**COMBINATORICA**
Bolyai Society – Springer-Verlag

COMBINATORICA **27** (1) (2007) 91–127

# FULLY-DYNAMIC MIN-CUT*

MIKKEL THORUP

Received June 22, 2001
Revised November 14, 2003

We show that we can maintain up to polylogarithmic edge connectivity for a fully-dynamic graph in $\tilde{O}(\sqrt{n})$ worst-case time per edge insertion or deletion. Within logarithmic factors, this matches the best time bound for 1-edge connectivity. Previously, no $o(n)$ bound was known for edge connectivity above 3, and even for 3-edge connectivity, the best update time was $O(n^{2/3})$, dating back to FOCS'92.

Our algorithm maintains a concrete min-cut in terms of a pointer to a tree spanning one side of the cut plus ability to list the cut edges in $O(\log n)$ time per edge.

By dealing with polylogarithmic edge connectivity, we immediately get a sampling based expected factor $(1 + o(1))$ approximation to general edge connectivity in $\tilde{O}(\sqrt{n})$ time per edge insertion or deletion. This algorithm also maintains a pointer to one side of a near-minimal cut, but if we want to list the cut edges in $O(\log n)$ time per edge, the update time increases to $\tilde{O}(\sqrt{m})$.

## 1. Introduction

We consider the problem of maintaining a min-cut of a fully-dynamic graph. Here by *fully-dynamic* we mean that the graph may be *updated* by insertion and deletion of edges. If we only allow insertions or only allow deletions, the graph is only *partially dynamic*. The updates are interspersed with *queries* to the current graph. The update and query *operations* are presented on-line, with no knowledge of future operations.

*Mathematics Subject Classification (2000):* 68Q25, 68W05, 68R10, 05C40, 05C85, 94C12, 94C15, 90B10, 90B25

* A preliminary version of this work was presented at the *The 33rd ACM Symposium on Theory of Computing (STOC)* [22], Crete, Greece, July 2001.

Like priority queues, dynamic graph algorithms may be both of direct interest, and of interest as data structures within algorithms for static problems. As an example of direct usage, Frederickson [9] suggests using his dynamic minimum spanning tree algorithm to tell how loaded links we need to use to get from one vertex to another in a dynamic communications network. As an example of usage as data structures, Biedl et al. [2] use the dynamic 2-edge connectivity algorithm of Holm et al. [13] to efficiently find the perfect matching of a bridgeless 3-regular graph asserted by Petersen's matching theorem [18]. The reader is directed elsewhere [5,8,23] for a more general introduction to the work on dynamic graph algorithms.

In this paper, we present a fully-dynamic graph algorithm for general edge connectivity and min-cut. Our goal is to maintain the *edge connectivity*, $\lambda$, which is the minimum number of edges whose removal disconnect the graph. These edges define a min-cut. Here, a *cut* is a partitioning of the vertex set into two sets, called *sides*. The *size of the cut* is the number of cross edges connecting the sides, and a *min-cut* is a cut of minimum size. Besides the edge connectivity, we wish to maintain some concrete min-cut $C$ in the sense that we want a pointer to a tree spanning one side of $C$, plus ability to check if any two vertices are on the same side of $C$ in $O(\log n)$ time, plus ability to list the cut edges in $O(\log n)$ time per edge. We note that the number of min-cuts may be quadratic, e.g., if the graph is a ring, but the ambition here is just to maintain a single min-cut $C$.

Edge connectivity is a basic sensitivity measure for graphs, and a min-cut points us to a concrete place where the graph is closest to falling apart. Karger [15] has shown that the static problem, finding the edge connectivity and some min-cut of a given graph, can be solved in near-linear expected time.

By a *$k$-edge connectivity algorithm*, we mean an algorithm that maintains if a graph is $k$-edge connected, that is, if its edge connectivity is at least $k$. For fully-dynamic 1-edge connectivity and 2-edge connectivity the best bound is $O(\sqrt{n})$ time per update, due to Eppstein et al. [6] by reduction to work of Frederickson [9,10]. This is a *worst-case* time bound satisfied for every single update, and it is such worst-case time bounds that are the focus of this paper. Faster amortized bounds will be discussed later. For 3-edge and 4-edge connectivity, the best time bounds are $O(n^{2/3})$ and $O(n\alpha(n))$ by Eppstein et al. [6]. For edge connectivity up to $k$, combining the sparsification Eppstein et al. [6], the certificates of Nagamochi and Ibaraki [16], and the Monte Carlo type randomized static min-cut algorithm of Karger [15] gives an $\tilde{O}(kn)$ expected time bound per update. Finally, a result of Thorup and Karger [23] implies a Monte Carlo type randomized factor $2+o(1)$ ap-

proximation to general edge connectivity in $\tilde{O}(\sqrt{n})$ time per update. This result does, however, not provide any small cut.

In this paper, we show that up to polylogarithmic edge connectivity can be maintained in $\tilde{O}(\sqrt{n})$ time per update. Within the same time bound, with high probability, we can maintain general edge connectivity within a factor $1 \pm o(1)$. The algorithms also maintain a concrete (approximate) min-cut. However, in the general approximate case, in order to list the cut edges in $O(\log n)$ time per edge, the algorithm has an increased update time of $\tilde{O}(\sqrt{m})$.

We note that if we allow amortization, that is, if we only care about the average update time, polylogarithmic time bounds are known for 1- and 2-edge connectivity. Here randomized bounds are due to Henzinger and King [12] and deterministic bounds due to Holm et al. [13]. Also, for the special case of planar graphs, Eppstein et al. [7] have shown that we can do 3-edge and 4-edge connectivity in $O(\sqrt{n})$ amortized time per operation. Finally, the above mentioned $2 + o(1)$ approximation of Karger and Thorup [23] is presented with polylogarithmic amortized bounds. We leave it as a major open problem to improve the time bounds of the current paper using amortization.

Concerning partially dynamic algorithms, the most general connectivity result by Dinitz and Nossenson [4] is that we can maintain 5-edge connectivity incrementally, ending with $m$ edges, in $O(m+n\log^2 n)$ total time. We show here how to maintain polylogarithmic edge connectivity incrementally (decrementally) ending (starting) with $m$ edges in $\tilde{O}(n^{3/2}+m)$ total time.

### 1.1. Techniques

The main obstacle in dealing with edge connectivity beyond 2 is that an arbitrary spanning tree may cross a min-cut many times. Karger [15] has pointed out that a certain Lagrangian greedy tree packing technique studied implicitly in [19,25] leads to trees crossing an arbitrary min-cut at most twice. This leads him to a static randomized near-linear time min-cut algorithm. Unfortunately, it seems very difficult to maintain the minimum cut crossed twice by a dynamic tree.

Our main technical contribution is to make a much more detailed analysis of the same greedy tree packing, showing that it leads to a spanning tree crossing some min-cut exactly once. Using this fact, we can then maintain a min-cut by creative combination of techniques from [1,6,9,10,23].

### 1.2. Preliminaries

We define *with high probability*, to mean with probability $1-1/n^{\omega(1)}$. For example, this definition implies that if $n$ events each succeed with high probability, they all succeed with high probability.

Let $G=(V,E)$, $V(G)=V$, and $E(G)=E$. Moreover, let $\lambda(G)$ be the edge connectivity $\lambda$ of $G$. Notationally, we will use '|' for restriction, '\' for subtraction, and '/' for contraction. Thus, if $U\subseteq V$, then $G|U$ is the subgraph of $G$ induced by $U$. Also, $G\backslash U$ is the graph we get from $G$ if we remove the vertices in $U$ and all their incident edges. Finally, if $\mathcal{P}$ partitions $V$, then $G/\mathcal{P}$ denotes $G$ with each set of $\mathcal{P}$ contracted. During the contraction we remember the original identity of the edges, so $E(G/\mathcal{P})$ may be viewed as the subset of the edges from $G$ that connect different sets in $\mathcal{P}$.

In most of this paper, the graph $G$ will be understood. If we want to consider a specific graph $H$, we will specify this with a subscript $_H$. For example, we will use $\lambda$ for the edge connectivity of the understood graph $G$ and $\lambda_H$ for the edge connectivity of the specific graph $H$.

As a special terminology of this paper, if $\mathcal{P}$ partitions $V$, then a *trivial* $\mathcal{P}$ *cut*, is a cut where one of the two sides is a set in $\mathcal{P}$.

## 2. Tree packings and edge connectivity

A *tree packing* of $G$ is a family $\mathcal{T}$ of spanning trees of $G$, allowing multiple occurrences of the same tree. It *loads* each edge $e\in E(G)$ with the number $L^{\mathcal{T}}(e)$ of trees containing $e$. The *relative load* is then $\ell^{\mathcal{T}}(e)=L^{\mathcal{T}}(e)/|\mathcal{T}|$. Hence $\ell^{\mathcal{T}}(e)$ is the average load of $e$ be the trees in $\mathcal{T}$. Having the reserved term "relative load" is convenient when we start using the relative edge load inside sums and averages over certain edges. If $D$ is a set of edges, we define the *total relative load* $\ell^{\mathcal{T}}(D)=\sum_{e\in D}\ell^{\mathcal{T}}(e)$.

The value of a packing is:

$$\mathsf{pack\_val}(\mathcal{T})=1/\max_{e\in E(G)}\ell^{\mathcal{T}}(e)$$

Note that if a packing consists of $t$ disjoint trees, its value is $t$.

A main goal of this paper is to construct a tree crossing some min-cut only once. To this end, we note

**Observation 1.** *If $C$ is a min-cut and $\mathcal{T}$ is a tree packing with* $\mathsf{pack\_val}(\mathcal{T})>\lambda/2$, *then $\mathcal{T}$ contains a tree crossing some min-cut at most once.*

**Proof.** The number of edges from $C$ used by an average tree in $\mathcal{T}$ is

$$\ell^{\mathcal{T}}(E(C)) = \sum_{e \in E(C)} \ell^{\mathcal{T}}(e) \leq |E(C)| \max_{e \in E(G)} \ell^{\mathcal{T}}(e) \tag{1}$$

$$= |E(C)| / \mathsf{pack\_val}(\mathcal{T}) < \lambda/(\lambda/2) = 2. \tag{2}$$

Consequently, some $T \in \mathcal{T}$ crosses $C$ strictly less than twice, and since the number of crossings is a positive integer for a given tree, this means that $T$ crosses $C$ at most once. ∎

For an algorithmic understanding of tree packings, we employ a dual concept for partitions. For a partition $\mathcal{P}$ of $V(G)$, we define its *partition value* as

$$\mathsf{part\_val}(\mathcal{P}) = \frac{|E(G/\mathcal{P})|}{|\mathcal{P}| - 1}$$

Note that if $\mathcal{P}$ has only two sets, then $\mathcal{P}$ is a cut and $\mathsf{part\_val}(\mathcal{P})$ the size of this cut. Since any spanning tree $T$ has at least $|\mathcal{P}|-1$ edges from $E(G/\mathcal{P})$, we get

$$\begin{aligned}
\mathsf{pack_val}(\mathcal{T}) &= 1/\max_{e \in E} \ell^{\mathcal{T}}(e) \\
&\leq |E(G/\mathcal{P})| \,/ \sum_{e \in E(G/\mathcal{P})} \ell^{\mathcal{T}}(e) \\
&\leq |E(G/\mathcal{P})| \,/ (|\mathcal{P}| - 1) \\
&= \mathsf{part_val}(\mathcal{P}).
\end{aligned}$$

Here the first inequality follows because the average is smaller than the maximum, and the second follows because any spanning tree loads at least $(|\mathcal{P}|-1)$ edges from $E(G/\mathcal{P})$.

Thus no tree packing value is bigger than any partition value. However, Tutte [24] and Nash-Williams [17] have shown that for any graph, there exists a tree packing and a partition of the same value, that is, we have the following duality result:

**Lemma 2 ([24,17]).** *The maximum value of a tree packing equals the minimum partition value of a partition.* ∎

The threshold between tree packing values and partition values is very important for this paper, and we denote it by $\Phi$, that is,

$$\Phi = \max_{\mathcal{T}} \mathsf{pack\_val}(\mathcal{T}) = \min_{\mathcal{P}} \mathsf{part\_val}(\mathcal{P})$$

From this equation, Karger [15] derived

**Lemma 3** ([15]). $\lambda/2 < \Phi \leq \lambda$.

**Proof.** First we note that $\Phi \leq \lambda$ because $\lambda$ is the minimum partition value over two set partitions. Second, consider an arbitrary partition $\mathcal{P}$. Then

$$\Phi \geq \mathsf{part\_val}(\mathcal{P}) > |E(G/\mathcal{P})|/|\mathcal{P}| \geq \lambda/2. \tag{3}$$

The last relation follows because $2|E(G/\mathcal{P})|/|\mathcal{P}|$ is the average value of a trivial $\mathcal{P}$ cut, and all cuts have value at least $\lambda$. ∎

Combining Observation 1 and Lemma 3, we get:

**Corollary 4.** *If $C$ is a min-cut and $\mathcal{T}$ is an optimal tree packing, that is,* $\mathsf{pack\_val}(\mathcal{T}) = \Phi$*, then $\mathcal{T}$ contains a tree crossing some min-cut at most once.* ∎

## 2.1. Greedy tree packings

A tree packing $\mathcal{T} = \{T_1, \ldots, T_k\}$ is *greedy* if each $T_i$ is a minimal spanning tree with respect to the loads induced by $\{T_1, \ldots, T_{i-1}\}$. As mentioned in [23], the work of Plotkin et al. [19] and Young [25] implies

**Lemma 5** ([19,25]). *A greedy tree packing $\mathcal{T}$ with $\geq 3\lambda \ln m/\varepsilon^2$ trees has* $\mathsf{pack\_val}(\mathcal{T}) \geq (1-\varepsilon)\Phi$. ∎

In combination with Lemma 3, we get

$$(1-\varepsilon)\lambda/2 < (1-\varepsilon)\Phi \leq \mathsf{pack\_val}(\mathcal{T}) \leq \Phi \leq \lambda. \tag{4}$$

## 2.2. Karger's near-linear time min-cut algorithm

To appreciate greedy tree packings in the context of min-cut algorithms, below we review a high-level version of Karger's [15] static near-linear time min-cut algorithm. Our version is simplified and slower by several log-factors. Our goal is to present some of the basic ideas to be used in our own dynamic min-cut algorithm. For example, we ignore his version based on directed spanning trees because we do not know how to maintain directed spanning trees dynamically.

