will be given in the two subsequent sections.

Before we start with the description of algortihms having a provable approximation guarantee, we mention an early primal-dual algorithm for UFL proposed by Erlenkotter [Erl78]. This algorithm is computationally efficient, but no analysis of the quality of the solutions it produces was provided. Nevertheless, many later algorithms use similar ideas.

LP-rounding algorithms. The LP-rounding algorithms first solve the LP-relaxation of the problem, and then round the (potentially fractional) solution to an integral one. Such rounding maintains the feasibility of the solution, i.e., no constraint gets violated, but also potentially increases the cost of the solution. In the analysis, one needs to prove that this cost increase is bounded, in this case by a multiplicative constant. The following two techniques are essential in construction of LP-rounding algorithms for the UFL problem.

Filtering. A method of modifying the LP-solution, called *filtering*, was introduced by Lin and Vitter [LV92]. Lin and Vitter considered a broad class of 0-1 problems having both covering and packing constraints. They start by solving the LP-relaxation of the problem, and in the subsequent filtering step they select a subset of the variables that have positive value in the LP solution and that have relatively large objective coefficients. These variables are set equal to zero, which results in a modified problem. The LP-relaxation of this modified problem is then solved and rounding is applied.

In the paper by Shmoys et al. [STA97] filtering was used in order to bound the connection costs. Here again a subset of the variables that have a positive value in the LP-solution are set equal to zero. The remaining positive variables were scaled so as to remain feasible for the original LP-relaxation. Another way to describe this process is as follows. The facility opening variables y_i are scaled up by a constant $\gamma > 1$ and then the connection variables x_{ij} are adjusted to use the closest possible facilities. In case the scaling factor γ is chosen randomly (from some specific distribution) we use the name *randomized filtering*.

The filtering done in our new algorithm (see Section 2.3) is slightly different as the filtered LP-solution is not necessarily feasible with respect to the LP-relaxation. There we will use the name *sparsening technique* for the combination of filtering with our new analysis.

Clustering. The name *clustering* was first used in [CS03], but the technique was already applied in the algorithm of Shmoys et al. [STA97]. Based on the fractional solution, the instance is cut into pieces called *clusters*, each of them having a distinct client called the *cluster center*. This is done by iteratively picking a client, not covered by the previous clusters, as the next cluster center, and by adding the facilities that serve the cluster center in the fractional solution, along with other clients served by these facilities to this cluster. The construction of clusters guarantees that the facilities in each cluster are open to a total extent of one, and therefore

after opening the facility with the smallest opening cost in each cluster, the total facility opening cost paid does not exceed the facility opening cost of the fractional solution. Moreover, by choosing clients for the cluster centers in a greedy fashion, the algorithm makes each cluster center the minimizer of a certain cost function among the clients in the cluster.

Having the clusters computed, the algorithm opens one facility for each cluster center. The facility opening variables for facilities fractionally serving a cluster center sum up to 1, hence the cost of the facilities opened by the algorithm may be bounded with the facility opening cost in the fractional solution. Each client in the cluster may now be connected, via the cluster center, to the open facility. The triangle inequality for connection costs bounds the cost of such a connection.

Algorithms. The first constant factor approximation algorithm for the UFL problem was given by Shmoys, Tardos, and Aardal [STA97]. It solves the LP relaxation and then modifies the obtained fractional solution as follows. First, the filtering technique is applied to change the fractional solution. Then, based on this modified fractional solution the instance is divided into clusters. Eventually, clients are connected to facilities, which are open in each cluster as described above. For UFL, this filtering and rounding algorithm is a 4-approximation algorithm. Shmoys et al. also show that if the filtering step is substituted by randomized filtering, an approximation guarantee of 3.16 is obtained.

As was observed later [ACS99], the filtering stage may be skipped, and instead, the values of the optimal LP-dual variables may be used to bound the maximal connection cost between a client and a facility fractionally serving him. Such a modified analysis gives an approximation factor of 3.

Since the algorithm of Shmoys et al. a couple of modifications to the rounding procedure have been proposed. Chudak and Shmoys [Chu98; CS03] developed a randomized version of the rounding procedure, which improved the approximation factor to $1 + 2/e \approx 1.736$. Later, Sviridenko [Svi02] combined randomized filtering with randomized rounding. By a complicated analysis, he proved that the resulting algorithm has the approximation ratio equal 1.58.

We propose yet another rounding procedure that is similar to the algorithm of Sviridenko, but uses a new argument in the analysis. The quality of the resulting algorithm is best described in terms of the bifactor approximation language (defined below). In particular, we prove that our algorithm is a (1.6774,1.3738)-approximation algorithm for the UFL problem. This property makes it possible for us to combine this algorithm with the algorithm of Mahdian et al. [MYZ06] to obtain a 1.5-approximation for the UFL problem.

Primal-dual algorithms. Another group of algorithms for the UFL problem is formed by the, so called, *primal-dual algorithms*. These algorithms, unlike the LP-rounding algorithms, do not solve the LP-relaxation of the problem. Nevertheless, the LP-relaxation is used in the analysis.

The main idea is to construct a feasible (integral) solution to the problem, using complementary slackness, and a certificate that the solution is not too expensive

in comparison with an optimal solution. The role of the certificate is played by a feasible solution to the dual of the LP-relaxation of the problem. Such a dual-feasible solution is, by weak LP duality, not more expensive than an optimal primal feasible solution to the LP-relaxation, which is not more expensive than an optimal integral solution.

A well known example of a primal dual algorithm for UFL is the algorithm of Jain and Vazirani [JV01]. It is only a 3-approximation algorithm, but its simplicity makes it possible to apply the algorithm, or its minor modification, to many variants of the facility location problem.

Another example is the algorithm of Jain, Mahdian and Saberi [JMS02] (the JMS algorithm). It is somewhat more careful when constructing the solution, which yields a better approximation guarantee of 1.61. The price that is paid for this accuracy is a more complicated analysis. To prove this approximation guarantee, the authors needed to model a potentially bad instance by another linear program, which they call the *factor revealing LP*. This LP models an instance of a fixed size. Finally, they had to prove that for any size of an instance, the resulting LP has a bounded value. For more details of this algorithm see Section 2.2.1.

An important property of the primal-dual algorithms is their running time. They are relatively simple to implement, and perform only simple operations, which is often emphasized by calling them *combinatorial algorithms*. It was proved [MYZ03] that the JMS algorithm may be implemented in *quasilinear* time.

Bifactor approximation. Before we proceed with the other algorithms it is useful to introduce one more piece of notation. The UFL problem has the particular property that the objective function is a sum of two different types of cost, the facility opening cost and the connection cost. While it is trivial to optimize only one of these costs, by simply opening only one or all the facilities, it is challenging to produce solutions with neither of the two costs being too big. We are facing a tradeoff situation, where it is to be decided how much of the cost should be allocated for opening facilities, and how much for connecting clients. Therefore, in the context of the UFL problem, we consider the following definition of bifactor approximation.

Definition 2.1.1 (bifactor approximation algorithm) *An (λ_f, λ_c)-approximation algorithm for the UFL problem is a polynomial time algorithm that produces a solution whose cost is bounded by $\lambda_f \cdot F^* + \lambda_c \cdot C^*$ for any feasible solution with facility cost F^* and connection cost C^*.*

Note that, unlike in the definition of the standard approximation algorithm, the comparison in the Definition 2.1.1 is not equivalent to bounding the cost in terms of the cost of an optimal solution. If we only bound the cost in terms of a particular solution, e.g. a fractional solution to the LP-relaxation of the problem, we obtain a potentially weaker result. It may be seen as a consequence of the fact that by scaling the facility opening costs in an instance we may change the optimal solution. The above definition requires the produced solutions to be good in comparison with any

feasible solution, and therefore also in comparison with solutions that are optimal for different objective functions.

An important result that supports the use of bifactor approximation algorithm for UFL is an approximation hardness result by Jain et al. [JMS02]. They proved that there exists no $(\lambda_f, 1 + 2e^{-\lambda_f} - \epsilon)$-approximation algorithm for UFL for any positive ϵ and $\lambda_f \geq 1$ unless $NP \subset DTIME(n^{\log \log n})$. Note the asymmetry in this hardness bound: it appears to be easier to approximate the facility opening cost than the connection cost.

Scaling and greedy augmentation. Besides the algorithms that produce new solutions from scratch, there are also techniques for modifying existing solutions for the UFL problem.

The *greedy augmentation* technique introduced by Guha and Khuller [GK98] (see also [CG99]) is the following. Consider an instance of the metric UFL problem and a feasible solution. For each facility $i \in \mathcal{F}$ that is not opened in this solution, we may compute the impact of opening facility i on the total cost of the solution, also called the *gain* of opening i, denoted by g_i. While there is a facility i with positive gain g_i, the greedy augmentation procedure opens a facility i_0 that maximizes the ratio of saved cost to the facility opening cost $\frac{g_i}{f_i}$, and updates values of g_i. The quality of the resulting solution is expressed in the form of the following lemma.

Lemma 2.1.2 ([CG99]) *For every instance I of the metric UFL problem, and for every solution SOL of I with facility cost F_{SOL} and connection cost C_{SOL}, if an initial solution has facility cost F and connection cost C, then after greedy augmentation the cost of the solution is at most*

$$F + F_{SOL} \max\{0, ln(\frac{C - C_{SOL}}{F_{SOL}})\} + F_{SOL} + C_{SOL}$$

This technique may be applied to a trivial solution, e.g. to a solution with only the cheapest facility opened, to produce an approximation algorithm. For some specific instances of the problem such an algorithm would actually be a pretty good one. In particular, it is an optimal 1.463..-approximation algorithm for instances with connection costs being equal either 1 or 3. Notice that such instances are produced by a reduction from the Set Cover problem proving 1.463.. hardness of approximation of the UFL problem.

Nevertheless, the greedy augmentation technique appears even more useful when applied together with a scaling of the cost function. Suppose we are given an approximation algorithm A for the metric UFL problem and a real number $\delta \geq 1$. Consider the following algorithm $S_\delta(A)$.

1. scale up all facility opening costs by a factor δ;
2. run algorithm A on the modified instance;
3. scale back the opening costs;

4. run the greedy augmentation procedure.

Following the analysis of Mahdian, Ye, and Zhang [MYZ06] one may prove the following lemma.

Lemma 2.1.3 *Suppose A is a (λ_f,λ_c)-approximation algorithm for the metric UFL problem, then $S_\delta(A)$ is a ($\lambda_f + ln(\delta)$,$1 + \frac{\lambda_c - 1}{\delta}$)-approximation algorithm for this problem.*

This method may be applied to balance a (λ_f,λ_c)-approximation algorithm with $\lambda_f << \lambda_c$, i.e., to obtain a (λ, λ)-approximation algorithm with $\lambda_f < \lambda < \lambda_c$. It was used by Mahdian et al. [MYZ06].

2.1.4 Our contribution to facility location approximation algorithms.

It is clear from the presentation above that many different approximation algorithms had been developed for the UFL problem before we started our work. But, there was still an approximation gap between the 1.463..-hardness of approximation by Guha and Kuller and the 1.52-approximation algorithm of Mahdian et al. The best LP-rounding result was the 1.58-approximation by Sviridenko.

In our first attempt to close the gap, we tried to improve the complicated analysis of Mahdian et al. As a result we proved that the approximation gap cannot be closed by just improving the analysis of the MYZ algorithm. This result is described in Section 2.2.

Next, we attempted to construct an improved LP-rounding algorithm. The primary goal was to obtain an algorithm with an approximation ratio better than 1.52. We failed to achieve this goal with a single LP-rounding algorithm, but we constructed an LP-rounding algorithm which, combined with the primal-dual JMS algorithm, gives the approximation ratio 1.5. This is currently the best known approximation ratio. A detailed presentation of this algorithm is given in Section 2.3. With this result, we have decreased the approximation gap by 1/3. The obvious remaining open problem is to close this gap completely. We hope that our contribution will help to eventually achieve this goal.

2.2 The algorithm of Mahdian et al. is not optimal

In the paper describing the 1.52-approximation algorithm, Mahdian et al. [MYZ02] posed the following two questions.

> "It might be possible to apply a method similar to the one used in [JMS02] (i.e., deriving a factor-revealing LP and analyzing it) to analyze both phases of our algorithm in one shot. This might give us a tighter bound on the approximation factor of the algorithm."

> "The important open question is whether or not our algorithm can close the gap with the approximability lover bound of 1.463."

In a later paper Mahdian et al. [MYZ03] answered the first question in the affirmative by presenting a new analysis of their algorithm using a single factor-revealing LP. In this section we provide another single factor-revealing LP that makes it possible to give a negative answer to the second open question by providing an instance of the metric UFL problem for which their algorithm returns a solution that is 1.4943 times more expensive than the optimal solution.

2.2.1 The algorithm

Let us first recall the following 1.62-approximation algorithm of Jain, Mahdian and Saberi [JMS02] (the JMS algorithm).

1. Set all clients to be not connected, all facilities closed, and the *budget* of every client j, denoted by B_j, to be equal 0. At every moment, each client j offers some money from its budget to each closed facility i. The amount offered equals $\max(B_j - c_{ij}, 0)$ if j is not connected, and $\max(c_{i'j} - c_{ij}, 0)$ if it is connected to some other facility i'.

2. While there is a not connected client, increase the budget of each such client at the same rate, until one of the following events occurs:

   (a) For some closed facility i, the total offer that it receives from clients is equal to the cost f_i of opening i. In this case, we open facility i, and for every client j (connected or not connected) with a non-zero offer to i, we connect j to i.

   (b) For some not connected client j, and some facility i that is already open, the budget of j is equal to the connection cost c_{ij}. In this case, we connect j to i.

3. For every client j, set α_j equal to the budget of j at the end of the algorithm.

The original analysis of the JMS algorithm [JMS02] shows that it is a 1.61-approximation algorithm for the UFL problem. For us a more important property of the JMS algorithm is that it is a $(1.11, 1.78)$-approximation algorithm for the metric UFL problem, i.e., for any instance I of the metric UFL problem, and any feasible solution with facility cost F_{SOL} and connection cost C_{SOL}, the cost of a solution computed by the JMS algorithm for I is at most $1.11F_{SOL} + 1.78C_{SOL}$. The proof of this bifactor approximation guarantee is highly nontrivial and technical. We refer an interested reader to the original work of the authors [MYZ02; MYZ03; MYZ06; Mah04].

Recall the *scaling + greedy augmentation* technique described in the previous section. The cost asymmetry of the JMS algorithm and properties of the greedy augmentation are exploited by the following algorithm of Mahdian, Ye and Zhang [MYZ02] (the MYZ algorithm).

1. Scale up the facility opening costs by a factor $\delta \geq 1$.

2. Run the JMS algorithm.

3. Scale back the facility opening costs (multiply them by a factor $\frac{1}{\delta}$).

4. Run the greedy augmentation procedure [CG99]

An equivalent formulation [MYZ03] of the MYZ algorithm describes the last two steps as gradually scaling back the facility costs and opening any facility that may be opened without increasing the total cost.

In the language of Lemma 2.1.3, the MYZ algorithm is the $S_\delta(JMS)$ algorithm. Lemma 2.1.3 together with the $(1.11, 1.78)$ approximation guarantee of the JMS algorithm implies that the MYZ algorithm with $\delta = 1.504$ is a (1.52,1.52)-approximation algorithm for the metric UFL problem.

2.2.2 Analysis: lower bound

The original analysis of the MYZ algorithm does not give an easy way to produce an instance of the metric UFL problem that would be "difficult" for the algorithm, and it was not clear how tight the analysis actually was. The difficulty comes from the fact that three techniques, namely a factor revealing LP, cost scaling, and greedy augmentation, are combined to get the 1.52 approximation ratio. Mahdian et al. [MYZ03] presented a new analysis of the algorithm that uses a single LP to prove the 1.52-approximation factor, but still there was no tight instance example provided.

Both the JMS and the MYZ algorithm are analyzed by modeling their local behavior. For every *star*, i.e., one facility and a subset of clients, the solution is proved to be locally not too expensive (see [JMS02] for details). Suppose that the analyzed star has a facility i with opening cost f, that the distance between client j and facility i is equal to d_j, and that the JMS algorithm ends with clients' budgets α_j, assigning client j to a facility at distance $r_{j,j'}$ at the moment of first assignment of client j'. Then the following LP, called a factor revealing LP, models this situation while maximizing the "local" proportion of the cost of the computed solution to any fractional solution cost.

$$
\begin{aligned}
& \max \frac{\sum_{j=1}^{k} \alpha_j}{f + \sum_{j=1}^{k} d_j} \\
\text{subject to} \quad & \alpha_j \le \alpha_{j+1} && \text{for all } 1 \le j < k \\
& r_{j,j+1} \ge r_{j,j+2} \ge \ldots \ge r_{j,k} && \text{for all } 1 \le j \le k \\
& \alpha_{j'} \le r_{j,j'} + d_j + d_{j'} && \text{for all } 1 \le j < j' \le k \\
& \sum_{j'=1}^{j-1} \max(r_{j',j} - d_{j'}, 0) + \sum_{j'=j}^{k} \max(\alpha_j - d_{j'}, 0) \le f && \text{for all } 1 \le j \le k \\
& f, d_j, \alpha_j, r_{j,j'} \ge 0 && \text{for all } 1 \le j \le j' \le k
\end{aligned}
$$

For our analysis of the MYZ algorithm we define another mathematical program and its optimal value as follows.

Definition 2.2.1 *For every $\delta \in \mathcal{R}_+$ and $k \in \mathcal{N}_+$, let $z(\delta, k)$ be defined as the the value of an optimal solution to the following LP, later referred to as the model LP.*

$$\max \frac{\frac{1}{\delta} \cdot \sum_{j=1}^{k} \alpha_j + (1 - \frac{1}{\delta}) \cdot \sum_{j=1}^{k} r_j}{\frac{1}{\delta} \cdot f + \sum_{j=1}^{k} d_j}$$

$$\text{subject to} \quad \alpha_j \leq \alpha_{j+1} \quad \text{for all} \quad 1 \leq j < k$$

$$\alpha_{j'} \leq r_j + d_j + d_{j'} \quad \text{for all} \quad 1 \leq j < j' \leq k \tag{2.6}$$

$$\sum_{j'=1}^{j-1} \max(r_{j'} - d_{j'}, 0) + \sum_{j'=j}^{k} \max(\alpha_j - d_{j'}, 0) \leq f \quad \text{for all} \quad 1 \leq j \leq k \tag{2.7}$$

$$\frac{f}{\delta} \geq \sum_{j=1}^{k} \max(r_j - d_j, 0) \tag{2.8}$$

$$r_j \leq \alpha_j \quad \text{for all} \quad 1 \leq j \leq k \tag{2.9}$$

$$f, d_j, \alpha_j, r_j \geq 0 \quad \text{for all} \quad 1 \leq j \leq k$$

To incorporate scaling into the analysis, the variables of our model LP represent parameters of a scaled instance, and the objective value represents the cost of the scaled back solution.

In order to derive a lower bound on the approximation factor of the algorithm, we add constraint (2.8). This constraint implies that the instance of the metric UFL problem that we will read from the solution of our LP is such that the MYZ algorithm does not open any facility during the augmentation phase.

One more difference between the two LP models is that the latter one uses r_j variables instead of the double-indexed $r_{j,j'}$ variables used in the original factor revealing LP of the JMS algorithm. This change may be interpreted as an assumption that clients do not change facilities they are assigned to during the JMS phase of the MYZ algorithm. This assumption did not affect the solution values obtained from the model LP. Even if it would affect the optimum value, it would not matter as we are computing a lower bound on the performance factor of the algorithm.

Lemma 2.2.2 *For every $\delta \in \mathcal{R}_+$ and every $k \in \mathcal{N}_+$, there exists an instance I of the metric UFL problem with k clients and $k + 1$ facilities, such that the solution computed by the MYZ algorithm for the instance I is $z(\delta, k)$ times more expensive than the optimal solution to the instance I.*

Proof: Consider the instance illustrated in Figure 2.1, where the variable values form an optimal solution to the model LP. It contains $k + 1$ facilities and k clients. Constraints (2.9) of the model LP imply that values $(\alpha_j - r_j)/\delta$ are positive.

The optimal solution is to open the left facility with opening cost $\frac{1}{\delta} \cdot f$, the connection cost is then $\sum_{j=1}^{k} d_j$, and the total cost of the optimal solution is the

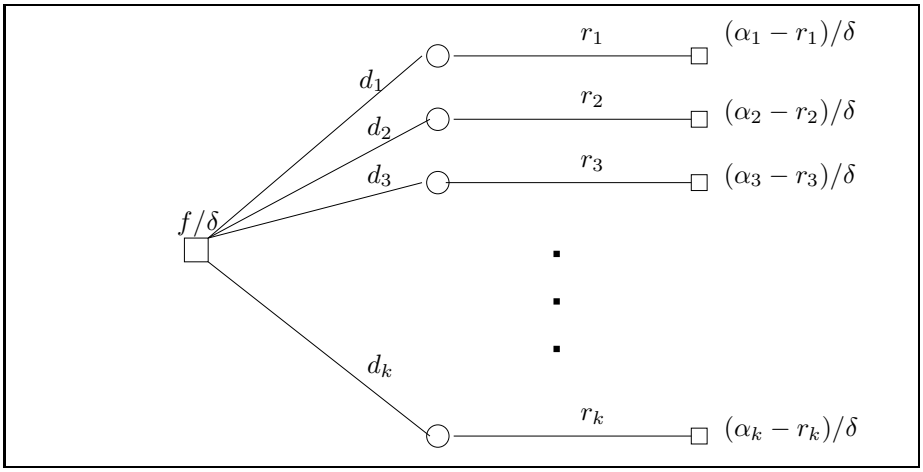


Figure 2.1: An instance of the metric UFL problem. Squares represent facilities, circles represent clients. The numbers are facility opening costs and connection costs. (Remaining connection costs are defined to be shortest paths' costs in the given graph)

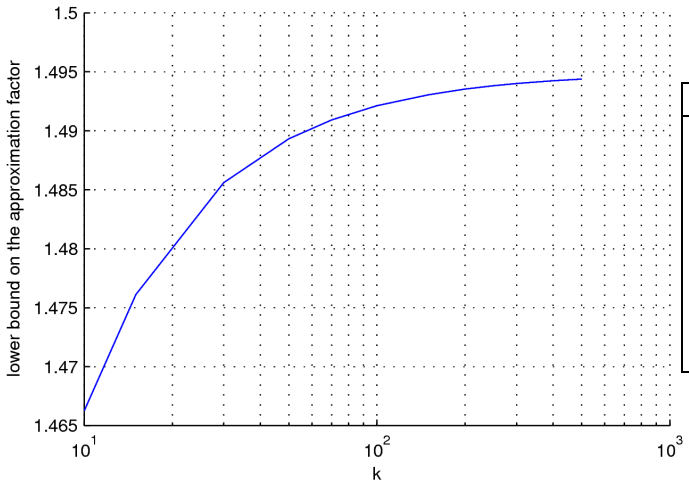
denominator of the objective function in the model LP. The numerator is the cost of the solution that opens all the other (right) facilities. It has connection cost $\sum_{j=1}^{k} r_j$ and facility cost $\frac{1}{\delta} \cdot (\sum_{j=1}^{k} \alpha_j - \sum_{j=1}^{k} r_j)$.

It remains to observe that the MYZ algorithm, when running on this instance, may open exactly all the right facilities. Constraints (2.7) imply that whenever there is enough budget offered to the left facility, there is a right facility with enough to offer as well. Constraints (2.6) imply that whenever the budget of client j is high enough to connect to an already opened facility, it also suffices to open another facility with scaled cost $(\alpha_j - r_j)$. Additionally, constraint (2.8) implies that the left facility is not opened during the scaling back and augmentation phase.

□

In the above proof we say that the MYZ algorithm may have a $z(\delta, k)$ error, but we may modify the instance, so that the MYZ algorithm would have to have a $z(\delta, k) - \epsilon$ error. In the analysis of the greedy augmentation we neglect the gain we could possibly get from closing unused facilities, and indeed, for our example closing facilities would solve the problem. However, we may easily construct an instance out of many copies of the instance that we have just analyzed, such that the MYZ algorithm has a $z(\delta, k) - \epsilon$ error even if we allow the algorithm to close unused facilities during the augmentation phase. Moreover, if we use a generalized augmentation that is allowed to open some constant number of facilities at once, there are still instances for which the modified MYZ algorithm has a $z(\delta, k) - \epsilon$ error.

In the following we analyze how "bad" our instances of the metric UFL problem



k	$z(1.504, k)$
2	1.335106383
5	1.441877216
10	1.466269191
50	1.489333800
100	1.492135015
200	1.493540895
300	1.494012828
500	1.494389666

Figure 2.2: Analysis of the approximation factor of the MYZ algorithm with $\delta = 1.504$ on single star instances (k is the number of clients in the instance).

may be for the MYZ algorithm with $\delta = 1.504$. The growth of the approximation factor with the instance size is presented in Figure 2.2. The figure suggests that the actual limit of the growth is somewhere between 1.495 and 1.50, which is essentially below 1.52, but on the other hand, quite above 1.463.. .

In Figure 2.3, the dependence of our analysis on the value of the scaling factor δ is presented. We may observe that $z(\delta, 50)$ has a local minimum at $\delta = 1.75$ and a local maximum at $\delta = 1.92$. It would be interesting to understand why these two values are so special. Since what we provide is a lower bound on the approximation factor of the MYZ algorithm, we may conclude that for $\delta \leq 2.3$ it does not close the gap with the inapproximability bound of 1.463.. . For larger values of δ our analysis is inappropriate, because then the algorithm is dominated by greedy augmentation and we did not model this precisely enough.

To conclude, our analysis of the algorithm of Mahdian at al. shows that the approximation factor is not better than 1.494369. The difficulty growth with the instance size is presented in figure 2.2. The figure suggests, that the actual limit of the growth is somewhere between 1.495 and 1.50, which is essentially below 1.52, but on the other hand, quite above (1.463..). As a consequence, the approximation gap may not be closed just by improving the analysis of this algorithm.

The way Mahdian at al. have modified the JMS algorithm can be seen as a method to make the algorithm also suitable for instaces dominated by connection costs. In the next section, we give a new LP-rounding algorithm which can be seen as an alternative solution to these instances. While JMS gives good approximation for instances dominated by facility opening costs, the new algorithm is better when

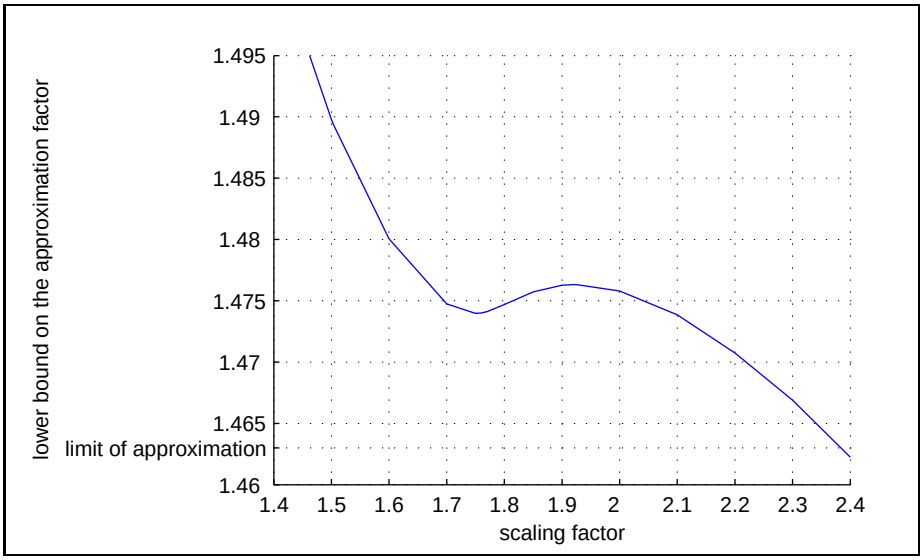


Figure 2.3: Analysis of the approximation factor of the MYZ algorithm on single star instances for different values of the scaling factor δ. ($k = 50$)

connection costs are critical. As a result the combination of these two algorithms will give an improved approximation ratio.

2.3 A new greedy rounding algorithm

2.3.1 Outline

Here, we modify the $(1 + 2/e)$-approximation algorithm of Chudak [Chu98], see also Chudak and Shmoys [CS03], to obtain a new (1.6774,1.3738)-approximation algorithm for the UFL problem. Our linear programming (LP) rounding algorithm is the first one that achieves an optimal bifactor approximation due to the matching lower bound of $(\lambda_f, 1 + 2e^{-\lambda_f})$ established by Jain et al. [JMS02]. In fact we obtain an algorithm for each point $(\lambda_f, 1+2e^{-\lambda_f})$ such that $\lambda_f \geq 1.6774$, which means that we have an optimal approximation algorithm for instances dominated by connection cost (see Figure 2.4).

One could view our contribution as an improved analysis of a minor modification of the algorithm by Sviridenko [Svi02], which also introduces filtering to the algorithm of Chudak and Shmoys. The filtering process that is used both in our algorithm and in the algorithm by Sviridenko is relatively easy to describe, but the analysis of the impact of this technique on the quality of the obtained solution is quite involved in each case. Therefore, we prefer to state our algorithm as an appli-

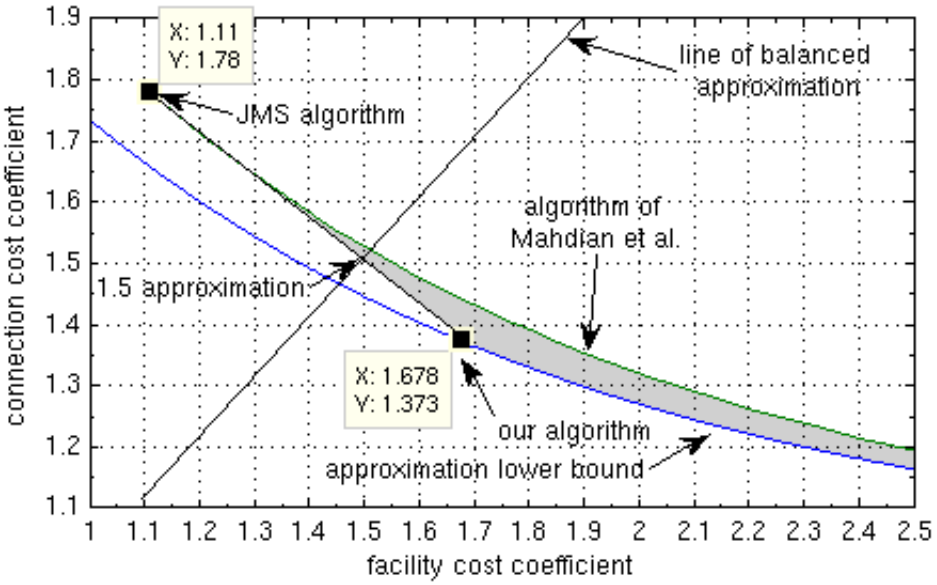


Figure 2.4: Bifactor approximation picture. The gray area corresponds to the improvement due to our algorithm.

cation of the sparsening technique to the algorithm of Chudak and Shmoys, which in our opinion is relatively easy do describe and analyze.

We start by observing that for a certain class of instances the analysis of the algorithm of Chudak and Shmoys may be improved. We call these instances *regular*, and for the other instances we propose a measure of their *irregularity*. The goal of the sparsening technique is to explore the irregularity of instances that are potentially tight for the original algorithm of Chudak and Shmoys. We cluster the given instance in the same way as in the 1.58-approximation algorithm by Sviridenko [Svi02], but we continue our algorithm in the spirit of Chudak and Shmoys' algorithm, and we use certain average distances to control the irregularities, which leads to an improved bifactor approximation guarantee.

Our new algorithm may be combined with the $(1.11, 1.7764)$-approximation algorithm of Jain et al. to obtain a 1.5-approximation algorithm for the UFL problem. This is an improvement over the previously best known 1.52-approximation algorithm of Mahdian et al., and it cuts of a $1/3$ off the gap with the approximation lower bound by Guha and Khuller [GK98].

In Section 2.3.2 we give a brief overview of the main ingredients of some known approximation algorithms for UFL. In particular we state the LP relaxation of UFL, and describe clustering in more details. Sparsening of the support graph of the LP solution is discussed in Section 2.3.3, where we also prove a lemma that

provides a bound on certain connection costs. Our bifactor algorithm together with its analysis is presented in Section 2.3.4, and the 1.5-approximation algorithm is stated in Section 2.3.5. In Section 2.3.6 we show that the new $(1.6774, 1.3738)$-approximation algorithm may be used to improve the approximation ratio for the 3-level version of the UFL problem to 2.492. A randomized approach to clustering is discussed in Section 2.3.7, and, finally, in Section 2.3.8 we present some concluding remarks and open problems.

2.3.2 Preliminaries

In Section 2.1.3 we gave a brief overview of the concept of LP-rounding algorithms for the metric UFL problem. We now need to dive into a somewhat more detailed description.

Recall that these algorithms first solve the linear relaxation of a given integer programming (IP) formulation of the problem, and then round the fractional solution to produce an integer solution with a value not too much higher than the starting fractional solution. Since the optimal fractional solution is at most as expensive as an optimal integral solution, we obtain an estimation of the approximation factor.

IP formulation and relaxation

Recall the IP formulation (2.1-2.4) of the UFL problem.

$$\begin{aligned} \min \sum_{i\in\mathcal{F}} f_i y_i + \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{C}} c_{ij}x_{ij} & \\ \text{subject to} \sum_{i\in\mathcal{F}} x_{ij} = 1, \qquad & \text{for all} \quad j \in \mathcal{C}, \\ x_{ij} - y_i \le 0, \qquad & \text{for all} \quad i \in \mathcal{F},\ j \in \mathcal{C}, \qquad (2.3) \\ x_{ij} \ge 0,\ y_i \in \{0,1\} \qquad & \text{for all} \quad i \in \mathcal{F},\ j \in \mathcal{C}. \qquad (2.4) \end{aligned}$$

A linear relaxation of this IP formulation is obtained by replacing the integrality constraints (2.4) by the constraint

$$x_{ij} \ge 0 \quad \text{for all } i \in \mathcal{F}, j \in \mathcal{C}. \tag{2.10}$$

Note, that $y_i \ge 0$ is implied by constraints (2.3) and (2.10). We do not explicitly require $y_i \le 1$. However, in any optimal fractional solution we have $y_i \le 1$ assuming $f_i > 0$. In one of the intermediate fractional solutions that are considered in the rounding procedure that we will describe in Section 2.3.3 the y variables may potentially take values grater that 1.

The value of the solution to the above LP relaxation will serve as a lower bound for the cost of the optimal solution. We will also make use of the following dual formulation of this LP.

$$\begin{aligned} \max & \sum_{j \in \mathcal{C}} v_j & \\ \text{subject to} & \sum_{j \in \mathcal{C}} w_{ij} \leq f_i & \text{for all } i \in \mathcal{F}, \\ & v_j - w_{ij} \leq c_{ij} & \text{for all } i \in \mathcal{F}, j \in \mathcal{C}, \\ & w_{ij} \geq 0 & \text{for all } i \in \mathcal{F}, j \in \mathcal{C}\,. \end{aligned}$$

Clustering

We now recall and describe in more detail the concept of clustering that was already discussed in Section 2.1.3. Suppose we are given an optimal solution to the LP relaxation of our problem. Consider the bipartite graph $G = ((V', V''), E)$ with vertices V' being the facilities and V'' the clients of the instance, and where there is an edge between a facility $i \in V'$ and a client $j \in V''$ if the corresponding variable x_{ij} in the optimal solution to the LP relaxation is positive. We call G a *support graph* of the LP solution. If two clients are both adjacent to the same facility in graph G, we will say that they are *neighbors* in G.

The clustering in this graph is a partitioning of clients into clusters together with a choice of a leading client for each of the clusters. This leading client is called a *cluster center*. Additionally we require that no two cluster centers are neighbors in the support graph. This property helps us to open one of the adjacent facilities for each cluster center. For a picture of a cluster see Figure 2.5.

The algorithms by Shmoys et al., Chudak and Shmoys, and by Sviridenko all use the following procedure to obtain the clustering: While not all the clients are clustered, choose greedily a new cluster center j, and build a cluster from j and all the neighbors of j that are not yet clustered. Obviously the outcome of this procedure is a proper clustering. Moreover, it has a desired property that clients are close to their cluster centers. Each of the mentioned LP-rounding algorithms uses a different greedy criterion for choosing new cluster centers. In our algorithm we will use the clustering with the greedy criterion of Sviridenko [Svi02]. Another way of clustering is presented in Section 2.3.7.

2.3.3 Sparsening the graph of the fractional solution

In this section we consider a technique that we use to control the expected connection cost of the obtained integer solution. Our technique is based on the concept of filtering, introduced by Lin and Vitter [LV92], briefly discussed in see Section 2.1.3. We will give an alternative analysis of the effect of filtering on a fractional solution to the LP relaxation of the UFL problem. We will use the name *sparsening* for the combination of filtering with our new analysis.