Using (4) in the averaging of (1), Karger argues that near-minimal cuts are crossed at most twice by a constant fraction of the trees in a greedy tree packing. More precisely,

**Lemma 6** ([15]). *Let $C$ be any cut with $<1.1\lambda$ edges and let $\mathcal{T}$ be a greedy tree packing with $\omega(\lambda \ln m)$ trees. Then a fraction 1/3 of the trees in $\mathcal{T}$ cross $C$ at most twice.* ∎

Our first goal is to find min-cuts of size up to some polylogarithmic $\lambda_{\max}$. A greedy tree packing $\mathcal{T}$ is produced in $\tilde{O}(m|\mathcal{T}|)$ time. To apply Lemma 6, we pack $|\mathcal{T}| = \lambda_{\max} \log^{1.1} n$ trees in $\tilde{O}(m)$ time.

Karger [15] shows that in $O(m \log n)$ time we can find the smallest cut crossed at most twice by a given tree $T$. We do this for all trees in $\tilde{O}(m)$ time, returning the overall smallest cut $C$. By Lemma 6, if $\lambda \leq \lambda_{\max}$, $C$ is a min-cut. Otherwise, $C$ has more than $\lambda_{\max}$ cross edges, and we report that the edge connectivity is above $\lambda_{\max}$.

For higher edge connectivity, Karger applies one of his previous results:

**Lemma 7** ([14]). *Let $p$ be a probability and $H = G_p$ be a random subgraph of $G$ including each edge independently with probability $p$. Let $\lambda_H$ be the edge connectivity of $H$. Suppose $p\lambda = \omega(\log n)$. Then, w.h.p., $\lambda_H = (1 \pm o(1))p\lambda$. Moreover, w.h.p., min-cuts of $G$ are near-minimal in $H$ and vice versa. More precisely, a min-cut $C$ of $G$ has $(1+o(1))\lambda_H$ cross edges in $H$. Conversely, a min-cut $C_H$ of $H$ has $(1+o(1))\lambda$ cross edges in $G$.* ∎

In a Monte-Carlo type randomized algorithm, for $i = 0, \ldots, \lfloor \log n \rfloor$, we pick $H_i = G_{1/2^i}$ and construct a greedy tree packing $\mathcal{T}_{H_i}$ of $H_i$ with $\Theta(\log^{2.2} n)$ trees. For all the trees $T$ in all the tree packings $\mathcal{T}_{H_i}$, we find the smallest cut in $G$ crossed at most twice by $T$, and return overall smallest such cut.

The running time of the above algorithm is near-linear. For the correctness we need:

**Lemma 8.** *The above randomized algorithm returns a min-cut with high probability.*

**Proof.** Even though we do not know $\lambda$, we know that there is a value of $i$ such that $\lambda/2^i = \Theta(\log^{1.1} n)$. Fix this value of $i$. By Lemma 7, w.h.p., the edge connectivity of $\lambda_{H_i}$ of $H_i$ is $\Theta(\log^{1.1} n)$, so $\mathcal{T}_{H_i}$ contains $\Theta(\log^{1.1} n)\lambda_{H_i}$ trees. Also, w.h.p., any min-cut $C$ of $G$ has $(1+o(1))\lambda_H$ edges. Then, by Lemma 6, $\mathcal{T}_{H_i}$ contains a tree $T$ crossing $C$ at most twice. Consequently, w.h.p., the min-cut $C$ is considered in the minimization of the algorithm. ∎

Karger's algorithm [15] that for a given tree $T$ computes a smallest cut in $G$ crossed at most twice by $T$ is rather complicated. We do not see any way to maintain the smallest such cut dynamically as $G$ and $T$ changes.

### 2.3. Why cross twice?

In Karger's algorithm, he uses Lemma 6, which states that any given near-minimal cut $C$ can be found considering cuts crossed twice by some tree in a greedy tree packing. It is natural to ask if it would suffice to consider cuts crossed only once by some tree in the greedy tree packing.

By Lemma 3, we have $\Phi > \lambda/2$. Hence, by Lemma 5, for a sufficiently large tree packing $\mathcal{T}$, we have $\mathsf{pack\_val}(\mathcal{T}) > \lambda/2$. Then by Observation 1, any min-cut is crossed once by some tree in $\mathcal{T}$.

Unfortunately, we may need a very large tree packing. To see this, consider the cycle $C_n$ over $n$ vertices. It has edge connectivity $\lambda_{C_n} = 2$. Any tree packing $\mathcal{T}_{C_n}$ of $C_n$ consists of paths $Q$ obtained from $C_n$ removing a single edge. With $|\mathcal{T}_{C_n}| < n$, there will be some edge $e$ used by all $Q \in \mathcal{T}_{C_n}$, hence with relative load $\ell^{\mathcal{T}_{C_n}}(e) = 1$. Thus $\mathsf{pack\_val}(\mathcal{T}_{C_n}) = \lambda_{C_n}/2$ if $|\mathcal{T}_{C_n}| < n$. Consequently, we cannot hope to use Observation 1 to get a min-cut crossed once by a tree.

A more fundamental obstacle is as follows. Suppose $|\mathcal{T}_{C_n}| \leq n-2$. We can then find two edges $e_1$ and $e_2$ that are not excluded by any path $Q \in \mathcal{T}_{C_n}$. However, $e_1$ and $e_2$ are the cross edges of some min-cut $C$, and then this min-cut is crossed twice by all $Q \in \mathcal{T}_{C_n}$. Thus we cannot hope to improve Lemma 6 to only consider cuts crossed once by some tree and yet cover all minimal cuts $C$.

## 3. Our new contribution

Our main technical result is that for an appropriately large greedy tree packing, there exists a min-cut crossed once by one of the trees. Not having to consider cuts crossed twice by tree will allow us to develop an efficient fully-dynamic min-cut algorithm. More precisely, our main technical contribution is the following purely combinatorial result:

**Theorem 9.** *A greedy tree packing with $\omega(\lambda^7 \log^3 m)$ trees contains a tree crossing some min-cut only once.*

This theorem will be proved in § 4. Before discussing a fully-dynamic min-cut algorithm derived from Theorem 9, we relate the theorem to the previous construction of Karger.

### 3.1. New sequential algorithms

From an algorithmic perspective, we get a new simpler static algorithm for edge connectivity below some polylogarithmic $\lambda_{\max}$. To apply Theorem 9,

we pack $\lambda_{\max}^7 \log^{3.1} n$ trees in $\tilde{O}(m)$ time. Now, for each tree $T$ in our tree packing, we just have to find the smallest cut crossed once by $T$. As described in [15], for a given $T$, this only takes $O(m)$ time, and is much simpler than finding the smallest cut crossed twice by $T$. However, the overall running time becomes slower by several log-factors because we have to consider many more trees.

For higher edge connectivity, we get a corresponding randomized algorithm for near-minimal cuts. For $i = 0, \ldots, \lfloor \log n \rfloor$, we pick $H_i = G_{1/2^i}$ and maintain a min-cut up to size $\log^{1.1} n$ of $H_i$. Let $j$ be minimal such that $H_j$ has edge connectivity below $\log^{1.1} n$. Then we approximate the edge connectivity of $G$ as $\lambda_{H_j} 2^j$. Also, we return a min-cut of $H_j$ as a near-minimal cut of $G$.

**Lemma 10.** *With high probability, the above randomized algorithm approximates the edge connectivity of $G$ within a factor $1 \pm o(1)$. Also, the returned min-cut of $H_j$ has $(1+o(1))\lambda$ cross edges in $G$.*

**Proof.** If $\lambda < \log^{1.1} n$, we get the exact answers from $H_0 = G$. Otherwise, there is an $i$ such that $\lambda/2^i = [1/3 \log^{1.1} n, 2/3 \log^{1.1} n)$. By Lemma 7, w.h.p., $\lambda_{H_i} < \log^{1.1} n$, so $j \leq i$. But then $\lambda/2^j \geq 1/3 \log^{1.1} n$. Hence by Lemma 7, w.h.p., $\lambda = (1 \pm o(1)) 2^j \lambda_{H_j}$ and the min-cuts of $H_j$ have $(1+o(1))\lambda$ cross edges in $G$. ∎

### 3.2. Fully-dynamic min-cut

The strength of the above approach over the original construction of Karger is that it can be implemented dynamically, both the tree packings and the cuts crossed once by a tree in the packing. This will be demonstrated in § 5. Putting everything together in § 6, we will show:

**Theorem 11.** *In $\tilde{O}(\sqrt{n})$ time per edge insertion or deletion in a fully-dynamic graph, we can maintain a min-cut of up to polylogarithmic size, and report if no such cut exists.*

*Within the same time bound, with high probability, for arbitrary edge connectivity $\lambda$, we can maintain a near-minimal cut of size $(1+o(1))\lambda$. However, if we want to list the cut edges in $O(\log n)$ time per edge, we increase the update time from $\tilde{O}(\sqrt{n})$ to $\tilde{O}(\sqrt{m})$.*

Improved amortized bounds for the partially dynamic case will be discussed in § 6.1.

## 4. Proof of Theorem 9

We want to show that a sufficiently large tree packing is going to contain a tree crossing some min-cut only once.

### 4.1. Idealized proof and outline

To illustrate our basic idea, we first study an optimal tree packing $\mathcal{T}^*$ together with an optimal partition $\mathcal{P}^*$, i.e., with $\Phi = \mathsf{part\_val}(\mathcal{P}^*) = \mathsf{pack\_val}(\mathcal{T}^*)$.

From Corollary 4, we know that any min-cut is crossed once by some tree in $\mathcal{T}^*$, but in § 2.3 we saw that his property does not hold for greedy tree packings of sub-linear size.

Below, we prove a weaker result for $\mathcal{T}^*$; namely that it contains a tree crossing some min-cut only once. We present this proof because it illustrates the basic ideas used later in a much more complicated proof of the same statement for a greedy tree packing $\mathcal{T}$ of size $(\lambda \log n)^{O(1)}$. As a first step in the proof, we note

**Lemma 12.** *If $\Phi = \mathsf{part\_val}(\mathcal{P}^*) = \mathsf{pack\_val}(\mathcal{T}^*)$, all edges $e$ in $G/\mathcal{P}^*$ have $\ell^{\mathcal{T}^*}(e) = 1/\Phi$. Moreover, $T/\mathcal{P}^*$ is a tree for all $T \in \mathcal{T}$.*

**Proof.** Since each $T \in \mathcal{T}$ spans $G$, it contains at least $|\mathcal{P}^*| - 1$ edges from $E(G/\mathcal{P}^*)$, so the average relative load over $E(G/\mathcal{P}^*)$ is at least $(|\mathcal{P}^*| - 1)/|E(G/\mathcal{P}^*)| = 1/\Phi$, which is also the maximal relative load. It follows that all relative loads in $E(G/\mathcal{P}^*)$ must be the same, and that no $T$ can contain more than $|\mathcal{P}^*| - 1$ edges from $E(G/\mathcal{P}^*)$. ∎

Using Lemma 12, we wish to show that there is a $T \in \mathcal{T}^*$ crossing some min-cut once. If all trivial $\mathcal{P}^*$ cuts are min-cuts, we take any tree $T \in \mathcal{T}^*$. By Lemma 12, $T/\mathcal{P}^*$ is a tree over $G/\mathcal{P}^*$, and any leaf edge of $T/\mathcal{P}^*$ is the unique edge in $T$ crossing the trivial $\mathcal{P}^*$ cut defined by the leaf. Hence we may assume that some trivial $\mathcal{P}^*$ cut is not a min-cut.

Let $T \in \mathcal{T}^*$. By Lemma 12, $T/\mathcal{P}^*$ is a tree, hence with average degree $2(|\mathcal{P}^*|-1)/|\mathcal{P}^*| < 2$. Thus, on the average, a trivial $\mathcal{P}^*$ cut is crossed less than twice by $T$. Suppose for a contradiction that $T$ crosses all min-cuts twice. This means that on the average, a trivial $\mathcal{P}^*$ cut which is not a min-cut is crossed less than twice.

The above argument holds for all $T \in \mathcal{T}^*$. Consequently, on the average, a trivial $\mathcal{P}^*$ cut $B$ which is not a min-cut has a total relative load $\ell^{\mathcal{T}^*}(B) < 2$. In particular, we can find a concrete such cut $B$. Since $B$ is not minimal, it has

at least $\lambda$+1 edges. Hence some edge $f \in B$ has relative load $\ell^{\mathcal{T}^*}(f) < 2/(\lambda+1)$. By Lemma 12, we have $\ell^{\mathcal{T}^*}(f) = 1/\Phi = \max_{e \in E} \ell^{\mathcal{T}}(e)^*$, so we conclude that the maximal relative load is less than $2/(\lambda+1)$.