Suppose that, for a given UFL instance, we have solved its LP relaxation, and that the optimal primal solution is (x^*, y^*) and the corresponding optimal dual

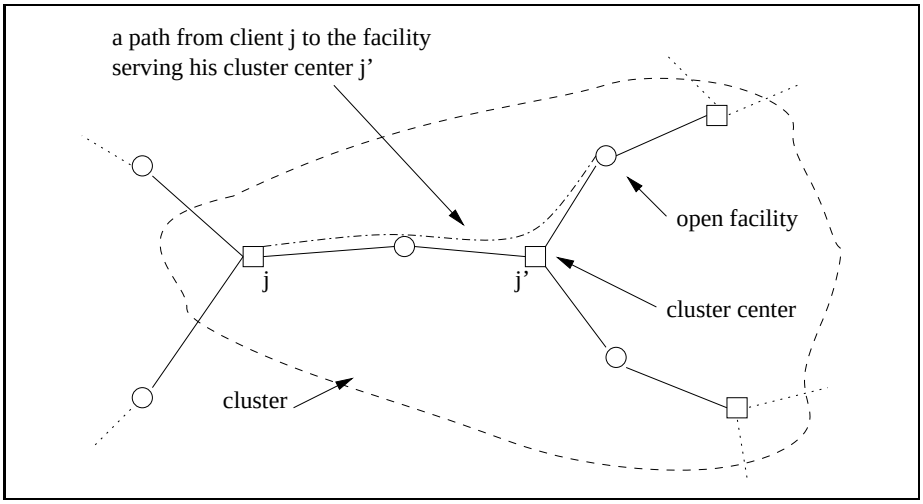


Figure 2.5: A cluster. If we make sure that at least one facility is open close to a cluster center j' , then any other client j from the cluster may use this facility. Because the connection costs are assumed to be metric, the distance to this facility is at most the length of the shortest path from j to the open facility.

solution is (v^*, w^*). Such a fractional solution has facility cost $F^* = \sum_{i \in \mathcal{F}} f_i y_i^*$ and connection cost $C^* = \sum_{i \in \mathcal{F}, j \in \mathcal{C}} c_{ij} x_{ij}^*$. Each client j has its share v_j^* of the total cost. This cost may again be divided into a client's fractional connection cost $C_j^* = \sum_{i \in \mathcal{F}} c_{ij} x_{ij}^*$, and his fractional facility cost $F_j^* = v_j^* - C_j^*$.

Motivation and intuition

The idea behind the sparsening technique is to make use of irregularities of an instance if they occur. We call an instance *locally regular* around client j if the facilities that serve j in the fractional solution (x^*, y^*) are all at the same distance from j. An instance which is locally regular around each client is called *regular*. We begin by observing that for such an instance the algorithm of Chudak and Shmoys produces a solution whose cost is bounded by $F^* + (1 + \frac{2}{e})C^*$, which is an easy consequence of the original analysis [CS03], but also follows from our analysis in Section 2.3.4. Although this observation might not be very powerful itself, the value $F^* + (1 + \frac{2}{e})C^*$ happens to be the intersection point between the bifactor approximation lower bound curve $(\lambda_f, 1 + 2e^{-\lambda_f})$ and the y-axis in Figure 2.4. Moreover, for regular instances we may apply the *scaling + greedy augmentation* technique described in Section 2.1.3 to obtain an approximation algorithm corresponding to any single point on this curve. In particular, we may simply use this construction to get an optimal 1.463...-approximation algorithm for regular instances of the metric UFL problem. Note, that the proof of the matching hardness of approximation also

uses instances that are essentially[2] regular.

The instances that are not regular are called *irregular* and these are the instances for which it is more difficult to create a feasible integer solution with good bounds on the connection cost. In fractional solutions of irregular instances there exist clients that are fractionally served by facilities at different distances. Our approach is to divide facilities serving a client into two groups, namely *close* and *distant* facilities. We will remove links to distant facilities before the clustering step, so that if there are irregularities, then distances to cluster centers will decrease.

We measure the local irregularity of an instance by comparing the fractional connection cost of a client to the average distance to his distant facilities. In the case of a regular instance, the sparsening technique gives the same results as the *scaling + greedy augmentation* technique, but for irregular instances sparsening makes it possible to construct an integer solution with a better bound on the connection costs.

Details

We will start by modifying the optimal fractional LP-solution (x^*, y^*) by scaling the y-variables by a constant $\gamma > 1$ to obtain a fractional solution $(x^*, \tilde{y})$, where $\tilde{y} = \gamma \cdot y^*$. Now suppose that the values of the y-variables are fixed, but that we now have the freedom to change the values of the x-variables in order to minimize the connection cost. For each client j we compute the values of the corresponding $\tilde{x}$-variables in the following way. We choose an ordering of facilities with nondecreasing distances to client j. We connect client j to the first facilities in the ordering so that among the facilities fractionally serving j, only the last one in the chosen ordering may be opened by more than that it serves j. Formally, for any facilities i and i' such that i' is later in the ordering, if $\tilde{x}_{ij} < \tilde{y}_i$ then $\tilde{x}_{i'j} = 0$.

In the next step, we will eliminate the occurrences of situations where $0 < \tilde{x}_{ij} < \tilde{y}_i$. We do so by creating an equivalent instance of the UFL problem, where facility i is split into two identical facilities i' and i''. In the new setting, the opening of facility i' is $\tilde{x}_{ij}$ and the opening of facility i'' is $\tilde{y}_i - \tilde{x}_{ij}$. The values of the $\tilde{x}$-variables are updated accordingly. By repeatedly applying this procedure we obtain a so-called *complete* solution $(\overline{x}, \overline{y})$, i.e., a solution in which no pair $i \in \mathcal{F}, j \in \mathcal{C}$ exists such that $0 < \overline{x}_{ij} < \overline{y}_i$ (see [Svi02][Lemma 1] for a more detailed argument).

In the new complete solution $(\overline{x}, \overline{y})$ we distinguish groups of facilities that are especially important for a particular client. For a client j we say that a facility i is one of its *close facilities* if it fractionally serves client j in $(\overline{x}, \overline{y})$; $\mathcal{C}_j = \{i \in \mathcal{F} | \overline{x}_{ij} > 0\}$ is the set of close facilities of j. If $\overline{x}_{ij} = 0$, but facility i was serving client j in solution (x^*, y^*), then we say, that i is a *distant* facility of client j; $\mathcal{D}_j = \{i \in \mathcal{F} | \overline{x}_{ij} = 0, x^*_{ij} > 0\}$ is the set of distant facilities of j.

[2]These instances come from a reduction from the SET COVER problem. Clients represent elements to be covered, and facilities represent subsets. The distance c_{ij} equals 1 if subset i contains element j and it equals 3 otherwise. To formally argue about the regularity of such an instance we would need to construct an optimal fractional solution using only facilities at distance 1.

We will extensively use the average distances between single clients and groups of facilities defined as follows.

Definition 2.3.1 *For any client $j \in \mathcal{C}$, and for any subset of facilities $\mathcal{F}' \subset \mathcal{F}$ such that $\sum_{i\in\mathcal{F}'} y_i^* > 0$, let*

$$d(j,\mathcal{F}') = \frac{\sum_{i\in\mathcal{F}'} c_{ij}\cdot y_i^*}{\sum_{i\in\mathcal{F}'} y_i^*}.$$

To interpret differences between certain average distances we will use the following parameter.

Definition 2.3.2 *Let*

$$r_\gamma(j) = \begin{cases} \frac{d(j,\mathcal{D}_j)-d(j,\mathcal{D}_j\cup\mathcal{C}_j)}{F_j^*} & \text{for } F_j^* > 0 \\ 0 & \text{for } F_j^* = 0. \end{cases}$$

The value $r_\gamma(j)$ is a measure of the irregularity of the instance around client j. It is the average distance to a distant facility minus the fractional connection cost C_j^* (note, that $C_j^* = d(j,\mathcal{D}_j\cup\mathcal{C}_j)$ is the general average distance to both close and distant facilities) divided by the fractional facility cost of a client j; or it is equal to 0 if $F_j^* = 0$. Since $d(j,\mathcal{D}_j) \le v_j^*$, $C_j^* = d(j,\mathcal{D}_j\cup\mathcal{C}_j)$ and $C_j^* + F_j^* = v_j^*$, $r_\gamma(j)$ takes values between 0 and 1. $r_\gamma(j) = 0$ means that client j is served in the solution (x^*, y^*) by facilities that are all at the same distance. If $r_\gamma(j) = 1$, then the facilities are at different distances and the distant facilities are all so far from j that j is not willing to contribute to their opening. In fact, for clients j with $F_j^* = 0$ the value of $r_\gamma(j)$ is not relevant for our analysis.

To get some more intuition for the values of F_j^* and $r_\gamma(j)$, imagine that one knows F_j^* and C_j^*, but an adversary is constructing the fractional solution, and he is deciding about distances to particular facilities fractionally serving client j. One could interpret F_j^* as a measure of freedom the adversary has in choosing those distances. In this language, $r_\gamma(j)$ is a measure of what fraction of this freedom is used to make distant facilities more distant than average facilities.

Consider yet another quantity, namely $r'_\gamma(j) = r_\gamma(j) * (\gamma - 1)$. Observe, that for a client j with $F_j^* > 0$ we have

$$r'_\gamma(j) = \frac{d(j,\mathcal{D}_j\cup\mathcal{C}_j) - d(j,\mathcal{C}_j)}{F_j^*}.$$

We may use $r_\gamma(j)$ and $r'_\gamma(j)$ to rewrite some distances from client j in the following form (see Figure 2.6):

- the average distance to a close facility is

$$D_{av}^C(j) = d(j,\mathcal{C}_j) = C_j^* - r'_\gamma(j)\cdot F_j^*,$$

- the average distance to a distant facility is

$$D_{av}^D(j) = d(j,\mathcal{D}_j) = C_j^* + r_\gamma(j)\cdot F_j^*,$$

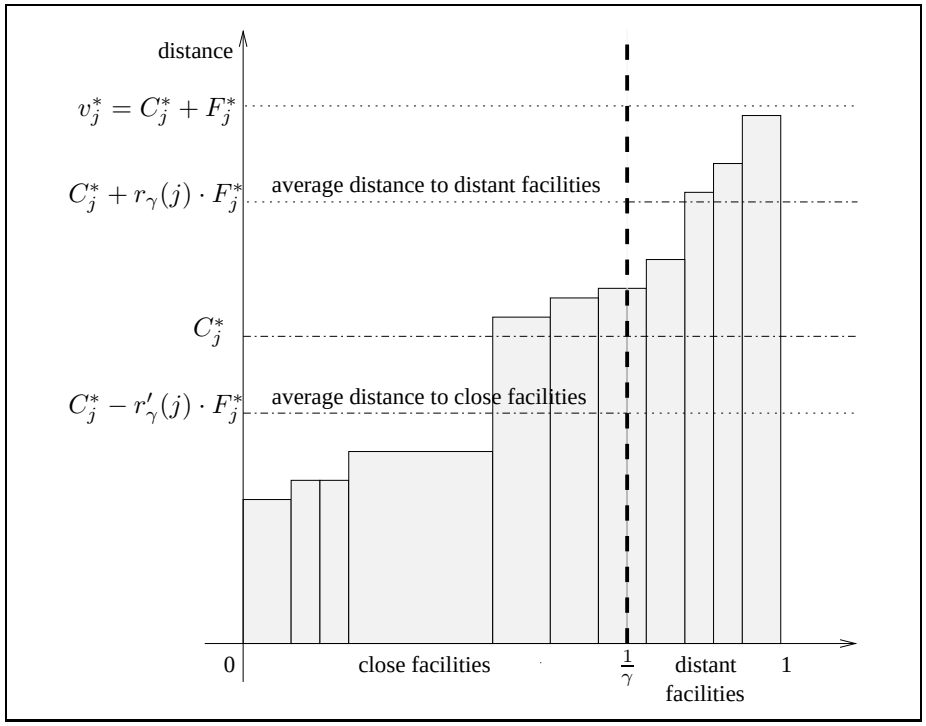


Figure 2.6: Distances to facilities serving client j; the width of a rectangle corresponding to facility i is equal to x_{ij}^*. The figure explains the meaning of $r_\gamma(j)$ and $r'_\gamma(j)$.

- the maximal distance to a close facility is

$$D_{max}^C(j) \leq D_{av}^D(j) = C_j^* + r_\gamma(j) \cdot F_j^*.$$

In the following lemma we will prove an upper bound on the average distance from client j to another group of facilities.

Lemma 2.3.3 *Suppose $\gamma < 2$ and that clients $j, j' \in \mathcal{C}$ are neighbors in $(\overline{x}, \overline{y})$, i.e., $\exists i \in \mathcal{F}$ s.t. $\overline{x}_{ij} > 0$ and $\overline{x}_{ij'} > 0$. Then, either $\mathcal{C}_{j'} \setminus (\mathcal{C}_j \cup \mathcal{D}_j) = \emptyset$ or*

$$d(j, \mathcal{C}_{j'} \setminus (\mathcal{C}_j \cup \mathcal{D}_j)) \leq D_{av}^D(j) + D_{max}^C(j') + D_{av}^C(j').$$

Proof: Assume that $\mathcal{C}_{j'} \setminus (\mathcal{C}_j \cup \mathcal{D}_j)$ is not empty, since otherwise we are done.

Case 1. Assume that the distance between j and j' is at most $D_{av}^D(j) + D_{av}^C(j')$. By a simple observation, that a maximum is larger that the average, we get

$$d(j', \mathcal{C}_{j'} \setminus (\mathcal{C}_j \cup \mathcal{D}_j)) \leq D_{max}^C(j'). \tag{2.11}$$

Combining the assumption with (2.11), we obtain

$$d(j, \mathcal{C}_{j'} \setminus (\mathcal{C}_j \cup \mathcal{D}_j)) \leq D^D_{av}(j) + D^C_{max}(j') + D^C_{av}(j')\,.$$

Case 2. Assume that the distance between j and j' is longer than $D^D_{av}(j)+D^C_{av}(j')$. Since $d(j, \mathcal{C}_j \cap \mathcal{C}_{j'}) \leq D^D_{av}(j)$, the assumption implies

$$d(j', \mathcal{C}_j \cap \mathcal{C}_{j'}) > D^C_{av}(j'). \tag{2.12}$$

Consider the following two sub-cases.

Case 2a. Assume that $d(j', \mathcal{C}_{j'} \cap \mathcal{D}_j) \geq D^C_{av}(j')$.
This assumption together with (2.12) gives

$$d(j', \mathcal{C}_{j'} \cap (\mathcal{C}_j \cup \mathcal{D}_j)) \geq D^C_{av}(j'). \tag{2.13}$$

Recall that $D^C_{av}(j') = d(j', \mathcal{C}_{j'})$. Hence (2.13) is equivalent to

$$d(j', \mathcal{C}_{j'} \setminus (\mathcal{C}_j \cup \mathcal{D}_j)) \leq D^C_{av}(j'). \tag{2.14}$$

Since j and j' are neighbors, the distance between them is at most $D^C_{max}(j) + D^C_{max}(j')$. By the triangle inequality (2.5) we may add this distance to (2.14) and get

$$d(j, \mathcal{C}_{j'} \setminus (\mathcal{C}_j \cup \mathcal{D}_j)) \leq D^D_{av}(j) + D^C_{max}(j') + D^C_{av}(j')\,.$$

Case 2b. In the remaining case we assume that $d(j', \mathcal{C}_{j'} \cap \mathcal{D}_j) < D^C_{av}(j')$.
This assumption may also be written as

$$d(j', \mathcal{C}_{j'} \cap \mathcal{D}_j) = D^C_{av}(j') - z \text{ for some } z > 0. \tag{2.15}$$

Now we combine (2.15) with the assumption of Case 2 to get

$$d(j, \mathcal{C}_{j'} \cap \mathcal{D}_j) \geq D^D_{av}(j) + z. \tag{2.16}$$

Let $\hat{y} = \sum_{i \in (\mathcal{C}_{j'} \cap \mathcal{D}_j)} \overline{y}_i$ be the total fractional opening of facilities in $\mathcal{C}_{j'} \cap \mathcal{D}_j$ in the modified fractional solution $(\overline{x}, \overline{y})$.

Observe that (2.16) together with the definition $d(j, \mathcal{D}_j) = D^D_{av}(j)$ implies that the set $(\mathcal{D}_j \setminus \mathcal{C}_{j'})$ is not empty. Moreover it contains facilities whose opening variables $\overline{y}$ sum up to $\gamma - 1 - \hat{y} > 0$. More precisely, inequality (2.16) implies $d(j, \mathcal{D}_j \setminus \mathcal{C}_{j'}) \leq D^D_{av}(j) - z \cdot \frac{\hat{y}}{\gamma - 1 - \hat{y}}$. Hence

$$D^C_{max}(j) \leq D^D_{av}(j) - z \cdot \frac{\hat{y}}{\gamma - 1 - \hat{y}}. \tag{2.17}$$

We combine (2.17) with the assumption of Case 2 to conclude that the minimal distance from j' to a facility in $\mathcal{C}_{j'} \cap \mathcal{C}_j$ is at least $D^D_{av}(j) + D^C_{av}(j') - D^C_{max}(j) \geq D^C_{av}(j') + z \cdot \frac{\hat{y}}{\gamma - 1 - \hat{y}}$. Hence

$$d(j', \mathcal{C}_{j'} \cap \mathcal{C}_j) \geq D^C_{av}(j') + z \cdot \frac{\hat{y}}{\gamma - 1 - \hat{y}}. \tag{2.18}$$

Recall that, by definition, $d(j', \mathcal{C}_{j'}) = D^C_{av}(j')$. Hence equality (2.15) may be written as

$$d(j', \mathcal{C}_{j'} \setminus \mathcal{D}_j) = D^C_{av}(j') + z \cdot \frac{\hat{y}}{1 - \hat{y}}. \tag{2.19}$$

Since, by the assumption that $\gamma < 2$, we have $\frac{\hat{y}}{1-\hat{y}} < \frac{\hat{y}}{\gamma - 1 - \hat{y}}$, we may also write

$$d(j', \mathcal{C}_{j'} \setminus \mathcal{D}_j) < D^C_{av}(j') + z \cdot \frac{\hat{y}}{\gamma - 1 - \hat{y}}. \tag{2.20}$$

We may now combine (2.20) with (2.18) to get

$$d(j', \mathcal{C}_{j'} \setminus (\mathcal{D}_j \cup \mathcal{C}_j)) < D^C_{av}(j') + z \cdot \frac{\hat{y}}{\gamma - 1 - \hat{y}}. \tag{2.21}$$

Finally, we bound the distance form j to j' by $D^C_{max}(j) + D^C_{max}(j')$ to get

$$\begin{aligned} d(j, \mathcal{C}_{j'} \setminus (\mathcal{C}_j \cup \mathcal{D}_j)) &\leq D^C_{max}(j) + D^C_{max}(j') + d(j', \mathcal{C}_{j'} \setminus (\mathcal{D}_j \cup \mathcal{C}_j)) \\ &\leq D^D_{av}(j) - z \cdot \tfrac{\hat{y}}{\gamma - 1 - \hat{y}} + D^C_{max}(j') + D^C_{av}(j') + z \cdot \tfrac{\hat{y}}{\gamma - 1 - \hat{y}} \\ &= D^D_{av}(j) + D^C_{max}(j') + D^D_{av}(j')\,, \end{aligned}$$

where the second inequality is an application of (2.21) and (2.17). □

2.3.4 Our new algorithm

Consider the following algorithm $A1(\gamma)$:

1. Solve the LP relaxation of the problem to obtain a solution (x^*, y^*).
2. Modify the fractional solution as described in Section 2.3.3 to obtain a complete solution $(\overline{x}, \overline{y})$.
3. Compute a greedy clustering for the solution $(\overline{x}, \overline{y})$, choosing as cluster centers unclustered clients minimizing $D^C_{av}(j) + D^C_{max}(j)$.
4. For every cluster center j, open one of his close facilities randomly with probabilities $\overline{x}_{ij}$.
5. For each facility i that is not a close facility of any cluster center, open it independently with probability $\overline{y}_i$.
6. Connect each client to an open facility that is closest to him.

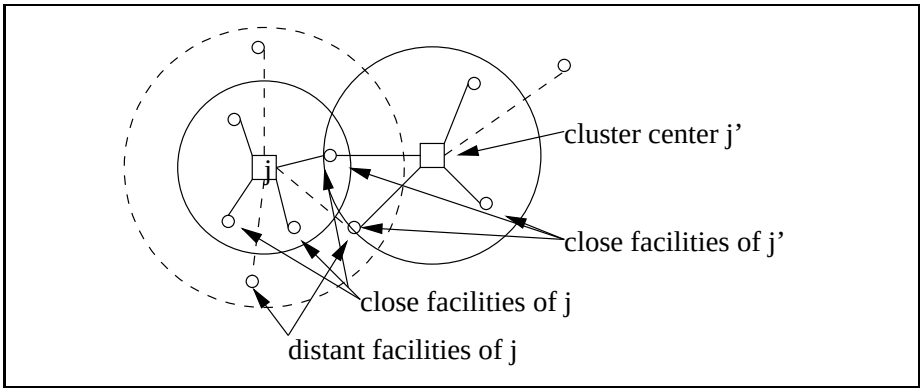


Figure 2.7: Facilities that client j may consider: his close facilities, distant facilities, and close facilities of cluster center j'.

In the analysis of this algorithm we will use the following result:

Lemma 2.3.4 *Given n independent events $e_1, e_2, \ldots, e_n$ that occur with probabilities $p_1, p_2, \ldots, p_n$ respectively, the event $e_1 \cup e_2 \cup \ldots \cup e_n$ (i.e., at least one of e_i) occurs with probability at least $1 - \frac{1}{e^{\sum_{i=1}^{n} p_i}}$, where e denotes the base of the natural logarithm.*

Let γ_0 be defined as the only positive solution to the following equation.

$$\frac{1}{e} + \frac{1}{e^{\gamma_0}} - (\gamma_0 - 1) \cdot (1 - \frac{1}{e} + \frac{1}{e^{\gamma_0}}) = 0 \tag{2.22}$$

An approximate value of this constant is $\gamma_0 \approx 1.67736$. As we will observe in the proof of Theorem 2.3.5, equation (2.22) appears naturally in the analysis of algorithm $A1(\gamma)$.

Theorem 2.3.5 *Algorithm $A1(\gamma_0)$ produces a solution with expected cost $E[cost(SOL)] \leq \gamma_0 \cdot F^* + 1 + \frac{2}{e^{\gamma_0}} \cdot C^*$.*

Proof: We start our analysis from observations valid for the arbitrary choice of the scaling parameter $\gamma > 1$. Later, we also assume $\gamma < 2$. The last argument requires $\gamma = \gamma_0$.

The expected facility opening cost of the solution is
$E[F_{SOL}] = \sum_{i \in \mathcal{F}} f_i \overline{y}_i = \gamma \cdot \sum_{i \in \mathcal{F}} f_i y_i^* = \gamma \cdot F^*$.

To bound the expected connection cost we show that for each client j there is an open facility within a certain distance with a certain probability. If j is a cluster center, one of his close facilities is open and the expected distance to this open facility is $D_{av}^C(j) = C_j^* - r'_\gamma(j) \cdot F_j^* \leq C_j^*$.

If j is not a cluster center, he first considers his close facilities (see Figure 2.7). If any of them is open, the expected distance to the closest open facility is at most

$D^C_{av}(j)$. From Lemma 2.3.4, with probability $p_c \geq (1-\frac{1}{e})$, at least one close facility is open.

Suppose none of the close facilities of j is open, but at least one of his distant facilities is open. Let p_d denote the probability of this event. The expected distance to the closest facility is then at most $D^D_{av}(j)$.

If neither any close nor any distant facility of client j is open, then j may connect himself to the facility serving his cluster center j'. Again from Lemma 2.3.4, such an event happens with probability $p_s \leq \frac{1}{e^\gamma}$. We will now use the fact that if $\gamma < 2$ then, by Lemma 2.3.3, the expected distance from j to the facility opened around j' is at most $D^D_{av}(j) + D^C_{max}(j') + D^C_{av}(j')$.

Finally, we combine the probabilities of particular cases with the bounds on the expected connection cost for each of the cases to obtain the following estimate of the expected total connection cost.

$$\begin{aligned}
E[C_{SOL}] &\leq \textstyle\sum_{j\in\mathcal{C}} \left(p_c \cdot D^C_{av}(j) + p_d \cdot D^D_{av}(j) + p_s \cdot (D^D_{av}(j) + D^C_{max}(j') + D^C_{av}(j'))\right) \\
&\leq \textstyle\sum_{j\in\mathcal{C}} \left((p_c + p_s) \cdot D^C_{av}(j) + (p_d + 2p_s) \cdot D^D_{av}(j)\right) \\
&= \textstyle\sum_{j\in\mathcal{C}} \left((p_c + p_s) \cdot (C^*_j - r'_\gamma(j) \cdot F^*_j) + (p_d + 2p_s) \cdot (C^*_j + r_\gamma(j) \cdot F^*_j)\right) \\
&= ((p_c + p_d + p_s) + 2p_s) \cdot C^* \\
&\quad + \textstyle\sum_{j\in\mathcal{C}} \left((p_c + p_s) \cdot (-r_\gamma(j) \cdot (\gamma - 1) \cdot F^*_j) + (p_d + 2p_s) \cdot (r_\gamma(j) \cdot F^*_j)\right) \\
&= (1 + 2p_s) \cdot C^* + \textstyle\sum_{j\in\mathcal{C}} \left(F^*_j \cdot r_\gamma(j) \cdot (p_d + 2p_s - (\gamma - 1) \cdot (p_c + p_s))\right) \\
&\leq (1 + \tfrac{2}{e^\gamma}) \cdot C^* + \textstyle\sum_{j\in\mathcal{C}} \left(F^*_j \cdot r_\gamma(j) \cdot (\tfrac{1}{e} + \tfrac{1}{e^\gamma} - (\gamma - 1) \cdot (1 - \tfrac{1}{e} + \tfrac{1}{e^\gamma}))\right).
\end{aligned}$$

In the above calculation we used the following properties. In the first inequality we explored the fact that cluster centers were chosen greedily, which implies $D^C_{max}(j') + D^C_{av}(j') \leq D^C_{max}(j) + D^C_{av}(j)$. For the last inequality, we used $p_d + 2p_s = 1 - p_c + p_s \leq 1 - (1 - \frac{1}{e}) + \frac{1}{e^\gamma} = \frac{1}{e} + \frac{1}{e^\gamma}$.

It remains to observe, that by setting $\gamma = \gamma_0 \approx 1.67736$ (see (2.22)) we eliminate the last term in the connection cost estimate, and we obtain $E[C_{SOL}] \leq (1 + \frac{2}{e^{\gamma_0}}) \cdot C^* \leq 1.37374 \cdot C^*$. □

The algorithm A1(γ) was described as a procedure of rounding a particular fractional solution to the LP relaxation of the problem. In the presented analysis we compared the cost of the obtained solution with the cost of the starting fractional solution. If we appropriately scale the cost function in the LP relaxation before solving the relaxation, we easily obtain an algorithm with a bifactor approximation guarantee in a stronger sense. Namely, we get a comparison of the produced solution with any feasible solution to the LP relaxation of the problem, and therefore a bifactor approximation algorithm in the sense of Definition 2.1.1. Such a stronger guarantee is, however, not necessary to construct the 1.5-approximation algorithm for the metric UFL problem, which is presented in the next section.

The algorithm A1(γ) with $\gamma = 1 + \epsilon$ (for a sufficiently small positive ϵ) is essentially the algorithm of Chudak and Shmoys. Observe, that for regular instances, namely those with $r_\gamma(j) = 0$ for every client j, we do not need to set $\gamma = \gamma_0$ to eliminate the dependence of connection cost of the produced solution on the facil-

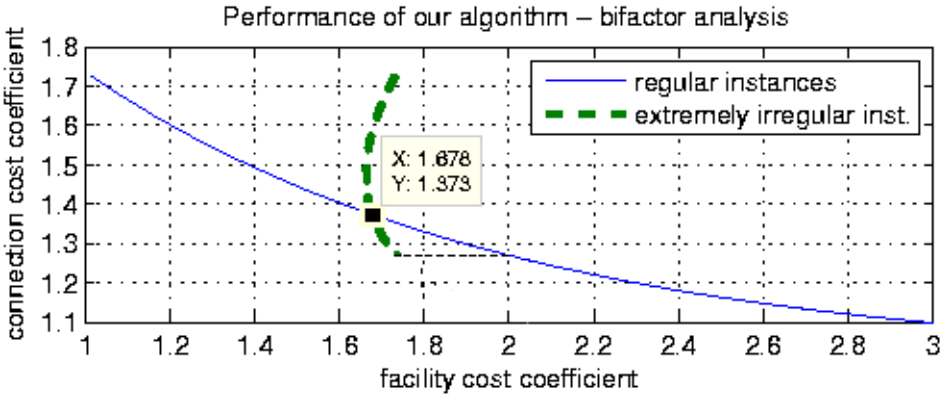


Figure 2.8: The performance of our algorithm for different values of parameter γ. The solid line corresponds to regular instances with $r_\gamma(j) = 0$ for all j and it coincides with the approximability lower bound curve. The dashed line corresponds to instances with $r_\gamma(j) = 1$ for all j. For a particular choice of γ we get a horizontal segment connecting those two curves; for $\gamma \approx 1.67736$ the segment becomes a single point. Observe that for instances dominated by connection cost only a regular instance may be tight for the lower bound.

ity opening cost of the fractional solution. Hence, for regular instances, we get a $(\gamma, \frac{2}{e^\gamma})$-approximation algorithm for each choice of $\gamma > 1$.

2.3.5 The 1.5-approximation algorithm

In this section we will combine our algorithm with an earlier algorithm of Jain et al. to obtain a 1.5-approximation algorithm for the metric UFL problem.

In 2002 Jain, Mahdian and Saberi [JMS02] proposed a primal-dual approximation algorithm (the JMS algorithm). Using a dual fitting approach they showed that it is a 1.61-approximation algorithm. Later Mahdian, Ye and Zhang [MYZ02; MYZ06] derived the following result.

Lemma 2.3.6 ([MYZ02]) *The cost of a solution produced by the JMS algorithm is at most* $1.11 \times F^* + 1.7764 \times C^*$*, where* F^* *and* C^* *are facility and connection costs in an optimal solution to the linear relaxation of the problem.*

Theorem 2.3.7 *Consider the solutions obtained with the A1(γ_0) and JMS algorithms. The cheaper of them is expected to have a cost at most* 1.5 *times the cost of the optimal fractional solution.*

Proof: Consider an algorithm A2 which does the following. With probability $p = 0.313$ runs the JMS algorithm and otherwise, with probability $1 - p$, runs the A1(γ_0) algorithm. Suppose that we are given an instance, and that F^* and C^* are

facility and connection costs in an optimal solution to the linear relaxation of this instance. Consider the expected cost of the solution produced by algorithm A2 for this instance.

$$\begin{aligned} E[cost] &\leq p \cdot (1.11 \cdot F^* + 1.7764 \cdot C^*) + (1-p) \cdot (1.67736 \cdot F^* + 1.37374 \cdot C^*) \\ &= 1.4998 \cdot F^* + 1.4998 \cdot C^* \\ &< 1.5 * (F^* + C^*) \\ &\leq 1.5 * OPT. \end{aligned}$$

□

Instead of the JMS algorithm we could take the algorithm of Mahdian et al. [MYZ06] - the MYZ(δ) algorithm that scales the facility costs by δ, runs the JMS algorithms, scales back the facility costs and finally runs the greedy augmentation procedure. With the notation introduced in Section 2.1.3, the MYZ(δ) algorithm is the $S_\delta(JMS)$ algorithm. The MYZ(1.504) algorithm was proven [MYZ06] to be a 1.52-approximation algorithm for the metric UFL problem. We may change the value of δ in the original analysis to observe that MYZ(1.1) is a (1.2053,1.7058)-approximation algorithm. This algorithm combined with our A1(γ_0) (1.67736,1.37374)-approximation algorithm gives a 1.4991-approximation algorithm, which is even better than just using JMS and A1(γ_0), but it gets more complicated and the additional improvement is tiny.

2.3.6 Multilevel facility location

In the k-level facility location problem the clients need to be connected to open facilities on the first level, and each open facility except on the last, k-th level, needs to be connected to an open facility on the next level. Aardal, Chudak, and Shmoys [ACS99] gave a 3-approximation algorithm for the k-level problem with arbitrary k. Ageev, Ye, and Zhang [AYZ03] proposed a reduction of a k-level problem to a $(k-1)$-level and a 1-level problem, which results in a recursive algorithm. This algorithm uses an approximation algorithm for the single level problem and has a better approximation ratio, but only for instances with small k. Using our new algorithm A1(γ_0) instead of the JMS algorithm within this framework, improves approximation for each level. In particular, in the limit as k tends to ∞, we get a 3.236-approximation which is the best possible for this construction.

By a slightly different method, Zhang [Zha06] obtained a 1.77-approximation algorithm for the 2-level problem. For the 3-level and the 4-level version of the problem he obtained 2.523-[3] and 2.81-approximation algorithms, by reducing to a problem with smaller number of levels. In the following we will modify the algorithm by Zhang for the 3-level problem, and use the new (1.67736,1.37374)-approximation algorithm for the single-level part, to obtain a 2.492-approximation, which improves on the previously best known approximation by Zhang. Note, that for $k > 4$ the best known approximation factor is still due to Aardal et al. [ACS99].

[3]This value deviates slightly from the value 2.51 given in the paper. The original argument contained a minor calculation error.

3-level facility location

We will now present the ingredients of the 2.492-approximation algorithm. We start from an algorithm to solve the 2-level version.

Lemma 2.3.8 (Theorem 2 in [Zha06]) *The 2-level UFL problem may be approximated by a factor of* $1.77 + \epsilon$ *in polynomial time for any given constant* $\epsilon > 0$.

Zhang [Zha06] also considered a scaling technique analogous to the one described in Section 2.1.3, but applicable to the 2-level version of the problem. An effect of using this technique is analyzed in the following lemma.

Lemma 2.3.9 (Theorem 3 in [Zha06]) *For any given* $\epsilon > 0$*, if there is an* (a, b)*-approximation algorithm for the 2-level UFL problem, then we can get an approximation algorithm for the 2-level UFL problem with performance guarantee*

$$\left(a + \frac{e}{e-1} ln(\Delta) + \epsilon, 1 + \frac{b-1}{\Delta}\right)$$

for any $\Delta \geq 1$.

He also uses the following reduction.

Lemma 2.3.10 (Lemma 7 in [Zha06]) *Assume, that the 1-level and 2-level UFL problems have approximation algorithms with factors* (a, b) *and* (α, β)*, respectively, then the 3-level UFL problem may be approximated by factors* $(max\{a, \frac{a+\alpha}{2}\}, \frac{3b+\beta}{2})$.

Zhang [Zha06] observed, that the above three statements may be combined with the MYZ algorithm to improve the approximation ratio for the 3-level UFL problem. In the following theorem we show that we may use our new (1.6774,1.3738)-approximation algorithm for the 1-level UFL subproblem to get even better approximation for the 3-level variant.

Theorem 2.3.11 *There is a 2.492-approximation algorithm for the 3-level UFL problem.*

Proof: We first use the algorithm from Lemma 2.3.8, and the scaling technique from Lemma 2.3.9, with $\Delta = 1.57971$, to obtain a $(2.492, 1.48743)$-approximation algorithm for the 2-level UFL problem.

Then we use our $(1.6774, 1.3737\ldots)$-approximation algorithm for the 1-level UFL problem with the scaling technique from Lemma 2.1.3, with $\gamma = 2.25827$, to obtain a $(2.492, 1.1655)$-approximation algorithm for the 1-level UFL problem.

Finally, we use Lemma 2.3.10 to combine these two algorithms into a $(2.492, 2.492)$-approximation algorithm for the 3-level UFL problem.

□

2.3.7 Universal randomized clustering procedure

In this section we discuss a different approach to clustering. We propose to modify the greedy clustering algorithm by choosing consecutive cluster centers randomly with uniform distribution. The output of such a process is obviously random, but we may still prove some statements about probabilities. A resulting clustering will be denoted by a function $g : \mathcal{C} \to \mathcal{C}$, which assigns to each client j the center of his cluster $j' = g(j)$. The following lemma states that the clustering g obtained with the randomized clustering procedure is expected to be "fair".

Lemma 2.3.12 *Given a graph $G = (\mathcal{F} \cup \mathcal{C}, E)$ and assuming that a clustering g was obtained by the above described random process, for every two distinct clients j and j', the probability that $g(j) = j'$ is equal the probability that $g(j') = j$.*

Proof: Let $C(G)$ denote the maximal (over the possible random choices of the algorithm) number of clusters that can be obtained from G with the random clustering procedure. The proof will be by induction on $C(G)$. Fix any $j, j' \in \mathcal{C}$ such that j is a neighbor of j' in G (if they are not neighbors, neither $g(j) = j'$ nor $g(j') = j$ can occur). Suppose $C(G) = 1$, then $Pr[g(j) = j'] = Pr[g(j') = j] = 1/|\mathcal{C}|$.

Let us now assume that $C(G) > 1$. There are two possibilities, either one of j, j' gets to the first cluster or they both avoid it. Consider the first case (the first chosen cluster center is either j or j' or one of their neighbors). If j (j') is chosen as a cluster center, then $g(j') = j$ ($g(j) = j'$). Since they are chosen with the same probability, the contribution of the first case to the probability of $g(j') = j$ is equal to the contribution to the probability of $g(j) = j'$. If neither of them gets chosen as a cluster center but at least one gets into the new cluster, then neither $g(j') = j$ nor $g(j) = j'$ is possible.

Now consider the second case (neither of j and j' gets into the first cluster). Consider the graph G' obtained from G by removing the first cluster. The random clustering proceeds like it has just started with the graph G', but the maximal number of possible clusters is smaller $C(G') \leq C(G)-1$. Therefore, by the inductive hypothesis, in a random clustering of G' the probability. that $g(j') = j$ is equal the probability that $g(j) = j'$. Hence, the second case contribution to those probabilities for the clustering of the original graph G is also equal. □

If $g(j) = j'$ in a clustering g of graph G we will say that client j' *offers a support* to client j. The main idea behind the clustering algorithms for the UFL problem is that we may afford to serve each cluster center directly (because they are never neighbors in G) and all the other clients are offered a support from their cluster centers. A non-central client may either accept a support and connect himself via his cluster center (that is what all non-central clients do in the algorithm of Shmoys et al.), or he may try to get served locally, and if it fails, he will accept the support (this is the way the Chudak and Shmoys' algorithm works). In both those algorithms the probability that an offer of support is accepted is estimated to be constant. Therefore, we may modify those algorithms to use the random clustering procedure and do the following analysis.

For any two clients j and j', the probability that j accepts a support of j' is equal to the probability that j' accepts the support of j. Let i be a facility on a shortest path from j to j'. When we compute the expected connection cost of a client j, we observe that with certain probability p he accepts a support of j'. In such a case he must pay for the route via i and j' to the facility directly serving j'. In this situation we will say that j is paying only for the part until facility i, and the rest is paid by j', but if j would be supporting j' he would have to pay a part of j''s connection cost, which is the length of the path from i via j to the facility serving j. We may think of it as each client having a bank account, and when he accepts a support he makes a deposit, and when he offers a support and the support is accepted, then he withdraws money to pay a part of the connection cost of the supported client. From Lemma 2.3.12 we know that for a client j the probability that he will earn on j' is equal to the probability that he will lose on j'. Therefore, if the deposed amount is equal to the withdrawal, the expected net cash flow is zero.

The above analysis shows that randomizing the clustering phase of the above mentioned algorithms would not worsen their approximation ratios. Although it does not make much sense to use a randomized algorithm if it has no better performance guarantee, the random clustering has an advantage of allowing the analysis to be more local and uniform.

2.3.8 Concluding remarks

With the 1.52-approximation algorithm of Mahdian et al. it was not clear to the authors if a better analysis of the algorithm could close the gap with the approximation lower bound of 1.463 by Guha and Khuller (see Section 2.2). Similarly, we now do not know if our new algorithm $A1(\gamma)$ could be analyzed better to close the gap. Construction of hard instances for our algorithm remains an open problem.

The *scaling + greedy augmentation* technique described in Section 2.1.3 enables us to move the bifactor approximation guarantee of an algorithm along the approximability lower bound of Jain et al. (see Figure 2.4) towards higher facility opening costs. If we developed a technique to move the analysis in the opposite direction, together with our new algorithm, it would imply closing the approximability gap for the metric UFL problem. It seems that with such an approach we would have to face the difficulty of analyzing an algorithm that closes some of the previously opened facilities.

Chapter 3

Phylogenetic trees/networks

3.1 Preliminaries

Since the famous theory of evolution by Charles Darwin [Dar59] was announced to the public in 1859, a great part of scientific activity has been devoted to exploring its consequences and, in particular, to drawing a map of relationships between the living species. Although, the theory of evolution itself is still, by many people, not accepted as a valid description of the development of complicated living organisms, it gives a great motivation to study many related problems. With the recent advance in technology it is suddenly possible for biologists to operate with great amounts of experimental data. The emerging interdisciplinary field of computational biology concerns, among others, the algorithmic problems arrising in the analysis of such large biological data sets.

One of the most commonly encountered problems in computational evolutionary biology is to plausibly infer the evolutionary history of a set of species, often abstractly modelled as a tree, using obtained biological data. Existing algorithms for directly constructing such a tree do not scale well (in terms of running time) and this has given rise to *supertree* methods: first infer trees for small subsets of the species and then puzzle them together into a bigger tree such that in some well-defined sense the information in the subset trees is preserved [BE04]. In the fundamental case where the subsets in question each contain exactly three species - subsets of two or fewer species cannot convey information - we speak of *rooted triplet* methods.

In recent years improved understanding of the complex mechanisms driving evolution has stimulated interest in reconstructing evolutionary *networks* [Doo99; KGDO05; Mar99; RL04]. Such structures are more general than trees and allow us to capture the phenomenon of reticulate evolution i.e. non tree-like evolution caused by processes such as hybrid speciation and horizontal gene transfer. A natural abstraction of reticulate evolution, used already in several papers, is to permit *recombination vertices*, vertices with indegree greater than one. Informally a *level-k phylogenetic network* is an evolutionary network in which each biconnected component contains at most k such recombination vertices (for definitions see Sec-

tion 3.2.1). Phylogenetic trees form the base: they are level-0 networks. The higher the level of a network, the more intricate the pattern of reticulate evolution that it can accommodate. Note that phylogenetic networks can also be useful for visualising two or more competing hypotheses about tree-like evolution.

Various authors have already studied the problem of constructing phylogenetic trees (and more generally networks) that are consistent with an input set of rooted triplets. Aho et al. [ASSU81] showed a simple polynomial-time algorithm which, given a set of rooted triplets, finds a phylogenetic tree consistent with all the triplets, or shows that no such tree exists. For the equivalent problem in level-1 and level-2 networks the problem becomes NP-hard [vIKK$^+$08; JNS00], although the problem becomes polynomial-time solveable if the input triplets are *dense* i.e. if there is at least one triplet in the input for each subset of three species [vIKK$^+$08; JS06].

Several authors have considered algorithmic strategies of use when the algorithms from [ASSU81] and [JS06] fail to find a tree or network. Gąsieniec et al. [GJLO99] gave a polynomial-time algorithm that always finds a tree consistent with at least 1/3 of the (weighted) input triplets, and furthermore showed that 1/3 is best possible when all possible triplets on n species (the *full triplet set*) are given as input. On the negative side, [Bry97; Jan01; Wu04] showed that it is NP-hard to find a tree consistent with a maximum number of input triplets. In the context of level-1 networks, [JNS00] gave a polynomial-time algorithm that produces a level-1 network consistent with at least $5/12 \approx 0.4166$ of the input triplets. They also described an upper-bound, which is a function of the number of distinct species n in the input, on the percentage of input triplets that can be consistent with a level-1 network. As in [GJLO99], this upper bound is tight in the sense that it is the best possible for the full triplet set on n species. They computed a value of n for which their upper bound equals approximately 0.4883, showing that in general no better fraction is possible. The apparent convergence of this bound from above to 0.4880... begs the question, however, whether a fraction better than 5/12 is possible for level-1 networks, and whether the full triplet set is in general always the worst-case scenario for such fractions.

In Section 3.2 we answer these questions in the affirmative, and in fact we give a much stronger result. In particular, we develop a probabilistic argument that (as far as such fractions are concerned) the full triplet set is indeed always the worst possible case, irrespective of the type of network being studied (Proposition 3.2.3, Corollary 3.2.4). Furthermore, by using a generic, derandomized polynomial-time (re)labeling procedure, we can convert a network $\mathcal{N}$ that achieves a fraction p' for the full triplet set into an isomorphic network $\mathcal{N}'(T)$ that achieves a fraction $\geq p'$ for a given input triplet set T (Theorem 3.2.5). In this way we can easily use the full triplet set to generate, for any network structure, a lower bound on the fraction that can be achieved for arbitrary triplet sets within such a network structure. The derandomization we give is fully general (with a highly optimized running time) and leads immediately to a simple extension of the 1/3 result from [GJLO99]. For level-1 networks we use the derandomization to give a polynomial-time algorithm that achieves a fraction *exactly equal* to the level-1 upper-bound identified in [JNS00],

and which is thus worst-case optimal for level-1 networks. We formally prove that this achieves a fraction of at least 0.48 for all n. Moreover, we demonstrate the flexibility of our technique by proving that for level-2 networks (see [vIKK$^+$08]) we can, for any triplet set T, find in polynomial time a level-2 network consistent with at least a fraction 0.61 of the triplets in T (Theorem 3.2.13).

We emphasize that for the above mentioned results we are optimizing (and thus defining worst-case optimality) with respect to $|T|$, the number of triplets in the input, not $Opt(T)$, the size of the optimal solution for that specific T. The latter formulation we call the MAX variant of the problem. The fact that $Opt(T)$ is always bounded above by $|T|$ implies that an algorithm that obtains a fraction p' of the input T is trivially also a p'-approximation for the corresponding MAX problem. Better approximation factors for the MAX problem might, however, be possible. We discuss this further in Section 3.2.7.

The results are given in terms of unweighted triplet sets. A natural extension, especially in phylogenetics, is to attach a weight to each triplet $t \in T$ i.e. a value $w(t) \in \mathbb{Q}_{\geq 0}$ denoting the relative importance of (or confidence in) t. In this weighted version of the problem fractions are defined relative to the total weight of T (defined as the sum of the weights of all triplets in T), not to $|T|$. It is easy to verify that all the results in this article also hold for the weighted version of the problem.

The above mentioned results achieve the best possible approximation guarantees against the trivial upper bound for a number of variants of the triplet consistency problem. A natural question is whether a nontrivial upper bound in some restricted setting exists. We address this question is Section 3.3, in which we also consider the problem of finding an optimal caterpillar - a very restricted tree. This leads to a number of partial results, including a reduction from the MAX SUBDAG problem, which suggest that approximation ratios better than 1/2 will be very difficult to obtain.

Finally, in Section 3.4 we study a related problem of comparing two leaf-labelled trees, assuming the sets of labels are equal. This problem naturally appears in the situation when two different models of evolution for a given set of species are considered. The algorithms we study do not calculate any distance measure. Instead, they suggest a way of drawing given trees in a plane in order to display the structural differences of the considered trees. We consider the following computational problem. A *binary tanglegram* is a pair $\langle S, T \rangle$ of binary trees whose leaf sets are in one-to-one correspondence; matching leaves are connected by inter-tree edges. We ask for a drawing in which both trees are drawn with no edge crossing and such that the inter-tree edges have as few crossings as possible. It is known that finding a drawing with the minimum number of crossings is NP-hard and that the problem is fixed-parameter tractable with respect to that number. We show that the problem is hard even if both trees are complete binary trees. For this case we give an $O(n^3)$-time 2-approximation and a new and simple fixed-parameter algorithm. We prove that under the Unique Games Conjecture there is no constant-factor ap-

proximation for general binary trees. We show that the maximization version of the problem for general binary trees can be reduced to a version of MaxCut for which the algorithm of Goemans and Williamson yields a 0.878-approximation. We also did an experimental study to evaluate our 2-approximation.

3.2 Constructing networks consistent with big fraction of triplets

3.2.1 Definitions

A *phylogenetic network* (*network* for short) $\mathcal{N}$ on species set X is defined as a pair (N, γ) where N is the *network topology* (*topology* for short) and γ is a *labeling* of the topology. The topology is a directed acyclic graph in which exactly one vertex has indegree 0 and outdegree 2 (the root) and all other vertices have either indegree 1 and outdegree 2 (*split vertices*), indegree 2 and outdegree 1 (*recombination vertices*) or indegree 1 and outdegree 0 (*leaves*). A labeling is a bijective mapping from the leaf set of N (denoted L^N) to X. Let $n = |X| = |L^N|$.

A directed acyclic graph is *connected* (also called "weakly connected") if there is an undirected path between any two vertices and *biconnected* if it contains no vertex whose removal disconnects the graph. A *biconnected component* of a network is a maximal biconnected subgraph.

Definition 3.2.1 *A network is said to be a* level-k network *if each biconnected component contains at most $k \in \mathbb{N}$ recombination vertices.*

We define *phylogenetic trees* to be the class of level-0 networks.

The unique *rooted triplet* (*triplet* for short) on a species set $\{x, y, z\} \subseteq X$ in which the lowest common ancestor of x and y is a proper descendant of the lowest common ancestor of x and z is denoted by $xy|z$ (which is identical to $yx|z$, see Figure 3.1). For any set T of triplets, define $X(T)$ as the union of the species sets of all triplets in T.

Definition 3.2.2 *A triplet $xy|z$ is* consistent *with a network $\mathcal{N}$ (interchangeably: $\mathcal{N}$ is consistent with $xy|z$) if $\mathcal{N}$ contains a subdivision of $xy|z$, i.e., if $\mathcal{N}$ contains vertices $u \neq v$ and pairwise internally vertex-disjoint paths $u \to x$, $u \to y$, $v \to u$ and $v \to z$*[1].

By extension, a set of triplets T is consistent with a network $\mathcal{N}$ (interchangeably: $\mathcal{N}$ is consistent with T) iff, for all $t \in T$, t is consistent with $\mathcal{N}$. Checking consistency can be done in polynomial time, see Lemmas 3.2.7 and 3.2.8.

[1] Where it is clear from the context, as in this case, we may refer to a leaf by the species that it is mapped to.

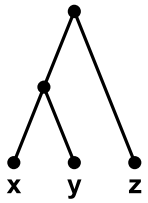


Figure 3.1: One of the three possible triplets on the set of species $\{x, y, z\}$. Note that, as with all figures in this article, all arcs are assumed to be directed downwards, away from the root.

3.2.2 Labeling a network topology

Suppose we are given a topology N with n leaves, and a set T of triplets where $L^N = \{l_1, l_2, \ldots, l_n\}$ and $X = X(T) = \{x_1, x_2, \ldots, x_n\}$. The specific goal of this section is to create a labeling γ such that the number of triplets from T consistent with (N, γ) is maximized.

Let $f(N, \gamma, T)$ denote the fraction of T that is consistent with (N, γ).

Consider the special set $T_1(n)$, the *full triplet set*, of all the possible $3\binom{n}{3}$ triplets with leaves labelled from $\{x_1, x_2, \ldots, x_n\}$. Observe that for this triplet set the number of triplets consistent with a phylogenetic network (N, γ) does not depend on the labeling γ. We may thus define $\#N = f(N, \gamma, T_1(n)) = f(N, T_1(n))$ by considering any arbitrary, fixed labeling γ. (Note that $\#N = 1/3$ if N is a tree topology.)

We will argue that the triplet set $T_1(n)$ is the worst-case input for maximizing $f(N, \gamma, T)$ for any fixed topology N on n leaves. In particular we prove the following:

Proposition 3.2.3 *For any topology N with n leaves and any set of triplets T, if the labeling γ is chosen uniformly at random, then the quantity $f(N, \gamma, T)$ is a random variable with expected value $E(f(N, \gamma, T)) = \#N$.*

Proof: Consider first the full triplet set $T_1(n) = \{t_1, t_2, \ldots, t_{3\binom{n}{3}}\}$ and an arbitrary fixed labeling γ_0. By labeling N we fix the *position* of each of the triplets in N. Formally, a position of a triplet $t = xy|z$ (with respect to γ_0) is a triplet $p = \gamma_0^{-1}(t) = \gamma_0^{-1}(x)\gamma_0^{-1}(y)|\gamma_0^{-1}(z)$ on the leaves of N. We may list possible positions for a triplet in N as those corresponding to $t_1, t_2, \ldots, t_{3\binom{n}{3}}$ in (N, γ_0). Since a $\#N$ fraction of $t_1, t_2, \ldots, t_{3\binom{n}{3}}$ is consistent with (N, γ_0), a $\#N$ fraction of these positions makes the triplet consistent. Now consider a single triplet $t \in T$ and a labeling γ that is chosen randomly from the set Γ of $n!$ possible bijections from L^N to X. Observe, that for each $t_i \in T_1(n)$, exactly $2 \cdot (n-3)!$ labelings $\gamma \in \Gamma$ make triplet t have the same position in (N, γ) as t_i has in (N, γ_0) (the factor of 2 comes from the fact that we think of $xy|z$ and $yx|z$ as being the same triplet). Any single labeling occurs with probability $\frac{1}{n!}$, hence triplet t takes any single position with probability $\frac{2\cdot(n-3)!}{n!} = \frac{1}{3\cdot\binom{n}{3}}$.

Since for an arbitrary $t \in T$ each of the $3 \cdot \binom{n}{3}$ positions have the same probability

and $\#N$ of them make t consistent, the probability of t being consistent with (N, γ) is $\#N$. The expectation is thus that a fraction $\#N$ of the triplets in T are consistent with (N, γ). □

From the expected value of a random variable we may conclude the existence of a realization that attains at least this value.

Corollary 3.2.4 *For any topology N and any set of triplets T there exists a labeling γ_0 such that $f(N, \gamma_0, T) \geq \#N$.*

We may deterministically find such a γ_0 by derandomizing the argument in a greedy fashion, using the method of conditional expectation (see e.g. [MR95]). In particular, we will show the following.

Theorem 3.2.5 *For any topology N and any triplet set T, a labeling γ_0 such that $f(N, \gamma_0, T) \geq \#N$ can be found in time $O(m^3 + n|T|)$, where m and n are the numbers of vertices and leaves of N.*

We use standard arguments from conditional expectation to prove that the solution is of the desired quality. It is somewhat more sophisticated to optimize the running time of this derandomization procedure. We therefore dedicate the following subsection to the proof of Theorem 3.2.5.

3.2.3 An optimized derandomization procedure

We start with a sketch of the derandomization procedure.

1. $\Gamma \leftarrow$ set of all possible labelings.
2. while there is a leaf l whose label is not yet fixed, do:
   - for every species x that is not yet used
     - let Γ_x be the set of labelings from Γ where l is labeled by x
     - compute $E(f(N, \gamma_x, T))$, where γ_x is chosen randomly from Γ_x
   - $\Gamma \leftarrow \Gamma_x$ s.t. $x = argmax_x E(f(N, \gamma_x, T))$
3. return $\gamma_0 \leftarrow$ the only element of Γ.

The following lemma shows that the solution produced is of the desired quality.

Lemma 3.2.6 $f(N, \gamma_0, T) \geq \#N$.

Proof: By Proposition 3.2.3 the initial random labeling γ has the property $E(f(N, \gamma, T)) = \#N$. It remains to show that this expectation is not decreasing when labels of leaves get fixed during the algorithm. Consider a single update $\Gamma \leftarrow \Gamma_x$ of the range of the random labeling. By the choice of the leaf l to get a fixed label, we choose a partition of Γ into blocks Γ_x. The expectation $E(f(N, \gamma, T))$ is an average of $f(N, \gamma, T)$ over Γ, and at least one of the blocks Γ_x of the partition has this average at least as big as the total average. Hence, by the choice of Γ_x with the highest expectation of $f(N, \gamma_x, T)$, we get $E(f(N, \gamma_x, T)) \geq E(f(N, \gamma, T))$. □

We will now propose an efficient implementation of the derandomization procedure. Since the procedure takes a topology N as an input, we need to express the running time of this procedure in terms of the size of N. We will use $m = |V(N)|$ for the number of vertices of N, and n for the number of leaves.

We will need a generalized definition of consistency. In Definition 3.2.2 it is assumed that x, y and z are leaves of the network $\mathcal{N}$. In the following we will use a predicate $consistent(x, y, z)$ defined on vertices of $\mathcal{N}$, which coincides with Definition 3.2.2 when x, y and z are leaves. The predicate $consistent(x, y, z)$ is defined to be true if $\mathcal{N}$ contains vertices $u \neq v$ and pairwise internally vertex-disjoint paths $u \to x$, $u \to y$, $v \to u$ and $v \to z$; paths are allowed to be of length 0 (e.g. $u = x$), but vertices x, y and z need to be pairwise different. We begin with a consistency checking algorithm.

Lemma 3.2.7 *Given a network $\mathcal{N}$ with m vertices, we can preprocess it in time $O(m^3)$ so that given any three vertices x, y, z we can check whether $xy|z$ is consistent with $\mathcal{N}$ in time $O(1)$.*

Proof: Assume that vertices of $\mathcal{N}$ are numbered according to a topological order, so that whenever (u, v) is an edge, $u < v$. Define $join(x, z)$ to be the predicate stating that $\mathcal{N}$ contains $t \neq x$ and internally vertex-disjoint paths $t \to x$ and $t \to z$. It can be verified in linear time, for example by checking if there is a path $r \to z$ (where r is the root) after removing x from the network. Observe that $consistent(x, y, z)$ can be expressed in terms of $join$ and $consistent(x', y', z')$ where $\max(x', y', z') < \max(x, y, z)$ as follows:

1. $x, y < z$. Then $consistent(x, y, z)$ holds iff for some $z' \neq x, y$ (z', z) is an edge and $consistent(x, y, z')$ holds.

2. $x, z < y$. Then $consistent(x, y, z)$ holds iff (x, y) is an edge and $join(x, z)$ holds or for some $y' \neq x, z$ (y', y) is an edge and $consistent(x, y', z)$ holds.

3. $y, z < x$. Then $consistent(x, y, z)$ holds iff (y, x) is an edge and $join(y, z)$ holds or for some $x' \neq y, z$ (x', x) is an edge and $consistent(x', y, z)$ holds.

So, determining all consistent triplets can be done by evaluating all predicates $consistent(x, y, z)$. The number of operations we have to perform for each x, y, z is not greater than sum of indegrees of those vertices which makes the overall complexity $O(m^3 + m^2|E(\mathcal{N})|) = O(m^3)$. □

Although we do not use this result any further, we note that the above lemma can be strengthened to obtain the following.

Lemma 3.2.8 *Given a level-k network $\mathcal{N}$, we can preprocess it in time $O(m+mk^2)$ so that given any three vertices x, y, z we can check whether $xy|z$ is consistent with $\mathcal{N}$ in time $O(1)$.*

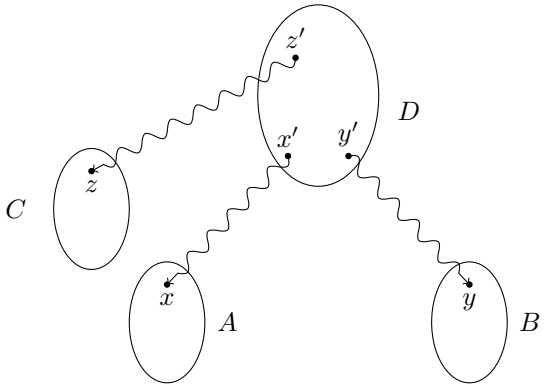


Figure 3.2: The case $LCA(A, B) = LCA(A, B, C)$ from the proof of Lemma 3.2.8.

Proof: We begin with a definition and two claims. A *chain* is a sequence of vertices $v_0 \to v_1 \to \ldots v_{k+1}$ such that both in and out degrees of $v_1, \ldots, v_k$ are all equal to 1. Now, consider an arbitrary biconnected component in $\mathcal{N}$. We claim that, if each chain within the component is replaced by an edge, the transformed component contains at most $\max(2, 3k)$ vertices. This is trivially true for the biconnected component consisting of only a single edge. To see that it is more generally true, note that the transformed component contains at most k recombination vertices and that all other vertices have outdegree 2 and indegree at most 1. Each such non-recombination vertex creates at least one new path that eventually has to terminate in a recombination vertex, and a recombination vertex can terminate at most two paths. So there are at most $2k$ non-recombination vertices, and the claim follows.

From now on, biconnected component (or simply component) refers to a biconnected component containing more than one edge. We claim that all biconnected components within $\mathcal{N}$ are vertex-disjoint. To see why this is true, note that each vertex in such a component must have degree at least 2. The value of indegree plus outdegree of all vertices in $\mathcal{N}$ is at most 3, so they cannot be a part of more than one such component.

The above reasoning shows that we could imagine the network as a rooted tree $\mathcal{T}$ in which each vertex corresponds to some bigger component that has relatively simple structure: contracting all chains in it gives us a graph of size $O(k)$.

First we focus on the case when x, y, z lie in one biconnected component. The obvious solution is to simply preprocess such queries for each triple of vertices from one biconnected component. This does not give us the desired complexity yet: while each component is small after contracting chains, it might originally contain as many as $\Omega(m)$ vertices. We may overcome this difficulty by observing that if some chain consist of more than 3 inner vertices, we can replace it by a chain containing exactly 3 of them. Then we preprocess the resulting graph using the $O(|V|^3)$ algorithm from Lemma 3.2.7 (observe that $|V| = O(k)$). Given a query concerning consistency of

some $xy|z$ we may have to replace some of x, y, z with other vertices lying on the shortened chains, which can be easily done in time $O(1)$. The whole preprocessing takes time $\sum_i O(k_i^3)$ where all $k_i = O(k)$ and $\sum_i k_i = O(m)$, giving us the desired complexity.

Now we can solve the general case. Let A, B, C be biconnected components such that $x \in A$, $y \in B$ and $z \in C$. We can preprocess $\mathcal{T}$ in linear time so that given its two vertices, we can find their lowest common ancestor (LCA) in time $O(1)$ [BFC00]. (This is already sufficient for the case $k = 0$ and contributes the m term in the running time.) If $xy|z$ is consistent there are only two possible situations:

1. $LCA(A, B, C)$ is a proper ancestor of $LCA(A, B)$. This is easy to detect.

2. $LCA(A, B) = LCA(A, B, C)$. Let $D = LCA(A, B)$. We can find x', y', z' - the entrance points within D (see Figure 3.2) - in time $O(1)$. It can be done by either using level ancestor queries or modifying the LCA algorithm [BFC04]. We can then check whether $x'y'|z'$ is consistent using the above preprocessing.

□

Armed with Lemma 3.2.7 (or Lemma 3.2.8) we are now ready to proceed with the derandomization. Assume that we assign labels to leaves in order of their position in the topological ordering of Lemma 3.2.7; for simplicity let us refer to the leaves as $1, 2, \ldots, n$. The most time-consuming part of the algorithm is calculating the probability that a given triplet $t = ab|c$ is consistent with a random labeling having already specified the labels of leaves $1, 2, \ldots, k-1$. Let γ be this partial labeling. We need to consider six cases:

1. Labels a, b, c have not been assigned yet. The probability is then simply the number of consistent triplets $xy|z$ with $k \leq x, y, z$ divided by $3\binom{n-k+1}{3}$.

2. $\gamma(x) = a$ but both b and c have not been assigned yet. The probability is the number of $k \leq y, z$ such that $xy|z$ is a consistent triplet divided by $2\binom{n-k+1}{2}$.

3. $\gamma(z) = c$ but both a and b have not been assigned yet. The probability is the number of $k \leq x < y$ such that $xy|z$ is a consistent triplet divided by $\binom{n-k+1}{2}$.

4. $\gamma(x) = a, \gamma(y) = b$ but c has not been assigned yet. The probability is the number of $k \leq z$ such that $xy|z$ is a consistent triplet divided by $n - k + 1$.

5. $\gamma(x) = a, \gamma(z) = c$ but b has not been assigned yet. The probability is the number of $k \leq y$ such that $xy|z$ is a consistent triplet divided by $n - k + 1$.

6. $\gamma(x) = a, \gamma(y) = b, \gamma(z) = c$. The probability is 0 or 1, depending on whether $xy|z$ is consistent or not.

In the first five cases, we can do rather better than simply counting all the $xy|z$ each time from scratch. Define:

$$\begin{aligned}
\mathit{count}3(k) &:= \#(\text{x,y,z}) \text{ such that } k \le x, y, z \text{ and } x < y \text{ and } \mathit{consistent}(x, y, z) \\
\mathit{count}2x(k, x) &:= \#(\text{y,z}) \text{ such that } k \le y, z \text{ and } \mathit{consistent}(x, y, z) \\
\mathit{count}2z(k, z) &:= \#(\text{x,y}) \text{ such that } k \le x < y \text{ and } \mathit{consistent}(x, y, z) \\
\mathit{count}1xy(k, x, y) &:= \#\text{z such that } k \le z \text{ and } \mathit{consistent}(x, y, z) \\
\mathit{count}1xz(k, x, z) &:= \#\text{y such that } k \le y \text{ and } \mathit{consistent}(x, y, z)
\end{aligned}$$

It is easy to see that we can compute all the above values in time $O(m^3)$ as follows:

$$\begin{aligned}
\mathit{count}3(k) &= \mathit{count}3(k+1) + \mathit{count}2x(k+1, k) + \mathit{count}2z(k+1, k) \\
\mathit{count}2x(k, x) &= \mathit{count}2x(k+1, x) + \mathit{count}1xz(k+1, x, k) + \mathit{count}1xy(k+1, x, k) \\
\mathit{count}2z(k, z) &= \mathit{count}2z(k+1, z) + \mathit{count}1xz(k+1, k, z) \\
\mathit{count}1xy(k, x, y) &= \mathit{count}1xy(k+1, x, y) + \mathit{consistent}(x, y, k) \\
\mathit{count}1xz(k, x, z) &= \mathit{count}1xz(k+1, x, z) + \mathit{consistent}(x, k, z)
\end{aligned}$$

Having preprocessed the above values, we can calculate each probability in time $O(1)$, giving us a total running time of $O(m^3 + n^2|T|)$. It turns out, however, that we can do slightly better. As it currently stands, for each leaf and each unused label we calculate the expected number of consistent triplets after assigning this label to this leaf separately. To further improve the complexity of the running time we can try to do all such calculations at once (for a fixed leaf): for each triplet and for each unused label we calculate the probability that this triplet is consistent after we use this label. (In other words, we switch from a leaf-label-triplet loop nesting to leaf-triplet-label). Then it turns out that those probabilities are the same for almost all unused labels. More formally:

Lemma 3.2.9 *Let Γ be the set of labelings that assign $\gamma(i)$ to leaf i for each $i = 1, 2, \ldots, k-1$ and Γ_x be the subset of Γ that assign x to leaf k. We can find x for which $E(f(N, \gamma_x, T))$ is maximum in time $O(|T|)$ assuming the above preprocessing.*

Proof: Let $e_x = E(f(N, \gamma_x, T))$. Each such e_x is a sum of probabilities corresponding to the elements of T. We will start with all e_x equal 0 and consider triplets one by one. For each triplet t we will increase the appropriate values of e_x by the corresponding probabilities. Again we should consider six cases depending on how $t = ab|c$ looks like. Let $A = \frac{1}{3\binom{n-k+1}{3}}$, $B = \frac{1}{\binom{n-k+1}{2}}$ and $C = \frac{1}{n-k+1}$:

1. Labels a, b, c have not been assigned yet. We should increase e_a and e_b by $\mathit{count}2x(k+1, k)\frac{B}{2}$, e_c by $\mathit{count}2z(k+1, k)B$ and the remaining e_l by $\mathit{count}3(k+1)A$.

2. $\gamma(x) = a$ but both b and c have not been assigned yet. We should increase e_b by $\mathit{count}1xy(k+1, a, k)C$, e_c by $\mathit{count}1xz(k+1, x, k)C$ and the remaining e_l by $\mathit{count}2x(k+1, x)\frac{B}{2}$.

3. $\gamma(z) = c$ but both a and b have not been assigned yet. We should increase e_a and e_b by $count1xz(k+1, k, z)C$ and the remaining e_l by $count2z(k, z)B$.

4. $\gamma(x) = a, \gamma(y) = b$ but c has not been assigned yet. We should increase e_c by $consistent(x, y, k)$ and the remaining e_l by $count1xy(k+1, x, y)C$.

5. $\gamma(x) = a, \gamma(z) = c$ but b has not been assigned yet. We should increase e_b by $consistent(x, k, z)$ and the remaining e_l by $count1xz(k+1, x, z)C$.

6. $\gamma(x) = a, \gamma(y) = b, \gamma(z) = c$. We should increase all e_l by $consistent(x, y, z)$.

The naive implementation of the above procedure would require time $O(|T|n)$. We can improve it by observing that we are interested only in the relative increment of the different e_x, not in their actual values. So, instead of increasing all e_l with $l \notin S$ by some δ (for some $S \subseteq X$), we can decrease all e_l with $l \in S$ by this δ. Then processing each triplet takes time $O(1)$ as we only have to change at most 3 values of e_x.

□

We may now compose these lemmas into the following.

Proof: [of Theorem 3.2.5] Consider the procedure sketched at the beginning of the subsection and implemented according to the above lemmas. By Lemma 3.2.6, it produces good labelings. By Lemma 3.2.7 and n applications of Lemma 3.2.9, it can be implemented to run in $O(m^3 + n|T|)$ time. □

3.2.4 Consequences

Theorem 3.2.5 gives a new perspective on the problem of approximately constructing phylogenetic networks. From the algorithm of Gąsieniec et al. [GJLO99] we can always construct a phylogenetic tree that is consistent with at least 1/3 of the the input triplets. In fact, the trees constructed by this algorithm are very specific - they are always caterpillars. (A caterpillar is a phylogenetic tree such that, after removal of leaves, only a directed path remains.) Theorem 3.2.5 implies that not only caterpillars, but all possible tree topologies have the property, that given any set of triplets we may find in polynomial time a proper assignment of species into leaves with the guarantee that the resulting phylogenetic tree is consistent with at least a third of the input triplets.

The generality of Theorem 3.2.5 makes it meaningful not only for trees, but also for any other subclass of phylogenetic networks (e.g. for level-k networks). Let us assume that we have focused our attention on a certain subclass of networks. Consider the task of designing an algorithm that for a given triplet set constructs a network from the subclass consistent with at least a certain fraction of the given triplets. A worst-case approach as described in this section will never give us a guarantee better than the maximum value of $\#N$ ranging over all topologies N in the subclass. Therefore, if we intend to obtain networks consistent with a big fraction of triplets and if our criteria is to maximize this fraction in the worst case,

then our task reduces to finding topologies within the subclass that are good for the full triplet set. Theorem 3.2.5 potentially has a further use as a mechanism for comparing the quality of phylogenetic networks generated by other methods, because it provides lower bounds for the fraction of T that a given topology and/or subclass of topologies can be consistent with. (Although a fundamental problem in phylogenetics [BSS04] [CRV07] [CMM05] [NSW$^+$03] [LR04], the science of network comparison is still very much in its infancy. In Section 3.4 we study a different approach to comparing networks. Notice that, the graphical comparison methods we consider are applicable mainly for the problem of comparing phylogenetic trees.)

For level-0 networks (i.e. phylogenetic trees) the problem of finding optimal topologies for the full triplet set is simple: any tree is consistent with exactly 1/3 of the full triplet set. For level-1 phylogenetic networks a topology that is optimal for the full triplet set was constructed in [JNS00]. We may use this network and Theorem 3.2.5 to obtain an algorithm that works for any triplet set and creates a network that is consistent with the biggest possible fraction of triplets in the worst case (see Section 3.2.5 for more details). For level-2 networks we do not yet know the optimal structure of a topology for the full triplet set, but we will show in Section 3.2.6 that we can construct a network that has a guarantee of being consistent with at least a fraction 0.61 of the input triplets.

3.2.5 Application to level-1 phylogenetic networks

In [JNS00] it was shown how to construct a special level-1 topology $C(n)$, which we call a *galled caterpillar*[2], such that $\#C(n) \geq \#N$ for all level-1 topologies N on n leaves. The existence of $C(n)$, which has a highly regular structure, was proven by showing that any other topology N can be transformed into $C(n)$ by local rearrangements that never decrease the number of triplets the associated network is consistent with. It was shown that $\#C(n) = S(n)/3\binom{n}{3}$, where $S(0) = S(1) = S(2) = 0$ and, for $n > 2$,

$$S(n) = max_{1 \leq a \leq n} \left\{ \binom{a}{3} + 2\binom{a}{2}(n-a) + a\binom{n-a}{2} + S(n-a) \right\}. \tag{3.1}$$

In Figure 3.3 an example of a galled caterpillar is shown. All galled caterpillars on $n \geq 3$ leaves consist of one or more *galls* chained together in linear fashion and terminating in a tail of one or two leaves. Observe that the recursive structure of $C(n)$ mirrors directly the recursive definition of $S(n)$ in the sense that the value of a chosen at recursion level k is equal to the number of leaves found in the kth gall, counting downwards from the root. In the definition of $C(n)$ it is not specified how the a leaves at a given recursion level are distributed within the gall, but it is easy to verify that placing them all on one side of the gall (as shown in the figure) is sufficient.

[2] In [JNS00] this is called a *caterpillar network*.

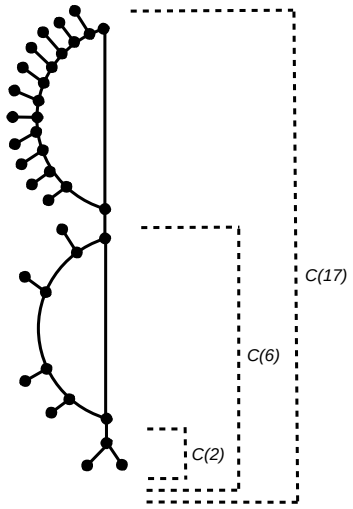


Figure 3.3: This is galled caterpillar $C(17)$. It contains two galls and ends with a tail of two leaves. $C(17)$ contains 11 leaves in the top gall because Equation 3.1 is maximized for $a = 11$.

Lemma 3.2.10 *Let T be a set of input triplets labelled by n species. Then, in time $O(n^3+n|T|)$, it is possible to construct a level-1 network $\mathcal{N}$, isomorphic to the galled caterpillar $C(n)$, consistent with at least a fraction $S(n)/3\binom{n}{3}$ of T.*

Proof: First we construct the level-1 topology $C(n)$. Using dynamic programming to compute all values of $S(n')$ for $0 \leq n' \leq n$ this can be done in time $O(n^2)$. Note that $C(n)$ contains in total $O(n)$ vertices. It remains only to choose an appropriate labeling of the leaves of $C(n)$, and this is achieved by substituting $C(n)$ for N in Theorem 3.2.5; this dominates the running time. □

Note that, because $C(n)$ achieves the best possible fraction for the input $T_1(n)$, the fraction achieved by Lemma 3.2.10 is worst-case optimal for all n. Empirical experiments suggest that the function $S(n)/3\binom{n}{3}$ is strictly decreasing and approaches a horizontal asymptote of 0.4880... from above; for values of $n = 10^1, 10^2, 10^3, 10^4$ the respective ratios are $0.511..., 0.490..., 0.4882..., 0.4880....$ It is difficult to formally prove convergence to 0.4880... so we prove a slightly weaker lower bound of 0.48 on this function. From this it follows that in all cases the algorithm described in Lemma 3.2.10 is guaranteed to produce a network consistent with at least a fraction 0.48 of T, improving considerably on the $5/12 \approx 0.4166$ fraction achieved in [JNS00].