On the other hand, if we take any min-cut $C$, since all trees cross it twice, there is an edge $e \in C$ with $\ell^{\mathcal{T}^*}(e) \geq 2/\lambda$. This contradicts that the maximal relative load is less than $2/(\lambda+1)$.

This completes our proof that some $T \in \mathcal{T}^*$ crosses some min-cut once. Below we will provide a similar but more complicated argument for the same statement for a greedy tree packing of size $(\lambda \log n)^{O(1)}$, thus proving Theorem 9. For the proof, we will first define some ideal relative edge loads. Next we prove that the relative edge loads of any concrete large greedy tree packing approximate the ideal relative loads. Using this approximation, we can mimic the above proof for the optimal tree packing $\mathcal{T}^*$ in a real proof of Theorem 9.

## 4.2. Ideal relative loads

Consider the following abstract recursive algorithm:

*Algorithm.* Assigns *ideal relative loads* $\ell^*(e)$ to the edges in $G$.

1. Let $\mathcal{P}^*$ be a partitioning of $V(G)$ with $\mathsf{pack\_val}(\mathcal{P}^*) = \Phi$ (c.f. Theorem 2).
2. For all $e \in E(G/\mathcal{P}^*)$, set $\ell^*(e) = 1/\Phi$.
3. For each $S \in \mathcal{P}^*$, recurse on the subgraph $G|S$ induced by $S$.

**Lemma 13.** *There is a distribution $\Pi$ of spanning trees of $G$ such that for each $e$,*

$$\Pr_{R \in \Pi}(e \in R) = \ell^*(e).$$

**Proof.** The distribution $\Pi$ is constructed in conjunction with the above recursive algorithm. Let $\mathcal{T}^*$ be an optimal tree packing for $G$. To pick a random tree $R$ from $\Pi_G$, we pick a uniformly random tree $U \in \mathcal{T}^*$, plus, for each $S \in \mathcal{P}^*$, a random $T_S \in \Pi_{G|S}$. Here $\Pi_{G|S}$ is the recursively defined distribution for $G|S$. We then construct $R$ as the union of the spanning trees $T_S$ of each set $S \in \mathcal{P}^*$ together with the edges from $U$ between different sets in $\mathcal{P}^*$. Here we note that $U/\mathcal{P}^*$ is a tree by Lemma 12, so $T$ is indeed a spanning tree. Further, for $e \in E(G/\mathcal{P})$, $\Pr(e \in U) = \Pr(e \in T) = \ell^{\mathcal{T}^*}(e) = 1/\Phi = \ell^*(e)$, as desired. For edges $e \in E(G|S)$, $\Pr(e \in T) = \Pr(e \in T_S)$, and by induction, this probability equals $\ell^*(e)$. ∎

**Lemma 14.** *The values of $\Phi$ are non-decreasing in the sense that for each $S \in \mathcal{P}^*$, $\Phi_{G|S} \geq \Phi$.*

**Proof.** Suppose for a contradiction that for some $G|S$ we have a partition $\mathcal{P}_S$ with $\mathsf{part\_val}_{G|S}(\mathcal{P}_S) < \Phi$. Then $\mathcal{P}' = (\mathcal{P}^* \setminus \{S\}) \cup \mathcal{P}_S$ would have $\mathsf{part\_val}(\mathcal{P}') < \Phi$, contradicting that $\Phi$ is the smallest partition value. More precisely, we have

$$\mathsf{part\_val}(\mathcal{P}') = \frac{|E(G/\mathcal{P}')|}{|\mathcal{P}'|-1} = \frac{|E(G/\mathcal{P}^*)| + |E((G|S)/\mathcal{P}_S)|}{|\mathcal{P}^*|-1+|\mathcal{P}_S|-1}.$$

This value is below $\Phi$ when $\mathsf{part\_val}_{G|S}(\mathcal{P}_S) = \frac{|E((G|S)/\mathcal{P}_S)|}{|\mathcal{P}_S|-1}$ is below $\Phi = \mathsf{part\_val}(\mathcal{P}^*) = \frac{|E(G/\mathcal{P}^*)|}{|\mathcal{P}^*|-1}$. ∎

**Lemma 15.** *Each tree $R$ in the distribution $\Pi_G$ from Lemma 13 is a minimum spanning tree with respect to the ideal relative loads $\ell^*(\cdot)$.*

**Proof.** From Lemma 14 it follows that the ideal relative loads are non-increasing as we recurse down on $G|S$, $S \in \mathcal{P}^*$. Moreover, inductively, we know that $R|S = T_S$ is a minimum spanning tree of $G|S$. Hence it follows that $R$ is a minimum spanning tree. ∎

It is convenient to introduce the following notation: for $\circ = <, >, \leq, \geq, =$ and $X = \mathcal{T}, *$, define

$$E^X_{\circ\delta} = \{e \in E | \ell^X(e) \circ \delta\}.$$

For example, $E^{\mathcal{T}}_{\leq\delta} = \{e \in E | \ell^{\mathcal{T}}(e) \leq \delta\}$. Similarly, $E^*_{=\delta} = \{e \in E | \ell^*(e) = \delta\}$.

### 4.3. Concrete relative loads approaching the ideal loads

In this subsection, we will study a concrete greedy tree packing $\mathcal{T}$. We will refer to its relative loads $\ell^{\mathcal{T}}(e)$ as *concrete* relative loads as opposed to the *ideal* relative loads $\ell^*(e)$. We will show that if $\mathcal{T}$ is sufficiently large, the concrete relative loads approximate the ideal relative loads. The result is a natural generalization of Lemma 5:

**Proposition 16.** *A greedy tree packing $\mathcal{T}$ with $t \geq 6\lambda \ln m/\varepsilon^2$ trees, $\varepsilon < 2$, has*

$$|\ell^{\mathcal{T}}(e) - \ell^*(e)| \leq \varepsilon/\lambda \tag{5}$$

*for all $e \in E(G)$.*

**Proof.** In the proof, we fix $\overline{\ell} = 2/\lambda$ and $\epsilon = \varepsilon/2 < 1$. Then the right-hand side of (5) is $\varepsilon/\lambda = \epsilon\overline{\ell}$. Moreover, by Lemma 3, the maximal ideal load is bounded by $\max_{f \in E} \ell^*(f) = 1/\Phi < 2/\lambda = \overline{\ell}$.

The proof uses the general packing and covering ideas from Young [25]. The basic idea is this. We consider the distribution $\Pi$ of spanning trees from Lemma 13. Then

$$\Pr_{T\in\Pi}(e\in T)=\ell^*(e).$$

Hence, picking $\mathcal{T}$ at random from $\Pi$, for each edge $e\in E$, we would have $\mathrm{E}[\ell^{\mathcal{T}}(e)]=\ell^*(e)$. Then, using standard Chernoff bounds (see, e.g., [3]), one can show that the expected number of violations to (5) is below 1. The proof uses a certain pessimistic estimator for the number of violations to (5). As the real greedy tree packing moves on, it chooses concrete trees for the packing. With each greedy choice, we revise the pessimistic estimator, conditioning it on the current choices but assuming that the remaining choices are random from $\Pi$. It turns out that greedy choices cannot increase the pessimistic estimator. When the greedy tree packing is complete with $t$ trees, there are no random choices left. Then an expected number of violation below 1 implies that there are no violations. Hence we end up concluding that (5) is satisfied for the complete greedy tree packing.

In order to present the basic technique, we first show a simpler statement which is similar to Lemma 5.

**Claim 16.1.** *For all edges $e\in E$,*

$$\ell^{\mathcal{T}}(e)\le\ell+\epsilon\overline{\ell} \tag{6}$$

*where $\ell=\max_{f\in E}\ell^*(f)=1/\Phi$.*

**Proof.** For the analysis of our greedy tree packing, let $\mathcal{T}$ be the set of trees packed so far. We are going to pack $t=6\lambda\ln m/\varepsilon^2$ trees total. Our analysis will be made in terms of the following quantity:

$$\sum_{e\in E}\frac{(1+\epsilon)^{L^{\mathcal{T}}(e)}(1+\epsilon\ell)^{t-|\mathcal{T}|}}{(1+\epsilon)^{\ell t+\epsilon\overline{\ell}t}} \tag{7}$$

The intuition behind (7) is that it is a Chernoff bound style pessimistic estimator of the number of violations to (6) if the remaining $|\mathcal{T}|-t$ trees were picked randomly from $\Pi$. For our analysis, we need to prove that (7) has the following three properties:

(a) When we start with $\mathcal{T}=\emptyset$, (7) is strictly below 1,
(b) when done with $|\mathcal{T}|=t$, (7) strictly below 1 implies that (6) is satisfied for all $e\in E$, and
(c) when we greedily add a tree to $\mathcal{T}$, (7) cannot increase.

Clearly (a), (b), and (c) imply (6) for all $e\in E$. We will now prove each of these statements.

*(a)* For $\mathcal{T}=\emptyset$, (7) becomes

$$\sum_{e\in E}\frac{(1+\epsilon\ell)^t}{(1+\epsilon)^{\ell t+\epsilon\overline{\ell}t}} < m\frac{e^{\epsilon\ell t}}{(1+\epsilon)^{\ell t+\epsilon\overline{\ell}t}} < m\frac{e^{\epsilon\overline{\ell}t}}{(1+\epsilon)^{\overline{\ell}t+\epsilon\overline{\ell}t}}$$
$$< me^{-\epsilon^2\overline{\ell}t/3} = me^{-(\varepsilon/2)^2(2/\lambda)(6\lambda\ln m/\varepsilon^2)/3} = 1.$$

Above, for the second inequality, we use that $1+\epsilon<e^\epsilon$ and $\ell<\overline{\ell}$, hence that $\left(\frac{e^\epsilon}{1+\epsilon}\right)^{\ell t}<\left(\frac{e^\epsilon}{1+\epsilon}\right)^{\overline{\ell}t}$.

*(b)* For $|\mathcal{T}|=t$, (7) becomes

$$\sum_{e\in E}(1+\epsilon)^{L^{\mathcal{T}}(e)-(\ell t+\epsilon\overline{\ell}t)}.$$

If $L^{\mathcal{T}}(e)\geq \ell t+\epsilon\overline{\ell}t$ for some $e$, the term for $e$ would be at least 1, and hence the sum would be at least 1. Hence, if (7) is below 1 when we finish, $L^{\mathcal{T}}(e)<\ell t+\epsilon\overline{\ell}t$ for all $e$.

*(c)* When adding a tree $U$ to $\mathcal{T}$, we study the quantity

$$q(U)=\sum_{e\in E}(1+\epsilon)^{L^{\mathcal{T}\cup\{U\}}(e)}.$$

We want to show that (7) is not increased when a greedy tree $U$ is added to $\mathcal{T}$. Since (7) is proportional to $\sum_{e\in E}(1+\epsilon)^{L^{\mathcal{T}}(e)}(1+\epsilon\ell)^{t-|\mathcal{T}|}$, this is equivalent to showing

$$q(U)\leq\sum_{e\in E}(1+\epsilon)^{L^{\mathcal{T}}(e)}(1+\epsilon\ell).$$

With a random tree $R\in\Pi$, we would get an expected quantity of

$$\mathrm{E}_{R\in\Pi}\left[q(R)\right]=\sum_{e\in E}\left((1+\epsilon)^{L^{\mathcal{T}}(e)}(1+\epsilon\Pr_{R\in\Pi}\left[e\in R\right])\right).$$

Here $\Pr_{R\in\Pi}[e\in R]=\ell^*(e)\leq\ell$, so we get

$$\mathrm{E}_{R\in\Pi}\left[q(R)\right]\leq\sum_{e\in E}(1+\epsilon)^{L^{\mathcal{T}}(e)}(1+\epsilon\ell).$$

Thus it suffices to show that a greedy tree $U$ has a smaller quantity than any tree $R\in\Pi$. Note that $q(U)=\sum_{e\in E}(1+\epsilon)^{L^{\mathcal{T}}(e)}+\epsilon\sum_{e\in U}(1+\epsilon)^{L^{\mathcal{T}}(e)}$. Hence

our quantity $q(U)$ grows with $\sum_{e\in U}(1+\epsilon)^{L^{\mathcal{T}}(e)}$. Thus (c) follows if we can show that

$$c(U) \le \min_{R\in\Pi} c(R) \text{ where } c(S) = \sum_{e\in S}(1+\epsilon)^{L^{\mathcal{T}}(e)}. \tag{8}$$

Since all $R \in \Pi$ are spanning trees, (8) is satisfied by a spanning tree $U$ minimizing $c(U)$, i.e., a minimum spanning tree where the weight of an edge $e$ is $(1+\epsilon)^{L^{\mathcal{T}}(e)}$. Now, for the choice of a minimum spanning tree, all that matters is the relative ordering of the edge weights, and clearly this ordering is preserved if we replace $(1+\epsilon)^{L^{\mathcal{T}}(e)}$ by $L^{\mathcal{T}}(e)$. Since our greedy tree packing does pick a minimal spanning tree $U$ with respect to the latter weights, it follows that (8), and hence (c) is satisfied. Having proved (a), (b), and (c), we conclude that (6) is satisfied for all $e\in E$. ∎

We are now ready to show the upper bound on $\ell^{\mathcal{T}}(e)$ in (5), that is,

**Claim 16.2.** *For all edges $e\in E$, we have $\ell^{\mathcal{T}}(e)\le \ell^*(e)+\epsilon\overline{\ell}$.*

**Proof.** To prove the claim, we consider an arbitrary $\ell\le 1/\Phi$. For all $e\in E^*_{\le\ell}$, we will show that (6) is satisfied, that is, $\ell^{\mathcal{T}}(e)\le\ell+\epsilon\overline{\ell}$. We recall that $e\in E^*_{\le\ell}$ means that $\ell^*(e)\le\ell$. Claim 16.1 handled the special case where $\ell=1/\Phi$.