Lemma 3.2.11 $S(n)/3\binom{n}{3} > 0.48$ *for all* $n \geq 0$.

Proof: This can easily be computationally verified for $n < 116$, we have done this with a computer program written in Java [Kel]. To prove it for $n \geq 116$, assume

by induction that the claim is true for all $n' < n$. Instead of choosing the value of a that maximizes $S(n)$ we claim that setting a equal to $z = \lfloor 2n/3 \rfloor$ is sufficient for our purposes. We thus need to prove the following inequality:

$$\frac{\binom{z}{3} + 2\binom{z}{2}(n-z) + z\binom{n-z}{2} + S(n-z)}{3\binom{n}{3}} > 48/100.$$

Combined with the fact that, by induction, $S(n-z)/3\binom{n-z}{3} > 48/100$, it is sufficient to prove that:

$$\frac{\binom{z}{3} + 2\binom{z}{2}(n-z) + z\binom{n-z}{2} + 144/100\binom{n-z}{3}}{3\binom{n}{3}} > 48/100$$

Using Mathematica we rearrange the a inequality to:

$$\frac{\lfloor 2n/3 \rfloor \left(22 + 9n + 33n^2 - 6(7+18n)\lfloor 2n/3 \rfloor + 86\lfloor 2n/3 \rfloor^2\right)}{n(2 - 3n + n^2)} < 0$$

Taking $(2n/3) - 1$ as a lower bound on $\lfloor 2n/3 \rfloor$, and $2n/3$ as an upper bound, it can be easily verified that the above inequality is satisfied for $n \geq 116$. □

To conclude this section we combine Lemmas 3.2.10 and 3.2.11 into the following Theorem.

Theorem 3.2.12 *Let T be a set of input triplets labelled by n species. In time $O(n^3 + n|T|)$ it is possible to construct a level-1 network $\mathcal{N}$ consistent with at least a fraction $S(n)/3\binom{n}{3} > 0.48$ of T, and this is worst-case optimal.*

3.2.6 A lower bound for level-2 networks

Theorem 3.2.13 *Let T be a set of input triplets labelled by n species. It is possible to find, in polynomial time, a level-2 network $\mathcal{N}(T)$ such that $\mathcal{N}(T)$ is consistent with at least a fraction 0.61 of T.*

Proof: We prove the theorem by showing how to construct a topology, which we call $LB_2(n)$, consistent with at least 0.61 of the triplets in $T_1(n)$. Using Theorem 3.2.5, $LB_2(n)$ can then be labelled to obtain the network $\mathcal{N}(T)$. We show by induction how $LB_2(n)$ can be constructed. We take $n < 16813$ as the induction base; for these values of n we refer to a simple computational proof written in Java [Kel]. We now prove the result for $n \geq 16813$. Let us assume by induction that, for any $n' < n$, there exists some topology $LB_2(n')$ such that $\#LB_2(n') \geq 0.61$. If we let $t(n')$ equal the number of triplets in $T_1(n')$ consistent with $LB_2(n')$, we have that $t(n')/3\binom{n'}{3} \geq 0.61$ and thus that $t(n') \geq 1.83\binom{n'}{3}$. Consider the structure in Figure 3.4. For $S \in \{A, B, C, D, E\}$, we define the operation *hanging l leaves from side S* as replacing the edge S with a directed path containing l internal vertices,

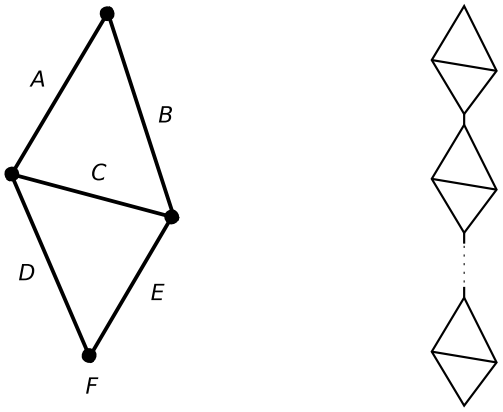


Figure 3.4: We construct the network $LB_2(n)$ by repeatedly chaining the structure on the left, a *simple level-2 network* together to obtain an overall topology resembling the structure on the right.

and then attaching a leaf to each internal vertex. We construct $LB_2(n)$ as follows. We create a copy of the structure from the figure and hang $c = \lfloor 0.385n \rfloor$ leaves from side C, $d = \lfloor 0.07n \rfloor$ from side D and $e = \lfloor 0.26n \rfloor$ from side E. We let $f = \lfloor 0.285n \rfloor$ and add the edge (F, r), where r is the root of the network $LB_2(f)$. Finally we hang $a = n - (c + d + e + f)$ leaves from side A; it might be that $a = 0$. (The only reason we hang leaves from side A is to compensate for the possibility that $c + d + e + f$ does not exactly equal n.) This completes the construction of $LB_2(n)$. Note that as in Section 3.2.5, the network is constructed by recursively chaining the same basic structure together.

We can use Mathematica to show that $LB_2(n)$ is consistent with at least 0.61 of the triplets in $T_1(n)$. In particular, by explicitly counting the triplets consistent with $LB_2(n)$ we derive an inequality expressed in terms of $n, c, d, e, f, t(f)$, which Mathematica then simplifies to a cubic inequality in n that holds for all $n \geq 16813$. (To simplify the inequality we take $x - 1$ as a lower bound on $\lfloor x \rfloor$ and assume that no leaves are hung from side A). The Mathematica script is reproduced in Figure 3.5, and can be downloaded from [Kel]. Finally, we comment that the networks computationally constructed for $n < 16813$ are, essentially, built in the same way as the networks described above. The only difference is that, to absorb inaccuracies arising from the floor function, we try several possibilities for how many leaves should be hung from each side; for side C, for example, we try also $(c - 1)$ and $(c + 1)$ leaves. □

3.2.7 Conclusions and open questions

With Theorem 3.2.5 we have described a method which shows how, in polynomial time, good solutions for the full triplet set can be efficiently converted into equally

```
c[n] = ((385 / 1000) * n) - 1
d[n] = ((70 / 1000) * n) - 1
e[n] = ((260 / 1000) * n) - 1
f[n] = ((285 / 1000) * n) - 1
t[n] = (3 * (610 / 1000) * Binomial[f[n], 3])


Simplify[
 (Binomial[c[n], 3] + Binomial[d[n], 3] + Binomial[e[n], 3] + t[n] + (Binomial[c[n], 2] * d[n]) +
     (Binomial[c[n], 2] * e[n]) + (Binomial[c[n], 2] * f[n]) + (c[n] * d[n] * e[n]) +
     (c[n] * d[n] * f[n]) + (Binomial[c[n], 2] * e[n]) + (c[n] * e[n] * d[n]) +
     (c[n] * e[n] * f[n]) + (Binomial[c[n], 2] * f[n]) + (c[n] * f[n] * d[n]) +
     (c[n] * f[n] * e[n]) + (Binomial[d[n], 2] * c[n]) + (Binomial[d[n], 2] * e[n]) +
     (Binomial[d[n], 2] * f[n]) + (d[n] * f[n] * c[n]) + (Binomial[d[n], 2] * f[n]) +
     (d[n] * f[n] * e[n]) + (Binomial[e[n], 2] * c[n]) + (Binomial[e[n], 2] * d[n]) +
     (Binomial[e[n], 2] * f[n]) + (e[n] * f[n] * c[n]) + (e[n] * f[n] * d[n]) +
     (Binomial[e[n], 2] * f[n]) + (Binomial[f[n], 2] * c[n]) + (Binomial[f[n], 2] * d[n]) +
     (Binomial[f[n], 2] * e[n])) / (3 * Binomial[n, 3]) > (610 / 1000) ]
```

$$\frac{-147\,984\,000\,000 + 94\,041\,640\,000\,n - 16\,470\,260\,400\,n^2 + 979\,319\,n^3}{n\left(2 - 3\,n + n^2\right)} > 0$$

```
Reduce[ (-147 984 000 000 + 94 041 640 000 n - 16 470 260 400 n^2 + 979 319 n^3) /
   (n (2 - 3 n + n^2)) > 0, n, Integers ]
```

$$n \in \text{Integers} \;\&\&\; (n \le -1 \;||\; n \ge 16\,813)$$

Figure 3.5: The Mathematica script used in the proof of Theorem 3.2.13.

good, or better, solutions for more general triplet sets. Where best-possible solutions are known for the full triplet set, this leads to worst-case optimal algorithms, as demonstrated by Theorem 3.2.12. An obvious next step is to use this method to generate algorithms (where possible worst-case optimal) for wider subclasses of phylogenetic networks. Finding the/an "optimal form" of level-2 networks for the full triplet set remains a fascinating open problem.

From a biological perspective (and from the perspective of understanding the relevance of triplet methods) it is also important to attach *meaning* to the networks that the techniques described in this section produce. For example, we have shown how, for level-1 networks, we can always find a network isomorphic to a galled caterpillar which is consistent with at least a fraction 0.48 of the input. If we do this, does the location of the species within this galled caterpillar communicate any biological information? Also, what does it say about the relevance of triplet methods, and especially the level-k hierarchy, if we know *a priori* that a large fraction (already for level 2 more than 0.61) of the input can be made consistent with some network from the subclass? And, as discussed in Section 3.2.4, how far can the techniques described in this section be used as a quality measure for networks produced by other algorithms?

As mentioned in the introduction, an algorithm guaranteed to find a network consistent with a fraction p' of the input trivially becomes a p'-approximation for the MAX variant of the problem (where we optimize not with respect to $|T|$ but with respect to the size of the optimal solution for T.) In fact, the best-known approximation factor for MAX-LEVEL-0 is 1/3, a trivial extension of the fact that $p = 1/3$ for trees [GJLO99]. On the other hand, the APX-hardness of this problem implies that an approximation factor arbitrarily close to 1 will not be possible. It remains a highly challenging open problem to determine whether better approximation factors

can be obtained for the latter problem via some different approach. (For some more discussion and partial results see the following section.) Alternatively, there could be some complexity theoretic reason why approximation factors better than p (where p is optimal in our formulation) are not possible. Under strong complexity-theoretic assumptions the best approximation factor possible for MAX-3-SAT, for example, uses a trivial upper bound of all the clauses in the input [Hås97], analogous perhaps to using $|T|$ as an upper bound.

3.3 An attempt to break the 1/3 barrier for trees

In this section we will list a number of partial results obtained when trying to improve the approximation ratio for the problem of finding the best tree for a given set of triplets, also called MAX-LEVEL-0.

As most of the results presented here are reductions between problems, let us first recall the names of the involved problems and the relevant approximation results.

MAX SUBDAG. Also called LINEAR ORDERING. The problem is to find a maximal cardinality, acyclic subset of edges in a given graph. There is a trivial 1/2-approximation: consider any ordering of vertices, try this and the opposite ordering; an edge is either consistent with one ordering or the other, hence one of these orderings induces a set of edges with cardinality at least half the total number of edges.

Concerning hardness: there is a lower bound of 65/66 [PY91].

FEEDBACK ARC SET. It is the dual of the above problem, it asks to minimize the number of edges necessary to remove. No constant factor approximation is known. There is a PTAS for tournaments [KMS07], and a Fixed Parameter Tractability (FPT) algorithm [CLL+08] (the number of edges to remove is the parameter, for a definition of FPT see Section 1.2).

MAX-LEVEL-0. In this problem we are given a set of triplets and we are supposed to find a tree that is consistent with the maximal number of triplets. By the algorithm of Gąsieniec et al. [GJLO99] we may find a tree consistent with 1/3 of the triplets, hence there is a 1/3-approximation algorithm.

By a reduction form MAX SUBDAG (see Section 3.3.2), there is no approximation better than 65/66.

MIN-LEVEL-0. It is dual of the above problem, the objective is to minimize the number of triplets that are not consistent with the resulting tree. Little is known about approximation for this problem.

MAX-LEVEL-0-LABELING. The problem is a variant of MAX-LEVEL-0, where the topology (i.e., the shape of the tree) is given as an input. By the ran-

domized relabeling methods described in the previous section we obtain a simple 1/3 approximation for this problem.

MAX-CATERPILLAR. A special case of MAX-LEVEL-0-LABELING, where the given shape is a caterpillar. Recall that caterpillar is a tree such that after removal of leaves (and edges incident to leaves) the remaining graph is a path.

The algorithm of Gąsieniec et al. [GJLO99] gives 1/3-approximation. Our reduction from MAX SUBDAG also proves 65/66-approximation hardness for this problem.

MIN-CATERPILLAR. As above, except that we minimize the number of inconsistent triplets.

3.3.1 A bottom-up 1/3-approximation algorithm for MAX-LEVEL-0

Here, we present a simple polynomial-time approximation algorithm for MAX-LEVEL-0 which always outputs a tree consistent with at least $\frac{1}{3}$ of the rooted triplets in the input set $\mathcal{T}$. The novel property of our algorithm is that it naturally combines the $\frac{1}{3}$ approximation guarantee with the property that, given a feasible set of triplets, the algorithm finds a tree consistent with all of them. Note, that the latter property is a feature of the algorithm of Aho et al. [ASSU81]. The structure of our algorithm is identical to that of the greedy bottom-up heuristic proposed by Wu [Wu04], but we use a new scoring function to give an easy proof of the approximation factor.

Intuitively, in each iteration the algorithm looks for two currently existing trees S_i, S_j whose leaves participate in many rooted triplets of the form $xy|z$ where x belongs to S_i, y belongs to S_j, and z does not belong to either S_i or S_j, and then merge S_i and S_j.

Let R be the final tree returned by Algorithm `Greedy_bottom-up`($\mathcal{T}$). To analyze the approximation factor, we introduce the following notation. For any node u of R, let $\mathcal{L}[u]$ be the set of leaf labels in the subtree of R rooted at u. For each internal node u in R, denote the two children of u by u_1 and u_2, and let $\mathcal{T}(u)$ be the subset of $\mathcal{T}$ defined by $\mathcal{T}(u) = \{xy|z \in \mathcal{T} : \exists a, b, c \in \{x, y, z\}$ such that $a \in \mathcal{L}[u_1]$, $b \in \mathcal{L}[u_2]$, and $c \notin \mathcal{L}[u_1] \cup \mathcal{L}[u_2]\}$. Note that for any two internal nodes u and v, $\mathcal{T}(u)$ and $\mathcal{T}(v)$ are disjoint. Also, each $xy|z \in \mathcal{T}$ belongs to $\mathcal{T}(u)$ for some internal node u. Thus, the internal nodes of R partition $\mathcal{T}$ into disjoint subsets. For each

Algorithm `Greedy_bottom-up`
Input: A set $\mathcal{T}$ of rooted triplets on a leaf set $L = \{\ell_1, \ell_2, \ldots, \ell_n\}$.
Output: A tree with leaf set L consistent with at least one third of the rooted triplets in $\mathcal{T}$.

1. Construct the set $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$, where each S_i is a tree consisting of a leaf labeled by ℓ_i.
2. Repeat $n-1$ times:
   (a) For every $S_i, S_j \in \mathcal{S}$, reset $score(S_i, S_j) := 0$.
   (b) For every $xy|z \in \mathcal{T}$ such that $x \in S_i$, $y \in S_j$, and $z \in S_k$ for three different trees S_i, S_j, S_k, update *score* as follows:
   $score(S_i, S_j) := score(S_i, S_j) + 2$;
   $score(S_i, S_k) := score(S_i, S_k) - 1$;
   $score(S_j, S_k) := score(S_j, S_k) - 1$.
   (c) Select $S_i, S_j \in \mathcal{S}$ such that $score(S_i, S_j)$ is maximum.
   (d) Create a tree S_k by connecting a new root node to the roots of S_i and S_j.
   (e) $\mathcal{S} := \mathcal{S} \cup \{S_k\} \setminus \{S_i, S_j\}$.
3. Return the tree in $\mathcal{S}$.

internal node u of R, we further partition the set $\mathcal{T}(u)$ into two disjoint subsets $\mathcal{T}(u)'$ and $\mathcal{T}(u)''$ where $\mathcal{T}(u)'$ are the rooted triplets in $\mathcal{T}(u)$ which are consistent with R and $\mathcal{T}(u)'' = \mathcal{T}(u) \setminus T(u)'$.

Lemma 3.3.1 *$|T(u)'| \geq \frac{1}{3} \cdot |T(u)|$ for all internal nodes u of R.*

Proof: Consider the iteration of Algorithm `Greedy_bottom-up`($\mathcal{T}$) in which the node u is created as a new root node for two trees S_i and S_j selected in Step 2c. Clearly, $score(S_i, S_j) \geq 0$. Moreover, by the definition of *score* in Steps 2a and 2b and the construction of R, we have $score(S_i, S_j) = 2 \cdot |\mathcal{T}(u)'| - |\mathcal{T}(u)''|$. Since $|\mathcal{T}(u)''| = |\mathcal{T}(u)| - |\mathcal{T}(u)'|$, we obtain $|T(u)'| \geq \frac{1}{3} \cdot |T(u)|$. □

Corollary 3.3.2 *For any set $\mathcal{T}$ of rooted triplets, Algorithm* `Greedy_bottom-up`($\mathcal{T}$) *returns a tree consistent with at least one third of the rooted triplets in $\mathcal{T}$.*

Proof: Follows directly from Lemma 3.3.1 and the fact that $\mathcal{T}$ is partitioned into disjoint subsets by the internal nodes of R. □

3.3.2 Reduction from MAX SUBDAG

Here we present a reduction that rules out, under standard complexity-theoretic assumptions, the existence of a PTAS (see Section 1.2 for a definition) for MAX-LEVEL-0, MAX-CATERPILLAR, and therefore also for MAX-LEVEL-0-LABELING. Moreover, this reduction also implies that constructing a $(2-\epsilon)$-approximation algorithm for any of these three problems would automatically give a nontrivial approximation algorithm for the MAX SUBDAG problem, which would be a major result.

Proposition 3.3.3 *MAX-LEVEL-0 and MAX-CATERPILLAR are both APX-complete.*

Proof: By Theorem 1 we may label any tree topology to make it consistent with 1/3 of the given triplets. Therefore, both problems are in the class APX. To prove APX-hardness we use a reduction proposed by Wu [Wu04] and we show that it is actually an *L-reduction* from the MAX SUBDAG problem. Both L-reductions and the MAX SUBDAG problem were studied by Papadimitiou and Yannakakis [PY91]. MAX SUBDAG, which is equivalent to the problem LINEAR ORDERING, has been proven APX-complete [NV01][PY91].

In the MAX SUBDAG problem we are given a directed graph $G = (V, A)$, and the goal is to find a maximal cardinality subset of arcs $A' \subset A$ such that $G' = (V, A')$ is acyclic.

In the reduction of Wu, one constructs an instance of the MAX-LEVEL-0 problem from a given directed graph $G = (V, A)$ as follows. Let $x \notin V$ and consider the set of triplets T containing a single triplet $t_{uv} = vx|u$ for every arc $(u, v) \in A$, where $X = X(T) = V \cup \{x\}$. To argue that it is an L-reduction it remains to prove the following two claims.

1) If there exists a subset of arcs $A' \subset A$ such that $G' = (V, A')$ is acyclic and $|A'| = k$, then there exists a phylogenetic tree consistent with at least k triplets from T. To prove this claim, we consider a topological sorting of vertices in graph G'. We construct a caterpillar with leaves labeled (top-down) by such sorted vertices, the lowest leaf is labelled by x. It remains to observe that for any arc $(u, v) \in A'$ the corresponding triplet t_{uv} is consistent with the obtained phylogenetic tree. Observe, that the fact that the constructed tree is actually a caterpillar makes the reduction only stronger and therefore applicable also to the MAX-CATERPILLAR problem.

2) Given a phylogenetic tree $\mathcal{B}$ consistent with l triplets from T, we may construct in polynomial time a subset of arcs $A' \subset A$ such that $G' = (V, A')$ is acyclic and $|A'| = l$. In fact we will show that it suffices to take A' consisting of the arcs (u, v) such that the corresponding triplet t_{uv} is consistent with $\mathcal{B}$. We only need to argue that for such a choice of A' the resulting graph $G' = (V, A')$ is acyclic. Consider the path in the tree $\mathcal{B}$ from the root node to the leaf labeled by the special species x. For any vertex $v \in A$, the species v has an internal node on this path where he branched out of the evolution of x, namely the lowest common ancestor of u and x ($LCA(u, x)$). Observe, that the position of $LCA(u, x)$ induces a partial ordering $>_{\mathcal{B}}$ on A. Recall, that if a triplet $t_{uv} = vx|u$ is consistent with $\mathcal{B}$, then $LCA(u, x)$

is a proper ancestor of $LCA(v, x)$. Therefore, the consistent triplets from T induce another partial ordering that may be extended to $>_{\mathcal{B}}$. This implies that for A' containing the arcs (u, v) such that a triplet t_{uv} is consistent with $\mathcal{B}$, the graph $G' = (V, A')$ is acyclic.

With the above construction we have shown that the existence of an ϵ-approximation algorithm for the MAX-LEVEL-0 problem implies existence of an ϵ-approximation algorithm for the MAXIMUM SUBDAG problem. In particular, existence of a (Polynomial-Time Approximation Scheme) PTAS for MAX-LEVEL-0 would imply existence of PTAS for MAXIMUM SUBDAG, which is unlikely due to the results in [NV01] and [PY91].

As mentioned in Claim 1), the networks produced by the reducion are caterpilars, hence the problem MAX-CATERPILLAR is also APX-hard.

□

3.3.3 MIN CATERPILLAR reduced to FEEDBACK ARC SET

We will now discuss a reduction form the MIN CATERPILLAR problem to the FEEDBACK ARC SET problem. The reduction is approximation preserving and it also translates the FPT algorithm for the FEEDBACK ARC SET problem to an FPT algorithm for the MIN CATERPILLAR problem.

Lemma 3.3.4 *Given a triplet set T on n species, we may construct a directed graph G with $n + |T|$ vertices and $3|T|$ edges such that there exists a feedback arc set of cardinality k if and only if there exists a "phylogenetic" caterpillar consistent with all but k triplets from t.*

Proof: The graph is constructed as follows. There are two groups of vertices, one represents the n species, and the other represent the triplets. For every triplet $t = xy|z \in T$ there are three directed edges in G: $z \to t$, $t \to x$, and $t \to y$.

Suppose there is a caterpillar C consistent with all but k triplets. We will show that there are k edges whose removal makes the graph acyclic. Consider the removal of the "$z \to t$" edge for each of the inconsistent triplets. We will construct a linear ordering consistent with the remaining edges. Let species be ordered as on the caterpillar C and let every consistent triplet t be placed just before its first short leg species. Finally, let the inconsistent triplets be placed before the first species in the ordering. It is easy to check that all the not deleted edges go along the chosen ordering.

Suppose there is a feedback arc set of size k. We may remove all the triplets incident to the feedback edges. The remaining graph is acyclic, hence we may take just the ordering of species and put them on the caterpillar accordingly. For any triplet $xy|z$ that was not removed, there are paths $z \to x$ and $z \to y$ in the graph, hence z is above x and y on the caterpillar, which makes $xy|z$ consistent. □

We may use the above reduction to translate approximation results for the FEEDBACK ARC SET to the MIN CATERPILLAR problem. In particular, the result of Even et al. [ENSS98] gives the following.

Corollary 3.3.5 *There exists a $O(\log n \log\log n)$-approximation algorithm for the MIN CATERPILLAR problem.*

We may also apply the recent result of Chen et al. [CLL+08] to obtain the following.

Corollary 3.3.6 *There exists an FPT algorithm for the MIN CATERPILLAR problem.*

3.3.4 Maximization reduced to minimization

In this section we prove that finding a constant factor approximation algorithm to the minimization version of a triplet consistency problem would automatically improve the approximation factor for the corresponding maximization problem. The following applies to both finding the best tree and finding the best caterpillar. Since the greedy algorithm of Gąsieniec et al. produces a network consistent with 1/3 of the input triplets (not only 1/3 of the triplets in the optimal solution), it is actually a better approximation when the optimum is small. Suppose there was a constant factor approximation algorithm for the minimization version, then we could combine it with the algorithm of Gąsieniec et al. to improve the factor of 1/3.

Lemma 3.3.7 *Given an α-approximation algorithm for the MIN-LEVEL-0 (MIN-CATERPILLAR), we may construct a $\frac{1}{3-2/\alpha}$-approximation algorithm for the MAX-LEVEL-0 (MAX-CATERPILLAR).*

Proof: The algorithm is simply to run both the α-approximation algorithm and the algorithm of Gąsieniec et al., and to return the better of the two solutions. Let $OPT(T)$ denote the fraction of triplets consistent with the optimal tree (caterpillar). We will argue that depending on the value of $OPT(T)$ either the first or the second algorithm gives a good approximation.

Observe that $1 - OPT(T)$ is the optimal fraction of inconsistent triplets, hence the α-approximation algorithm will return a tree (caterpillar) inconsistent with at most $\alpha(1 - OPT(T))$ fraction of triplets, hence consistent with at least $1 - \alpha(1 - OPT(T))$. Comparing it to the optimum gives the quality of $\frac{1-\alpha(1-OPT(T))}{OPT(T)} = \alpha - \frac{\alpha-1}{OPT(T)}$. Note, that this quantity is monotone increasing with $OPT(T)$.

The quality of the solution given by the algorithm of Jasper et al. compared to the optimum is $\frac{1/3}{OPT(T)}$ which is monotone decreasing with $OPT(T)$.

The worst possible value of $OPT(T)$ for the combination of these algorithms is when they give equally good solutions. We may calculate this value from $1/3 = 1 - \alpha(1 - OPT(T))$, which gives $OPT(T) = 1 - \frac{2}{3\alpha}$. Substituting this value to the analysis of one of the algorithms gives $\frac{1/3}{OPT(T)} = \frac{1/3}{1-\frac{2}{3\alpha}} = \frac{1}{3-2/\alpha}$. □

3.3.5 Additional remarks

It is a fascinating open problem, whether it is possible to approximate the best phylogenetic tree better. This seems to be related to the question about the nature

of the triplet constrains.

A natural next step is to try to apply semidefinite programming techniques to the problem. We shell report here, that SDP was successfully used in the context of a related problem called BETWEENNESS. In this problem we are asked to find and ordering of elements subject to constraints of the form "a should be between b and c". The result of Chor and Sudan [CS98] is that, in case of satisfiable instances, we may produce in polynomial time the ordering consistent with $1/2$ of the constraints. Note, that this result is weaker than what we can do for triplet consistency problems, where it is possible to actually construct the feasible tree is such exists. Therefore, the BETWEENNESS problem is not any easier than triplet consistence, and reducing triplet consistency to BETWEENNESS does not really help.

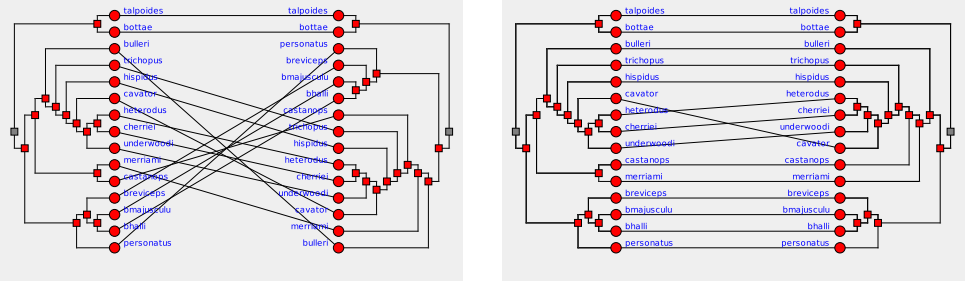
Another related result that should be mentioned is the heavy inconsistency of random triplet sets. Guillemot [Gui] studied the instances produced by taking each possible triplet on a fixed set of species with a fixed probability p. He proved, using existential arguments, that for such a random instance, with high probability, there is no phylogenetic network consistent with a nontrivial fraction of the given triplets (e.g. fraction less then $1/3$ for trees).

Having all these partial results we are still unable to make any step forward. A possible explanation would be, that the problem is actually hard to approximate any better than with a random solution. The relation between the MAX-CATERPILLAR problem and the MAX SUBDAG problem could probably be even more exploited. It is still possible, that the existence of $(3-\epsilon)$-approximation for MAX-CATERPILLAR is related to existence of $(2-\epsilon)$-approximation for MAX SUBDAG.

3.4 Comparing two trees

3.4.1 Introduction

In this section we are interested in drawing so-called *tanglegrams* [Pag02], that is, pairs of trees whose leaf sets are in one-to-one correspondence. The need to visually compare pairs of trees arises in applications such as the analysis of software projects, phylogenetics, or clustering. In the first application, trees may represent package-class-method hierarchies or the decomposition of a project into layers, units, and modules. The aim is to analyze changes in hierarchy over time or to compare human-made decompositions with automatically generated ones. Whereas trees in software analysis can have nodes of arbitrary degree, trees from our second application, that is, (rooted) phylogenetic trees, are binary trees. This makes binary tanglegrams an interesting special case, see Figure 3.6. Hierarchical clusterings, our third application, are usually visualized by a binary tree-like structure called *dendrogram*, where elements are represented by the leaves and each internal node of the tree represents the cluster containing the leaves in its subtree. Pairs of dendrograms stemming from different clustering processes of the same data can be compared visually using tanglegrams.



(a) arbitrary drawing

(b) drawing of our 2-approximation

Figure 3.6: A binary tanglegram showing two evolutionary trees for pocket gophers

In this section we consider binary tanglegrams if not stated otherwise. From the application point of view it makes sense to insist that (a) the trees under consideration are drawn planarly (namely, with no edge crossing), (b) each leaf of one tree is connected by an additional edge to the corresponding leaf in the other tree, and (c) the number of crossings among the additional edges is minimized. As in the bioinformatics literature (e.g., [Pag02; LPR$^+$07]), we call this the *tanglegram layout* (TL) problem; Fernau et al. [FKP05] refer to it as *two-tree crossing minimization*. Note that we are interested in the minimum number of crossings for visualization purposes. The number is not intended to be a tree-distance measure. Examples for such measures are nearest-neighbor interchange and subtree transfer [DHJ$^+$97].

Related problems. In graph drawing the so-called *two-sided crossing minimization problem* (2SCM) is an important problem that occurs when computing layered graph layouts. Such layouts have been introduced by Sugiyama et al. [STT81] and are widely used for drawing hierarchical graphs. In 2SCM, vertices of a bipartite graph are to be placed on two parallel lines (called *layers*) such that vertices on one line are incident only to vertices on the other line. As in TLs the objective is to minimize the number of edge crossings provided that edges are drawn as straight-line segments. In one-sided crossing minimization (1SCM) the order of the vertices on one of the layers is fixed. 1SCM is also NP-hard [EW94]. In contrast to TLs, a vertex in an instance of 1SCM or 2SCM can have several incident edges and the linear order of the vertices in the non-fixed layer is not restricted by the internal structure of a tree. The following is known about 1SCM in terms of approximation and exact algorithms. The median heuristic of Eades and Wormald [EW94] yields a 3-approximation and a randomized algorithm of Nagamochi [Nag05] yields an expected 1.4664-approximation. Dujmovič et al. [DFK04] gave an FPT algorithm that runs in $O^\star(1.4664^k)$ time, where k is the minimum number of crossings in any 2-layer drawing of the given graph that respects the vertex order of the fixed layer. The $O^\star(\cdot)$-notation ignores polynomial factors (see Section 1.2).

Previous work. Dwyer and Schreiber [DS04] studied drawing a series of tanglegrams in 2.5 dimensions, i.e., the trees are drawn on a set of stacked two-dimensional planes. They considered a one-sided version of the TL problem by fixing the layout of the first tree in the stack, and then, layer-by-layer, computing the leaf order of the next tree in $O(n^2 \log n)$ time each. Fernau et al. [FKP05] showed that the TL problem is NP-hard and gave a fixed-parameter algorithm that runs in $O^\star(c^k)$ time, where c is a constant that they estimate to be 1024 and k is the minimum number of crossings in any drawing of the given tanglegram. They showed that the problem can be solved in $O(n \log^2 n)$ time if the leaf order of one tree is fixed. This improves the result of Dwyer and Schreiber [DS04]. They also made the simple observation that the edges of the tanglegram can be directed from one root to the other. Thus the existence of a planar drawing can be verified using a linear-time upward-planarity test for single-source directed acyclic graphs [BDMT98]. Nearly ten years later, apparently not knowing these previous results, Lozano et al. [LPR$^+$07] gave a quadratic-time algorithm for the same special case, to which they refer as *planar tanglegram layout*.

Our results. We first take a closer look at the complexity of the TL problem, see Section 3.4.2. By a new reduction from Max2Sat we show that the TL problem is NP-hard even when restricted to *complete* binary trees. We further show that without this restriction, the TL problem is essentially as hard as the MinUncut problem. If the (widely accepted) Unique Games Conjecture holds, it is NP-hard to approximate MinUncut and thus TL within any constant factor.

Our main result is a 2-approximation for complete binary TLs that runs in $O(n^3)$ time, see Section 3.4.3. It can be generalized to complete d-ary trees, where it yields a factor-$(1 + \binom{d}{2})$ approximation in $O(n^{1+2\log_d(d!)})$ time. For $d \geq 3$ this is upper-bounded by $O(n^{2d-1.7})$.

Next we give a new fixed-parameter algorithm for complete binary TLs that is both much simpler and much faster than the FPT algorithm for *general* binary TLs by Fernau et al. The running time of our algorithm is $O^\star(4^k)$, see Section 3.4.4.

We have implemented our 2-approximation algorithm and have applied it to complete binary trees and, as a heuristic, to general binary trees. Our data consists of real-world phylogenetic trees and clustering dendrograms and of randomly generated examples. We compare our results with the optimal solutions computed with an integer quadratic program, see Section 3.4.5.

Formalization. We denote the set of leaves of a tree T by $L(T)$. We are given two rooted trees S and T with n leaves each. We require that S and T are *uniquely leaf-labeled*, that is, there are bijective labeling functions $\lambda_S : L(S) \to \Lambda$ and $\lambda_T : L(T) \to \Lambda$, where Λ is a set of labels, for example, $\Lambda = \{1, \ldots, n\}$. These labelings define a set of new edges $\{uv \mid u \in L(S), v \in L(T), \lambda_S(u) = \lambda_T(v)\}$, the *inter-tree edges*. The TL problem is to find plane drawings of S and T that minimize the number of induced crossings of the inter-tree edges, assuming that edges are drawn as straight-line segments. We additionally insist that the leaves in $L(S)$ are placed

on the vertical line $x = 0$ and those in $L(T)$ on the line $x = 1$. The trees S and T themselves are drawn to the left of $x = 0$ and to the right of $x = 1$, respectively. For an example, see Figure 3.6. We use notation $\langle S, T\rangle$ when referring to this instance of the TL problem.

The TL problem is purely combinatorial: Given a tree T, we say that a linear order of $L(T)$ is *compatible* with T if for each node v of T the nodes in the subtree of v form an interval in the order. Given a permutation π of $\{1, \ldots, n\}$, we call (i, j) an *inversion* in π if $i < j$ and $\pi(i) > \pi(j)$. For fixed orders σ of $L(S)$ and τ of $L(T)$ we define the permutation $\pi_{\tau,\sigma}$, which for a given position in τ returns the position in σ of the leaf having the same label. Now the TL problem consists of finding an order σ of $L(S)$ compatible with S and an order τ of $L(T)$ compatible with T such that the number of inversions in $\pi_{\tau,\sigma}$ is minimum.

3.4.2 Complexity

In this section we consider the complexity of the TL problem for complete and for general binary trees. Fernau et al. [FKP05] have shown that the TL problem is NP-complete for general binary trees. Their proof, however, uses extremely unbalanced trees and does not extend to complete binary trees. We show that the TL problem remains hard even when restricted to complete binary trees. We reduce from Max2Sat with at most 3 occurrences of each variable.

Our proof is completely different from that of Fernau et al., who reduce from MaxCut. We construct a TL instance (see Figure 3.7) in which one pair of aligned subtrees contains the variable gadgets. The two pairs of aligned subtrees to both sides of the variable gadgets contain the clause gadgets. The fourth pair of aligned subtrees on the same level has no crossings. Each clause gadget is modeled by a pair of smaller subtrees, see Figure 3.8. These are connected by inter-tree edges to the gadgets of the two corresponding variables. These edges cause exactly one additional crossing for each unsatisfied clause in an optimal solution. Thus we can infer the maximum number of satisfied clauses from an optimal TL solution.

Theorem 3.4.1 *The TL problem is NP-complete even for complete binary trees.*

Proof: Recall the Max2Sat problem which is defined as follows. Given a set $U = \{x_1, \ldots, x_n\}$ of Boolean variables, a set $C = \{c_1, \ldots, c_m\}$ of disjunctive clauses containing two literals each, and an integer K, the question is whether there is a truth assignment of the variables such that at least K clauses are satisfied. We consider a restricted version of Max2Sat, where each variable appears in at most three clauses. This version remains NP-complete [RRR98].

Our reduction constructs two complete binary trees S and T, in which certain aligned subtrees serve as variable gadgets and others as clause gadgets. We further determine an integer K' such that the instance $\langle S, T\rangle$ has less than K' crossings if and only if the corresponding Max2Sat instance has a truth assignment that satisfies at least K clauses.

The high-level structure of the two trees is depicted in Figure 3.7. From top to bottom, the four subtrees at level 2 on both sides are a clause subtree, a variable

subtree, another clause subtree, and finally a dummy subtree. The subtrees are connected to each other by edges such that in any optimal solution they must be aligned in the depicted (or mirrored) order. Each clause gadget appears twice, once in each clause subtree, and is connected to the variable gadgets belonging to its two literals. Pairs of corresponding gadgets in S and T are connected to each other. Finally, non-crossing dummy edges connect unused leaves to complete S and T. In the following we describe the gadgets in more detail.

Variable gadgets. The basic structure of a variable gadget consists of two complete binary trees with 32 leaves each as shown in Figure 3.8. Each tree has three highlighted subtrees of size 2 labeled a, b, c and a', b', c', respectively. From each of these subtrees there is one red *connector* edge leaving the gadget at the top and one leaving it at the bottom. As long as two connector edges from the same tree do not cross each other, they transfer the vertical order of the labeled subtrees towards a clause gadget. We define the configuration in Figure 3.8a as *true* and the configuration in Figure 3.8b as *false*. If the configuration is in its *true* state, the induced vertical order of the connector edges is $a < b < c$, otherwise the order is inverse: $c < b < a$. It can easily be verified that both states have the same number of crossings. To see that it is optimal observe that each pair of connector edges from the same subtree (for example, subtree a) always crosses all 26 gray edges in

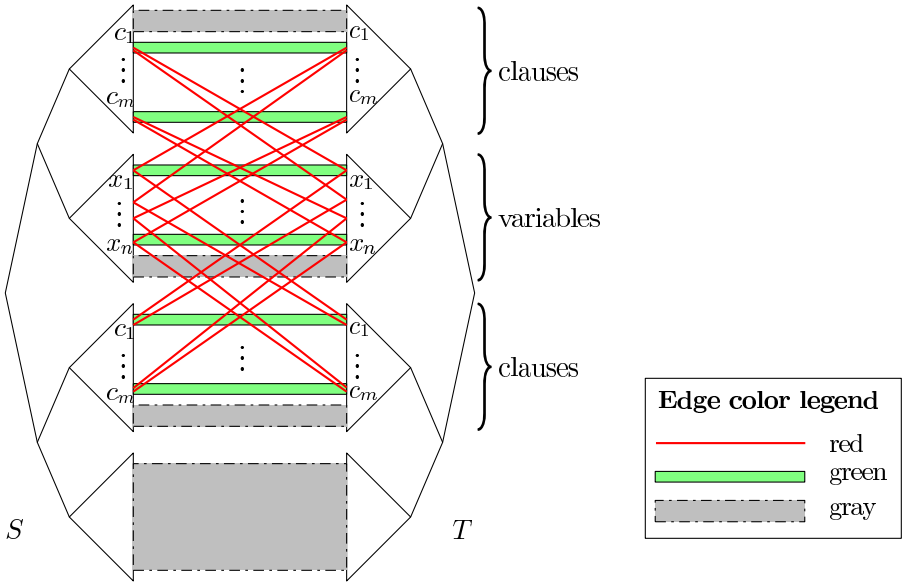
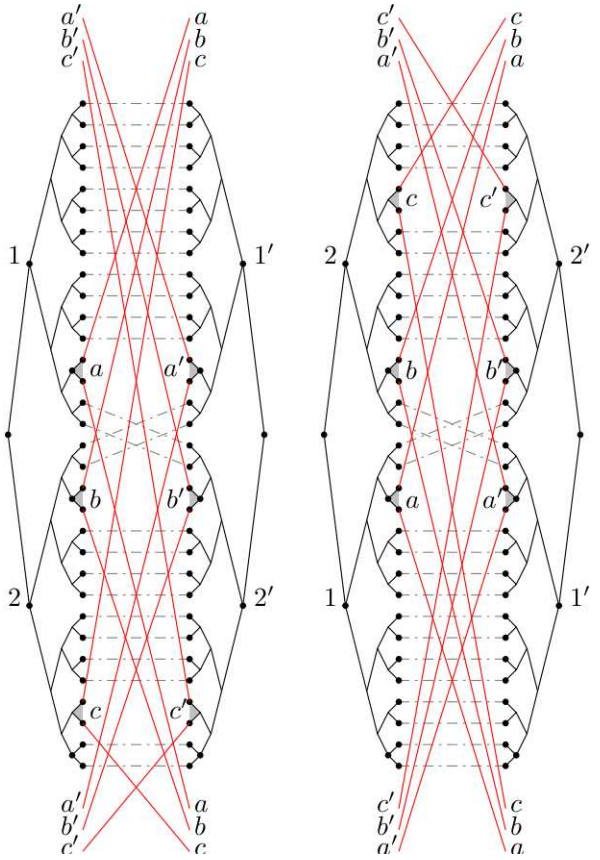
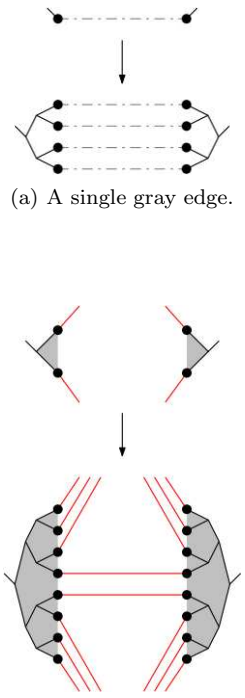


Figure 3.7: High-level structure of the two trees S and T. Red edges connect clause and variable gadgets, green edges connect corresponding gadget halves, and gray edges are dummy edges to complete the trees.



(a) $x = true$ (b) $x = false$

Figure 3.8: The variable gadget in its two optimal configurations with 184 crossings. Red edges are drawn solid, whereas dash-dot style is used for gray edges.



(a) A single gray edge.

(b) Two pairs of connector edges for a variable used in three clauses.

Figure 3.9: Replacing each edge by four edges.

the gadget. Furthermore all 24 crossings of two connector edges in the figure are mandatory. Finally, the four crossings among the gray edges between subtrees 1 and $2'$ and subtrees 2 and $1'$ are also optimal. (Otherwise, if subtree 1 is opposite of subtree $2'$, then there are at least 120 gray–gray crossings in addition to the 24 red–red crossings and the 156 red–gray crossings as opposed to a total of 184 crossings in either configuration of Figure 3.8.)

Note that so far the gadget in the figure is designed for a single appearance of the variable since the four connector-edge triplets are required for a single clause.

However, for the MAX2SAT reduction each variable can appear up to three times in different clauses. By appending a complete binary tree with four leaves as in Figure 3.9 to each leaf of the gadget in Figure 3.8 and copying each edge accordingly the above arguments still hold for the enlarged trees with 128 leaves each. Unused connector edges in opposite subtrees are linked to each other (a to a' etc.) as in Figure 3.9b such that the number of crossings in the gadget remains balanced for both states.

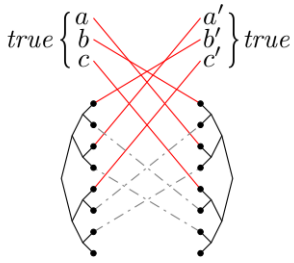
Clause gadgets. For each clause $c_i = l_{i1} \vee l_{i2}$, where l_{i1} and l_{i2} denote the two literals, we create two clause gadgets: one in the upper clause subtrees and one in the lower clause subtrees (recall Figure 3.7). Each gadget itself consists of two parts: one part that uses the connectors from the first variable in the left tree and those from the second variable in the right tree and vice versa. Figure 3.10 shows one such part of the gadget in the lower clause subtrees, where the connector edges lead upwards. The gadget in the upper clause subtree is simply a mirrored version.

The basic structure consists of two aligned subtrees with eight leaves as depicted in Figure 3.10. Three of the leaves on each side serve as the missing endpoints for the triplets of connector edges from the corresponding variables. Recall that for a positive literal with value *true* the order of the connector edges is $a < b < c$, and for a positive literal with value *false* it is $c < b < a$. (For negative literals the meaning of the orders is inverted.) The two connector leaves for the edges labeled a and b are in the same subtree with four leaves, the connector leaf for c is in the other subtree. Three cases need to be distinguished. If (1) both literals are *true* then the configuration in Figure 3.10a is optimal with 21 crossings. If (2) only one literal is *true* then Figure 3.10b shows an optimal configuration with 21 crossings again. Here the tree on the right side is rotated in its root node. Finally, if (3) both literals are *false* then there are at least 22 crossings in the gadget as shown in Figure 3.10c. Since this substructure is repeated four times for each clause we have 84 induced crossings for satisfied clauses and 88 induced crossings for unsatisfied clauses.

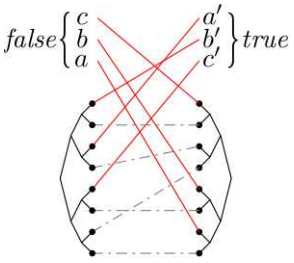
We construct the gadgets for all variables and clauses and link them together as two trees S and T, which are filled up such that they become complete binary trees. The general layout is as depicted in Figure 3.7, where each dummy leaf in S is connected to the opposite dummy leaf in T such that there are no crossings among dummy edges. In each of the four main subtrees all dummy edges are consecutive. Thus of all dummy edges only those in the variable subtree have crossings with exactly half the connector edges.

It remains to compute the minimum number M of crossings that are always necessary, even if all clauses are satisfied. Then the MAX2SAT instance has a solution with at least K satisfied clauses if and only if the constructed TL instance has a solution with at most $K' = M + 4(|C| - K)$ crossings. We get the corresponding variable assignment directly from the layout of the variable gadgets.

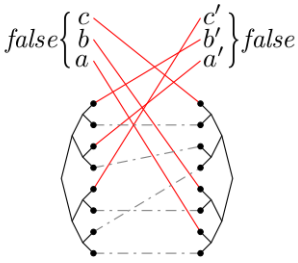
The first step for computing M is to fix an order for the variable gadgets in the variable subtree. Let this order be $x_1 < x_2 < \ldots < x_n$. To enforce this as the vertical order of the variable gadgets we need to establish links between adjacent



(a) *true* ∨ *true*: 21 crossings.



(b) *false* ∨ *true*: 21 crossings.



(c) *false* ∨ *false*: 22 crossings.

Figure 3.10: The clause gadget for a clause $c_i = l_{i1} \vee l_{i2}$.

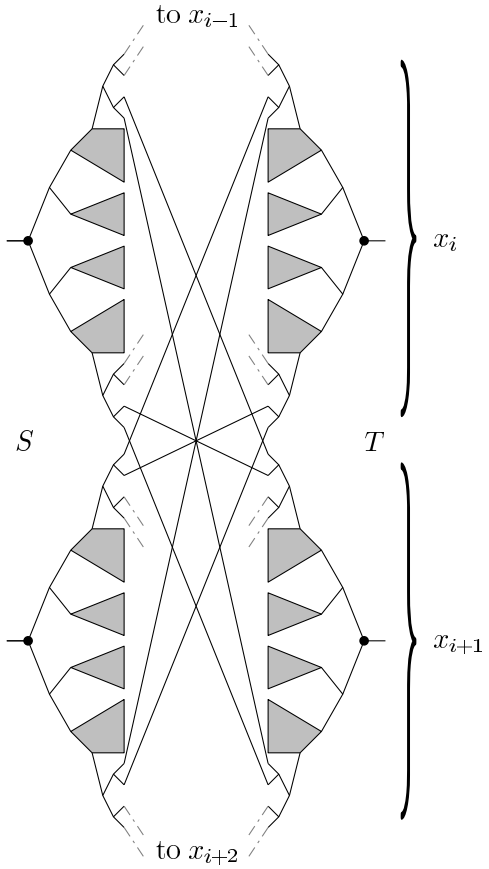


Figure 3.11: Linking adjacent variable gadgets for x_i and x_{i+1}.

gadgets such that any other order would increase the number of crossings. For these neighbor links we need eight of the 128 leaves in each half of each variable gadget as shown in Figure 3.11. Since both subtrees below the root of x_i in S and both subtrees below the root of x_{i+1} in T are connected to each other, the minimum number of crossings of those edges is independent of the truth state of each gadget. However, separating two adjacent variables by tree rotations at higher levels in S and T leads to a large number of extra crossings since the eight neighbor links would cross all variable gadgets between x_i and x_{i+1}.

With the order of the variables fixed we sort all clauses lexicographically and place smaller clauses towards the top of the clause subtrees. Consider two clause

gadgets in the same clause subtree. Then in the given clause order there are crossings between their connector-edge triplets if and only if the intervals between their respective variables intersect in the variable order. Since these crossings are unavoidable, the number of connector-triplet crossings in the lexicographic order of the clauses is optimal. Now we can finally compute all necessary crossings between connector edges, dummy edges and intra-gadget edges which yields the number M.

Since each gadget is of constant size the two trees and the number M can be computed in polynomial time.

The fact that the complete binary TL problem belongs to the class $\mathcal{NP}$ follows immediately from the NP-completeness of the general TL problem [FKP05]. □

Next we consider the complexity of the TL problem for two (not necessarily complete) binary trees. We show that this problem is essentially as hard as the MinUncut problem. As a result, we relate the existence of a constant-factor approximation for TL to the Unique Games Conjecture (UGC) by Khot [Kho02]. The UGC became famous when it was discovered that it implies optimal hardness-of-approximation results for problems such as MaxCut and VertexCover, and forbids constant factor-approximation algorithms for problems such as MinUncut and SparsestCut. We reduce the MinUncut problem to the TL problem, which, by the result of Khot and Vishnoi [KV05], makes it unlikely that an efficient constant-factor approximation for TL exists.

The MinUncut problem is defined as follows. Given an undirected graph $G = (V, E)$, find a partition (V_1, V_2) of the vertex set V that minimizes the number of edges that are not cut by the partition, that is, $\min_{(V_1,V_2)} |\{uv \in E : u, v \in V_1 \text{ or } u, v \in V_2\}|$. Note that computing an optimal solution to MinUncut is equivalent to computing an optimal solution to MaxCut. Nevertheless, the MinUncut problem is more difficult to approximate.

Theorem 3.4.2 *Under the Unique Games Conjecture it is NP-hard to approximate the TL problem for general binary trees within any constant factor.*

Proof: As mentioned above we reduce from the MinUncut problem. Note that our reduction is similar to the one in the NP-hardness proof by Fernau et al. [FKP05].

Consider an instance $G = (V, E)$ of the MinUncut problem. We will construct a TL instance $\langle S, T\rangle$ as follows. The two trees S and T are identical and there are three groups of edges connecting leaves of S to leaves of T. For simplicity we define multiple edges between a pair of leaves. In the actual trees we can replace each such leaf by a binary tree with the appropriate number of leaves.

Suppose $V = \{v_1, v_2, \ldots, v_n\}$, then both S and T are constructed as follows. There is a *backbone* path $(v_1^1, v_1^2, v_2^1, v_2^2, \ldots, v_n^1, v_n^2, a)$ from the root node v_1^1 to a leaf a. Additionally, there are leaves $l_S(v_i^j)$ and $l_T(v_i^j)$ attached to each node v_i^j for $i \in \{1, \ldots, n\}$ and $j \in \{1, 2\}$ in S and T, respectively. The edges form the following three groups.

Group A contains n^{11} edges connecting $l_S(a)$ with $l_T(a)$.

Group B contains for every $v_i \in V$ n^7 edges connecting $l_S(v_i^1)$ with $l_T(v_i^2)$, and n^7 edges connecting $l_S(v_i^2)$ with $l_T(v_i^1)$.

Group C contains for every $v_iv_j \in E$ a single edge from $l_S(v_i^1)$ to $l_T(v_j^1)$.

Suppose that in the optimal partition (V_1^*, V_2^*) of G there are k edges that are not cut. Then we claim that there exists a drawing of $\langle S, T\rangle$ such that $k \cdot n^{11} + O(n^{10})$ pairs of edges cross. It suffices to draw, for each vertex $v_i \in V_1^*$ ($v_i \in V_2^*$), the leaves $l_S(v_i^1)$ and $l_T(v_i^2)$ above (below) the backbones, and the nodes $l_S(v_i^2)$ and $l_T(v_i^1)$ below (above) the backbones. It remains to count the crossings: there are $k \cdot n^{11}$ A–C crossings, no A–B crossings, $O(n^{10})$ B–C crossings, and $O(n^4)$ C–C crossings.

Now suppose there exists an α-approximation algorithm for the TL problem with some constant α. Then it can produce a drawing $D(S, T)$ with at most $\alpha \cdot k \cdot n^{11} + O(n^{10})$ crossings. Let's assume that n is much larger than α. We show that from such a drawing $D(S, T)$ we would be able to reconstruct a cut (V_1, V_2) in G with at most $\alpha \cdot k$ edges uncut. First, observe that if a node $l_S(v_i^1)$ is drawn above (below) the backbone in $D(S, T)$, then $l_T(v_i^2)$ must be drawn on the same side of the backbone, otherwise it would result in n^{18} A–B crossings. Similarly $l_S(v_i^2)$ must be on the same side as $l_T(v_i^1)$. Then observe that if a node $l_S(v_i^1)$ is drawn above (below) the backbone in $D(S, T)$, then $l_S(v_i^2)$ must be drawn below (above) the backbone, otherwise there would be $O(n^{14})$ B–B crossings. Finally, observe that if we interpret the set of vertices v_i for which $l_S(v_i^1)$ is drawn above the backbone as a set V_1 of a partition of G, then this partition leaves at most $\alpha \cdot k$ edges from E uncut.

Hence, an α-approximation for the TL problem provides an α-approximation for the MinUncut problem, which contradicts the UGC. □

3.4.3 Approximation

We now present our main result, a 2-approximation algorithm for TLs that runs in $O(n^3)$ time. The idea is to split the problem recursively at the root of the trees into two subproblems, each consisting of a pair of complete binary trees.

Let $\langle S_0, T_0\rangle$ be the TL instance we want to solve. At a given level $\langle S, T\rangle$ in the recursion, we have two trees S and T, typically part of larger trees (that is, $S \subseteq S_0$ and $T \subseteq T_0$). Let the roots of S and T be v_S and v_T, respectively. Besides the two tress, we will use some additional information.

Firstly, associated with v_S and v_T we will have labels ℓ_S and ℓ_T that indicate what choices in the recursion so far led to the current subproblems. A label is a binary string, where '0' or '1' represents each of the two choices at each node in the path from the root of the original tree, to the current root. The length of the labels (denoted $|\ell_S|$ and $|\ell_T|$) gives the depth of the recursion (see Fig. 3.13).

We also assign labels to some other subtrees of $\langle S_0, T_0\rangle$ besides S and T. Given a leaf $v \in T_0 \setminus T$, we define the *nc-subtree* of v, with respect to T, as the largest complete binary subtree of T_0 that does not contain T and contains v (defined analogously for leaves in S_0). Each different nc-subtree receives a label, in the same

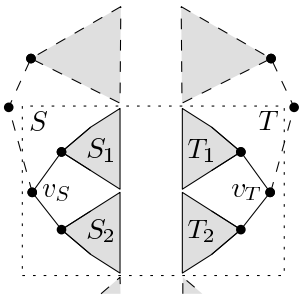


Figure 3.12: Context of subproblem $\langle S, T\rangle = \langle (S_1, S_2), (T_1, T_2)\rangle$.

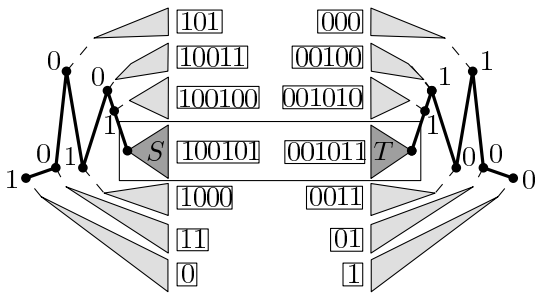


Figure 3.13: Labels for a particular subproblem $\langle S, T\rangle$. The numbers at the nodes show the choice taken (swap/do not swap children) at that step of the recursion that led to S and T.

way as S and T. For a given $\langle S, T\rangle$, there are $2(|\ell_S| + 1) = 2(|\ell_T| + 1)$ different labels. Note that the labels of the nc-subtrees are relative to the labels of v_S and v_T (different S or T will lead to different labels). We will sometimes refer to the label of leaf v, meaning the label of the nc-subtree of v.

Secondly, since S and T are part of a larger tree, some of the leaves of S may not have the matching leaf in T (and vice versa). This means that at some previous step of the algorithm, it was decided that such leaves will be matched to leaves in some other subtrees, above or below $\langle S, T\rangle$. We will not know exactly to which leaves they are matched, but we will know, for each leaf, the label of the subtree that contains the matching leaf.

At each level of the recursion we have to choose between one out of four configurations. At each node v_S on the left side, we must choose between having S_1 above S_2 or the other way around. On the right side for v_T, there are also two different ways of placing T_1 and T_2. We will try each of them, invoking the algorithm recursively for the top half and for the bottom half. Then we will return the configuration with the lowest number of crossings.

When counting the crossings that each option creates, we will distinguish two types: *current-level* and *lower-level* crossings.

Current-level crossings are crossings that can be avoided at this level by choosing one of the four configurations for the subtrees, independently of the choices to be done elsewhere in the recursion. Figure 3.14 illustrates the different types of current-level crossings. For the fourth type, (d), shown in Figure 3.14, we remark that the crossings are considered to be *current-level* only if the nc-subtrees that contain the endpoints of the edges outside S and T are different. Crossings that have the shape of type (d) but with both endpoints going to the same nc-subtree cannot be counted at this point, and will be called *indeterminate crossings*.

Lower-level crossings are crossings that appear based on choices taken by solving the subproblems of S and T recursively. We cannot do anything about them at this

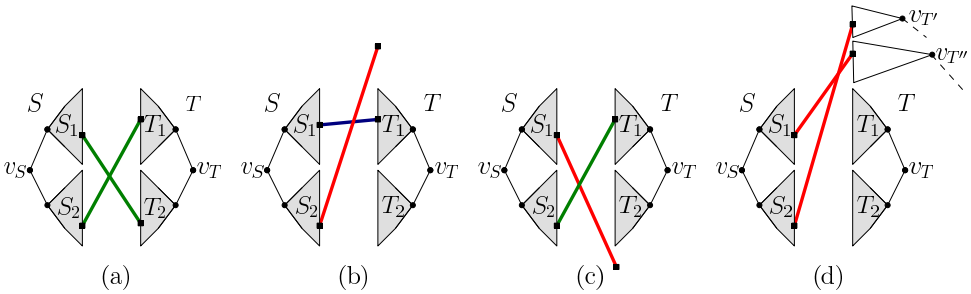


Figure 3.14: Different types of current-level crossings. For the fourth type, (d), the crossing is considered current-level only if the right leaves of the edges that cross have different labels, that is, $\ell_{T'} \neq \ell_{T''}$.

level, but we know their exact number after solving the subproblems.

Here's a sketch of the algorithm.

1. For all four choices of arranging $\{S_1, S_2\}$ and $\{T_1, T_2\}$, compute the total number of lower-level crossings recursively. Before each recursive call $\langle S_i, T_j \rangle$, we assign proper labels to some of the leaves of S and T, as follows. We label all leaves in S_i that connect to T_{3-j} (that is, T_1 if $j = 2$, T_2 otherwise) with $2\ell_T$ plus 0 or 1 depending on whether T_j is above or below T_{3-j}. Then we do the analogue for all leaves of T_j connected to S_{3-i}.

2. For each choice $\langle S_i, T_j \rangle$ compute the number of current-level crossings (details below).

3. Return the choice that has the smallest sum of lower-level and current-level crossings.

It is important to notice that the labels are needed to propagate as much information as possible to the smaller subproblems. For example, even though at this stage of the recursion it is clear that the leaves of, say T_{3-j}, are above the leaves of the subtrees below T, once we recurse into the top subproblem, this information will be lost, implying that what was a current-level crossing at this stage, will become an indeterminate crossing later. The labeling allows to prevent this loss of information.

It remains to describe how to compute the number of current-level crossings efficiently. This can be done as follows. We go through all inter-tree edges incident to leaves of each of the four subtrees and put each edge into one of at most $O(\log n)$ different classes depending on the labels of the other endpoints of the edges. Depending on where (that is, above or below) the nc-subtrees go, all edge pairs belonging to a specific pair of labels do or do not intersect. Hence we can count the total number of current-level crossings in linear time.

The running time of the algorithm satisfies the recurrence relation $T(n) \leq 8T(n/2) + O(n)$, which resolves to $T(n) = O(n^3)$ by the master method [CLRS01].

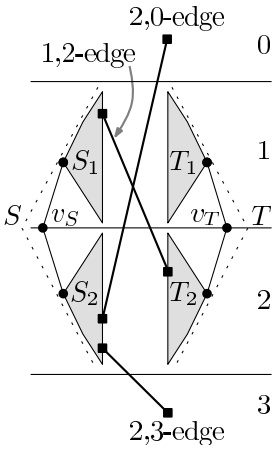


Figure 3.15: The possible locations for the endpoints of the edges are divided into four areas (numbered from 0 to 3). Each edge can be classified according to the areas of its endpoints.

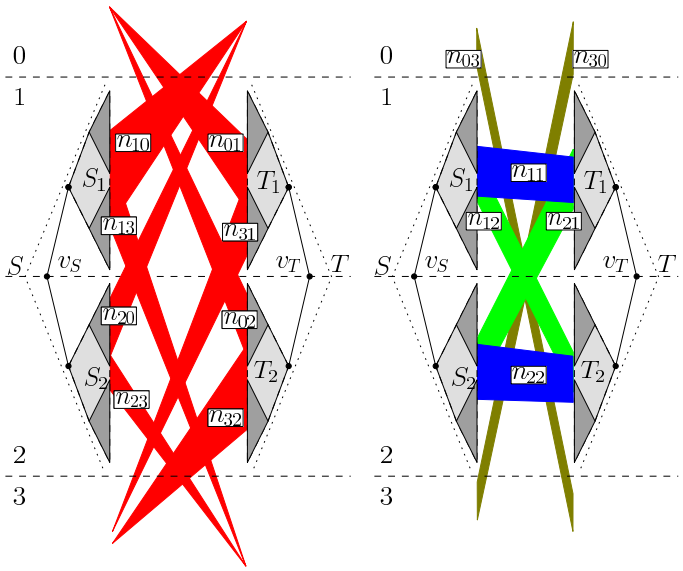


Figure 3.16: The 14 different (relevant) groups of edges in $\langle (S_1, S_2), (T_1, T_2) \rangle$.

Theorem 3.4.3 *The recursive algorithm computes a solution to the complete binary TL problem in $O(n^3)$ time. The resulting drawing has at most twice as many crossings as an optimal drawing.*

Proof: The algorithm will try, for a given subproblem $\langle S, T \rangle$, all four possible layouts of $S = (S_1, S_2)$ and $T = (T_1, T_2)$. Hence we can assume we know the order of the children of v_S and v_T in an optimal solution. Assume, w.l.o.g., that it is $\langle (S_1, S_2), (T_1, T_2) \rangle$. We distinguish between four different areas for the endpoints of the edges: above $\langle S, T \rangle$, in $\langle S_1, T_1 \rangle$, in $\langle S_2, T_2 \rangle$, and below $\langle S, T \rangle$. We number these regions from 0 to 3 (see Figure 3.15). This allows us to classify the edges into 16 groups (two of which, 0–0 and 3–3, are not relevant). We will denote the number of edges from area i to area j by n_{ij} (for $i, j \in \{0, 1, 2, 3\}$). Figure 3.16 shows the 14 different groups of edges.

The only edge crossings that our recursive algorithm cannot take into account are the indeterminate crossings, which occur when the two edges connect to leaves above/below $\langle S, T \rangle$, that are in the same nc-subtree (thus both leaves have the same label). The occurrence of such a crossing cannot be determined from the current subproblem because it depends on the relative location of the other two endpoints of the edges. However, we can bound the number of these crossings.

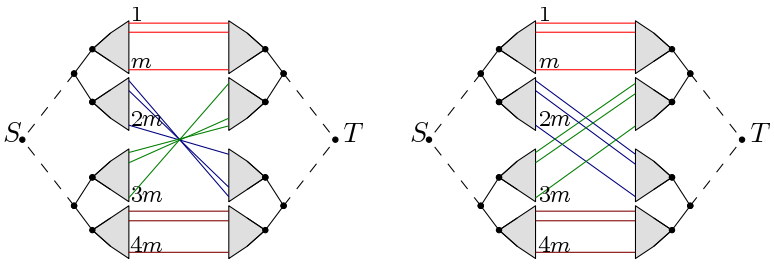


Figure 3.17: Example of trees for which the approximation algorithm can output a solution (left) that has roughly twice as many crossings as the optimal one (right).

We observe that any crossing of that type at the current subproblem was, in some previous step of the recursion, a crossing between two 1,2-edges or two 2,1-edges. We can upper-bound the number of these crossings by $\binom{n_{12}}{2} + \binom{n_{21}}{2}$. Let ALG be the number of crossings in the solution produced by the algorithm, and OPT the one in an optimal solution. We have

$$ALG \leq OPT + \binom{n_{12}}{2} + \binom{n_{21}}{2} \leq OPT + (n_{12}^2 + n_{21}^2)/2 \tag{3.2}$$

Since our (sub)trees are complete, we have $n_{10} + n_{12} + n_{13} = n_{01} + n_{21} + n_{31}$ and $n_{01}+n_{02}+n_{03} = n_{10}+n_{20}+n_{30}$. These two equalities yield $n_{12} \leq n_{01}-n_{10}+n_{21}+n_{31}$ and $n_{01}-n_{10} \leq n_{20}+n_{30}$, respectively, and thus we obtain $n_{12} \leq n_{20}+n_{30}+n_{21}+n_{31}$ or, equivalently, $n_{12}^2 \leq n_{12} \cdot (n_{20} + n_{30} + n_{21} + n_{31})$.

It is easy to verify that all the terms on the right-hand side of the last inequality count crossings that cannot be avoided and must be present in the optimal solution as well. Hence $n_{12}^2 \leq OPT$, and symmetrically $n_{21}^2 \leq OPT$. Plugging this into (3.2), we get $ALG \leq 2 \cdot OPT$. □

The approximation factor of 2 is tight: let $n = 4m$ and let S have leaves ordered $1, \ldots, 4m$ and let T have leaves ordered $1, \ldots, m, 3m, \ldots, 2m+1, m+1, \ldots, 2m, 3m+1, \ldots, 4m$. Then our algorithm can construct a drawing with $m^2+2\binom{m}{2} = m(2m-1)$ crossings, while the optimal drawing has only m^2 crossings (see Figure 3.17).

Generalization to d-ary trees. The recursive algorithm can be generalized to complete d-ary trees. The recurrence relation of the algorithm's running time changes to $T(n) \leq d \cdot (d!)^2 \cdot T(n/d) + O(n)$ since we need to consider all $d!$ subtree orderings of both trees, each of which triggers d subinstances of size n/d. Again, by the master method, this resolves to $T(n) = O(n^{1+2\log_d(d!)})$. At the same time the approximation factor increases to $1 + \binom{d}{2}$.

Maximization version. Instead of the original TL problem, which minimizes the number of pairs of edges that cross each other, we may consider the dual problem TL* of maximizing the number of pairs of edges that do not cross. The tasks of

finding optimal solutions for these problems are equivalent, but from the perspective of approximation it makes quite a difference which of the two problems we consider. Now we do not assume that we draw *binary* trees. Instead, if an internal node has more than two children, we assume that we may only choose between a given permutation of the children and the reverse permutation obtained by flipping the whole block of children.

In contrast to the TL problem, which is hard to approximate as we have shown in Theorem 3.4.2, the TL* problem has a constant-factor approximation algorithm. We show this (see the appendix) by reducing TL* to a constrained version of the MaxCut problem, which can be approximately solved with a semidefinite programming rounding algorithm by Goemans and Williamson [GW95].

Theorem 3.4.4 *There exists a* 0.878*-approximation algorithm for the TL* problem.*

Proof: Fix any drawing of the two trees S and T in an instance of the TL* problem. Any internal node of each of the trees corresponds to a decision variable. The decision to make in each such node is whether to flip the subtree rooted in that node or not. We model this situation by a graph; a flip decision corresponds to deciding to which side of a cut the corresponding vertex is assigned.

For each internal node v of a tree in the instance of TL* the constructed graph G contains two vertices v and v'. For each pair of edges connecting leaves of the two trees, there is one edge in G. Let l_1 and l_2 (r_1 and r_2) denote the leaves of S (T) incident to this pair of edges. Let l be the lowest common ancestor of l_1 and l_2 in S ($l = \mathrm{LCA}(l_1, l_2)$) and let $r = \mathrm{LCA}(r_1, r_2)$ in T. If the considered pair of edges crosses in the initial drawing, then we have an edge $\{l, r\}$ in G. If the pair of edges does not cross in the initial drawing, then there is an edge $\{l, r'\}$ in G.

It remains to observe that cuts in G that separate each pair v, v' correspond to drawings of S and T in the instance of the TL* problem. Moreover, edges that are cut in G correspond to the pairs of edges that do not cross in the drawing of the two trees.

The resulting optimization problem is the MaxResCut problem (that is, the MaxCut problem with additional constraints forcing certain pairs of vertices to be separated by the cut) studied by Goemans and Williamson [GW95]. Therefore, we may use their semidefinite programming rounding algorithm to compute a 0.878-approximation of the largest constrained cut in the graph G. This cut determines which of the subtrees in the initial drawing must be flipped to obtain a drawing that is a 0.878-approximation to TL*. □

3.4.4 Fixed-Parameter Tractability

We consider the following parametrized problem. Given an instance $\langle S, T\rangle$ of the complete binary TL problem and a non-negative integer k, decide whether there exist drawings of S and T with at most k induced crossings. Our algorithm for this problem uses a labeling strategy, just as our approximation algorithm in Section 3.4.3. However, here we do not select the subinstance that gives the minimum

number of lower-level crossings, but we consider all subinstances and recurs on them. Thus, our algorithm traverses a search tree of branching factor 4. For the search tree to have bounded height, we need to ensure that whenever we go to a subinstance, the parameter value decreases at least by one. For efficient bookkeeping we only consider current-level crossings. At first sight this seems problematic: if a subinstance does not incur any current-level crossings, the parameter will not drop. The following lemma shows that there is a way out. It says that if there is a subinstance without current-level crossings, then we can ignore the other three subinstances and do not have to branch. Note that the lemma does not hold for general binary trees.

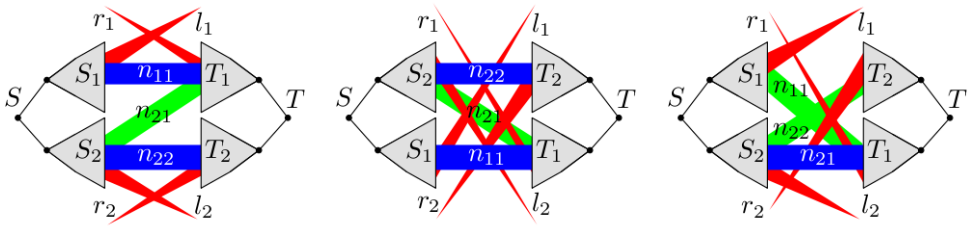
Lemma 3.4.5 *Given a pair $\langle S, T\rangle$ of two complete binary trees as an instance of the TL problem and two nodes v_S, v_T of S, T, respectively, with the same distance to their respective root. Let (S_1, S_2) be the subtrees incident to v_S and (T_1, T_2) the subtrees incident to v_T. If the subinstance $\langle (S_1, S_2), (T_1, T_2)\rangle$ does not incur any current-level crossings, then any ordering of the leaves of this subinstance does not have more crossings than the same ordering of the leaves of one of the other subinstances $\langle (S_1, S_2), (T_2, T_1)\rangle$, $\langle (S_2, S_1), (T_1, T_2)\rangle$, or $\langle (S_2, S_1), (T_2, T_1)\rangle$.*

Proof: If the subinstance $\langle (S_1, S_2), (T_1, T_2)\rangle$ does not incur any current-level crossings, the edges originating from these four subtrees are edges of the types shown in Figure 3.18a (or the symmetric case with no edges between S_2 and T_1). Let $n_{11}, n_{21}, n_{22}, l_1, l_2, r_1, r_2$ be the numbers of edges as in Figure 3.18. Since we consider complete binary trees we obtain the following equalities: $l_1 = r_1 + n_{21}$, $r_2 = l_2 + n_{21}$, and $r_1 + n_{11} = l_2 + n_{22}$.

Take any fixed ordering of the leaves of the subtrees S_1, S_2, T_1, T_2. We first compare the number of crossings of the subinstance $\langle (S_1, S_2), (T_1, T_2)\rangle$ with the number of crossings of the subinstance $\langle (S_2, S_1), (T_2, T_1)\rangle$ in Figure 3.18b. The subinstance $\langle (S_1, S_2), (T_1, T_2)\rangle$ can have at most $n_{21}(n_{11} + n_{22})$ crossings that do not occur in $\langle (S_2, S_1), (T_2, T_1)\rangle$. However, $\langle (S_2, S_1), (T_2, T_1)\rangle$ has at least $l_1(l_2 + n_{21} + n_{22}) + l_2 n_{11} + r_2(r_1 + n_{21} + n_{11}) + r_1 n_{22}$ crossings that do not appear in $\langle (S_1, S_2), (T_1, T_2)\rangle$. Inserting the above equalities for l_1 and r_2 we get $(r_1 + n_{21})(l_2 + n_{21} + n_{22}) + l_2 n_{11} + (l_2 + n_{21})(r_1 + n_{21} + n_{11}) + r_1 n_{22} \geq n_{21}(n_{11} + n_{22})$. Thus, the same ordering of leaves does not give more crossings for $\langle (S_1, S_2), (T_1, T_2)\rangle$ than it does for $\langle (S_2, S_1), (T_2, T_1)\rangle$.