Consider an arbitrary component $C$ in the subgraph with edge set $E^*_{\le\ell}$. We wish to apply our previous argument, but restricting our attention to the edges in $C$. That is, we are considering the packing $\mathcal{T}|C$ and the probability distribution $\Pi|C$, where $\cdot|C$ denotes that each tree is restricted to edges in $C$. Note that such a restricted tree may be disconnected and hence not span $C$. We also restrict the estimator (7) to the edges in $C$:

$$\sum_{e\in C} \frac{(1+\epsilon)^{L^{\mathcal{T}}(e)}(1+\epsilon\ell)^{t-|\mathcal{T}|}}{(1+\epsilon)^{\ell t+\epsilon\overline{\ell}t}}$$

The only non-trivial change is in the proof of (8), i.e., that

$$c(U|C) \le \min_{R\in\Pi} c(R|C).$$

By Lemma 15, each $R\in\Pi$ is a minimum spanning tree with respect to the ideal relative loads. Moreover, $C$ is spanned by edges whose ideal relative loads are strictly smaller than those of any edge leaving $C$. Hence $R|C$ is a spanning tree of $C$, that is, $\Pi|C$ is a distribution of spanning trees of $C$.

On the other hand, $U$ is a minimum spanning tree of $G$ with respect to the loads $L^{\mathcal{T}}(\cdot)$ defining our tree cost $c(\cdot)$. We note here that $U|C$ may not span $C$; for when it comes to the loads $L^{\mathcal{T}}(\cdot)$, there may by lighter edges

leaving $C$ than those spanning $C$. Nevertheless, $U|C$ must be contained in some minimum spanning tree $T_C$ of $C$ with respect to the loads $L^{\mathcal{T}}(\cdot)$, and then $c(U|C) \le c(T_C) \le \min_{R \in \Pi} c(R|C)$, as required. Thus we conclude that (6) is satisfied for all edges in the component $C$, and all edges in $E^*_{\le \ell}$ are in one such component.

Finally, for every edge $e$ in the graph, we use (6) with $\ell = \ell^*(e)$ to conclude that $\ell^{\mathcal{T}}(e) \le \ell^*(e) + \epsilon\overline{\ell}$. ∎

Finally, we need to show the lower bound on $\ell^{\mathcal{T}}(e)$ in (5), that is,

**Claim 16.3.** *For all edges $e \in E$, we have $\ell^{\mathcal{T}}(e) \le \ell^*(e) - \epsilon\overline{\ell}$.*

**Proof.** As in the proof of Claim 16.2, we pick an arbitrary $\ell \le 1/\Phi < \overline{\ell}$, and show that

$$\ell^{\mathcal{T}}(e) \ge \ell - \epsilon\overline{\ell} \tag{9}$$

is satisfied for all $e \in E^*_{\ge \ell} = E(G/E_{<\ell})$. The proof is a kind of dual to the upper bound proofs for Claims 16.1 and 16.2. In some parts, we simply replace $\epsilon$ by $-\epsilon$. In particular, our pessimistic estimator becomes

$$\sum_{e \in E^*_{\ge \ell}} \frac{(1-\epsilon)^{L^{\mathcal{T}}(e)}(1-\epsilon\ell)^{t-|\mathcal{T}|}}{(1-\epsilon)^{\ell t - \epsilon\overline{\ell} t}}. \tag{10}$$

However, there will also be combinatorial changes in the proof. More precisely, the proof of Claim 16.2 used that if we restrict a minimum spanning tree $U$ to any connected subgraph $C$, the result $U|C$ is contained in a minimum spanning tree of $C$. Below, instead we contract a set of edges $D$, and use that $U/D$ contains a minimum spanning tree of $G/D$.

Corresponding to the previous conditions (a), (b), and (c), we have

($a^-$) when we start with $\mathcal{T} = \emptyset$, (10) is strictly below 1,
($b^-$) when done with $|\mathcal{T}| = t$, (10) strictly below 1 implies that (9) is satisfied for all $e \in E^*_{\ge \ell}$, and
($c^-$) when we greedily add a tree to $\mathcal{T}$, (10) cannot increase.

The above three statements imply (9). The calculation proving (a) can be used directly to prove ($a^-$). We just have to note that the calculation also holds when $\epsilon$ is negative and $e$ is restricted to any subset of $E$. For ($b^-$) the proof is also almost identical to that for (b). For $|\mathcal{T}| = t$, (10) becomes $\sum_{e \in E^*_{\ge \ell}} (1-\epsilon)^{L^{\mathcal{T}}(e) - (\ell t - \epsilon\overline{\ell} t)}$. If $L^{\mathcal{T}}(e) \le \ell t - \epsilon\overline{\ell} t$ for some $e$, the term for $e$ would be at least 1, and hence the sum would be at least 1. Thus ($b^-$) follows.

We will now prove ($c^-$). When adding a tree $U$ to $\mathcal{T}$, we study the quantity

$$q^-(U) = \sum_{e \in E^*_{\geq \ell}} (1-\epsilon)^{L^{\mathcal{T} \cup \{U\}}(e)}.$$

We want to show that (10) is not increased when a greedy tree $U$ is added to $\mathcal{T}$. Since (10) is proportional to $\sum_{e \in E^*_{\geq \ell}} (1-\epsilon)^{L^{\mathcal{T}}(e)} (1-\epsilon\ell)^{t-|\mathcal{T}|}$, this is equivalent to showing

$$q^-(U) \leq \sum_{e \in E^*_{\geq \ell}} (1-\epsilon)^{L^{\mathcal{T}}(e)} (1-\epsilon\ell).$$

With a random tree $R \in \Pi$, we would get an expectation quantity of

$$\mathrm{E}_{R \in \Pi}\left[q^-(R)\right] = \sum_{e \in E^*_{\geq \ell}} \left( (1-\epsilon)^{L^{\mathcal{T}}(e)} (1 - \epsilon \Pr_{R \in \Pi}[e \in R]) \right).$$

Here $\Pr_{R \in \Pi}[e \in R] = \ell^*(e) \geq \ell$ for $e \in E^*_{\geq \ell}$. Hence

$$\mathrm{E}_{R \in \Pi}\left[q^-(R)\right] \leq \sum_{e \in E^*_{\geq \ell}} (1-\epsilon)^{L^{\mathcal{T}}(e)} (1-\epsilon\ell).$$

Thus it suffices to show that a greedy tree $U$ has a smaller quantity than any tree $R \in \Pi$. Note that $q^-(U) = \sum_{e \in E^*_{\geq \ell}} (1-\epsilon)^{L^{\mathcal{T}}(e)} - \epsilon \sum_{e \in U \cap E^*_{\geq \ell}} (1-\epsilon)^{L^{\mathcal{T}}(e)}$. Since $E(U) \cap E^*_{\geq \ell} = E(U/E_{<\ell})$, we conclude that our quantity $q^-(U)$ grows with $-\sum_{e \in U/E^*_{<\ell}} (1-\epsilon)^{L^{\mathcal{T}}(e)}$. Thus ($c^-$) follows if we can show that

$$c^-(U/E^*_{<\ell}) \leq \min_{R \in \Pi} c^-(R/E^*_{<\ell}) \text{ where } c^-(S) = \sum_{e \in S} -(1-\epsilon)^{L^{\mathcal{T}}(e)}. \tag{11}$$

By Lemma 15, each $R \in \Pi$ is a minimum spanning tree with respect to the ideal relative loads, and this implies that $R/E^*_{<\ell}$ is a spanning tree of $G/E^*_{<\ell}$.

On the other hand, $U$ is a minimum spanning tree of $G$ with respect to the loads $L^{\mathcal{T}}(\cdot)$. In particular, $U/E_{<\ell}$ must contain a minimum spanning tree $T_{G/E_{<\ell}}$ of $G/E_{<\ell}$ with respect to these loads. Since our edge costs $-(1-\epsilon)^{L^{\mathcal{T}}(e)}$ in $c^-$ are negative and grow with the loads $L^{\mathcal{T}}(\cdot)$, we have

$$c^-(U/E^*_{<\ell}) \leq c^-(T_{G/E_{<\ell}}) \leq \min_{R \in \Pi} c(R/E^*_{<\ell}).$$

Hence (11) is satisfied, completing the proof of ($c^-$), which in turn establishes (9) for all edges in $E^*_{\leq \ell}$. ∎

Since $\epsilon\overline{\ell} = \varepsilon/\lambda$, Claim 16.2 and Claim 16.3 immediately imply Proposition 16. ∎

### 4.4. Not all trees cross all min-cuts more than once

We are considering a tree packing $\mathcal{T}$ with at least $6\lambda \ln m/\varepsilon^2$ trees, as in Proposition 16. We are later going to fix $\varepsilon = o(1/(\lambda^3 \log n))$, but for now, $\varepsilon$ could be any positive value below $1/2$. In the proof, we are going to assume

**Assumption A.** *All trees in our greedy tree packings cross all min-cuts at least twice.*

Based on Assumption A, we will arrive at a contradiction for any sufficiently large greedy tree packing, thus proving that some tree crosses some min-cut once, as claimed in Theorem 9.

Let $MC$ be the set of all edges in minimum cuts and let $\ell_0$ be the smallest ideal relative load of edges in $MC$. Then $\ell_0$ is also the largest value such that $E^*_{\geq \ell_0} \supseteq MC$. For $i = 1, 2, \ldots$, let $\ell_i = \ell_0 - i\varepsilon/\lambda$. By Proposition 16, for $i \geq 0$,

$$E^*_{\geq \ell_i} \subseteq E^{\mathcal{T}}_{\geq \ell_{i+1}} \subseteq E^*_{\geq \ell_{i+2}}. \tag{12}$$

Note that (12) holds regardless of the concrete tree packing $\mathcal{T}$, as long as it has at least $6\lambda \ln m/\varepsilon^2$ trees as in Proposition 16. Hence (12) will remain true when we later expand $\mathcal{T}$.

For $i = 0, 1, 2, \ldots$, we define $\mathcal{P}_i$ to be the partition whose sets are the components of $(V, E^*_{<\ell_i})$.

**4.4.1. An easy case.** Suppose all trivial $\mathcal{P}_2$ cuts are min-cuts. In this case, the next greedy tree $T$ added to $\mathcal{T}$ will cross some min-cut once. To see this, first we note that since all trivial $\mathcal{P}_2$ cuts are min-cuts, $E^*_{\geq \ell_2} \subseteq MC$. However, by (12), $E^*_{\geq \ell_2} \supseteq E^{\mathcal{T}}_{\geq \ell_1} \supseteq E^*_{\geq \ell_0} \supseteq MC$. Hence all containments are met with equality, and in particular, $E^*_{\geq \ell_2} = E^{\mathcal{T}}_{\geq \ell_1}$. Consequently, the sets of $\mathcal{P}_2$ are the components of $(V, E^{\mathcal{T}}_{<\ell_1})$. Since the greedy tree $T$ minimizes the concrete relative loads of $\mathcal{T}$, we conclude that $T/\mathcal{P}_2$ is a minimum spanning tree of $G/\mathcal{P}_2$. Any leaf edge of $T/\mathcal{P}_2$, is then the only edge from $T$ crossing the trivial $\mathcal{P}_2$ cut whose one side is defined by the leaf. Since all trivial $\mathcal{P}_2$ cuts are assumed to be min-cuts, we have found a min-cut crossed only once by $T$, contradicting Assumption A.

**Lemma 17.** *For any $i \geq 2$, some trivial $\mathcal{P}_i$ cut is not a min-cut.*

**Proof.** Above we saw that Assumption A implied the lemma for $i = 2$. However, if $i > 2$ and $\mathcal{P}_i \neq \mathcal{P}_2$, we have $E^*_{\geq \ell_i} \supset E^*_{\geq \ell_2} \supseteq MC$. Then $E^*_{\geq \ell_i}$ contains an edge that is not in a min-cut. The end-points of this edge define two trivial $\mathcal{P}_i$ cuts that are not min-cuts. ∎

**4.4.2. The harder case.** We are going to identify an $a>1$ with $E^*_{\geq \ell_a}$ close to $E^{\mathcal{T}}_{\geq \ell_{a+1}}$. We will argue that all edges in $E^*_{\geq \ell_a}$ have concrete relative loads close to $2/\lambda$. On the other hand, by Lemma 17, some edges of $E^*_{\geq \ell_a}$ participate in a trivial $\mathcal{P}_a$ cut that has at least $\lambda+1$ edges. Based on that we will argue that some of them should have concrete loads close to $1/(\lambda+1)$, contradicting that all these loads are close to $2/\lambda$.