Next, we compare the number of crossings of the subinstance $\langle (S_1, S_2), (T_1, T_2)\rangle$ with the number of crossings of the subinstance $\langle (S_1, S_2), (T_2, T_1)\rangle$ in Figure 3.18c. Now the number of additional crossings of $\langle (S_1, S_2), (T_1, T_2)\rangle$ is at most $n_{21} n_{22}$, and the subinstance $\langle (S_1, S_2), (T_2, T_1)\rangle$ has at least $(r_1 + n_{11})(r_2 + n_{22}) + r_2 n_{21}$ crossings more. With the equality $r_1 + n_{11} = l_2 + n_{22}$ and the inequality $r_2 + n_{22} \geq n_{21}$ we get $(r_1 + n_{11})(r_2 + n_{22}) + r_2 n_{21} \geq n_{22} n_{21}$. Thus, again $\langle (S_1, S_2), (T_1, T_2)\rangle$ does not have more crossings than $\langle (S_1, S_2), (T_2, T_1)\rangle$ for the same leaf ordering. By symmetric reasoning the same holds for $\langle (S_2, S_1), (T_1, T_2)\rangle$. □

At each leaf of our bounded-height search tree, we obtain a certain layout of $\langle S, T\rangle$ and the accumulated number of current-level crossings is at most k. This, however, does not mean that the total number of crossings is at most k since we



(a) $\langle (S_1, S_2), (T_1, T_2) \rangle$ (b) $\langle (S_2, S_1), (T_2, T_1) \rangle$ (c) $\langle (S_1, S_2), (T_2, T_1) \rangle$

Figure 3.18: Edge types and crossings of the instance $\langle S, T \rangle$.

did not keep track of the indeterminate crossings. Therefore, at each leaf we need to examine how many crossings the corresponding layout has. This can be done in polynomial time. If one of the leaves yields at most k crossings, the algorithm outputs "Yes" and the layout; otherwise it outputs "No." We summarize:

Theorem 3.4.6 *The algorithm sketched above solves the parametrized version of the TL problem in $O^\star(4^k)$ time.*

3.4.5 Experiments

We have implemented the 2-approximation algorithm of Section 3.4.3 and have applied it to randomly generated complete and general binary tanglegrams, as well as to some real-world instances. The algorithms were written in Java and executed in a SuSE Linux 10.1 environment running on an AMD Opteron 248 2.2GHz system with 4GB RAM. In order to compare the quality of our results to optimal solutions, we solved the following simple integer quadratic program (IQP) with the mathematical programming software CPLEX 9.1 on the same Linux system to produce TLs.

We introduce a binary variable x_u for each inner node of $S \cup T$. If $x_u = 1$, the two subtrees of u change their order with respect to the input drawing. Let ab and cd be two inter-tree edges with $a, c \in S$ and $b, d \in T$. Let $v \in S$ and $w \in T$ be the lowest common ancestors of the leaves a and c, and of b and d, respectively. Assume that ab and cd cross each other in the original drawing. Then ab and cd cross each other in the solution encoded by the IQP if and only if $x_u \cdot x_v = 1$ or $(1 - x_u) \cdot (1 - x_v) = 1$. Two similar conditions hold in the case that ab and cd do not cross originally. Thus the total number of edge crossings can be expressed as the sum of these products for all pairs of edges. The IQP minimizes this sum.

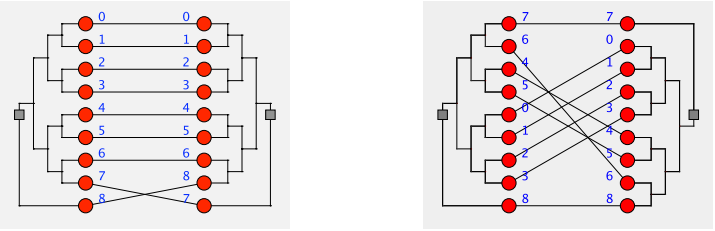
Our 2-approximation algorithm can not only be applied to complete binary tanglegrams but also, albeit only as a heuristic, to arbitrary binary tanglegrams. For arbitrary trees, the way of dividing an instance into two subinstances can lead to unnecessarily bad estimates of the number of crossings. Figure 3.19 shows an example that "fools" our algorithm. It aligns the single leaf attached to the root of each tree with the larger complete subtree on the other side since this causes no current-level crossings. All 14 crossings in Figure 3.19b are indeterminate crossings that

the algorithm does not take into account. A small modification of our algorithm weakens this effect (and yields the optimum solution in the given example). Instead of always dividing an instance $\langle (S_1, S_2), (T_1, T_2) \rangle$ into the subinstances $\langle S_1, T_1 \rangle$ and $\langle S_2, T_2 \rangle$ consisting of the two upper and the two lower subtrees we can also consider the *diagonal* subinstances $\langle S_1, T_2 \rangle$ and $\langle S_2, T_1 \rangle$. We divide the instance diagonally if $n_{11}^2 + n_{22}^2 \leq n_{12}^2 + n_{21}^2$ (using the notation introduced in Section 3.4.3). The approximation factor still holds in the case of complete trees.

In the following we denote our original algorithm by Algorithm 1 and the modified algorithm by Algorithm 2. We use n to denote the size of an instance, that is, the number of inter-tree edges.

We generated four sets (A–D) of random tanglegrams. Set A contains ten pairs of complete binary trees with random leaf orders for each $n = 16, 32, \ldots, 512$. In set B we simulated mutations by starting with two identical complete binary trees and then randomly swapping the positions of up to 20% of the leaves of one tree. Set C contains ten pairs of trees for each $n = 20, 30, \ldots, 80$. The trees are constructed from a set of nodes, initially containing the n leaves, by iteratively joining two random nodes by a new parent node that replaces its children in the set. This process generates trees that resemble phylogenetic trees or clustering dendrograms. Set D is similar to set C but again in each tanglegram the second tree is a mutation of the first tree, where up to 10% of the leaves can swap positions and up to 25% of the subtrees can reattach to another edge. A new edge is selected in a random walk starting at the subtree's old position. Such trees are of interest since real-world tanglegrams often consist of two rather similar trees.

To each tanglegram we applied Algorithm 1, Algorithm 2, and the IQP, and recorded the numbers c_1, c_2, and c_{opt} of crossings in respective solutions. We then computed for each tanglegram the performance ratios c_1/c_{opt} and c_2/c_{opt}. The results are shown in Figure 3.20. For complete binary trees (sets A and B) Algorithms 1 and 2 achieve similar ratios that tend to 1 as the size of the trees grows. On average both algorithms perform slightly better on mutated trees (B) than on random trees (A). Note that the absolute number of crossings is lower for mutated trees; thus a difference of only 1 or 2 to the optimum can already lead to relatively



(a) optimal TL: 1 crossing

(b) heuristic TL: 14 crossings

Figure 3.19: Example of a binary tree for which the heuristic performs badly.

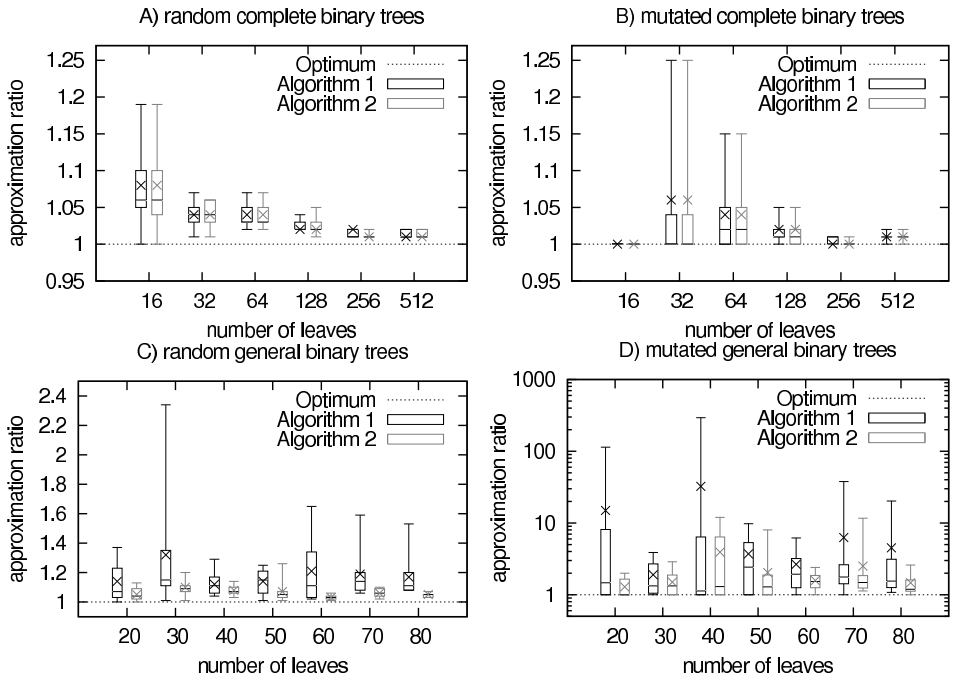


Figure 3.20: Results of comparing two variants of our 2-approximation. An IQP yielded the optimum. The plots show medians, first and third quartiles, minimum and maximum values. Arithmetic means are indicated by crosses.

large ratios for small n.

For general binary trees the performance ratios are no longer upper-bounded by 2 but at least for random trees (C) the ratios were well below 2 for $n \geq 40$. As expected, Algorithm 2 outperforms Algorithm 1. Algorithm 1 does attain performance ratios close to 1 for most random instances but it has some outliers as well. The solutions of Algorithm 2 are not only closer to the optimum, they also spread much less. For the last set of the mutated trees with relatively fewer crossings in the optimal solution, the results spread a lot more for both algorithms, but still Algorithm 2 is generally better with the third quartile of the ratios still below 2.

While the quality of the solutions of Algorithm 2 is better, Algorithm 1 is faster. This is due to the fact that the depth of the trees influences the running time exponentially; for complete trees the depth is $\log n$ while arbitrary trees can have depth $\Theta(n)$. Going back to the example in Figure 3.19, the recursion in Algorithm 2 goes down the complete heiWheneverght of the tree whereas the original algorithm only recurses one level. This behavior is also the reason why we could compute solutions for complete trees with up to 512 leaves (and depth 9) within the same time as arbitrary trees with 80 leaves. Running-time analysis is not the focus of this section and our implementations are not optimized for speed. But note that

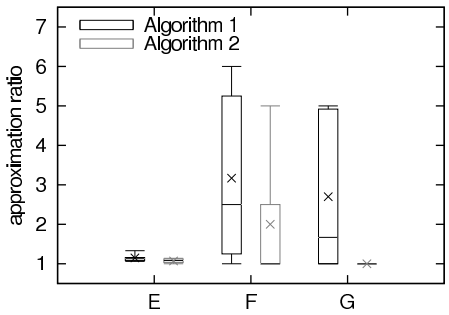


Figure 3.21: Results of real-world examples.

especially for non-complete trees, Algorithm 2 was by a factor of up to 13 slower than Algorithm 1 for random tanglegrams (85.2 vs. 6.6 sec for $n = 80$) and even up to 57 for mutated instances (337 vs. 6 sec for $n = 80$). At the same time it is interesting that the IQP could solve all mutated instances optimally within a second, and all random instances for $n \leq 60$ within 4 minutes. For complete trees the same observation holds, but for $n \geq 128$ the IQP could no longer solve the instances optimally within a time limit of 4 minutes. Unlike our algorithms, the IQP is by construction not sensitive to the completeness of the input tanglegram.

Our real-world examples comprise three sets (E–F) of tanglegrams. Set E contains 6 pairs of dendrograms of a hierarchically clustered social network based on email communication of 21 subjects. Sets F and G contain 6 and 10 pairs of phylogenetic trees for 15 species of pocket gophers and 17 species of lice [HSV+94]. Note that Figure 3.6 in the introduction shows a tanglegram in Set F. While the email tanglegrams have between 23 and 45 crossings in an optimal solution, the phylogenetic trees can be drawn with at most two crossings, most of them even without any crossings. The results are plotted in Figure 3.21, where in the performance ratio we added one to both crossing numbers to avoid divisions by zero. On the clustering data (E) both algorithms perform well, Algorithm 2 slightly better. In set F, all but one tanglegram can be drawn crossing-free. Algorithm 1 finds the optimum in two, Algorithm 2 in four of the six cases. In set G, six of the ten TLs can be drawn without crossings. Algorithm 1 solves five TLs optimally, Algorithm 2 all ten. Each of the real-world instances was solved within 0.5–2 sec.

To conclude, our experiments showed that on the one hand our 2-approximation performs very well on complete binary TLs. For general binary TLs the performance ratios are close to 1 and below 2 in most cases for the modified version of our algorithm. However, running times for non-complete trees are growing exponentially with the depth of the trees which makes the algorithms less applicable for very unbalanced trees. Interestingly, the IQP can quickly find an optimal solution for such trees with up to 80 leaves.

Acknowledgments. We thank Robert Görke for supplying us with clustering data, and Markus Völker for implementing both versions of our 2-approximation.

Chapter 4

Nash equilibria

4.1 Introduction

Modern game theory divides games into two categories: *transferable utility games* (also called cooperative games), and *noncooperative games*. In this chapter we study only the latter group.

A game is defined by a set of players, sets of strategies available for players, and the payoff functions of the players. A payoff function is a map from the Cartesian product of the strategy spaces of players to real numbers. It represents the information of how much (money, points, etc.) a particular player wins when all the players choose to play particular strategies. In noncooperative games it is assumed that players are interested only in maximizing their own payoff, i.e., they cannot profit from other player's outcomes.

We will only discuss the single round games (also called one shot games). Players in these games neither have the information about the history of players' behavior, nor can they plan to take recourse actions in the later rounds. Each of the players must choose one of his/her strategies not knowing the choices of the others. We assume that the set of available strategies is finite for each player. We also assume that all the players know the game, in particular, they know the payoff functions of all the players.

If we restrict our attention to games with only two players, we may simplify the notation significantly. To represent payoff functions it is now sufficient to give two matrices R and C. (Noncooperative two-player games are often called *bimatrix games*.) Rows of both matrices are indexed by strategies of one player, called *row player*, columns are indexed by strategies of the second player, called *column player*. Entry r_{ij} of matrix R (c_{ij} of C) represents the payoff to the row (column) player in a scenario when the row player plays strategy i and the column player plays strategy j. For an example of a simple two-player game see Figure 4.1.

In his famous work [Nas51] John Nash introduced a solution concept of Nash equilibrium for noncooperative games. A Nash equilibrium is a choice of strategies, one for each player, such that no player has an incentive to deviate (unilaterally).

$$R = \begin{pmatrix} 0 & 5 \\ -1 & 4 \end{pmatrix} \quad C = \begin{pmatrix} 0 & -1 \\ 5 & 4 \end{pmatrix}$$

Figure 4.1: A *prisoners dilemma* game. Each of the players has the choice either to confess to the police about a together committed crime (a strategy called *defection*, as the other prisoner is defeated) or to keep silent (a strategy called *cooperation*). A phenomenon captured by this game is that a single player gets a marginal gain from defeating the opponent, but he looses a lot when being defeated.

Nash proved the existence of an equilibrium for any finite noncooperative game. Nash equilibrium has been the dominant and most well studied solution concept in the noncooperative game theory.

It has been a long standing open problem to find algorithms that efficiently compute Nash equilibria. In a recent series of papers [GP06; DGP06; CD06], it was established that the problem of computing a Nash equilibrium is complete for the class PPAD[1] even for two-player games. This result reduces computation of fixed points of continues functions to the problem of finding Nash equilibria. As a consequence, it is unlikely that efficient algorithms for Nash equilibria exist. Since then the focus has been on algorithms for approximate equilibria.

By contrast to general games, Nash equilibria are easy to find in *zero-sum* two-player games. A zero sum game is a game with the property that the total payoff to the players is always zero. In two players case, it means that what one of the players wins is exactly what the other looses. It is not difficult to observe, that finding Nash equilibria in zero-sum two-player games is as hard as solving linear programs. We will use this observation in our construction of approximate equilibria for general (non-zero-sum) games.

In this work we address the notion of *additive approximation* and consider the problem of computing approximate Nash equilibria in bimatrix games. Under the usual assumption that the payoff matrices are normalized to be in $[0,1]^{n\times n}$ (where n is the number of available pure strategies), we say that a pair of mixed strategies is an ϵ-Nash equilibrium if no player can gain more than ϵ by unilaterally deviating to another strategy. In [CDT06] it was proved that it is PPAD-complete to find an ϵ-Nash equilibrium when ϵ is of the order $\frac{1}{poly(n)}$. For constant ϵ however, the problem is still open. In [LMM03], it was shown that for any constant $\epsilon > 0$, an ϵ-Nash equilibrium can be computed in subexponential time ($n^{O(\log n/\epsilon^2)}$). As for polynomial time algorithms, it is fairly simple to obtain a 3/4-approximation (see [KPS06] for a slightly better result) and even better a 1/2-approximation [DMP06]. An im-

[1] PPAD is a complexity class introduced by Christos Papadimitriou in 1994 [Pap94]. The name stands for "Polynomial Parity Arguments on Directed graphs". It is a class of search problems. By a specyfic parity argument, we know that each instance has a solution, but exhaustive search for such a solution would require time that is exponential in the instance size. Note that, in contrast to the NP problems, in PPAD problems it is not the existance of a solution that is difficult to determine.

proved approximation with $\epsilon = \frac{3-\sqrt{5}}{2} + \zeta \approx 0.38197 + \zeta$ for any $\zeta > 0$ was obtained by Daskalakis, Mehta and Papadimitriou in [DMP07].

Our results. We provide two new algorithms for approximate Nash equilibria. The first one achieves exactly the same factor as [DMP07] but with a simpler and faster technique. The second one, which is an extension of the first and has a more involved analysis, achieves an improved approximation of 0.36392. Regarding the running time, both algorithms are based on solving a single linear program in contrast to [DMP07], which may require to solve up to $n^{O(\frac{1}{\zeta^2})}$ linear programs for a $(0.38197 + \zeta)$-approximation.

Our technique is based on the fact that we can compute exact Nash equilibria for zero-sum games in polynomial time via linear programming. In particular, the main idea in both of our algorithms is as follows: we first find an equilibrium (say x^*, y^*) in the zero-sum game $R - C$, where R and C are the payoff matrices of the two players. If x^*, y^* is not a good solution for the original game, then the players take turns and switch to some appropriately chosen strategies. The probabilities of switching are chosen such that the final incentives to deviate become the same for both players. As a result, these probabilities are functions of the parameters of the problem. The final part of the analysis is to choose these functions so as to minimize the approximation error. The intuition behind using the auxiliary zero-sum game $R - C$ is that a unilateral switch from x^*, y^* that improves the payoff of one player also improves the payoff of the other player, since x^*, y^* is chosen to be an equilibrium with respect to $R - C$. This allows us to estimate upper bounds on the final incentive of both players to deviate, which we can later optimize. We explain this further in the proof of Theorem 4.3.1. We should note here that the use of zero-sum games has also been considered in [KS96] for obtaining *well-supported* approximate equilibria, which is a stronger notion of approximation.

Recently, in a work completed in parallel with ours, Spirakis and Tsaknakis [ST07] have obtained another algorithm achieving an improved approximation of 0.3393. This is the currently best known approximation factor for this problem. Their algorithm is based on a different methodology from ours and requires solving a polynomial number of linear programs.

Finally in Section 4.6, we show a simple reduction that allows us to compute approximate equilibria for games with more than two players by using algorithms for two-player games. We obtain a 0.60205-approximation for three-player games and 0.71533-approximation for four-player games. To the best of our knowledge these are the first nontrivial polynomial time approximation algorithms for multi-player games.

4.2 Notation and Definitions

Consider a two person game G, where for simplicity the number of available (pure) strategies for each player is n. Our results still hold when the players do not

have the same number of available strategies. We will refer to the two players as the row and the column player and we will denote their $n \times n$ payoff matrices by R, C respectively. Hence, if the row player chooses strategy i and the column player chooses strategy j, the payoffs are R_{ij} and C_{ij} respectively.

A *mixed strategy* for a player is a probability distribution over the set of his pure strategies and will be represented by a vector $x = (x_1, x_2, ..., x_n)^T$, where $x_i \geq 0$ and $\sum x_i = 1$. Here x_i is the probability that the player will choose his ith pure strategy. The *support* of x is the set of pure strategies that are used with positive probability. The ith pure strategy will be represented by the unit vector e_i, that has 1 in the ith coordinate and 0 elsewhere. For a mixed strategy pair x, y, the payoff to the row player is the expected value of a random variable which is equal to R_{ij} with probability $x_i y_j$. Therefore the payoff to the row player is $x^T R y$. Similarly the payoff to the column player is $x^T C y$.

A Nash equilibrium [Nas51] is a pair of strategies x^*, y^* such that no player has an incentive to deviate unilaterally. Since mixed strategies are convex combinations of pure strategies, in the definition of the Nash equilibrium, it suffices to consider only deviations to pure strategies:

Definition 4.2.1 *A pair of strategies x^*, y^* is a Nash equilibrium if the following two conditions hold:*

(i) For every pure strategy e_i of the row player, $e_i^T R y^* \leq (x^*)^T R y^*$.

(ii) For every pure strategy e_i of the column player, $(x^*)^T C e_i \leq (x^*)^T C y^*$.

Assuming that we normalize the entries of the payoff matrices so that they all lie in $[0, 1]$, we can define the notion of an additive ϵ-approximate Nash equilibrium (or simply ϵ-Nash equilibrium) as follows:

Definition 4.2.2 *For any $\epsilon > 0$, a pair of strategies x^*, y^* is an ϵ-Nash equilibrium if the following two conditions hold:*

(i) For every pure strategy e_i of the row player, $e_i^T R y^* \leq (x^*)^T R y^* + \epsilon$.

(ii) For every pure strategy e_i of the column player, $(x^*)^T C e_i \leq (x^*)^T C y^* + \epsilon$.

In other words, no player will gain more than ϵ by unilaterally deviating to another strategy. Other approximation concepts have also been studied. In particular, [DGP06] introduced the stronger notion of ϵ-well-supported equilibria, in which every pure strategy in the support of the chosen strategies should be an approximate best response to the strategies chosen for the other players. Another stronger notion of approximation is that of being geometrically close to an exact Nash equilibrium and was studied in [EY07]. We do not consider these concepts here. For more on these concepts, we refer the reader to [KS96] and [EY07].

4.3 A $(\frac{3-\sqrt{5}}{2})$-approximation

In this section, we provide an algorithm that achieves exactly the same factor as in [DMP07], which is $(3-\sqrt{5})/2$, but by using a different and simpler method. In the next section we show how to modify our algorithm in order to improve the approximation.

Given a game $G = (R, C)$, where the entries of R and C are in $[0, 1]$, let $A = R - C$. Our algorithm is based on solving the zero-sum game $(A, -A)$ and then modifying the solution appropriately, if it does not provide a good approximation. It is well known that zero-sum games can be solved efficiently using linear programming. The decision on when to modify the zero-sum solution depends on a parameter of the algorithm $\alpha \in [0, 1]$. We first describe the algorithm parametrically and then show how to obtain the desired approximation.

Algorithm 1
Let $\alpha \in [0, 1]$ be a parameter of the algorithm.

1. Compute an equilibrium (x^*, y^*) for the zero-sum game defined by the matrix $A = R - C$.

2. Let g_1, g_2 be the incentive to deviate for the row and column player respectively if they play (x^*, y^*) in the original game (R, C), i.e.,
   $g_1 = \max_{i=1,\ldots,n} e_i^T R y^* - (x^*)^T R y^*$ and $g_2 = \max_{i=1,\ldots,n} (x^*)^T C e_i - (x^*)^T C y^*$.
   Without loss of generality, assume, that $g_1 \geq g_2$ (the statement of the algorithm would be completely symmetrical if $g_1 < g_2$).

3. Let $r_1 \in \operatorname{argmax}_{e_i} e_i^T R y^*$ be an optimal response of the row player to the strategy y^*. Let $b_2 \in \operatorname{argmax}_{e_i} r_1^T C e_i$ be an optimal response of the column player to the strategy r_1.

4. Output the following pair of strategies, $(\hat{x}, \hat{y})$, depending on the value of g_1 with respect to the value of α:

$$(\hat{x}, \hat{y}) = \begin{cases} (x^*, y^*), & \text{if } g_1 \leq \alpha \\ (r_1, (1-\delta_2) \cdot y^* + \delta_2 \cdot b_2), & \text{otherwise} \end{cases}$$

   where $\delta_2 = \frac{1-g_1}{2-g_1}$.

Theorem 4.3.1 *Algorithm 1 outputs a* $\max\{\alpha, \frac{1-\alpha}{2-\alpha}\}$*-approximate Nash equilibrium.*

Proof: If $g_1 \leq \alpha$ (recall that we assumed $g_1 \geq g_2$), then clearly (x^*, y^*) is an α-approximate Nash equilibrium.

Suppose $g_1 > \alpha$. We will estimate the satisfaction of each player separately. Suppose b_1 is an optimal response for the row player to $\hat{y}$, i.e., $b_1 \in \operatorname{argmax}_{e_i} e_i^T R \hat{y}$. The row player plays r_1, which is a best response to y^*. Hence b_1 can be better than r_1 only when the column player plays b_2, which happens with probability δ_2.

Formally, the amount that the row player can earn by switching is at most:

$$\begin{aligned} b_1^T R\hat{y} - r_1^T R\hat{y} &= (1-\delta_2)(b_1^T R y^* - r_1^T R y^*) + \delta_2(b_1^T R b_2 - r_1^T R b_2) \\ &\leq \delta_2 \cdot b_1^T R b_2 \leq \delta_2 = \tfrac{1-g_1}{2-g_1} \end{aligned}$$

The first inequality above comes from the fact that r_1 is a best response to y^* and the second comes from our assumption that the entries of R and C are in $[0,1]$.

Consider the column player. The critical observation, which is also the reason we started with the zero-sum game $(R-C, C-R)$, is that the column player also benefits (when he plays y^*) from the switch of the row player from x^* to r_1. In particular, since (x^*, y^*) is an equilibrium for the zero-sum game $(R-C, C-R)$, the following inequalities hold:

$$(x^*)^T R e_j - (x^*)^T C e_j \geq (x^*)^T R y^* - (x^*)^T C y^* \geq e_i^T R y^* - e_i^T C y^*, \ \forall\, i,j = 1,...,n \tag{4.1}$$

If $e_i = r_1$, we get from (4.1) that $r_1^T C y^* \geq r_1^T R y^* - (x^*)^T R y^* + (x^*)^T C y^*$. But we know that $r_1^T R y^* - (x^*)^T R y^* = g_1$, which implies:

$$r_1^T C y^* \geq g_1 + (x^*)^T C y^* \geq g_1 \tag{4.2}$$

Inequality (4.2) shows that any deviation of the row player from x^*, y^*, that improves his payoff, guarantees at least the same gain to the column player as well. We can now use the lower bound of (4.2) to estimate the incentive of the column player to change his strategy. He plays $\hat{y}$ while he would prefer to play an optimal response to $\hat{x}$ which is b_2. Since b_2 is played with probability δ_2, by switching he could earn:

$$\begin{aligned} \hat{x}^T C b_2 - \hat{x}^T C \hat{y} &= r_1^T C b_2 - r_1^T C \hat{y} \\ &= r_1^T C b_2 - ((1-\delta_2) r_1^T C y^* - \delta_2 \cdot r_1^T C b_2) \\ &= (1-\delta_2)(r_1^T C b_2 - r_1^T C y^*) \\ &\leq (1-\delta_2)(1-g_1) = \delta_2 = \tfrac{1-g_1}{2-g_1} \end{aligned}$$

The last inequality above follows from (4.2). The probability δ_2 was chosen so as to equalize the incentives of the two players to deviate in the case that $g_1 > \alpha$. It is now easy to check that the function $(1-g_1)/(2-g_1)$ is decreasing, hence the incentive for both players to deviate is at most $(1-\alpha)/(2-\alpha)$. Combined with the case when $g_1 \leq \alpha$, we get a $\max\{\alpha, \frac{1-\alpha}{2-\alpha}\}$-approximate equilibrium.

□

In order to optimize the approximation factor of Algorithm 1, we only need to equate the two terms, α and $\frac{1-\alpha}{2-\alpha}$, which then gives:

$$\alpha^2 - 3\alpha + 1 = 0 \tag{4.3}$$

The solution to (4.3) in the interval $[0,1]$ is $\alpha = \frac{3-\sqrt{5}}{2} \approx 0.38197$. Note that the approximation ratio of our algorithm may be expressed as $\alpha = 1 - 1/\phi$, where ϕ is

the *golden ratio*[2]. Since α is an irrational number, we need to ensure that we can still do the comparison $g_1 \leq \alpha$ to be able to run Algorithm 1 (note that this is the only point where the algorithm uses the value of α). But to test $g_1 \leq 3 - \sqrt{5}/2$, it suffices to test if $(3 - 2g_1)^2 \geq 5$ and clearly g_1 is a polynomially sized rational number. Concerning complexity, zero-sum games can be solved in polynomial time by linear programming. All the other steps of the algorithm require only polynomial time. Therefore, Theorem 4.3.1 implies:

Corollary 4.3.2 *We can compute, in polynomial time, a* $\frac{3-\sqrt{5}}{2}$*-approximate Nash equilibrium for bimatrix games.*

4.4 An Improved Approximation

In this section we obtain a better approximation of $1/2 - 1/(3\sqrt{6}) \approx 0.36392$ by essentially proposing a different solution in the cases where Algorithm 1 approaches its worst case guarantee. We first give some motivation for the new algorithm. From the analysis of Algorithm 1, one can easily check that as long as g_1 belongs to $[0, 1/3] \cup [1/2, 1]$, we can have a 1/3-approximation if we run the algorithm with any $\alpha \in [1/3, 1/2)$. Therefore, the bottleneck for getting a better guarantee is when the maximum incentive to deviate is in $[1/3, 1/2]$. In this case, we will change the algorithm so that the row player will play a mix of r_1 and x^*. Note that in Algorithm 1, the probability of playing r_1 is either 0 or 1 depending on the value of g_1. This probability will now be a more complicated function of g_1, derived from a certain optimization problem. As for the column player, we again compute b_2 which is now the best response to the *mixture* of r_1 and x^*- not only to r_1. Then we compute an appropriate mixture of b_2 and y^*. Again, the probability of playing b_2 is chosen so as to equate the incentives of the two players to defect. Finally we should note that the modification of the algorithm will not be on the interval $[1/3, 1/2]$ but instead on a subinterval of the form $[1/3, \beta]$, where β is derived from the optimization that we perform in our analysis.

Algorithm 2

1. Compute an equilibrium (x^*, y^*) for the zero-sum game defined by the matrix $A = R - C$.

2. As in Algorithm 1, let g_1, g_2 be the incentive to deviate for the row and column player respectively if they play (x^*, y^*) in the original game, i.e., $g_1 = \max_{i=1,\ldots,n} e_i^T R y^* - (x^*)^T R y^*$ and $g_2 = \max_{i=1,\ldots,n} (x^*)^T C e_i - (x^*)^T C y^*$. Without loss of generality, assume, that $g_1 \geq g_2$.

3. Let $r_1 \in \operatorname{argmax}_{e_i} e_i^T R y^*$ be an optimal response of the row player to the strategy y^*.

[2]The golden ratio is a solution (in positive variables) of $\phi = \frac{a+b}{a} = \frac{a}{b}$. The numerical value of the ratio is $\phi = \frac{1+\sqrt{5}}{2} \approx 1.6180339887$.

4. The row player will play a mixture of r_1 and x^*, where the probability of playing r_1 is given by:

$$\delta_1 = \delta_1(g_1) = \begin{cases} 0, & \text{if } g_1 \in [0, 1/3] \\ \Delta_1(g_1), & \text{if } g_1 \in (1/3, \beta] \\ 1, & \text{otherwise} \end{cases}$$

where $\Delta_1(g_1) = (1 - g_1)\left(-1 + \sqrt{1 + \frac{1}{1-2g_1} - \frac{1}{g_1}}\right)$.

5. Let b_2 be an optimal response of the column player to $((1-\delta_1)x^* + \delta_1 r_1)$, i.e., $b_2 \in \text{argmax}_{e_i}\ ((1-\delta_1)x^* + \delta_1 r_1)^T C e_i$. Let also $h_2 = (x^*)^T C b_2 - (x^*)^T C y^*$, i.e., the gain from switching to b_2 if the row player plays x^*.

6. The column player will play a mixture of b_2 and y^*, where the probability of playing b_2 is given by:

$$\delta_2 = \delta_2(\delta_1, g_1, h_2) = \begin{cases} 0, & \text{if } g_1 \in [0, 1/3] \\ \max\{0, \Delta_2(\delta_1, g_1, h_2)\}, & \text{if } g_1 \in (1/3, \beta] \\ \frac{1-g_1}{2-g_1}, & \text{otherwise} \end{cases}$$

where $\Delta_2(\delta_1, g_1, h_2) = \frac{\delta_1 - g_1 + (1-\delta_1)h_2}{1+\delta_1-g_1}$.

7. Output $(\hat{x}, \hat{y}) = ((1-\delta_1)x^* + \delta_1 r_1, (1-\delta_2)y^* + \delta_2 b_2)$.

In our analysis, we will take β to be the solution to $\Delta_1(g_1) = 1$ in $[1/3, 1/2]$, which coincides with the root of the polynomial $x^3 - x^2 - 2x + 1$ in that interval and it is:

$$\beta = \frac{1}{3} + \frac{\sqrt{7}}{3}\cos\left(\frac{1}{3}\tan^{-1}\left(3\sqrt{3}\right)\right) - \frac{\sqrt{21}}{3}\sin\left(\frac{1}{3}\tan^{-1}\left(3\sqrt{3}\right)\right) \tag{4.4}$$

Calculations show $0.445041 \leq \beta \leq 0.445042$. The emergence of β in our analysis is explained in Lemma 4.4.2.

Remark 4.4.1 *The actual probabilities δ_1 and δ_2 can be irrational numbers (and so is β). However, for any constant $\epsilon > 0$, we can take approximations of high enough accuracy of all the square roots that are involved in the calculations so that the final loss in the approximation ratio will be at most ϵ. From now on, for ease of exposition, we will carry out the analysis of Algorithm 2, as if we can compute all the expressions involved exactly.*

Note that for $g_1 \in [\frac{1}{3}, \frac{1}{2}]$ and $\delta_1 \in [0, 1]$ the denominators that appear in the functions Δ_1, Δ_2 do not vanish. The following lemma ensures that $\hat{x}$ is a valid strategy. It will be proved in Section 4.5. That $\hat{y}$ is also a valid strategy is proved in Lemma 4.4.3.

Lemma 4.4.2 *For $g_1 \in (1/3, \beta]$ we have $\Delta_1(g_1) \in [0, 1]$.*

Now we bound the incentives of players to deviate. Let F be the following function:

$$F(\delta_1, g_1, h_2) := \frac{(\delta_1 (1 - g_1 - h_2) + h_2)(1 - (1 - \delta_1)h_2)}{1 + \delta_1 - g_1} \tag{4.5}$$

Lemma 4.4.3 *The pair of strategies $(\hat{x}, \hat{y})$ is a λ-Nash equilibrium for game (R, C) with*

$$\lambda \leq \begin{cases} g_1 & \text{if } g_1 \leq 1/3 \\ \max_{h_2 \in [0,g1]} \begin{cases} F(\delta_1, g_1, h_2) & \text{if } \Delta_2(\delta_1, g_1, h_2) \geq 0 \\ (1 - \delta_1)g_1 & \text{if } \Delta_2(\delta_1, g_1, h_2) < 0 \end{cases} & \text{if } g_1 \in (1/3, \beta] \\ \frac{1-g_1}{2-g_1} & \text{if } g_1 > \beta \end{cases} \tag{4.6}$$

Proof: In the case that $g_1 \in [0, 1/3] \cup [\beta, 1]$, the answer essentially follows from the proof of Theorem 4.3.1. The interesting case is when $g_1 \in [1/3, \beta]$.

Case 1: $g_1 \leq 1/3$
$(\hat{x}, \hat{y}) = (x^*, y^*)$ which is by definition a g_1-approximate Nash equilibrium.

Case 2a: $g_1 \in (1/3, \beta]$ and $\Delta_2(\delta_1, g_1, h_2) \geq 0$
Recall that Lemma 4.4.2 implies $\hat{x}$ is a valid strategy in Case 2. Observe, that $\delta_2(g_1, \delta_1, h_2) = \Delta_2(g_1, \delta_1, h_2) = \frac{\delta_1 - g_1 + (1-\delta_1)h_2}{1+\delta_1-g_1} \leq 1$ is a valid probability, and therefore $\hat{y}$ is a valid mixed strategy too.