In order to define closeness between $E^*_{\geq \ell_a}$ and $E^{\mathcal{T}}_{\geq \ell_{a+1}}$, we need a positive parameter $\alpha<1$ that we are later going to fix as $\alpha=o(1/\lambda^2)$, hence small but much bigger than $\varepsilon=o(1/(\lambda^3\log n))$. Whatever the choice of $\alpha$, there has to be some $a$ between 2 and $2+2\log_{1+\alpha} m$ such that $|E^*_{\geq \ell_{a+2}}\setminus MC|\leq(1+\alpha)|E^*_{\geq \ell_a}\setminus MC|$; otherwise, we would have more than $(1+\alpha)^{\log_{(1+\alpha)} m}=m$ edges. Let such an $a$ be fixed relative to $\alpha$. Then, since $MC\subseteq E^*_{\geq \ell_a}\subseteq E^{\mathcal{T}}_{\geq \ell_{a+1}}\subseteq E^*_{\geq \ell_{a+2}}$, we have

$$\begin{aligned}|E^{\mathcal{T}}_{\geq \ell_{a+1}}\setminus E^*_{\geq \ell_a}| &= |E^{\mathcal{T}}_{\geq \ell_{a+1}}\setminus MC| - |E^*_{\geq \ell_a}\setminus MC| \\ &\leq |E^*_{\geq \ell_{a+2}}\setminus MC| - |E^*_{\geq \ell_a}\setminus MC| \leq \alpha|E^*_{\geq \ell_a}\setminus MC|.\end{aligned} \tag{13}$$

As for (12) we note that (13) holds regardless of the concrete tree packing $\mathcal{T}$, as long as it has at least $6\lambda\ln m/\varepsilon^2$ trees as in Proposition 16.

**Lemma 18.** $\ell_a\geq 2/\lambda-\frac{O((\varepsilon/\alpha)\log n)}{\lambda}$.

**Proof.** First, we argue that $\ell_0$ is close to $2/\lambda$. Since $\ell_0$ was the minimal load of an edge in a min-cut, we can find a min-cut $C$ intersecting $E^*_{=\ell_0}$. Since $C\subseteq MC\subseteq E^*_{\geq \ell_0}$, we have $C\cap E^*_{=\ell_0}=C\cap E^*_{\leq \ell_0}$, so $C\cap E^*_{=\ell_0}$ cuts a component $H$ of $(V,E^*_{\leq \ell_0})$. Then $\Phi_H\leq|C\cap E^*_{=\ell_0}|$. Moreover, by Lemma 14, we have $\Phi_H\geq\Phi$, and by Lemma 3, we have $\Phi>\lambda/2=|C|/2$. Thus $|C\cap E^*_{=\ell_0}|>|C|/2$, so more than half the edges $e\in C$ have $\ell^*(e)=\ell_0$.

Since no edge has ideal relative load bigger than $1/\Phi<2/\lambda$, the average ideal relative edge load over edges in $C$ is $<\frac{\ell_0+2/\lambda}{2}=\ell_0/2+1/\lambda$. Then, by Lemma 16, the average concrete relative load over edges in $C$ is $<\ell_0/2+1/\lambda+\varepsilon/\lambda$. Now, $C$ has $\lambda$ edges, so the total concrete relative load over $C$ is $\ell^{\mathcal{T}}(C)<\ell_0\lambda/2+1+\varepsilon$.

On the other hand, $\ell^{\mathcal{T}}(C)$ is the average number of crossings of $C$ by trees $T\in\mathcal{T}$, and by Assumption A, each $T$ crosses $C$ at least twice, so $\ell^{\mathcal{T}}(C)\geq 2$. Thus, $2<\ell_0\lambda/2+1+\varepsilon$, or equivalently $\ell_0>(1-\varepsilon)2/\lambda$.

We defined $\ell_i=\ell_0-i\varepsilon/\lambda$, so $\ell_a>2/\lambda-(2+a)\varepsilon/\lambda$ where $2+a\leq 2+\log_{1+\alpha} m=O((1/\alpha)\log n)$ since $\alpha<1$. ∎

We are now going to consider an arbitrary greedy tree packing $\mathcal{T}'$ expanding $\mathcal{T}$ with at least one tree.

**Lemma 19.** *There exists an edge $f \in E^*_{\geq \ell_a}$ with $\ell^{\mathcal{T}'\setminus\mathcal{T}}(f) \leq 2/(\lambda+1)+\alpha$.*

**Proof.** We are going to exploit Lemma 17 stating that some trivial $\mathcal{P}_a$ cut is not a min-cut. The edge $f$ will be from one of these non-minimal trivial $\mathcal{P}_a$ cuts.

Let $p$ be the number of components in $(V, E^*_{<\ell_a})$ and let $p'$ be the number of components in $(V, E^{\mathcal{T}'}_{<\ell_{a+1}})$. By (12), $E^*_{<\ell_a} \supseteq E^{\mathcal{T}'}_{<\ell_{a+1}}$, so $p \leq p'$. By definition of greedy, whenever it adds a new tree $T$ to $\mathcal{T}'$, $T/E^{\mathcal{T}'}_{<\ell_{a+1}}$ is a spanning tree of $G/E^{\mathcal{T}'}_{<\ell_{a+1}}$. Hence, $|T \cap E^{\mathcal{T}'}_{\geq \ell_{a+1}}| = p' - 1$, implying $|T \cap E^*_{\geq \ell_a}| \leq p' - 1$.

On the other hand, each edge of $E^{\mathcal{T}'}_{\geq \ell_{a+1}} \setminus E^*_{\geq \ell_a}$ can reduce the number of components of $(V, E^*_{\geq \ell_a})$ by at most 1, so $p' \leq p + |E^{\mathcal{T}'}_{\geq \ell_{a+1}} \setminus E^*_{\geq \ell_a}|$. Also, by (13), $|E^{\mathcal{T}}_{\geq \ell_{a+1}} \setminus E^*_{\geq \ell_a}| \leq \alpha |E^*_{\geq \ell_a} \setminus MC|$. Thus we have

$$|T \cap E^*_{\geq \ell_a}| \leq p - 1 + \alpha |E^*_{\geq \ell_a} \setminus MC|.$$

Let $q$ be the number of trivial $\mathcal{P}_a$ cuts that are also min-cuts, that is, of size $\lambda$. By Assumption A, each of the these min-cuts is crossed at least twice by $T$, that is, by two edges in $T \cap E^*_{\geq \ell_a}$. Also, each edge in $T \cap E^*_{\geq \ell_a}$ is part of exactly two trivial $\mathcal{P}_a$ cuts. Hence the sum of crossings by $T$ over all non-minimal trivial $\mathcal{P}_a$ cuts is at most

$$2|T \cap E^*_{\geq \ell_a}| - 2q < 2(p - q) + 2\alpha |E^*_{\geq \ell_a} \setminus MC|.$$

On the other hand, the sum of the number of edges over all non-minimal trivial $\mathcal{P}_a$ cuts is bounded from below by both $(\lambda+1)(p-q)$ and $2|E^*_{\geq \ell_a} \setminus MC|$; the former because non-minimal cuts have at least $\lambda+1$ edges, and the latter because each edge in $E^*_{\geq \ell_a} \setminus MC$ contributes to two trivial $\mathcal{P}_a$ cuts, neither of which are minimal.

Thus, on the average, $T$ uses an edge in a non-minimal trivial $\mathcal{P}_a$ cut at most

$$\frac{2(p-q)}{(\lambda+1)(p-q)} + \frac{2\alpha |E^*_{\geq \ell_a} \setminus MC|}{2|E^*_{\geq \ell_a} \setminus MC|} = 2/(\lambda+1) + \alpha$$

times. Since this average is guaranteed by all trees $T$ added to $\mathcal{T}'$, starting from $\mathcal{T}$, there must exist an edge $f \in E^*_{\geq \ell_a}$ in some non-minimal trivial $\mathcal{P}_a$ cut with $\ell^{\mathcal{T}'\setminus\mathcal{T}}(f) \leq 2/(\lambda+1)+\alpha$. ∎

Let $\mathcal{T}'$ be the continuation to four times the size of the tree packing $\mathcal{T}$. That is, $t' = |\mathcal{T}'| = 4t$ where $t = 6\lambda \ln m/\varepsilon^2$. For any edge $e \in E$,

$$|L^{\mathcal{T}'\setminus\mathcal{T}}(e) - |\mathcal{T}' \setminus \mathcal{T}| \cdot \ell^*(e)| = \left| \sum_{T \in \mathcal{T}'\setminus\mathcal{T}} (L^{\{T\}}(e) - \ell^*(e)) \right|$$

$$\begin{aligned}
&\leq \left| \sum_{T \in \mathcal{T}'} (L^{\{T\}}(e) - \ell^*(e)) \right| \\
&\qquad + \left| \sum_{T \in \mathcal{T}} (L^{\{T\}}(e) - \ell^*(e)) \right| \\
&= |L^{\mathcal{T}'}(e) - |\mathcal{T}'| \cdot \ell^*(e)| + |L^{\mathcal{T}}(e) - |\mathcal{T}| \cdot \ell^*(e)|.
\end{aligned}$$

By Proposition 16, $|\ell^{\mathcal{T}}(e) - \ell^*(e)| \leq \varepsilon/\lambda$. Also, since $t' = 4t = 6\lambda \ln m/(\varepsilon/2)^2$, we have $|\ell^{\mathcal{T}'}(e) - \ell^*(e)| \leq (\varepsilon/2)/\lambda$. Hence

$$\begin{aligned}
|\ell^{\mathcal{T}' \setminus \mathcal{T}}(e) - \ell^*(e)| &= \frac{|L^{\mathcal{T}' \setminus \mathcal{T}}(e) - |\mathcal{T}' \setminus \mathcal{T}| \cdot \ell^*(e)|}{|\mathcal{T}' \setminus \mathcal{T}|} \\
&\leq \frac{|L^{\mathcal{T}'}(e) - |\mathcal{T}'| \cdot \ell^*(e)| + |L^{\mathcal{T}}(e) - |\mathcal{T}| \cdot \ell^*(e)|}{|\mathcal{T}' \setminus \mathcal{T}|} \\
&\leq \frac{t' \cdot |\ell^{\mathcal{T}'}(e) - \ell^*(e)| + t \cdot |\ell^{\mathcal{T}}(e) - \ell^*(e)|}{t' - t} \\
&= \frac{t' \cdot (\varepsilon/2)/\lambda + t \cdot \varepsilon/\lambda}{t' - t} = \varepsilon/\lambda.
\end{aligned}$$

In particular, we have

$$\ell^*(f) - \ell^{\mathcal{T}' \setminus \mathcal{T}}(f) \leq \varepsilon/\lambda$$

for the $f \in E^*_{\geq \ell_a}$ from Lemma 19, which had $\ell^{\mathcal{T}' \setminus \mathcal{T}}(f) \leq 2/(\lambda+1) + \alpha$. However, $\ell^*(f) \geq \ell_a$, and by Lemma 18, $\ell_a \geq 2/\lambda - \frac{O((\varepsilon/\alpha)\log n)}{\lambda}$. Thus we have

$$\begin{aligned}
&2/\lambda - \tfrac{O((\varepsilon/\alpha)\log n)}{\lambda} - (2/(\lambda+1) + \alpha) \leq \varepsilon/\lambda \\
\iff \qquad &2/(\lambda(\lambda+1)) \leq \tfrac{O((\varepsilon/\alpha)\log n)}{\lambda} + \alpha\,.
\end{aligned}$$

To balance, we set $\alpha = \frac{(\varepsilon/\alpha)\log n}{\lambda}$, and to reach a contradiction, we set $\alpha = o(1/\lambda^2)$. Then $\varepsilon = \lambda\alpha^2/\log n = o(1/(\lambda^3 \log n))$. Thus, with this choice of $\varepsilon$, we conclude that Assumption A is false, hence that any greedy tree packing $\mathcal{T}'$ with $t' = 4t = 24\lambda \ln m/\varepsilon^2 = \omega(\lambda^7 \log^3 n)$ trees has a tree crossing some min-cut only once. *This completes the proof of Theorem 9.*

## 5. Maintaining tree packings and cuts

In this section, we will consider how to maintain a tree packing of polylogarithmic size, including the minimum cut crossed once by some tree in the packing. First we consider the maintenance of the tree packing itself.

### 5.1. Tree packings

For a dynamic graph, we maintain a greedy tree packing of size $t$ as a list of $t$ dynamic minimum spanning tree structures, each with weights being the total loads over the previous spanning trees. Thorup and Karger [23] have shown that each edge insertion or deletion translates into at most $t^2$ updates for the trees in the tree packing. In [23] they implemented each tree with the dynamic minimum spanning tree algorithm of Holm et al. [13] supporting each update in polylogarithmic amortized time. Here, instead, we use the dynamic minimum spanning tree algorithm of Frederickson [9] with the sparsification of Eppstein et al. [6] supporting updates in $O(\sqrt{n})$ worst case time. Thus, we get:

**Lemma 20 ([6,9,23]).** *We can maintain a greedy tree packing of polylogarithmic size for a fully-dynamic graph in $\tilde{O}(\sqrt{n})$ time per edge insertion or deletion.* ∎

Our remaining problem is now, independently for each dynamic spanning tree $T$ in the tree packing, to maintain the minimum cut crossed once by $T$. To this end, we will use top trees which is a data structure for maintaining information about dynamic trees. We note that we will be reusing many of the ideas from the 2-edge connectivity algorithm in [10] which maintains tree cuts of size 1.