We estimate the incentive for the row player to deviate from $\hat{x}$. If b_1 is an optimal response to $\hat{y}$, then the gain from switching is at most:

$$\begin{aligned} b_1^T R\hat{y} - \hat{x}^T R\hat{y} = \quad & (b_1 - \hat{x})^T R\hat{y} = \\ = \ \delta_2(b_1 - \hat{x})^T Rb_2 & +(1-\delta_2)(b_1 - \hat{x})^T Ry^* \\ \leq \ \delta_2(1 - \hat{x}^T Rb_2) & +(1-\delta_2)(b_1 - \hat{x})^T Ry^* \\ = \ \delta_2(1 - \delta_1 r_1^T Rb_2 - (1-\delta_1)(x^*)^T Rb_2) & +(1-\delta_2)(\quad \delta_1 (b_1 - r_1)^T Ry^* \\ & \qquad +(1-\delta_1)(b_1 - x^*)^T Ry^*) \end{aligned}$$

By (4.1) we have $(x^*)^T Rb_2 \geq (x^*)^T Cb_2 - (x^*)^T Cy^* + (x^*)^T Ry^* \geq h_2$. Also r_1 is a best response to y^*, hence $(b_1 - r_1)^T Ry^* \leq 0$ and $(b_1 - x^*)^T Ry^* \leq g_1$. Therefore, the gain from deviating is at most:

$$b_1^T R\hat{y} - \hat{x}^T R\hat{y} \leq \delta_2(1 - (1-\delta_1)h_2) + (1-\delta_2)(1-\delta_1)g_1 = \mathrm{EST}_1.$$

We now estimate the incentive of the column player to switch. The best response to $\hat{x}$ for the column player is b_2, which is played with probability δ_2. Thus the incentive to deviate from $\hat{y}$ is:

$$\begin{aligned} \hat{x}^T Cb_2 - \hat{x}^T C\hat{y} \ &= (1-\delta_2)(\hat{x}^T Cb_2 - \hat{x}^T Cy^*) \\ &= (1-\delta_2)((1-\delta_1)((x^*)^T Cb_2 - (x^*)^T Cy*) + \delta_1(r_1^T Cb_2 - r_1^T Cy^*)) \\ &\leq (1-\delta_2)((1-\delta_1)h_2 + \delta_1(1-g_1)) = \mathrm{EST}_2 \end{aligned}$$

The last inequality follows from the definitions of g_1 and h_2. It remains to observe that our choice of $\delta_2(\delta_1, g_1, h_2) = \frac{\delta_1 - g_1 + (1-\delta_1)h_2}{1+\delta_1-g_1}$ makes these estimates both equal to $F(\delta_1, g_1, h_2)$:

$$\mathrm{EST}_1 = \mathrm{EST}_2 = \frac{(\delta_1(1-g_1-h_2)+h_2)(1-(1-\delta_1)h_2)}{\delta_1+1-g_1} = F(\delta_1, g_1, h_2).$$

Case 2b: $g_1 \in (1/3, \beta]$ and $\Delta_2(\delta_1, g_1, h_2) < 0$
Then $\hat{y} = y^*$ and the best response of the row player is r_1. Hence he can improve his payoff by at most

$$r_1^T R y^* - \hat{x}^T R y^* = r_1^T R y^* - (\delta_1 \cdot r_1^T R y^* + (1-\delta_1)((x^*)^T R y^*)) = (1-\delta_1)g_1$$

while the column player can improve by at most

$$\hat{x}^T C b_2 - \hat{x}^T C y^* = \delta_1(r_1^T C b_2 - r_1^T C y^*) + (1-\delta_1)((x^*)^T C b_2 - (x^*)^T C y^*)$$

By (4.1) we can see that $r_1^T C y^* \geq g_1$. Hence

$$\hat{x}^T C b_2 - \hat{x}^T C y^* \leq \delta_1(1-g1) + (1-\delta_1)h_2$$

It is easy to check that $\Delta_2(g_1, \delta_1, h_2) < 0$ implies $\delta_1(1-g1) + (1-\delta_1)h_2 < (1-\delta_1)g_1$. Therefore the maximum incentive to deviate in this case is at most $(1-\delta_1)g_1$. Combining Case 2a and Case 2b, and taking the worst possible case over the range of h_2 (recall that $h_2 \leq g_2 \leq g_1$), we get precisely the expression in the statement of Lemma 4.4.3.

Case 3: $g_1 > \beta$
Notice that in this case, the players are playing the same strategies as in Algorithm 1, when $g_1 \geq \alpha$. By the analysis in the proof of Theorem 4.3.1, we see that the maximum incentive is $(1-g_1)/(2-g_1)$. □

We will now argue that our choice of $\Delta_1(g_1)$ is optimal for any $g_1 \in (\frac{1}{3}, \beta]$ and that the expression (4.6) from Lemma 4.4.3 achieves an improvement over Algorithm 1. For this, we need to find the worst possible approximation in Case 2 of Lemma 4.4.3. In particular, we need to look at the maxima of the following function:

$$P(g_1) := \min_{\delta_1 \in [0,1]} \max_{h_2 \in [0, g_1]} \begin{cases} F(\delta_1, g_1, h_2) & \text{if } \Delta_2(\delta_1, g_1, h_2) \geq 0 \\ (1-\delta_1)g_1 & \text{if } \Delta_2(\delta_1, g_1, h_2) < 0 \end{cases} \tag{4.7}$$

Lemma 4.4.4 *The pair $(\delta_1, h_2) = (\Delta_1(g_1), g_1)$ is an optimal solution for the expression $P(g_1)$. Furthermore, the maximum of $P(g_1)$ over g_1 is $\frac{1}{2} - \frac{1}{3\sqrt{6}}$, i.e., the following holds*

$$P(g_1) = F(\Delta_1(g_1),\, g_1, g_1) \quad \forall g_1 \in [\frac{1}{3}, \frac{1}{2}] \tag{4.8}$$

$$\max_{g_1 \in [\frac{1}{3}, \beta]} P(g_1) = \frac{1}{2} - \frac{1}{3\sqrt{6}} \leq 0.36392. \tag{4.9}$$

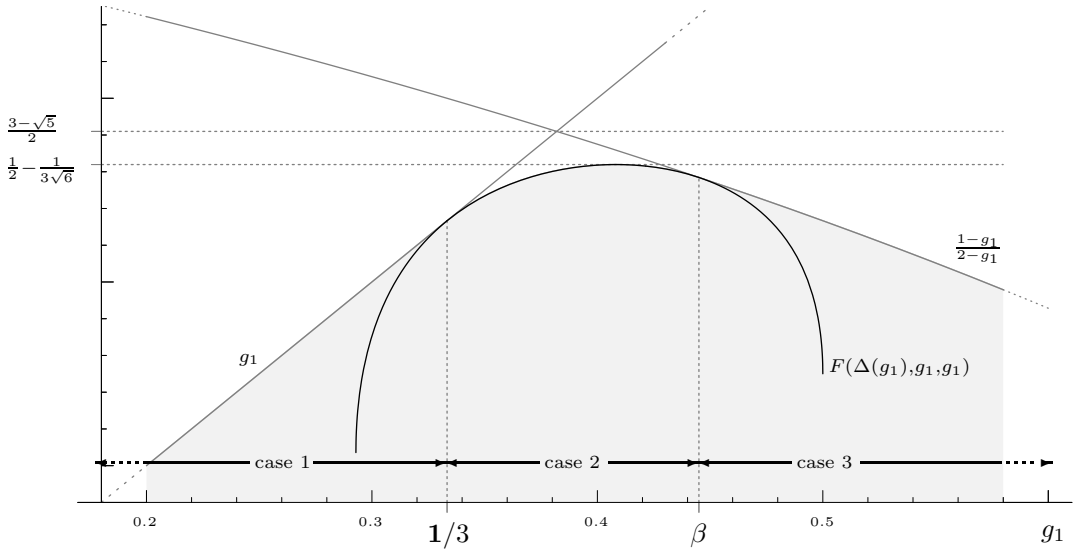


Figure 4.2: How the approximation factor depends on g_1.

The lemma will be proved in Section 4.5. Given Remark 4.4.1, we are now ready to conclude with the following:

Theorem 4.4.5 *For any $\epsilon > 0$, Algorithm 2 computes a $(0.36392+\epsilon)$-approximate Nash equilibrium.*

Proof: By Lemma 4.4.3 the output of Algorithm 2, $(\hat{x}, \hat{y})$ is a pair of mixed strategies for players, such that the incentive of players to deviate is bounded by (4.6). By Lemma 4.4.4 we have that for $g_1 \in (1/3, \beta]$ the expression (4.6) is bounded by $\frac{1}{2} - \frac{1}{3\sqrt{6}} \leq 0.36392$. It is easy to observe, that for other values of g_1 the expression (4.6) takes only smaller values. In particular, it is at most $1/3$ when $g_1 \in [0, 1/3]$ and at most $\frac{1-\beta}{2-\beta} \approx 0.3569$ when $g_1 > \beta$. The dependence of the approximation on the variable g_1 is presented in Figure 4.2. □

A Tight Example: The analysis that we have presented is tight. Tracing all inequalities used, we constructed the following worst-case example, on which Algorithm 2 yields a 0.36392-approximate equilibrium:

$$R = \begin{pmatrix} 0 & \alpha & \alpha \\ \alpha & 0 & 1 \\ \alpha & 1 & 0 \end{pmatrix} \quad C = \begin{pmatrix} 0 & \alpha & \alpha \\ \alpha & 1 & 1/2 \\ \alpha & 1/2 & 1 \end{pmatrix} \quad \text{where } \alpha = 1/\sqrt{6}.$$

4.5 Proof of Lemma 4.4.2 and Lemma 4.4.4

Proof of Lemma 4.4.2:

We show that Δ_1 maps $[1/3, \beta]$ into $[0, 1]$, where Δ_1 (see Algorithm 2) is defined as

$$\Delta_1(g_1) := (1 - g_1)\left(-1 + \sqrt{1 + \frac{1}{1 - 2g_1} - \frac{1}{g_1}}\right).$$

It is easy to check that $\Delta_1(1/3) = 0$. We will show that Δ_1 is real-valued and monotone increasing on the interval $[1/3, 1/2)$. Then we show that $1/3 < \beta < 1/2$, and $\Delta_1(\beta) = 1$.

To check that $\Delta_1(g_1)$ takes real values on $[1/3, 1/2)$, it is easy to verify that the radicand, i.e., the expression under the square root, is nonnegative in this domain.

$$\left(1 + \frac{1}{1 - 2g_1} - \frac{1}{g_1}\right) \;\geq\; 1 \quad \text{for all } \; g_1 \in [1/3, 1/2). \tag{4.10}$$

To check the monotonicity of $\Delta_1(g_1)$, we calculate $\Delta_1'(g_1)$ and find

$$\Delta_1'(g_1) = 1 + \frac{1 - 3g_1 - 2g_1^2 + 14g_1^3 - 8g_1^4}{2(1 - 2g_1)^2 g_1^2 \sqrt{1 + \frac{1}{1-2g_1} - \frac{1}{g_1}}} \;> 0 \quad \text{for all} \quad g_1 \in [1/3, 1/2). \tag{4.11}$$

The inequality in (4.11) is obtained as follows: Inequality (4.10) shows that the radicand in (4.11) is strictly positive on $[1/3, 1/2)$. So the denominator appearing in $\Delta_1'(g_1)$ is real and positive. For the numerator appearing in $\Delta_1'(g_1)$ the following estimation holds for all $g_1 \in [1/3, 1/2)$:

$$\begin{aligned} 1 - 3g_1 - 2g_1^2 + 14g_1^3 - 8g_1^4 &= \tfrac{1}{2}(3 + g_1 \; + (1 - g_1)(4g_1 + 1)(\, -2 + (1 - 2g_1)^2)\;) \\ &\geq \tfrac{1}{2}(3 + g_1 \; + (1 - g_1)(4g_1 + 1)(\, -2\;)\;) \\ &= \tfrac{1}{2}\left((1 - \tfrac{5}{2}g_1)^2 + \tfrac{7}{4}g_1^2\right) \;> 0. \end{aligned}$$

Here the first inequality holds since $g_1 \in [1/3, 1/2)$ implies $(1 - g_1)(4g_1 + 1) > 0$. This proves (4.11) showing that Δ_1 is strictly increasing on the interval $[1/3, 1/2)$.

Now we calculate $g \in [1/3, 1/2)$ for which $\Delta_1(g) = 1$ holds. In the following let $x \in [1/3, 1/2)$. This implies $0 < 2 - x$ and $0 < 1 - x$, which together with (4.10) gives rise to the second equivalence in the following:

$$\begin{aligned} \Delta_1(x) = 1 \;&\Leftrightarrow\; (2 - x) \;&&=\; (1 - x)\sqrt{1 + \tfrac{1}{1-2x} - \tfrac{1}{x}} \\ &\Leftrightarrow\; (2 - x)^2 \;&&=\; (1 - x)^2\left(1 + \tfrac{1}{1-2x} - \tfrac{1}{x}\right) \;\Leftrightarrow\; 1 - 2x - x^2 + x^3 = 0. \end{aligned}$$

The polynomial $p(x) := 1 - 2x - x^2 + x^3$ has *exactly one* zero in $[1/3, 1/2]$, since p is monotone decreasing on this domain: One calculates $p'(x) = -2x - 2(1 - 3x^2) \leq -2x < 0$ for all $x \in [1/3, 1/2]$. Moreover one has $p(1/3) = 7/27$ and $p(1/2) = -1/8$, showing that p has a root within the interval.

Substituting $x = \frac{1}{3}\left(1 + \sqrt{7}\cos(\alpha) - \sqrt{21}\sin(\alpha)\right)$ and $\alpha = \arctan(t)/3$ leads to

$$1 - 2x - x^2 + x^3 = \frac{7}{27}\left(1 - \sqrt{28}\cos(3\alpha)\right) = \frac{7}{27}\left(1 - \frac{\sqrt{1+27}}{\sqrt{1+t^2}}\right)$$

where the last term is zero for $t = 3\sqrt{3}$. Resubstitution shows that $p(\beta) = 0$ holds for

$$\beta = \frac{1}{3}\left(1 + \sqrt{7}\cos(\alpha) - \sqrt{21}\sin(\alpha)\right)$$

where $\alpha = \frac{1}{3}\arctan\left(3\sqrt{3}\right)$. Taylor expansion of the corresponding terms leads to $0.445041 < \beta < 0.445042$, proving $\beta \in [1/3, 1/2)$. This shows $\Delta_1(\beta) = 1$, which proves the lemma. □

In the proof of Lemma 4.4.4 we will make repeated use of the following simple observation:

Observation 1 *Let $a, b \in \mathbb{R}$, $a, b \geq 0$ then $a - b \geq 0 \Leftrightarrow a^2 - b^2 \geq 0$.*

We solved the univariate minimization problems that arise in Lemma 4.4.4 in the classic manner, eventually leading to the minimizer $\Delta_1(g)$. This procedure is lengthy, so here we give an uninspiring but shorter proof. The proof is based on the following Lemma:

Lemma 4.5.1 *For every pair $(g, \delta) \in [1/3, \beta] \times [0, 1]$ the following holds*

$$F(\delta, g, g) = \max_{h \in [0,g]} \begin{cases} F(\delta, g, h) & \text{if } \Delta_2(\delta, g, h) \geq 0 \\ (1-\delta)g & \text{if } \Delta_2(\delta, g, h) < 0 \end{cases} \tag{4.12}$$

$$F(\Delta_1(g), g, g) = \min_{d \in [0,1]} F(d, g, g). \tag{4.13}$$

We postpone the proof of Lemma 4.5.1 to the end of this Section.

Proof of Lemma 4.4.4: Combining (4.12) and (4.13) from Lemma 4.5.1 we obtain

$$F(\Delta_1(g_1), g_1, g_1) = \min_{\delta_1 \in [0,1]} \max_{h_2 \in [0, g_1]} \begin{cases} F(\delta_1, g_1, h_2) & \text{if } \Delta_2(\delta_1, g_1, h_2) \geq 0 \\ (1-\delta_1)g_1 & \text{if } \Delta_2(\delta_1, g_1, h_2) < 0. \end{cases}$$

For ease of exposition, we drop the subscripts of the variables from now on. Hence we are left to prove $\max_{g \in [\frac{1}{3}, \beta]} F(\Delta_1(g), g, g) = \frac{1}{2} - \frac{1}{3\sqrt{6}} \leq 0.36392$ where

$$F(\Delta_1(g), g, g) =$$
$$\tfrac{1}{4} - \tfrac{1}{4}(1-2g)(3-2g)(4g-1) + 2(1-g)\sqrt{g(1-2g)(-1+4g-2g^2)}$$

It is easy to check that (4.10) implies that the radicand $g(1-2g)(-1+4g-2g^2)$ is nonnegative for all $g \in [1/3, \beta]$. We now prove that the maximum of $F(\Delta(g), g, g)$ on $[\frac{1}{3}, \beta]$ is assumed for $g = 1/\sqrt{6}$: Straightforward calculation leads to

$$\mathcal{F}^* := F\left(\Delta(1/\sqrt{6}),\ 1/\sqrt{6},\ 1/\sqrt{6}\right) = \frac{1}{2} - \frac{1}{3\sqrt{6}}\,.$$

Fixing $g \in [1/3, \beta]$ (arbitrarily), one finds:

$$\mathcal{F}^* - F\left(\Delta_1(g),\ g,\ g\right) =$$
$$= \underbrace{\frac{1}{4} - \frac{1}{3\sqrt{6}} + \frac{1}{4}(1-2g)(3-2g)(4g-1)}_{\geq 0 \ \ (*)} \ \underbrace{-2\,(1-g)\,\sqrt{g(1-2g)(-1+4g-2g^2)}}_{\geq 0 \ \ (**)} .$$

Here $(*)$ and $(**)$ are implied by the choice of g, i.e., $(3-2g) \geq 2(1-g) \geq (1-2g) \geq 0$, and $4g - 1 \geq 1/3 > 0$ hold. Finally, since $\sqrt{6} > 2$ we have $\frac{1}{4} - \frac{1}{3\sqrt{6}} > \frac{1}{12} > 0$.

The inequalities in $(*)$ and $(**)$ together with Observation 1 lead to the equivalence

$$\mathcal{F}^* - F\left(\Delta_1(g),\ g,\ g\right) \geq 0 \quad \Leftrightarrow$$
$$\underbrace{\left(\frac{1}{4} - \frac{1}{3\sqrt{6}} + \frac{1}{4}(1-2g)(3-2g)(4g-1)\right)^2 - 4(1-g)^2\left(g(1-2g)(-1+4g-2g^2)\right)}_{= \left(\frac{11}{18} + \frac{2}{3\sqrt{6}}(3-g) + (1-g)^2\right)\left(g - \frac{1}{\sqrt{6}}\right)^2} \geq 0.$$

Here the second inequality holds for the chosen g, since the term can be reformulated as shown under the brace, where $(3-g) > 0$ holds by the restriction $g \in [1/3, \beta]$.

Thus we showed $\mathcal{F}^* = F(\Delta_1(1/\sqrt{6}), 1/\sqrt{6}, 1/\sqrt{6}) \geq F(\Delta_1(g), g, g)$, proving the lemma, since $g \in [1/3, \beta]$ was chosen arbitrarily and $1/\sqrt{6} \in [1/3, \beta]$ is implied by $0.40 \leq 1/\sqrt{6} \leq 0.41 < \beta$.

□

It now remains to prove Lemma 4.5.1.

Proof of Lemma 4.5.1:

Fix some pair $(g, \delta) \in [1/3, \beta] \times [0, 1]$. We rewrite (4.12) as

$$F(\delta, g, g) \leq \left(\max_{h \in [0,g]} \left\{ \begin{array}{ll} F(\delta, g, h) & \text{if } \Delta_2(\delta, g, h) \geq 0 \\ (1-\delta)g & \text{if } \Delta_2(\delta, g, h) < 0 \end{array} \right. \right) \leq \max_{h \in [0,g]} \ F(\delta, g, g) \tag{4.14}$$

and prove it as follows: A brief calculation, together with $(1-g) > 0$, lead to $\Delta_2(\delta, g, g) = (1-g)\delta/(1-g+\delta) \geq 0$. So there is a $h^* \in [0, g]$, namely $h^* := g$, such that $\Delta_2(\delta, g, h^*) \geq 0$. This implies the first inequality in (4.14).

Observe that to prove the second inequality in (4.14), it suffices to show that

$$F(\delta, g, g) \geq (1-\delta)g \quad \text{and} \quad F(\delta, g, g) \geq F(\delta, g, h) \quad \text{for all } h \in [0, g] \tag{4.15}$$

both hold – independently of the value of Δ_2. Quick calculation proves the first inequality of (4.15): Recall that the choice on (g, δ) implies $(1-g) \geq 0$, $2\delta g \geq 0$, and $(1-2g) \geq 0$, yielding

$$F(\delta, g, g) - (1-\delta)g = \frac{(1-g)\,\delta}{(1-g)+\delta}\ (2\delta g + (1-2g)\ ) \geq 0.$$

To obtain the second inequality of (4.15), we show that for the chosen δ, g, the function $F(\delta, g, h)$ is monotone non-decreasing on $h \in [0, g]$: Recalling $h \leq g \leq 1/2$ we find $(1-2h) \geq 0$, implying

$$\frac{\mathrm{d}F(\delta,g,h)}{\mathrm{d}h} = \frac{(1-2h)(1-\delta)^2 + g\delta(1-\delta)}{(1-g)+\delta} \geq 0.$$

This finally proves (4.15), and thus the second inequality in (4.14), concluding the proof of (4.12).

To prove (4.13) fix some $d \in [0,1]$ arbitrarily and define $\mathfrak{p}(g) := g(1-2g)(-1+4g-2g^2)$, which is the radicand appearing in $F(\Delta_1(g), g, g)$. A Bbief calculation leads to

$$\begin{aligned}&(F(d,g,g) - F(\Delta_1(g),g,g)) \quad (1-g+d) = \\ &\underbrace{\left((4g-1)(1-g)^3 + 2g(1-2g)(1-g)d + g(1-2g)d^2\right)}_{\geq 0 \ \ (\star)} \ - \ \underbrace{2(1-g+d)(1-g)\sqrt{\mathfrak{p}(g)}}_{\geq 0 \ \ (\star\star)} \ .\end{aligned}$$

To obtain $(\star)$, recall that $1/3 < \beta < 1/2$ and observe that the restrictions on g, d imply $g, d \geq 0$ as well as $(4g-1) \geq 0$, $(1-g) \geq 0$, and $(1-2g) \geq 0$. Moreover we have $(1-g+d) > (1-g) \geq 0$, showing $(\star\star)$. Recall also that (4.10) implies that $\mathfrak{p}(g) \geq 0$ for the chosen g. Hence, by exploiting $(1-g+d) > 0$ and Fact 1, we obtain:

$$\begin{gathered}F(d,g,g) - F(\Delta_1(g),g,g) \geq 0 \\ \Leftrightarrow \\ \left((4g-1)(1-g)^3 + 2g(1-2g)(1-g)d + g(1-2g)d^2\right)^2 - 4(1-g+d)^2(1-g)^2\mathfrak{p}(g) \geq 0 \\ \Leftrightarrow \\ \left((1-3g)(1-g)^2 + 2g(1-2g)(1-g)d + g(1-2g)d^2\right)^2 \geq 0.\end{gathered}$$

The last inequality is trivially true, which finally proves (4.13) since $(g,d) \in [1/3, \beta] \times [0,1]$ were chosen arbitrarily.

□

4.6 Games with more than 2 players

In this section we consider games with more than two players. A Nash equilibrium for multi-player games is defined in the same way as for two-player games. It is a choice of strategies such that no agent has a unilateral incentive to deviate. We show now how the simple 1/2-approximation algorithm for two players by Daskalakis *et al.* [DMP06] may be generalized to a procedure that reduces the number of players in the computation of an approximate equilibrium.

Lemma 4.6.1 *Given an α-approximation algorithm for games with $k-1$ players, we can construct a $\frac{1}{2-\alpha}$-approximation algorithm for k-player games.*

Proof: Suppose we are given a game with k players. Pick any player, e.g. the first player, and fix any strategy x_1 for this player. If the first player's strategy is fixed at x_1, the game becomes a $k-1$ player game. Hence we may use the α-approximation algorithm to obtain an α-approximate equilibrium $(x_2, \ldots, x_k)$ for the players $2, ..., k$ in this restricted game. Finally, player 1 computes his optimal response r_1 to $(x_2, \ldots, x_k)$ and plays a mix of his original strategy x_1 and the new strategy r_1. Let δ be the probability that player 1 plays r_1. Hence the output of this construction is $((1-\delta)x_1 + \delta r_1, x_2, ..., x_k)$.

We will now measure the quality of this construction. The incentive to deviate for player 1 may be bounded by $1-\delta$. For the other players the incentive may be bounded by $\alpha(1-\delta)+\delta$. By equalizing the incentives we get $\delta = \frac{1-\alpha}{2-\alpha}$, which gives the upper bound for the incentive $1-\delta = \frac{1}{2-\alpha}$. □

We may now repeatedly apply Lemma 4.6.1 combined with the 0.3393-approximation for two-player games of Spirakis and Tsaknakis [ST07] to get constant factor approximations for any fixed number of players. In particular, we get 0.60205-approximation for three player games and 0.71533-approximation for four-player games. To the best of our knowledge this is the first nontrivial polynomial time approximation for multiplayer normal form games.

4.7 Discussion

In general, our algorithms produce solutions with large support. This is to no surprise, as implied by negative results on the existence of approximate equilibrium strategies with small support [Alt94; FNS07].

The major remaining open question here is whether a polynomial time algorithm for any constant $\epsilon > 0$ is possible. It would be interesting to investigate if we can exploit further the use of zero-sum games to obtain better approximations. We would also like to study if our techniques can be used for the stronger notions of approximation discussed in [KS96] and [EY07].

Chapter 5

Set Multicover

5.1 Preliminaries

In this chapter, we discuss the problem of covering elements with sets. The problem of covering each element at least once is the classical Set Cover problem. We show that the covering problem becomes easier for approximation if we require that each of the elements is covered many times.

The problem we study can be thought of as an abstraction of the following situation. Suppose we need to provide a certain service at a number of sites, e.g., we need to locate ambulances, so that a number of locations can be reached within a certain driving time. In this example, the multiple coverage requirement corresponds to the reliability of the service or, in other words, to the robustness of the rescue system. In such terms, we may interpret the results presented here as follows. Providing logarithmic (in the number of nodes to serve) robustness in the system is a problem for which we may find constant factor approximations. Moreover, if the single coverage version has a large integrality gap, then we may compute a solution for the multiple coverage version that is not much more expensive than the optimal single coverage. Finally, because approximating the single coverage is a difficult problem, the theoretical bounds on the number of sets (ambulances) used in the two versions of the problem are not much different, i.e., they differ by at most a multiplicative constant.

Our technique is based on adding sets to the current solution until the proper level of coverage is achieved. The choice of a set to be added is based on a specific weight function that takes into account the previously chosen sets. The construction of the weight function guarantees efficient usage of sets if the initial coverage requirement is big enough.

The chapter is organized as follows. First, we discuss the Set Cover and the Set Multicover problems. In Section 5.3, we present the main technique in a form of a strategy for a certain game. Then we apply the technique, in its simplest form, to the Set Multicover problem in Section 5.4. In Section 5.5 we study the optimal choice of a parameter of our algorithm.

5.2 The covering problems

The Set Cover problem is defined as follows. Given a collection $\mathcal{C}$ of subsets of U, such that $\bigcup_{S\in\mathcal{C}} S = U$, find a collection $\mathcal{C}' \subseteq \mathcal{C}$, such that $\bigcup_{S\in\mathcal{C}'} S = U$ and $|\mathcal{C}'|$ is minimal. We assume that the ground set (also called universum) U has n elements, numbered from 1 to n. We denote such a set by $[n]$.

The Set Cover problem is NP-complete. Moreover, it was proved by Feige [Fei98] that the existence of an $(c \cdot \ln n)$-approximation algorithm for Set Cover with $c < 1$ would imply that $NP \subseteq DTIME(n^{\log\log n})$. Raz and Safra [RS97] proved a slightly weaker bound on c but using a weaker assumption that $P \neq NP$. On the other hand, Johnson [Joh73] and Lovász [Lov75] showed two different algorithms, both of which approximate Set Cover within a factor of $H_n = \sum_{i=1}^{n} \frac{1}{i} = \Theta(\log n)$.

A natural generalization of Set Cover is the Set Multicover problem, where each element x of U needs to be covered by at least l_x sets, and each subset $S \in \mathcal{C}$ may be used an arbitrary number of times. This problem was addressed by Rajagopalan and Vazirani in [RV99], where an H_n-approximation algorithm was presented. We will, however, concentrate on the particular case where each element must be covered by at least k subsets (i.e., $l_x = k$ for all $x \in U$). We call this problem k-SetCover. An instance $\mathcal{SC}$ of the k-SetCover is then defined by a triplet $(n, \mathcal{C}, k)$, where $\mathcal{C}$ is a collection of $[n]$ subsets. When $k = 1$, then it is an instance of a Set Cover problem, which we denote $(n, \mathcal{C})$.

A similar problem – a version where each set may be used at most once (known as Constrained Set Multicover) – was recently shown [BDS07] to have applications to reverse engineering of protein and gene networks.

In this chapter, the logarithms are of base 2 unless stated otherwise.

5.3 Bucket Game

In this section we present a simple two-player *Bucket Game* [BB05], which is used as a showcase for the *Exponential Balancing* (EB) technique. The game was originally used to construct an asymptotically optimal online randomized algorithm for the Dynamic Page Migration problem. Meanwhile, it turned out that the EB technique is also perfectly applicable to the k-SetCover problem. In this chapter we discuss the latter application only.

The game is defined as follows.

Definition 5.3.1 (Bucket Game) *Assume we have a set of n buckets, which are initially empty, numbered from 1 to n.*[1] *We also have an infinite set of balls. For any $i \in [n]$, let c_i be the current number of balls in bucket i. Let $0 < c < 1$ be any fixed constant. The* Bucket Game *is played in rounds by two players A and B. Each round of the game is defined as follows.*

[1] Note, that the set of buckets corresponds to the ground set in the Set (Multi)Cover problem. Nevertheless, the game is meant to be more abstract than that, and the relation with the Set Cover problem is not essential for the strategies of the players.

- *Player A defines a sequence of non-negative weights $\{w_i\}_{i=1}^n$ and shows it to player B.*
- *Player B chooses some subset $X \subseteq [n]$ of buckets, s.t. $\sum_{i \in X} w_i \geq c \cdot \sum_{i=1}^n w_i$, and throws exactly one ball into each bucket from X.*

The game ends when each of the buckets contains at least T balls (i.e., $c_i \geq T$ for all $i \in [n]$). The goal of player A is to minimize the number of rounds, while B wants to play as long as possible.

Let us make the following simple observations. First of all, to throw at least one ball into each bucket (i.e., for the case $T = 1$), $\mathcal{O}(\log n)$ rounds are sufficient. Player A simply defines $w_i = 1$ for empty buckets and $w_i = 0$ for non-empty ones. Then in each round at least a fraction c of empty buckets gets a ball. Hence, after at most $\log_{1/(1-c)} n$ rounds there is no empty bucket left. Moreover, to fill each bucket to the threshold T, player A can repeat this scheme T times, which yields an upper bound of $\mathcal{O}(T \cdot \log n)$.

Second, for any T, there exists a player B strategy which prevents finishing the game in less than $\Omega(T + \log n)$ rounds. Since each bucket may get at most one ball per round, the number of rounds cannot be smaller than T. On the other hand, suppose that in every round B chooses the subset with the smallest number of empty buckets. With this strategy, in the i-th round, at most $\lceil c \cdot e_i \rceil$ of empty buckets get a ball, where e_i is the number of empty buckets at the beginning of the i-th round. Thus $\Omega(\log n)$ rounds are also necessary.

Surprisingly, there is a simple player A strategy, which we call *Exponential Balancing*, and which asymptotically matches the above-mentioned lower bound.

Theorem 5.3.2 *For $T = \log n$, there exists a player A strategy which guarantees finishing the game in $\mathcal{O}(\log n)$ rounds.*

Proof: In each round, A defines $w_i = 2^{-c_i}$. We call w_i a *weight of a bucket* i, and define *the total weight* as $\mathcal{W} = \sum_{i \in [n]} w_i$.

Initially, all buckets are empty, $w_i = 1$ for all i. Hence, the initial total weight is n. In any round each bucket "offers" half of its current weight to player B for putting a ball into this bucket. According to the rules of the game, B must collect at least a fraction c of this offered weight. Thus, in every round, the a fraction $c/2$ of the total weight is removed, i.e., if $\mathcal{W}$ and $\mathcal{W}'$ denote the total weights in two consecutive rounds, then $\mathcal{W}' \leq (1 - c/2) \cdot \mathcal{W}$.

If in a round it holds that $\mathcal{W} \leq 1/n$, then the threshold T is reached and the game ends. Precisely, $\mathcal{W} = \sum_i 2^{-c_i} \leq 1/n$ implies $2^{-c_i} \leq 1/n$ for all $i \in [n]$, and thus $c_i \geq \log n$ for all $i \in [n]$.

It remains to observe that the weight of the game is reduced from n to $1/n$ in at most $\log_{1/(1-c/2)} n^2 = \frac{2}{\log 1/(1-c/2)} \cdot \log n$ rounds. □

If threshold T is larger than $\log n$, then player A may act as if he was playing $\lceil T / \log n \rceil$ times a game with threshold $\log n$. By Theorem 5.3.2, each of these subgames lasts $\mathcal{O}(\log n)$ rounds, and thus the whole game ends after at most $\mathcal{O}(T)$ rounds. Thus, we get the following.

Corollary 5.3.3 *For any T, there exists a player A strategy which guarantees finishing the game in $\mathcal{O}(T + \log n)$ rounds.*

5.4 Simple constant factor approximation

Consider the relaxation of the Set Cover problem, where each covering set can be taken a fractional number of times, and call feasible solutions to this relaxation *fractional covers* [Lov75]. Formally, a fractional cover is a function $f : \mathcal{C} \to [0, 1]$, such that for all elements $i \in U$, $\sum_{S:i\in S} f(S) \geq 1$.

For any instance $(n, \mathcal{C})$ of the Set Cover problem, let $z(n, \mathcal{C}, f) = \sum_{S\in\mathcal{C}} f(S)$ denote the "size" of the fractional cover f, and let $z^*(n, \mathcal{C}) = \min_f z(n, \mathcal{C}, f)$ denote the minimal size of a fractional cover.

As the cost of the optimal fractional solution of the k-SetCover instance is exactly k times the cost of the optimal fractional solution of the corresponding Set Cover instance, we obtain the following lemma.

Lemma 5.4.1 *Let $(n, \mathcal{C}, k)$ be an instance of the k-SetCover problem. Then the cost of its optimal solution is at least $k \cdot z^*(n, \mathcal{C})$.*

Consider a positive weight function $w : U \to \mathbb{R}^{\geq 0}$ defined on the elements of the ground set. The notion of weights extends naturally to sets of elements, i.e., the weight of set S is $w(S) = \sum_{x\in S} w(x)$. We may bound the weight of the "heaviest" set as follows.

Lemma 5.4.2 *For any Set Cover instance $(n, \mathcal{C})$, and any weight function w, it holds that*

$$\max_{S\in\mathcal{C}} w(S) \geq \frac{\sum_{x\in U} w(x)}{z^*(n, \mathcal{C})} \quad .$$

Proof: Let f be a fractional cover of size $z^*(n, \mathcal{C})$.

$$\begin{aligned}\max_{S\in\mathcal{C}} w(S) &\geq \frac{\sum_{S\in\mathcal{C}} f(S)w(S)}{\sum_{S\in\mathcal{C}} f(S)} \\ &\geq \frac{\sum_{x\in U} w(x)}{\sum_{S\in\mathcal{C}} f(S)} \\ &= \frac{\sum_{x\in U} w(x)}{z^*(n, \mathcal{C})}.\end{aligned}$$

□

Algorithm EBCover
Input: $(n, \mathcal{C}, k)$, instance of k-SetCover
Output: multiset $\mathcal{C}_{\text{EB}}$ covering each element k times
1. $\mathcal{C}_{\text{EB}} = \emptyset$
2. For any $x \in U$, let $c_x = |\{S \in \mathcal{C}_{\text{EB}} : x \in S\}|$ denote the number of times $\mathcal{C}_{\text{EB}}$ covers x
3. **while** there exists $x \in U$ with $c_x < k$
4. $\quad w(x) \leftarrow 2^{-c_x}$
5. $\quad \mathcal{C}_{\text{EB}} \leftarrow \mathcal{C}_{\text{EB}} \cup \{\arg\max_{S \in \mathcal{C}} w(S)\}$ /* add the heaviest set */
6. return $\mathcal{C}_{\text{EB}}$;

Figure 5.1: Algorithm EBCover

The following technical lemma is useful for bounding the approximation ratios.

Lemma 5.4.3 *For any $a, x > 0$, it holds that*

$$\frac{1}{\log_a(1/(1-x))} \leq \frac{\ln a}{x} \ .$$

Proof: Fix any $a, x > 0$. Using the relation $(1-x)^{1/x} \leq \frac{1}{\mathrm{e}}$, we obtain

$$\begin{aligned}
\frac{1}{\log_a(1/(1-x))} &= \frac{1}{x \cdot \frac{1}{x} \cdot \log_a\left((1-x)^{-1}\right)} \\
&= \frac{1}{x \cdot \log_a\left((1-x)^{-1/x}\right)} \\
&\leq \frac{1}{x \cdot \log_a \mathrm{e}} \\
&= \frac{\ln a}{x} \ .
\end{aligned}$$

□

Consider the algorithm EBCover described in Figure 5.1.

Theorem 5.4.4 *The approximation ratio of the* EBCover *algorithm on a* $(\log n)$*-SetCover instance is at most* $4 \cdot \ln 2 \approx 2.772$.

Proof: Fix any instance of $(\log n)$-SetCover, $(n, \mathcal{C}, \log n)$. We reformulate the proof of Theorem 5.3.2 in terms of set covers.

Let $c = 1/z^*(n, \mathcal{C}) \leq 1$. The algorithm EBCover takes the role of player A defining weights on items that we want to cover. The set chosen in step 5 corresponds to a valid choice of player B (by Lemma 5.4.2, such choice removes at least a fraction of $c/2$ from the total weight). Thus, by the same argument as in the proof of

Theorem 5.3.2, the algorithm terminates after adding $\frac{2}{\log(1/(1-c/2))} \cdot \log n$ sets to $\mathcal{C}_{\text{EB}}$.