### 5.2. Top Trees

Top trees is a generic data structure that maintains a binary hierarchy of subtrees, called clusters, over each tree in a dynamic forest. This hierarchy can be used to organize information for a diverse set of applications such as the tree packings and cuts needed in this paper. For the cuts, the dynamic forest will span the components of the dynamic graph, and we will maintain all graph edges between any pair of clusters in the hierarchy. If a tree edge is deleted, we then get the cut edges between the two subtrees. Top trees were introduced by Alstrup et al. [1] as a simpler-to-use variant of Frederickson's topology trees [10].

A top tree is defined over a labeled tree, that is, a tree $T$ where each vertex is associated with a possibly empty set of labels, and where all labels are distinct. In our cut application, a label will correspond to an end-point of an edge in the underlying graph $G=(V,E)$. Thus, for each $(v,w)\in E$, we will have a label $[v,w]$ associated with $v$ and a symmetric label $[w,v]$ associated with $w$. Note that as we define a top tree over $T$, edges are generally

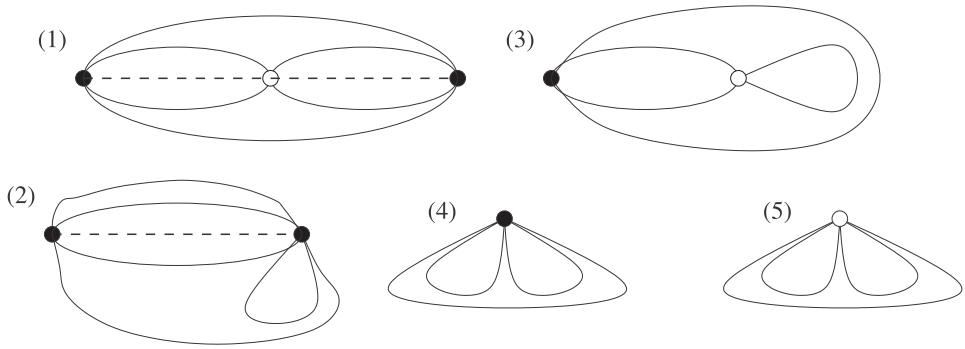


**Fig. 1.** The cases of joining two neighboring clusters into one. The • are the boundary vertices of the joined cluster and the ∘ are the boundary vertices of the children clusters that did not become boundary vertices of the joined cluster. Finally the dashed line is the cluster path of the joined cluster.

understood to be from $T$. The graph edges from $G$ are only mentioned as one of many potential denotations of the labels.

In a sub-tree $U$ of $T$, each vertex is associated a subset of its labels in $T$. The *boundary* $\partial U$ of $U$ is the set of vertices in $U$ that have strictly less incident edges or associated labels in $U$ than in $T$. In particular, $\partial T = \emptyset$. We call $U \setminus \partial U$ the *interior* of $U$. We say that $U$ is a *cluster* of $T$ if $U$ has at least one edge or label and at most two boundary vertices.

If a cluster $C$ has two boundary vertices $a$ and $b$, we call $C$ a *path cluster*. The path from $a$ to $b$ is called the *cluster path* of $C$, denoted $\pi(C)$. If $C$ has only one boundary vertex $a$, $C$ is called a *point cluster*, and then $\pi(C) = a$. Note that we get a new labeled tree if we replace a path cluster with an edge between its boundary vertices, or if we replace a point cluster with a label at its boundary vertex.

A *top tree* $\mathcal{R}$ over $T$ is a binary tree such that:

1. The nodes of $\mathcal{R}$ are clusters of $T$.
2. Sibling clusters are *neighbors* in the sense that they intersect in a single vertex but have no edges or labels in common. Then their parent cluster is their union (see Fig. 1).
3. The root of $\mathcal{R}$ is $T$ itself.

The *size* of a labeled tree or cluster is its total number of edges and labels. The basic philosophy is that clusters are induced by their edges and labels, the vertices only being included as their end-points. This is also why a tree consisting of a single vertex with no labels has no clusters, hence no top tree.

The leaves of the top tree $\mathcal{R}$ are referred to as *base clusters*. In one extreme, $\mathcal{R}$ consists of the single cluster $T$ acting as both root and base cluster. In the other extreme, each edge and label induces its own base cluster of size 1.

We will typically operate with a parameter $q$ limiting the size of the base clusters. If the underlying tree $T$ has size $s$, we want the top tree $\mathcal{R}$ to have $O(\lceil s/q \rceil)$ base clusters. Also, we want the height of $\mathcal{R}$ to be $O(\log s)$. This way, $\mathcal{R}$ provides us with a balanced binary decomposition of $T$, down to a certain granularity parameterized by $q$. Our problem is that we want to operate on a dynamic labeled forest and we wish to maintain balanced top trees over all the labeled trees, modifying the top trees as little as possible.

More precisely, our top trees have to be maintained under the following *forest updates*, updating the underlying dynamic labeled forest:

$\mathsf{link}((v,w))$ where $v$ and $w$ are in different trees, links these trees by adding the edge $(v,w)$ to our dynamic forest.
$\mathsf{cut}(e)$ removes the edge $e$ from our dynamic forest.
$\mathsf{attach}(v,a)$ attaches a label $a$ to the vertex $v$.
$\mathsf{detach}(a)$ detaches the label $a$ from whatever vertex it was attached to.

To accommodate these forest updates, the top trees are changed by a sequence of local *top tree modifications* described below. During these modifications, we will temporarily accept a *partial* top tree whose root cluster may not be a whole underlying tree $T$ but just a cluster of $T$.

$C := \mathsf{create}()$ creates a top tree consisting of the single cluster $C$.
$C := \mathsf{join}(A,B)$ where $A$ and $B$ are neighboring root clusters of two top trees $\mathcal{R}_A$ and $\mathcal{R}_B$. Creates a new cluster $C = A \cup B$ and makes it the common root of $A$ and $B$, thus turning $\mathcal{R}_A$ and $\mathcal{R}_B$ into a single new top tree $\mathcal{T}_C$. Finally, the new root cluster $C$ is returned.
$\mathsf{split}(C)$ where $C$ is the root cluster of a top tree $\mathcal{T}_C$ and has children $A$ and $B$. Deletes $C$, thus turning $\mathcal{T}_C$ into the two top trees $\mathcal{T}_A$ and $\mathcal{T}_B$. Finally, the root clusters of $\mathcal{T}_A$ and $\mathcal{T}_B$ are returned.
$\mathsf{destroy}(C)$ eliminates the top tree consisting of $C$.

**5.2.1. Discipline for modifying top trees.** Top tree modifications have to be applied in the following order:

1. First, top-down, we perform a sequence of splits.
2. Then we destroy some base clusters.
3. Then we update the labeled forest.
4. Then we create some base clusters.
5. Finally, with joins, we recreate complete top trees bottom-up.

The above order implies that when we do a split or join, we know that all parts of the underlying labeled forest is partitioned into base clusters.

It is an important rule that *a forest update may not change any current cluster*. Here, a cluster is changed by a forest update if the update changes its labels or edges or its set of boundary vertices. To appreciate the latter, consider an update $\mathsf{attach}(v, a)$. This update would change any cluster $C$ where $v$ was an interior vertex; either adding $a$ as a label in $C$, or making $v$ a boundary vertex. Satisfying the rule means that when we get to the update in step 3, the previous steps 1–2 should have eliminated all clusters that would be changed by the update.

It is often natural to perform a *composite sequence of updates* in step 3. For example, if dealing with a spanning tree $T$, we might want to swap one tree edge $(v_1, w_1)$ with another edge $(v_2, w_2)$. If we do $\langle \mathsf{cut}((v_1, w_1)); \mathsf{link}((v_2, w_2)) \rangle$ as a composite update rather than as two separate updates, we avoid dealing with a disconnected spanning tree during the top tree modifications in steps 1–2 and 4–5. Knowing that the spanning tree is a connected spanning tree may help avoiding pathological special cases in applications.

By a simple reduction to Frederickson's topology trees [10], Alstrup et al. [1] show:

**Lemma 21 ([1,10]).** *Consider a fully-dynamic labeled forest and let $q$ be a positive integer parameter. For the trees in the forest, we can maintain top trees with base clusters of size at most $q$ and such that if a tree has size $s$, its top tree has height $O(\log s)$ and $O(\lceil s/q \rceil)$ clusters. Each* link, cut, attach, *or* detach *is supported with $O(1)$* creates *and* destroys*, and $O(\log s)$* joins *and* splits*. These top tree modifications are identified in $O(\log s)$ time. For a composite sequence of $k$ updates, the number of each top tree modifications is multiplied by $O(k)$.* ∎

**5.2.2. Representation and usage of top trees.** A top tree is represented as a standard binary rooted tree with parent and children pointers. The nodes used to represent the top tree are called *top nodes*. These top nodes represent clusters, and with each top node is associated the set of at most two boundary vertices of the represented cluster. With a top leaf we have a representation of a base cluster as a labeled subtree of the underlying forest. With an internal top node is stored how it is decomposed into its children (c.f. Fig. 1). Thus, considering the information descending from a top node, we can construct the cluster it represents.

From each label or edge, there is a pointer to the base cluster containing it, if any. Also, from each vertex $v$, there is a pointer to the smallest cluster

that $v$ is internal to. Using these pointers along with the parent pointers in the top tree, for any edge, label, or vertex $x$, we can find the root, $\mathsf{top\_root}(x)$, of the top tree over the underlying tree $T$ containing $x$. With top trees of logarithmic height as in Theorem 21, we identify $\mathsf{top\_root}(x)$ in $O(\log s)$ time. A typical application in a forest is that two vertices $v$ and $w$ are in the same underlying tree if and only if $\mathsf{top\_root}(v) = \mathsf{top\_root}(w)$.

An *application* of the top tree data structure has direct access to the above representation, and will typically associate some extra information with the top nodes. The application employs an *implementation* of top trees, which is an algorithm like the one described in Theorem 21, converting each forest update into a sequence of top tree modifications. In connection with each top tree modification, the application is notified and given pointers to the top nodes representing the involved clusters. The application can then update its information associated with these top nodes. We note that a top tree may only be modified with the defined top tree modifications: create, join, split and destroy. This discipline is important if we have several applications running over the same top trees.

### 5.3. Finding given cut sizes and edges between clusters

We are considering a given spanning tree $T$ from our tree packing of the dynamic graph $G = (V, E)$. By a *tree cut*, we mean a cut defined by deleting a tree edge, the sides being spanned by the two resulting subtrees of $T$. We will now show how to find the size of a given tree cut. Afterward we will also list the cut edges. The techniques developed will all help us toward our real goal; namely that of finding the minimal tree cut, that is, the smallest cut crossed once by $T$.

We shall refer to edges in $T$ as *tree edges* and all edges in $G$ as *graph edges*. As mentioned previously, for each graph edge $(v, w)$, we will have labels $[v, w]$ and $[w, v]$ representing its end-points. The label $[v, w]$ is attached to $v$ and the label $[w, v]$ is attached to $w$. The edge $(v, w)$ is only viewed as incident to a cluster $C$ if one of these end-point labels are in $C$. When the graph edge $(v, w)$ is inserted in $G$, these two labels are attached with a composite update $\langle \mathsf{attach}(v, [v, w]), \mathsf{attach}(w, [w, v]) \rangle$. Similarly, the graph edge $(v, w)$ is deleted with $\langle \mathsf{detach}([v, w]), \mathsf{detach}([w, v]) \rangle$. This way we never have one label but not the other present in our top trees (c.f. § 5.2.1). This property will simplify our later algorithms, avoiding pathological special cases. Similarly, when the tree packing swaps a tree edge $e_1$ with another $e_2$ in $T$, we perform the composite update $\langle \mathsf{cut}(e_1), \mathsf{link}(e_2) \rangle$ so that the tree does not get disconnected.

With the above labels, the size of $T$ is $\Theta(m)$. We will maintain top trees over the current trees in $T$ using Theorem 21 with $q=\Theta(\sqrt{m})$. For now, we think of $q$ as fixed. Thus we have $O(\sqrt{m})$ base clusters of size $O(\sqrt{m})$, and the height of the top trees is $O(\log n)$.

**5.3.1. Cut sizes.** The idea for cut sizes is very simple. A graph edge $(v,w)\in E$ connects cluster $A$ and $B$ if label $[v,w]$ is in $A$ and label $[w,v]$ is in $B$, or vice versa. Suppose for each cluster pair $(A,B)$, we know the number $m(A,B)$ of graph edges $(v,w)\in E$ connecting $A$ and $B$. Then, to find the size of the tree cut of $(v,w)\in T$, we simply $\mathsf{cut}((v,w))$ and return $m(\mathsf{top\_root}(v),\mathsf{top\_root}(w))$. Having found the desired size, we restore $T$ with $\mathsf{link}((v,w))$.

We assume that clusters have distinct numbers up to $O(\sqrt{m})$. Then the numbers $m(\cdot,\cdot)$ are stored as symmetric 2-dimensional array of size $O(m)$. The symmetry means that we identify $m(A,B)$ with $m(B,A)$, so changing one changes the other implicitly.