By Lemma 5.4.1 the optimal solution has at least $k \cdot z^*(\mathcal{SC}) = \frac{1}{c} \cdot \log n$ sets. Hence, the approximation ratio is bounded by

$$\frac{\frac{2}{\log(1/(1-c/2))} \cdot \log n}{\frac{1}{c} \cdot \log n} = \frac{2c}{\log(1/(1-c/2))} \leq \frac{2c \cdot \ln 2}{c/2} = 4 \cdot \ln 2 \ .$$

□

5.5 Improvements by parameter adjustments

In this section, we consider a different choice of constants for the EBCover algorithm. In particular, we may change the base 2 in the weight function formula $w(x)$ occurring in step 4 to a parameter α. We call a resulting algorithm EBCover$_\alpha$. It turns out that for large coverage parameters, and with an appropriate choice of parameter α, the approximation ratio tends to 1.

Theorem 5.5.1 *The approximation ratio of the* EBCover$_\alpha$ *algorithm on a* $(\log_\gamma n)$*-SetCover instance is at most* $\frac{\alpha}{\alpha-1} \cdot (\ln \gamma + \ln \alpha)$.

Proof: The proof is a parametrized version of the proof of Theorem 5.4.4. Fix any instance $(n, \mathcal{C}, \log_\gamma n)$. Again, let $c = 1/z^*(n, \mathcal{C})$. The initial total weight is equal to $n \cdot \alpha^{-0} = n$, and the algorithm may stop when the total weight drops below $\alpha^{-\log_\gamma n}$. Moreover, in each step at least a fraction of $c \cdot \frac{\alpha-1}{\alpha}$ is removed from the total weight. Thus the number E of sets used by the algorithm is at most

$$E \leq \log_{\frac{1}{1-c\cdot\frac{\alpha-1}{\alpha}}} \left(n \cdot \alpha^{\log_\gamma n}\right) = \frac{\log_\gamma n + \log_\gamma n \cdot \log_\gamma \alpha}{\log_\gamma \left(1/\left(1 - c \cdot \frac{\alpha-1}{\alpha}\right)\right)} \ .$$

At this time, the lower bound on the optimum is $\frac{1}{c} \cdot \log_\gamma n$, and we may bound the approximation factor by

$$\frac{c \cdot (1 + \log_\gamma \alpha)}{\log_\gamma \left(1/\left(1 - c \cdot \frac{\alpha-1}{\alpha}\right)\right)} \leq \frac{c \cdot (1 + \log_\gamma \alpha) \cdot \ln \gamma}{c \cdot \frac{\alpha-1}{\alpha}} = \frac{\alpha}{\alpha - 1} \cdot (\ln \gamma + \ln \alpha) \ .$$

□

For a fixed $\gamma > 1$, let α_γ denote the optimal choice of $\alpha > 1$, i.e., the value of α that minimizes $f(\alpha, \gamma) = \frac{\alpha}{\alpha-1} \cdot (\ln \gamma + \ln \alpha)$.

Lemma 5.5.2 *For any* $\gamma > 1$, α_γ *is defined by:*

$$1 + \ln \gamma + \ln \alpha_\gamma - \alpha_\gamma = 0.$$

Moreover, $f(\alpha_\gamma, \gamma) = \alpha_\gamma$.

Proof: To find the value of α_γ we calculate the derivative of $f(\alpha, \gamma)$ with respect to α.

$$\frac{d(f(\alpha,\gamma))}{d\alpha} = \frac{-1+\alpha-\ln\gamma-\ln\alpha}{(-1+\alpha)^2}.$$

As we restrict our attention to the case $\alpha > 1$, the enumerator of the derivative is monotone increasing in α and it is equal 0 when $1+\ln\gamma+\ln\alpha-\alpha=0$. As we assume that $\alpha > 1$, the denominator of the derivative is always positive, hence the minimum of $f(\alpha, \gamma)$ with respect to $\alpha > 1$ is attained when the derivative vanishes.

As we use the equation defining α_γ to simplify $f(\alpha_\gamma, \gamma)$, we obtain

$$f(\alpha_\gamma,\gamma) = \frac{\alpha_\gamma(\ln\gamma+\ln\alpha_\gamma)}{\alpha_\gamma-1} = \frac{\alpha_\gamma((\alpha_\gamma-1-\ln\alpha_\gamma)+\ln\alpha_\gamma)}{\alpha_\gamma-1} = \frac{\alpha_\gamma(\alpha_\gamma-1)}{\alpha_\gamma-1} = \alpha_\gamma.$$

□

As we have just shown, for a given value of γ, α_γ gives both the optimal choice of α and the resulting approximation guarantee. The following table shows values of α_γ for various choices of γ.

γ	α_γ
1.001	1.046
1.01	1.148
1.1	1.502
1.2	1.731
1.5	2.189
2	2.678
e	3.146
3	3.289

Consider a more universal version of our algorithm, namely the algorithm EBCoverU defined as follows. The algorithm EBCoverU on an instance $(n, \mathcal{C}, k)$ first computes $\gamma = n^{1/k}$ so that $\log_\gamma n = k$. Then it computes α_γ as a solution to $1+\ln\gamma+\ln\alpha_\gamma-\alpha_\gamma=0$. Finally it runs the algorithm EBCover$_\alpha$ with $\alpha = \alpha_\gamma$.

We may think of the EBCoverU algorithm as of a universal algorithm for the k-SetCover problem (for general k). Additionally, by Theorem 5.5.1 and Lemma 5.5.2, under the assumption that $k = \Omega(\log n)$, EBCoverU is a constant factor approximation algorithm for the k-SetCover problem.

Suppose we could assume that the required coverage is even larger, namely that there exists $k_0(n) = \omega(\log n)$ and that $k \geq k_0(n)$ in all the considered instances. In such a situation, as $\lim_{\gamma\to 1} f(\alpha_\gamma, \gamma) = 1$, we get the following.

Corollary 5.5.3 *There is a PTAS for the k-SetCover problem with $k \geq k_0(n) = \omega(\log n)$.*

Proof: To get an ϵ-approximation algorithm for some $\epsilon > 1$ we proceed as follows. We compute γ such that $\alpha_\gamma = \epsilon$ and $1+\ln\gamma+\ln\alpha_\gamma-\alpha_\gamma=0$. Then we compute n_0,

such that $log_\gamma n_0 = k_0(n_0)$. Finally, we construct an algorithm that on an instance $(n, \mathcal{C}, k)$ of the k-SetCover problem proceeds as follows:

- if $n < n_0$, the algorithms enumerates all the possible solutions and finds the optimal solution for the instance;
- if $n \geq n_0$, the algorithm runs the EBCoverU algorithm.

Observe, that for big enough values of n, the required coverage is also big, so the obtained values of γ in the algorithm EBCoverU are small enough. As a result the values of α_γ are small enough, in particular, $\alpha_\gamma \leq \epsilon$. □

References

[Abe76] R. Abel, *Man is the measure: A cordial invitation to the central problems of philosophy*, pp. 1–17, The Free Press, 1976.

[ABM08] K. Aardal, J. Byrka, and M. Mahdian, *Facility location*, Encyclopedia of Algorithms (to appear) (M.-Y. Kao, ed.), Springer, 2008.

[ABUvH04] A. Anagnostopoulos, R. Bent, E. Upfal, and P. van Hentenryck, *A simple and deterministic competitive algorithm for online facility location*, Information and Computation **194** (2004), no. 2, 175–202.

[ACS99] K. Aardal, F. A. Chudak, and D. B. Shmoys, *A 3-approximation algorithm for the k-level uncapacitated facility location problem*, Information Processing Letters **72** (1999), 161–167.

[AGK+01] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit, *Local search heuristics for k-median and facility location problems*, Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC), ACM, 2001, pp. 21–29.

[AKK99] S. Arora, D. Karger, and M. Karpinski, *Polynomial time approximation schemes for dense instances of NP-hard problems*, Journal of Computer and System Sciences **58** (1999), no. 1, 193–210.

[Alt94] I. Althöfer, *On sparse approximations to randomized strategies and convex combinations*, Linear Algebra and Applications **199** (1994), 339–355.

[ARR98] S. Arora, P. Raghavan, and S. Rao, *Approximation schemes for euclidean k-medians and related problems*, Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC), ACM, New York, NY, USA, 1998, pp. 106–113.

[AS99] A. A. Ageev and M. I. Sviridenko, *An 0.828-approximation algorithm for the uncapacitated facility location problem*, Discrete Applied Mathematics **93** (1999), 149–156.

[ASSU81] A. Aho, Y. Sagiv, T. Szymanski, and J. Ullman, *Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions*, SIAM Journal on Computing **10** (1981), no. 3, 405–421.

[AYZ03] A. Ageev, Y. Ye, and J. Zhang, *Improved combinatorial approximation algorithms for the k-level facility location problem*, Proceedings of the 30th International Colloquium on Automata, Languages and Programming (ICALP), LNCS, vol. 2719, Springer, 2003, pp. 145–156.

[BA07] J. Byrka and K. Aardal, *The approximation gap for the metric facility location problem is not yet closed*, Operations Research Letters **35** (2007), no. 3, 379–384.

[Bal66] M. L. Balinski, *On finding integer solutions to linear programs*, Proceedings of the IBM Scientific Computing Symposium on Combinatorial Problems, 1966, pp. 225–248.

[BB05] M. Bienkowski and J. Byrka, *Bucket game with applications to set multicover and dynamic page migration*, Proceedings of the 13th European Symposium on Algorithms (ESA), LNCS, vol. 3669, Springer, 2005, pp. 815–826.

[BBB+08] K. Buchin, M. Buchin, J. Byrka, M. Nöllenburg, R. I. Silveira, and A. Wolff, *Drawing (complete) binary tanglegrams: Hardness, approximation, fixed-parameter tractability*, arXiv:0806.0920v1 [cs.CG], 2008, To appear in Proceedings of the 16th International Symposium on Graph Drawing (GD).

[BBM07] H. Bosse, J. Byrka, and E. Markakis, *New algorithms for approximate Nash equilibria*, Proceedings of the 3rd Workshop on Internet and Network Economics (WINE), LNCS, vol. 4858, Springer, 2007, pp. 17–29.

[BDMT98] P. Bertolazzi, G. Di Battista, C. Mannino, and R. Tamassia, *Optimal upward planarity testing of single-source digraphs*, SIAM Journal on Computing **27** (1998), no. 1, 132–169.

[BDS07] P. Berman, B. DasGupta, and E. Sontag, *Randomized approximation algorithms for set multicover problems with applications to reverse engineering of protein and gene networks*, Discrete Applied Mathematics **155** (2007), no. 6-7, 733–749.

[BE04] O. Bininda-Emonds, *Phylogenetic supertrees: combining information to reveal the tree of life*, Computational Biology Series, vol. 4, The MIT Press, 2004.

[BFC00] M. Bender and M. Farach-Colton, *The lca problem revisited*, Proceedings of Latin American Theoretical INformatics (LATIN), LNCS, vol. 1776, Springer, 2000, pp. 88–94.

[BFC04] M. Bender and M. Farach-Colton, *The level ancestor problem simplified*, Theoretical Computer Science **321** (2004), no. 1, 5–12.

[BGHK08] J. Byrka, P. Gawrychowski, K. T. Huber, and S. Kelk, *Worst-case optimal approximation algorithms for maximizing triplet consistency within phylogenetic networks*, arXiv:0710.3258v3 [q-bio.PE], 2008.

[BGJ08] J. Byrka, S. Guillemot, and J. Jansson, *New results on optimizing rooted triplets consistency*, To appear in Proceedings of the 19th International Symposium on Algorithms and Computation (ISAAC), 2008.

[Bry97] D. Bryant, *Building trees, hunting for trees, and comparing trees: theory and methods in phylogenetic analysis*, Ph.D. thesis, University of Canterbury, Christchurch, New Zealand, 1997.

[BSS04] M. Baroni, C. Semple, and M. Steel, *A framework for representing reticulate evolution*, Annals of Combinatorics **8** (2004), no. 4, 391–408.

[BW63] M. L. Balinski and P. Wolfe, *On benders decomposition and a plant location problem*, ARO-27, Mathematica Inc. Princeton, NJ, USA, 1963.

[Byr07] J. Byrka, *An optimal bifactor approximation algorithm for the metric uncapacitated facility location problem*, Proceedings of the 10th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX), LNCS, vol. 4627, Springer, 2007, pp. 29–43.

[CD06] X. Chen and X. Deng, *Settling the complexity of 2-player Nash equilibrium*, Proceeding of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, 2006, pp. 261–272.

[CDT06] X. Chen, X. Deng, and S. Teng, *Computing Nash equilibria: Approximation and smoothed complexity*, Proceeding of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, 2006, pp. 603–612.

[CFN77] G. Cornuéjols, M. L. Fisher, and G. L. Nemhauser, *Location of bank accounts to optimize float: An analytic study of exact and approximate algorithms*, Management Science **8** (1977), 789–810.

[CG99] M. Charikar and S. Guha, *Improved combinatorial algorithms for facility location and k-median problems*, Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, 1999, pp. 378–388.

[CGTS99] M. Charikar, S. Guha, E. Tardos, and D. B. Shmoys, *A constant-factor approximation algorithm for the k-median problem*, Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC), ACM, 1999, pp. 1–10.

[Chu98] F. Chudak, *Improved approximation algorithms for uncapacited facility location*, Integer Programming and Combinatorial Optimization (R. Bixby, E. Boyd, and R. Ríos-Mercado, eds.), Lecture Notes in Computer Science, vol. 1412, Springer, Berlin, 1998, pp. 180–194.

[CK] P. Crescenzi and V. Kann, *A compendium of NP optimization problems*, `http://www.nada.kth.se/~viggo/problemlist/compendium.html`.

[CKMN01] M. Charikar, S. Khuller, D. Mount, and G. Narasimhan, *Facility location with outliers*, Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), SIAM, 2001, pp. 642–651.

[CLL+08] J. Chen, Y. Liu, S. Lu, B. O'Sullivan, and I. Razgon, *A fixed-parameter algorithm for the directed feedback vertex set problem*, Proceedings of 40th Annual ACM Symposium on Theory of Computing (STOC), ACM, 2008, pp. 177–186.

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*, 2nd ed., The MIT Press, 2001.

[CMM05] I. Cassens, P. Mardulyn, and M. Milinkovitch, *Evaluating intraspecific "network" construction methods using simulated sequence data: Do existing algorithms outperform the global maximum parsimony approach?*, Systematic Biology **54** (2005), 363–372.

[CNW90] G. Cornuéjols, G. L. Nemhauser, and L. A. Wolsey, *The uncapacitated facility location problem*, Discrete Location Theory (P. Mirchandani and R. Francis, eds.), John Wiley & Sons Inc., 1990, pp. 119–171.

[Coo71] S. Cook, *The complexity of theorem proving procedures*, Proceedings of the Third Annual ACM Symposium on Theory of Computing (STOC), ACM, 1971, pp. 151–158.

[CRV07] G. Cardona, F. Rossello, and G. Valiente, *Tripartitions do not always discriminate phylogenetic networks*, arXiv:0707.2376v1 [q-bio.PE], 2007.

[CS98] B. Chor and M. Sudan, *A geometric approach to betweenness*, SIAM Journal on Discrete Mathematics **11** (1998), no. 4, 511–523.

[CS03] F. A. Chudak and D. B. Shmoys, *Improved approximation algorithms for the uncapacitated facility location problem*, SIAM Journal on Computing **33** (2003), no. 1, 1–25.

[CW99] F. Chudak and D. Wiliamson, *Improved approximation algorithms for capacitated facility location problems*, Proceedings of the 7th Conference on Integer Programing and Combinatorial Optimization (IPCO), LNCS, vol. 1610, Springer, 1999, pp. 99–113.

[Dar59] C. Darwin, *The origin of species by means of natural selection or the preservation of favoured races in the struggle for life*, J. Murray, 1859.

[DF99] R. Downey and M. Fellows, *Parameterized complexity*, Springer, 1999.

[DFK04] V. Dujmovič, H. Fernau, and M. Kaufmann, *Fixed parameter algorithms for one-sided crossing minimization revisited*, Proceedings of the 11th International Symposium on Graph Drawing (GD'03), LNCS, vol. 2912, Springer, 2004, pp. 332–344.

[DGP06] C. Daskalakis, P. Goldberg, and C. Papadimitriou, *The complexity of computing a Nash equilibrium*, Proceedinds of the 38th Annual ACM Symposium on Theory of Computing (STOC), ACM, 2006, pp. 71–78.

[DHJ+97] B. DasGupta, X. He, T. Jiang, M. Li, J. Tromp, and L. Zhang, *On distances between phylogenetic trees*, Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA), SIAM, 1997, pp. 427–436.

[DMP06] C. Daskalakis, A. Mehta, and C. Papadimitriou, *A note on approximate Nash equilibria*, Workshop on Internet and Network Economics (WINE), LNCS, vol. 4286, Springer, 2006, pp. 297–306.

[DMP07] C. Daskalakis, A. Mehta, and C. Papadimitriou, *Progress on approximate Nash equilibria*, Proceedings of the 8th ACM Conference on Electronic Commerce (EC), ACM, 2007, pp. 355–358.

[Doo99] W. Doolittle, *Phylogenetic classification and the universal tree*, Nature **284** (1999), 2124–2128.

[DS04] T. Dwyer and F. Schreiber, *Optimal leaf ordering for two and a half dimensional phylogenetic tree visualization*, Proceedings of the 2004 Australasian symposium on Information Visualisation (InVis.au), CRPIT, vol. 35, Australian Computer Society, 2004, pp. 109–115.

[ENSS98] G. Even, J. Naor, B. Schieber, and M. Sudan, *Approximating minimum feedback sets and multicuts in directed graphs*, Algorithmica **20** (1998), no. 2, 151–174.

[Erl78] D. Erlenkotter, *A dual-based procedure for uncapacitated facility location problems*, Operations Research **26** (1978), 992–1009.

[EW94] P. Eades and N. Wormald, *Edge crossings in drawings of bipartite graphs*, Algorithmica **10** (1994), 379–403.

[EY07] K. Etessami and M. Yannakakis, *On the complexity of Nash equilibria and other fixed points*, Proceedings of the 48th Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, 2007, pp. 113–123.

[Fei98] U. Feige, *A threshold of lnn for approximating set cover*, Journal of the ACM **45** (1998), no. 4, 634–652.

[FKP05] H. Fernau, M. Kaufmann, and M. Poths, *Comparing trees via crossing minimization*, Proceedings of the 25th Foundations of Software Technology and Theoretical Computer Science (FSTTCS), LNCS, vol. 3821, Springer, 2005, pp. 457–469.

[FNS07] T. Feder, H. Nazerzadeh, and A. Saberi, *Approximating Nash equilibria using small-support strategies*, Proceedings of the 8th ACM Conference on Electronic Commerce (EC), ACM, 2007, pp. 352–354.

[Fot03] D. Fotakis, *On the competitive ratio for online facility location*, Proceedings of the 30th International Colloquium on Automata, Languages and Programming (ICALP), LNCS, vol. 2719, Springer, 2003, pp. 637–652.

[GJLO99] L. Gąsieniec, J. Jansson, A. Lingas, and A. Östlin, *On the complexity of constructing evolutionary trees*, Journal of Combinatorial Optimization **3** (1999), 183–197.

[GK98] S. Guha and S. Khuller, *Greedy strikes back: Improved facility location algorithms*, Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms (SODA), SIAM, 1998, pp. 228–248.

[GMM00] S. Guha, A. Meyerson, and K. Munagala, *Hierarchical placement and network design problems*, Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, 2000, pp. 603–612.

[GP06] P. Goldberg and C. Papadimitriou, *Reducibility among equilibrium problems*, Proceedinds of the 38th Annual ACM Symposium on Theory of Computing (STOC), ACM, 2006, pp. 61–70.

[GPRS04] A. Gupta, M. Pál, R. Ravi, and A. Sinha, *Boosted sampling: approximation algorithms for stochastic optimization*, Proceedings of the 36st Annual ACM Symposium on Theory of Computing (STOC), ACM, 2004, pp. 417–426.

[Gui] S. Guillemot, Private communication.

[Gur90] Y. Gurevich, *Nondeterministic linear-time tasks may require substantially nonlinear deterministic time in the case of sublinear work space*, J. ACM **37** (1990), no. 3, 674–687.

[Gvo08] N. Gvozdenović, *Approximating the stability number and the chromatic number of a graph via semidefinite programming*, Ph.D. thesis, Universiteit van Amsterdam, 2008.

[GW95] M. X. Goemans and D. P. Williamson, *Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming*, Journal of ACM **42** (1995), no. 6, 1115–1145.

[Hås97] J. Håstad, *Some optimal approximability results*, Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC), ACM, 1997, pp. 1–10.

[Has99] J. Hastad, *Clique is hard to approximate within* $n^{1-\varepsilon}$, Acta Mathematica **182** (1999), 105–142.

[HMM03] M. Hajiaghayi, M. Mahdian, and V. Mirrokni, *The facility location problem with general cost functions*, Networks **42** (2003), no. 1, 42–47.

[Hoc82] D. S. Hochbaum, *Heuristics for the fixed cost median problem*, Mathematical Programming **22** (1982), no. 2, 148–162.

[HS85] D. S. Hochbaum and D. B. Shmoys, *A best possible approximation algorithm for the k-center problem*, Mathematics of Operations Research **10** (1985), 180–184.

[HSV+94] M. S. Hafner, P. D. Sudman, F. X. Villablanca, T. A. Spradling, J. W. Demastes, and S. A. Nadler, *Disparate rates of molecular evolution in cospeciating hosts and parasites*, Science **265** (1994), 1087–1090.

[Jan01] J. Jansson, *On the complexity of inferring rooted evolutionary trees*, Proceedings of the Brazilian Symposium on Graphs, Algorithms, and Combinatorics, Electronic Notes in Discrete Mathematics, vol. 7, Elsevier, 2001, pp. 121–125.

[JMS02] K. Jain, M. Mahdian, and A. Saberi, *A new greedy approach for facility location problems*, Proceedings of the 34st Annual ACM Symposium on Theory of Computing (STOC), ACM, 2002, pp. 731–740.

[JNS00] J. Jansson, N. Nguyen, and W. Sung, *Algorithms for combining rooted triplets into a galled phylogenetic network*, SIAM Journal on Computing **35** (200), no. 5, 1098–1121.

[Joh73] D. S. Johnson, *Approximation algorithms for combinatorial problems*, Proceedings of the 5th Annual ACM Symposium on Theory of Computing (STOC), ACM, 1973, pp. 38–49.

[JS06] J. Jansson and W. Sung, *Inferring a level-1 phylogenetic network from a dense set of rooted triplets*, Theoretical Computer Science **363** (2006), no. 1, 60–68.

[JV00] K. Jain and V. V. Vazirani, *An approximation algorithm for the fault tolerant metric facility location problem*, Proceedings of the 3rd International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX), LNCS, vol. 1913, Springer, 2000, pp. 177–183.

[JV01] K. Jain and V. V. Vazirani, *Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation*, Journal of the ACM **48** (2001), 274–296.

[Kel] S. Kelk, `http://homepages.cwi.nl/~kelk/tripletverify/`.

[KGDO05] V. Kunin, L. Goldovsky, N. Darzentas, and C. A. Ouzounis, *The net of life: Reconstructing the microbial phylogenetic network*, Genome Research **15** (2005), 954–959.

[KH63] A. A. Kuehn and M. J. Hamburger, *A heuristic program for locating warehouses*, Management Science **9** (1963), 643–666.

[Kha79] L. Khachiyan, *A polynomial algorithm in linear programming (in russian)*, Doklady Akademii Nauk SSSR **244** (1979), 1093 – 1096, English translation: Soviet Mathematics Doklady 20 (1979) 191 – 194.

[Kho02] S. Khot, *On the power of unique 2-prover 1-round games*, Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC), ACM, 2002, pp. 767–775.

[KM00] D. Karger and M. Minkoff, *Building steiner trees with incomplete global knowledge*, Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, 2000, pp. 613–623.

[KMS07] C. Kenyon-Mathieu and W. Schudy, *How to rank with few errors*, Proceedings of the 39th Annual ACM Symposium on Theory of computing (STOC), ACM, 2007, pp. 95–103.

[KP83] J. Krarup and P. M. Pruzan, *The simple plant location problem: Survey and synthesis*, European Journal of Operational Research **12** (1983), 38–81.

[KP90] J. Krarup and P. M. Pruzan, *Ingredients of locational analysis*, Discrete Location Theory (P. Mirchandani and R. Francis, eds.), John Wiley & Sons, 1990, pp. 1–54.

[KPS06] S. Kontogiannis, P. Panagopoulou, and P. Spirakis, *Polynomial algorithms for approximating Nash equilibria of bimatrix games*, Proceedings of the 2nd Workshop on Internet and Network Economics (WINE), LNCS, vol. 4286, Springer, 2006, pp. 286–296.

[KS96] K. Kent and D. Skorin-Kapov, *Population monotonic cost allocation on MST's*, In Operational Reearch Proceedings KOI (1996), 43–48.

[KV05] S. Khot and N. K. Vishnoi, *The unique games conjecture, integrality gap for cut problems and embeddability of negative type metrics into l_1*, Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, 2005, pp. 53–62.

[Lev73] L. Levin, *Universal search problems (Russian: Universal'nye perebornye zadachi)*, Problems of Information Transmission (Russian: Problemy Peredachi Informatsii) **9** (1973), no. 3, 265 – 266, translated into English by B. A. Trakhtenbrot (1984). "A survey of Russian approaches to perebor (brute-force searches) algorithms". Annals of the History of Computing **6** (1984), no. 4, 384 – 400.

[LMM03] R. Lipton, E. Markakis, and A. Mehta, *Playing large games using simple strategies*, Proceedings of the 4th ACM Conference on Electronic Commerce (EC), ACM, 2003, pp. 36–41.

[Lov75] L. Lovász, *On the ratio of the optimal integral and fractional covers*, Discrete Mathematics **13** (1975), 383–390.

[LPR+07] A. Lozano, R. Y. Pinter, O. Rokhlenko, G. Valiente, and M. Ziv-Ukelson, *Seeded tree alignment and planar tanglegram layout*, Proceedings of the 7th International Workshop on Algorithms in Bioinformatics (WABI), LNCS, vol. 4645, Springer, 2007, pp. 98–110.

[LR04] C. R. Linder and L. H. Rieseberg, *Reconstructing patterns of reticulate evolution in plants*, Reconstructing patterns of reticulate evolution in plants **91** (2004), 1700–1708.

[LSS04] R. Levi, D. Shmoys, and C. Swamy, *Lp-based approximation algorithms for capacitated facility location*, Proceedings of the 10th Conference on Integer Programming and Combinatorial Optimization (IPCO), LNCS, vol. 3064, Springer, 2004, pp. 206–218.

[LV92] J. Lin and J. Vitter, *ϵ-approximations with minimum packing constraint violation*, Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC), ACM, 1992, pp. 771–782.

[Mah04] M. Mahdian, *Facility location and the analysis of algorithms through factor-revealing programs*, Ph.D. thesis, MIT, 2004.

[Man64] A. S. Manne, *Plant location under economies-of-scale – decentralization and computation*, Management Sciences **11** (1964), 213–235.

[Mar99] W. Martin, *Mosaic bacterial chromosomes: a challenge on route to a tree of genomes*, BioEssays **21** (1999), 99–104.

[Mey01] A. Meyerson, *Online facility location*, Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, 2001, pp. 426–431.

[MF90] P. B. Mirchandani and R. L. Francis (eds.), *Discrete location theory*, John Wiley & Sons, 1990.

[MP03] M. Mahdian and M. Pál, *Universal facility location*, Proceedings of the 11th Annual European Symposium on Algorithms (ESA), LNCS, vol. 2832, Springer, 2003, pp. 409–421.

[MR95] R. Motwani and P. Raghavan, *Randomized algorithms*, Cambridge University Press, 1995.

[MYZ02] M. Mahdian, Y. Ye, and J. Zhang, *Improved approximation algorithms for metric facility location problems*, Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX), LNCS, vol. 2462, Springer, 2002, pp. 229–242.

[MYZ03] M. Mahdian, Y. Ye, and J. Zhang, *A 2-approximation algorithm for the soft-capacitated facility location problem*, Proceedings of the 6th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX), LNCS, vol. 2764, Springer, 2003, pp. 149–162.

[MYZ06] M. Mahdian, Y. Ye, and J. Zhang, *Approximation algorithms for metric facility location problems*, SIAM Journal on Computing **36** (2006), no. 2, 411–432.

[Nag05] H. Nagamochi, *An improved bound on the one-sided minimum crossing number in two-layered drawings*, Discrete Computational Geometry **33** (2005), no. 4, 565–591.

[Nas51] J. F. Nash, *Non-cooperative games*, Annals of Mathematics **54** (1951), 286–295.

[NSW+03] L. Nakhleh, J. Sun, T. Warnow, C. Linder, B. Moret, and A. Tholse, *Towards the development of computational tools for evaluating phylogenetic network reconstruction methods*, Proceedings of Pacific Symposium on Biocomputing, vol. 8, 2003, pp. 315–326.

[NV01] A. Newman and S. Vempala, *Fences are futile: On relaxations for the linear ordering problem*, Proceedings of the 8th International Conference on Integer Programming and Combinatorial Optimization (IPCO), LNCS, vol. 2081, Springer, 2001, pp. 333–347.

[NW88] G. Nemhauser and L. Wolsey, *Integer and combinatorial optimization*, John Wiley & Sons, 1988.

[Pag02] R. D. M. Page (ed.), *Tangled trees: Phylogeny, cospeciation, and coevolution*, University of Chicago Press, 2002.

[Pap94] C. Papadimitriou, *On the complexity of the parity argument and other inefficient proofs of existence*, Journal of Computer and System Sciences **48** (1994), no. 3, 498–532.

[PTW01] M. Pál, E. Tardos, and T. Wexler, *Facility location with nonuniform hard capacities*, Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, 2001, pp. 329– 338.

[PY91] C. H. Papadimitriou and M. Yannakakis, *Optimization, approximation, and complexity classes*, Journal of Computer and System Sciences **43** (1991), 425–440.

[RAM] `http://en.wikipedia.org/wiki/Random_access_machine`.

[RL04] M. C. Rivera and J. A. Lake, *The ring of life provides evidence for a genome fusion origin of eukaryotes*, Nature **43** (2004), 152–155.

[RRR98] V. Raman, B. Ravikumar, and S. S. Rao, *A simplified NP-complete MAXSAT problem*, Information Processing Letters **65** (1998), 1–6.

[RS97] R. Raz and S. Safra, *A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP*, Prrroceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC), ACM, 1997, pp. 475–484.

[RS02] R. Ravi and A. Sinha, *Integrated logistics: Approximation algorithms combining facility location and network design*, Proceedings of the 9th Conference on Integer Programming and Combinatorial Optimization (IPCO), LNCS, vol. 2337, Springer, 2002, pp. 212–229.

[RS04] R. Ravi and A. Sinha, *Multicommodity facility location*, Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), SIAM, 2004, pp. 342–349.

[RS06] R. Ravi and A. Sinha, *Hedging uncertainty: Approximation algorithms for stochastic optimization problems*, Mathematical Programming **108** (2006), no. 1, 97–114.

[RV99] S. Rajagopalan and V. V. Vazirani, *Primal-dual RNC approximation algorithms for set cover and covering integer programs*, SIAM Journal on Computing **28** (1999), no. 2, 525–540.

[Sch86] A. Schrijver, *Theory of linear and integer programming*, John Wiley & Sons, 1986.

[Shm00] D. B. Shmoys, *Approximation algorithms for facility location problems*, Proceedings of the 3rd International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX), LNCS, vol. 1913, Springer, 2000, pp. 27–33.

[Shm04] D. B. Shmoys, *The design and analysis of approximation algorithms: Facility location as a case study*, Proceedings of Symposia in Applied Mathematics, vol. 61, AMS, 2004, pp. 85–97.

[SK04] C. Swamy and A. Kumar, *Primal-dual algorithms for connected facility location problems*, Algorithmica **40** (2004), no. 4, 245–269.

[ST06] Z. Svitkina and E. Tardos, *Facility location with hierarchical facility costs*, Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), SIAM, 2006, pp. 153–161.

[ST07] P. Spirakis and H. Tsaknakis, *An optimization approach for approximate Nash equilibria*, 3rd Workshop on Internet and Network Economics (WINE), LNCS, vol. 4858, Springer, 2007, pp. 42–56.

[STA97] D. B. Shmoys, E. Tardos, and K. Aardal, *Approximation algorithms for facility location problems*, Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC), ACM, 1997, pp. 265–274.

[Sto63] J. F. Stollsteimer, *A working model for plant numbers and locations*, Journal of Farm Economics **45** (1963), 631–645.

[STT81] K. Sugiyama, S. Tagawa, and M. Toda, *Methods for visual understanding of hierarchical system structures*, IEEE Transactions on Systems, Man, and Cybernetics **11** (1981), no. 2, 109–125.

[Svi02] M. Sviridenko, *An improved approximation algorithm for the metric uncapacitated facility location problem*, Proceedings of the 9th Conference on Integer Programming and Combinatorial Optimization (IPCO), LNCS, vol. 2337, Springer, 2002, pp. 240–257.

[Tur37] A. Turing, *On computable numbers, with an application to the entscheidungsproblem*, Proceedings of the London Mathematical Society, Series 2 **42** (1937), 115–154.

[Vaz01] V. V. Vazirani, *Approximation algorithms*, Springer-Verlag, 2001.

[vIKK+08] L. van Iersel, J. Keijsper, S. Kelk, L. Stougie, F. Hagen, and T. Boekhout, *Constructing level-2 phylogenetic networks from triplets*, Proceedings of the 12th Annual International Conference on Research

in Computational Molecular Biology (RECOMB), LNCS, vol. 4955, Springer, 2008, pp. 450–462.

[Vyg05] J. Vygen, *Approximation algorithms for facility location problems (lecture notes)*, Tech. Report 05950-OR, Research Institute for Discrete Mathematics, University of Bonn, 2005, Available at `http://www.or.uni-bonn.de/~vygen/fl.pdf`.

[Woe] G. Woeginger, `http://www.win.tue.nl/~gwoegi/P-versus-NP.htm`.

[Wu04] B. Wu, *Constructing the maximum consensus tree from rooted triples*, Journal of Combinatorial Optimization **8** (2004), 29–39.

[ZCY05] J. Zhang, B. Chen, and Y. Ye, *A multiexchange local search algorithm for the capacitated facility location problem*, Mathematics of Operations Research **30** (2005), no. 2, 389–403.

[Zha04] J. Zhang, *Approximating the two-level facility location problem via a quasi-greedy approach*, Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), SIAM, 2004, pp. 808–817.

[Zha06] J. Zhang, *Approximating the two-level facility location problem via a quasi-greedy approach*, Mathematical Programming, Ser. A **108** (2006), 159–176.

Summary

Randomized Approximation Algorithms:
Facility Location, Phylogenetic Networks, Nash Equilibria

In this thesis, we study combinatorial optimization problems, including metric Uncapacitated Facility Location (UFL), construction of phylogenetic networks, computation of Nash equilibria, and a generalization of the Set Cover problem.

The studied problems are known to be hard to solve optimally. An efficient computation of approximate solutions is a challenge. A number of constant factor approximation algorithms is provided in the thesis. The given algorithms, typically, improve the best known approximation ratio for a particular problem. Our main contributions are as follows.

The metric UFL problem is to decide about location of facilities that need to be opened to serve a given set of clients. The objective is to find a solution that minimizes the total cost of opening facilities and servicing clients. We provide a 1.5-approximation algorithm for the UFL problem, which has the currently best known approximation guarantee.

One of the central problems in computational biology is to construct models of the evolutionary development of species in the nature. In the simplest setting, we aim at reconstructing the evolutionary tree of a set of species, given information on how closely related certain pairs of species are. A natural algorithmic problem is to efficiently construct phylogenetic networks consistent with most of the provided information. Our main contribution to the field of phylogenetics, is an improved method for constructing approximate level-1 and level-2 phylogenetic networks. We also consider the problem of comparing trees. We propose algorithms that draw trees in the plane to facilitate a graphical comparison.

A central concept in the noncooperative game theory is the Nash equilibrium. An equilibrium is a set of strategies (one for each player) that are stable, i.e., no player has an incentive to unilaterally change his strategy. John Nash proved that for any finite noncooperative game there exist an equilibrium. One of the currently most fascinating open problems in theoretical computer science is to efficiently find such equilibria. The problem has been recently shown to be complete for the class PPAD even in the case of only two players. Therefore, a natural approach is to develop efficient algorithms that find approximate equilibria. In this thesis, we present an

LP-rounding algorithm that computes 0.364-approximate Nash equilibria.

The classic Set Cover problem of covering elements with a minimal number of subsets is known to be difficult to approximate. We study a generalization of the problem, where each of the elements needs to be covered a certain number of times. We provide an algorithm that, under the assumption that the level of coverage is at least logarithmic in the number of elements to cover, is a constant factor approximation algorithm for the problem.

Curriculum Vitae

Jaroslaw Byrka was born on the 30^{th} of May 1980 in Wroclaw, Poland. From 1999 to 2004 he studied Computer Science at the University of Wroclaw. During his studies he spent one semester in 2002 in the Department of Computer Science at the University of Paderborn, Germany; and one semester in 2003 in the AI group at the Autonomous University of Madrid, Spain.

In October 2004, he started as a Ph.D. student in the PNA1 group at the Centrum voor Wiskunde en Informatica (CWI) in Amsterdam within the ADONET Marie-Curie Research Training Network. From October 2007 he was a Ph.D. student in the Computer Science Department at the Eindhoven University of Technology (TU/e). Since June 2008 he works as a postdoc fellow in the Mathematics Department at TU/e within the EU project ARRIVAL.