When a base cluster is created with $A:=\mathsf{create}()$, first we set $m(A,\cdot)=0$. Next, for each label $[v,w]$ in $A$, we take the base cluster $B$ of the symmetric label $[w,v]$ and increment $m(A,B)$ by one. Also, for we increment $m(A,C)$ for each of the $O(\log n)$ strict ancestors $C$ of $B$. Since $A$ has $O(\sqrt{m})$ labels, the create takes $O(\sqrt{m}\log n)$ time. When a higher cluster is created with $C:=\mathsf{join}(A,B)$, we set $m(C,C)=m(A,B)+m(A,A)+m(B,B)$. Also for each cluster $D\neq C$, we set $m(C,D)=m(A,D)+m(B,D)$. Thus, with $O(\sqrt{m})$ clusters, $\mathsf{join}$ takes $O(\sqrt{m})$ time. When a cluster $C$ is $\mathsf{split}$ or $\mathsf{destroy}$ed, we do not need to do anything, except to recycle its number when a new cluster is created or merged. Plugging the above time bounds into Theorem 21, we get

**Lemma 22.** *For each cluster pair $(C,D)$, we can maintain the number $m(C,D)$ of graph edges between $C$ and $D$, all in $O(\sqrt{m}\log n)$ time per forest update. In particular, we can identify the size of a tree cut in $O(\sqrt{m}\log n)$ time.* ∎

**5.3.2. Cut edges.** For each cluster pair $(C,D)$, let $E(C,D)$ be the edges connecting $C$ and $D$. Then $|E(C,D)|=m(C,D)$. We want to be able to list $E(C,D)$ in $O(\log n)$ time per edge. Then, to list the edges of a tree cut by a given tree edge $(v,w)$, we $\mathsf{cut}((v,w))$, list $E(\mathsf{top\_root}(v),\mathsf{top\_root}(w))$, and $\mathsf{link}((v,w))$.

As our only new information, for each pair $(A,B)$ of base clusters, we will store $E(A,B)$ as a list. Then, for a given cluster pair $(C,D)$, we can list

$E(C,D)$ recursively. More precisely, if $m(C,D)=0$, we list nothing. Otherwise, if $C$ and $D$ are both base clusters, we return the stored list $E(C,D)$. Otherwise, one of them, say $C$ has child clusters $A$ and $B$. Then, recursively, we first list $E(A,D)$, second $E(B,D)$. The depth of the recursion is the height of the top trees, so we spend $O(\log n)$ time per edge listed.

To maintain $E(A,B)$ for each pair of base clusters, when a base cluster $A$ is created, we just have to partition the $O(\sqrt{m})$ incident edges between the $O(\sqrt{m})$ other base clusters, all in $O(\sqrt{m})$ time. Thus, we get

**Lemma 23.** *In Lemma 22, we can further list the edges in $E(C,D)$ or the edges of the tree cut in $O(\log n)$ time per edge.* ∎

Above, we have thought of the parameter $q=\Theta(\sqrt{m})$ as fixed. However, if $m$ deviates from $q^2$ by more than $q^2/4$, then, over the next $q^2/4$ updates, we can build new top trees in the background with a new value of $q$. This completes our first simple application of top trees.

Two things are worth noting. First note how we tag more and more information to the same top trees as we address more and more complicated applications – from cut size to listing of edges. Second note that we could not directly maintain the lists $E(C,D)$ for each cluster pair, but only for pairs of base clusters. Instead we used the numbers $m(\cdot,\cdot)$ to direct a recursive search for the edges.

## 5.4. Maintaining cuts crossed once

We now return to our original problem of maintaining the minimum cut of $G$ crossed once by the dynamic spanning tree $T$.

**5.4.1. Covering.** For any two vertices $v$ and $w$, we will use $v\cdots w$ to denote the unique path from $v$ to $w$ in $T$. We say a graph edge $(v,w)\in E$ *covers* the edges in $v\cdots w$. The cover $c(e)$ of a tree edge $e\in T$ is the number of edges covering it, including itself as a graph edge. The tree cut induced by removing $e$ has size $c(e)$, so $\min_{e\in T} c(e)$ is exactly the minimum size of a cut crossed once by $T$. Thus we would like to maintain the covering of all edges as well as an edge with minimum cover.

In order to update the $c$-values of the edges in the dynamic spanning tree $T$ efficiently, we use a data structure of Sleator and Tarjan [20]. For any two vertices $v$ and $w$, it allows us to add a common value to the $c$-values of all edges in $v\cdots w$ in $O(\log n)$. The time bound holds regardless of the length of $v\cdots w$, so the $c$-values are only represented implicitly be the data structure. The $c$-value of an edge can be retrieved in $O(\log n)$ time. Within

this time bound, we can also get an edge with minimum $c$-value in $v \cdots w$. Changes to $T$ itself are also supported in $O(\log n)$ time per update. The $c$-values of the individual edges are not affected by such changes. An extension by Goldberg et al. [11] allows us to find an edge with minimum $c$-value in a tree $O(\log n)$ time. Thus we have three $O(\log n)$ time operations on our spanning tree $T$: $\mathsf{add\_cover}(v, w, x)$ adds $x$ to the cover of all edges in $v \cdots w$, $\mathsf{cover}(e)$ returns the cover of the edge $e$, and $\mathsf{min\_cover}()$ returns an edge with minimal cover.

Note that if our tree $T$ did not change, we would be done. More precisely, when a graph edge $(v, w)$ is added, we call $\mathsf{add\_cover}(v, w, 1)$, and when it is deleted, we call $\mathsf{add\_cover}(v, w, -1)$. To find the edge inducing the minimum tree cut, we return $e := \mathsf{min\_cover}()$, and if we want the value of the cut, we return $\mathsf{cover}(e)$.

Our problems start when we change $T$, swapping a tree edge $e$ with a tree edge $f$. This operation changes the path covered by the $c(e)$ graph edges covering $e$, and there could be $\Theta(m)$ such graph edges. Part of our efficient solution is that we only maintain covering up to $q = \Theta(\sqrt{m})$. Here $q$ is the same parameter as it was in the last subsection. *By up to* $q$, we mean that if the cover is larger than $q$, we just report that it is larger without providing the exact cover. This relaxation will allow us recover from a swap in $\tilde{O}(\sqrt{m})$ time.

Note that, really, we are only interested in covering up to polylogarithmic size, so getting up to $\Theta(\sqrt{m})$ is more than we need. However, due to $\Theta(\sqrt{m})$ bottlenecks associated with the time bounds in Lemma 22, we would not get a faster algorithm with less covering.

**5.4.2. Local covering.** For each cluster $C$ in the top tree, we will maintain the *local covering* obtained as follows.

- If an edge $(v, w)$ has both end-points $[v, w]$ and $[w, v]$ in $C$, cover $v \cdots w$.
- If an edge $(v, w)$ has only one end-point $[v, w]$ in $C$ then:
  - If $C$ is a point cluster, that is, $C$ has a single boundary vertex $a$, then cover $v \cdots a$.
  - If $C$ is a path cluster, that is, $C$ has two boundary vertices $a$ and $b$, then cover $v \cdots c$ where $c$ is the point closest to $v$ on the cluster path $\pi(C) = a \cdots b$.
- If an edge $(v, w)$ has no end-point $[v, w]$ in $C$, it does not contribute to the local covering of $C$.

Note that it is only on the cluster path that there is any difference between the regular covering and the local covering of $C$.

The point in the local covering is that it is exactly part of the regular covering of $C$ which is independent of the rest of $T$. For example, consider the case where $C$ has only one end-point $[v,w]$ in $C$ and $C$ is a path cluster with boundary vertices $a$ and $b$. Depending on where the other end-point $[w,v]$ is attached in $T\setminus C$, the regular covering of $v\cdots w$ will contain either $v\cdots a$ or $v\cdots b$. However, $v\cdots a\cap v\cdots b=v\cdots c$, so our local covering of $v\cdots c$ is exactly the part of the regular covering by $(v,w)$ that is not affected by the structure of $T\setminus C$.

Recall that as long as $C$ is a current cluster, it cannot change, neither in its labels, edges, or boundary vertices. Any update that would change $C$, would first have to split or destroy $C$. Consequently, there cannot be any changes to the local covering of any current cluster.

Below we show how to maintain the local covering (up to $q$) of clusters as clusters are allocated and freed with create, join, split, and destroy.

**5.4.3. Base clusters.** When a base cluster is created by $A:=\mathsf{create}()$, we just follow the above definition of local covering. We consider each of the $q$ incident edges $(v,w)$, with $[v,w]\in A$. If $[w,v]\in A$, we call $\mathsf{add\_cover}(v,w,1)$. If $[w,v]\notin A$ and $a$ is the only boundary vertex of $A$, we call $\mathsf{add\_cover}(v,a,1)$. Finally, if $[w,v]\notin A$ and $a$ and $b$ are the boundary vertices of $A$, we want to cover the edges between $v$ and the cluster path. This is done by $\mathsf{add\_cover}(v,a,1/2)$, $\mathsf{add\_cover}(v,b,1/2)$, and $\mathsf{add\_cover}(a,b,-1/2)$.

Thus we spend $O(\log n)$ time per incident edge, hence $O(\sqrt{m}\log n)$ time in total when creating a base cluster. When later $A$ is destroyed by $\mathsf{destroy}(A)$, we do exactly as above but with all cover values negated.

**5.4.4. Above the base.** We now consider the problem of updating the local covering in connection with a join $C:=\mathsf{join}(A,B)$. We already have the local covering of $A$ and $B$, and we want to extend this to a local covering of $C$. We note that we may have $\Theta(m)$ graph edges incident to $C$, so we cannot consider all of them as we did for base clusters. However, a carefully selected subset of size $O(q)$ will suffice because we only have to do the local covering up to $q$.

By symmetry, it suffices to consider the new covering that has to be done of the edges in $A$. We know that we have already done the local covering for $A$ alone. If $A$ is a point cluster, there is nothing to be done because it is only on the cluster path $\pi(A)$ that there can be a difference between the local and the regular covering. Hence we may assume that $A$ is a path cluster. We now consider the case where the resulting cluster $C$ is also a path cluster. Thus, in Fig. 1, $A$ could be either child cluster in (1) or the left child cluster

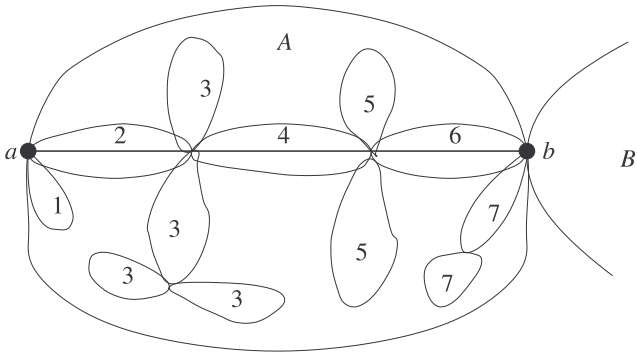


**Fig. 2.** The ordering in which leaf cluster in $A$ are visited.

in (2). In all these cases, the cluster path of $A$ is contained in the cluster path of $C$. Let $a$ and $b$ be the boundary vertices of $A$ with $b$ the boundary vertex shared with $B$.

To complete the local covering of $C$ in $\pi(A)$, for each edge $(v,w) \in E(A,B)$, we would like to cover $v \cdots w \cap \pi(A) = v \cdots b \cap a \cdots b$. Note that $(v,w)$ will also be used to cover part of $\pi(B)$ when $B$ is considered. The covering of $v \cdots b \cap a \cdots b$ is done in $O(\log n)$ time by calling $\mathsf{add\_cover}(v,b,1/2)$, $\mathsf{add\_cover}(a,b,1/2)$, and $\mathsf{add\_cover}(v,a,-1/2)$. Also, by Lemma 23, the edges are listed in $O(\log n)$ time per edge. Our problem is that $E(A,B)$ may be very large with $\Theta(m)$ edges.

**5.4.5. Reducing the set of edges to cover with.** If $E(A,B) > q$, we will only cover as above with a *reduced edge set* $E^-(A,B) \subseteq E(A,B)$. The idea is that when done with $E^-(A,B)$, the remaining edges in $E(A,B)$ would only increase the covering of edges $e$ with a current covering $c(e) \geq q$, and that is not necessary.

The basic idea is that the edges in $E(A,B)$ are only used to cover suffixes of $a \cdots b$. The set $E^-(A,B)$ contains between $q$ and $2q$ edges from $E(A,B)$ maximizing the length of the covered suffix of $a \cdots b$. Let $P$ be the shortest suffix of $a \cdots b$ covered by an edge in $E^-(A,B)$. Then all edges in $E^-(A,B)$ cover $P$, so when done with $E^-(A,B)$, each edge $e \in P$ has covering $c(e) \geq |E^-(A,B)| \geq q$. On the other hand, edges in $E(A,B) \setminus E^-(A,B)$ will only add to the covering of edges in $P$.

To identify the edges for $E^-(A,B)$, we visit the base clusters below $A$ in the order indicated in Fig. 2, visiting base clusters with the same number in an arbitrary order. Formally, the point in the ordering is that $X$ is before $Y$ if for some boundary vertex $x$ of $X$ and $y$ of $Y$, we have $x \cdots b \cup a \cdots b$

containing $y\cdots b \cup a\cdots b$. In that case, all edges from $X$ to $B$ cover as long a suffix of the cluster path $a\cdots b$ as does any edge from $Y$. Such an ordering of the base clusters below $A$ is easily derived following the decomposition of $C$ in the top tree. This takes time proportional to the number of base clusters below $A$, hence $O(\sqrt{m})$ time.

When we visit a base cluster $X$ below $A$, we add all of $E(X,B)$ to $E^-(A,B)$, visiting no more leaf clusters if $|E^-(A,B)| \geq q$. Since $X$ has no more than $q$ incident edges, we end up either with $E^-(A,B) = E(A,B)$ or $q \leq |E^-(A,B)| < 2q$, as desired. We can now cover $a\cdots b$ with the edges from $E^-(A,B)$ in $O(q\log n) = O(\sqrt{m}\log n)$ time. This completes the local covering up to $q$ of $C$ restricted to edges in $A$.

The set $E^-(A,B)$ is saved at $C$ so if later the cluster $C = A\cup B$ is split, we can just reverse the above process doing the same covering with negated values. Note that there are only $O(\sqrt{m})$ clusters in the top tree, so the total space used by the sets $E^-(A,B)$ is $O(m)$.

### 5.5. The last join case

In connection with $C := \mathsf{join}(A,B)$ it remains to consider the case where $A$ is path cluster but where the resulting cluster $C$ is not a path cluster. This is the case where $A$ is the left cluster in Fig. 1 (3). Again we let $a$ and $b$ be the boundary vertices of $A$ with $b$ the boundary vertex shared with $B$. We now want to cover the cluster path $a\cdots b$ with all edges leaving $A$. For the edges going to $B$ we use exactly the same procedure as before, identifying the edge set $E^-(A,B)$ and covering with these edges. For the other edges going to $D = T\setminus C$ we use a symmetric approach. Each of these edges covers a prefix of $a\cdots b$. We now visit the leaf clusters $X$ of $A$ in the reverse order of that in § 5.4.5 (see Fig. 2), accumulating $E(X,D)$ in $E^-(A,D)$ until it gets more than $q$ edges or all leaf clusters have been visited. When done, for each $(v,w)\in E^-(X,D)$, we call $\mathsf{add\_cover}(v,a,1/2)$, $\mathsf{add\_cover}(a,b,1/2)$, and $\mathsf{add\_cover}(v,b,-1/2)$.

Our only remaining problem is how to identify $E(L,D)$ since $D = T\setminus C$ is not a cluster in the top tree. From Lemma 21 we know that the top tree is restored in $O(\log n)$ joins, so when $C := \mathsf{join}(A,B)$ is called, the current top tree can have at most $O(\log n)$ root clusters to be merged. One of these is $C$, and the other $C_1,\ldots,C_k$ partition $D = T\setminus C$. Therefore, to list $E(X,D)$, for $i = 1,\ldots,k$, we list $E(X,C_i)$ as described in Lemma 23. Note that using $m(X,C_i)$, we can decide if $E(X,C_i)$ is empty in constant time. Hence, we construct $E^-(A,D)$ in $O(q\log n) = O(\sqrt{m}\log n)$ time.

We have now shown that in connection with a join $C := \mathsf{join}(A, B)$, we can extend the local covering up to $q$ from $A$ and $B$ to $C$ in $O(\sqrt{m}\log n)$ time using reduced edge sets. The reduced edge sets are stored in $O(m)$ space, and using them, we can revert the extension when $C$ is later $\mathsf{split}$.

### 5.6. Wrapping up the minimum tree cut

Since all local top modifications are supported in $\tilde{O}(\sqrt{m})$ time, by Theorem 21, we support each forest update in $\tilde{O}(\sqrt{m})$ time. When the top tree is complete, the root cluster contains the whole tree $T$, and the local covering of $T$ is the same as the regular covering of $T$. With $e := \mathsf{min\_cover}()$, we find the edge $e$ with the smallest cover in $O(\log n)$ time, including its cover $c(e)$. If $c(e) \geq q = \Theta(\sqrt{m})$, we know that all edges have true cover at least $q$, hence that all tree cuts are of size at least $q$. If $c(e) < q$, we know that $e$ defines the smallest tree cut and that it is of size $c(e)$. As in § 5.3, we can list the edges of the tree cut by cutting $e = (v, w)$ temporarily, and then listing $E(\mathsf{top\_root}(v), \mathsf{top\_root}(w))$ as in Lemma 23 in $O(\log n)$ time per edge. Cutting $e$ takes $\tilde{O}(\sqrt{m})$ time, but we can absorb that in the general update time. More precisely, after each external update to the tree $T$ or the graph $G$, we can identify $e$ and $c(e)$, and internally $\mathsf{cut}(e)$. Then the cut edges are available for listing, and also, we have the subtrees representing the two sides. Next an external update is made, we first internally $\mathsf{link}(e)$.

Summing up, we have now proved

**Proposition 24.** *For a dynamic graph $G$ with a dynamic spanning tree $T$, we can maintain a minimum tree cut up to size $\tilde{O}(\sqrt{m})$ in $\tilde{O}(\sqrt{m})$ time per update.* ∎

## 6. Maintaining the min-cut of a dynamic graph

In this final section, we combine the previous results so as to maintain the min-cut or near-minimal cut of a dynamic graph. First, we turn toward our main result on deterministic maintenance of polylogarithmic min-cuts. To get the best bounds, we consider a sparse witness of $k$-edge connectivity proposed by Nagamochi and Ibaraki [16]. Here, a *$k$-edge connectivity witness* is a subgraph $H$ of $G$ with at most $k(n-1)$ edges and such that either $\lambda_H = \lambda < k$ or $k \leq \lambda_H \leq \lambda$. Eppstein et al. [6] have shown that the above witness can be maintained dynamically. More precisely, we have

**Lemma 25 ([6,16]).** *In $O(k\sqrt{n})$ time per update, we can maintain a $k$-edge connectivity witness $H$ of $G$. The graph $H$ has at most $k(n-1)$ edges*

*and each insertion or deletion of an edge from $G$ results in at most two insertions or deletions of edges in $H$.* ∎

We are now ready to prove:

**Theorem 26.** *For any polylogarithmic $k$, in $\tilde{O}(\sqrt{n})$ time per edge insertion or deletion in a fully-dynamic graph, we can maintain a min-cut of up to size $k-1$, or report that the graph is $k$-edge connected.*

**Proof.** First, we apply Lemma 25, paying $O(k\sqrt{n})$ time per update, but getting a sparse graph with $\tilde{O}(\sqrt{n})$ edges. For this graph, using Lemma 20, we maintain a greedy tree packing $\mathcal{T}$ of size $k^7 \log^4 n$. By Theorem 9, there is a min-cut which is a tree cut of some tree in $\mathcal{T}$. We find such a min-cut applying Proposition 24 to each tree in the tree packing, returning the smallest tree cut over all the trees. ∎

We now show how to maintain a near-minimal cut for arbitrarily high edge connectivity.

**Theorem 27.** *For a dynamic graph $G$, w.h.p., we can maintain a near-minimal cut of size $(1+o(1))\lambda$, supporting updates in $\tilde{O}(\sqrt{n})$ time per edge insertion or deletion. Here we only maintain the size of the near-minimal cut within a factor $(1\pm o(1))$ and a spanning tree of each side. If we increase the update time to $\tilde{O}(\sqrt{m})$, we can maintain the exact size of the near-minimal cut and list the cut edges in $O(\log n)$ time per edge.*

**Proof.** First we ignore the listing of the edges. As in § 3.1, we consider the $\log n$ subgraphs $H_i$ where each edge of $G$ participates independently in $H_i$ with probability $1/2^i$. We apply Theorem 26 to each $H_i$, maintaining its min-cut up to size $\log^2 n$. Let $j$ be the minimum value of $i$ such that $\lambda_{H_i} < \log^2 n$. Then the min-cut of $H_j$ is our near-minimal cut of $G$, and in $G$, this cut has size $(1 \pm o(1))2^j \lambda_{H_i}$. The correctness follows from Lemma 10.

To get the exact size of the cut found and to list the cut edges, for each tree $T$ in any of the tree packings $\mathcal{T}_i$ of $H_i$, we make a separate maintenance of all the edges of $G$ as in Lemma 22 and 23. This increases the update cost to $\tilde{O}(\sqrt{m})$, but now, when we have decided on a tree cut, we have both its size in $G$ and the ability to list its edges. ∎

Theorems 26 and 27 cover Theorem 11. This completes the description of our fully-dynamic min-cut algorithms.

### 6.1. Partially dynamic min-cut algorithms

In a purely incremental or decremental setting, we can reduce the total time over all the updates. In the incremental setting, when we have no edge deletions, the witness from Lemma 25 never accepts more than $nk$ inserted edges, and edges that do not get into the witness can be excluded in constant time. Hence, by Theorem 26, starting with an empty graph and inserting $m$ edges, we can maintain up to polylogarithmic min-cuts in $\tilde{O}(n\sqrt{n}+m)$ total time. A randomized reduction of Thorup [21] gives a corresponding result in the decremental case where no edge is inserted. There we can delete the $m$ edges of a graph while maintaining up to polylogarithmic min-cuts in $\tilde{O}(n\sqrt{n}+m)$ total expected time. Applying the above technique to each of the sampled subgraphs $H_i$ in the proof of Theorem 27, we get corresponding time bounds for near-minimal cuts with arbitrary edge connectivity.

## 7. Concluding remarks an open problems

First, as our main combinatorial result, we showed that in a graph with polylogarithmic edge connectivity, a greedy tree packing with a sufficiently large but polylogarithmic number of trees would have a min-cut crossed once by some tree.

The tree packing itself could be maintained dynamically with known techniques. Next we showed how to maintain the minimum cut crossed once by a given tree in the tree packing. Putting the results together, we got our fully-dynamic min-cut results: deterministic maintenance of min-cuts of up to polylogarithmic size and randomized maintenance of near-minimal cuts of arbitrary size. Our $\tilde{O}(\sqrt{n})$ update time matches, within log-factors, the best update time for 1- and 2-edge connectivity.

Our work leaves several natural open problems:

*How large should a greedy tree packing be to guarantee that some min-cut is crossed once by some tree?* Theorem 9 says that $\omega(\lambda^7 \log^3 m)$ trees will do, but can we get a substantially better bound? This question is interesting from a purely combinatorial view-point, and a positive answer would immediately reduce the number of log-factors hidden in our $\tilde{O}(\sqrt{n})$ time bound.

*For a dynamic tree in a dynamic graph, can we in polylogarithmic amortized time maintain the minimum tree cut up to polylogarithmic size?* Recall here that a tree cut is a cut crossed once by a given tree. This would give us polylogarithmic amortized time bounds for all our algorithms. Holm et al. [13]

have shown how to get polylogarithmic amortized time bounds maintaining a spanning forest including all bridges, that is, tree cuts of size 1. However, their algorithm gets to pick its own forest and that is crucial for their amortization. In our case, it is a tree packing that picks and changes the trees, and then their approach breaks down.

*Is there an efficient dynamic algorithm that maintains exact edge connectivity and min-cuts of arbitrary size?* In the exact case, our approach is only relevant for very small edge connectivity. For example, for $\lambda = n^{1/7}$, Theorem 9 would require a tree packing with a super-linear number of trees, and then our update time would end up much slower than computing a min-cut from scratch with Karger's [15] near-linear time algorithm.

## Acknowledgment

I would like to thank Neal Young for explaining me the intuition behind the analysis from [25]. Also, I would like to thank the referees from *Combinatorica* for some very thorough and useful comments.

## References

[1] S. Alstrup, J. Holm, K. de Lichtenberg and M. Thorup: Maintaining information in fully dynamic trees with top trees, *ACM Trans. Algorithms* **1(2)** (2005), 243–264.

[2] T. C. Biedl, P. Bose, E. D. Demaine and A. Lubiw: Efficient algorithms for Petersen's matching theorem, *J. Algorithms* **38** (2001), 110–134.

[3] L. G. Valiant and D. Angluin: Fast probabilistic algorithms for Hamiltonian circuits and matchings, *J. Comput. System Sci.* **18** (1979), 155–193.

[4] Y. Dinitz and R. Nossenson: Incremental maintenance of the 5-edge-connectivity classes of a graph, in *Proc. 7th Scandinavian Workshop Algorithms Theory, LNCS 1851*, pages 272–285, 2000.

[5] D. Eppstein, Z. Galil and G. F. Italiano: Dynamic graph algorithms, in *Algorithms and Theory of Computation Handbook*, chapter 8, CRC Press, 1999.

[6] D. Eppstein, Z. Galil, G. F. Italiano and A. Nissenzweig: Sparsification – a technique for speeding up dynamic graph algorithms, *J. ACM* **44(5)** (1997), 669–696. Announced at FOCS'92.

[7] D. Eppstein, Z. Galil, G. F. Italiano and T. H. Spencer: Separator-based sparsification II: Edge and vertex connectivity; *SIAM J. Comput.* **28** (1998), 341–381.

[8] J. Feigenbaum and S. Kannan: Dynamic graph algorithms, in Kenneth H. Rosen, editor, *Handbook of Discrete & Combinatorial Mathematics*, chapter 17.1, pages 1142–1148, CRC Press, 2000.

[9] G. N. Frederickson: Data structures for on-line updating of minimum spanning trees, with applications; *SIAM J. Comput.* **14(4)** (1985), 781–798. Announced at STOC'83.
[10] G. N. Frederickson: Ambivalent data structures for dynamic 2-Edge-Connectivity and $k$ smallest spanning trees, *SIAM J. Comput.* **26(2)** (1997), 484–538. Announced at FOCS'91.
[11] A. V. Goldberg, M. D. Grigoriadis and R. E. Tarjan: Use of dynamic trees in a network simplex algorithm for the maximum flow problem, *Math. Programming* **50** (1991), 277–290.
[12] M. R. Henzinger and V. King: Randomized dynamic graph algorithms with polylogarithmic time per operation, *J. ACM* **46** (1999), 502–536. Announced at STOC'95.
[13] J. Holm, K. Lichtenberg and M. Thorup: Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge and biconnectivity; *J. ACM* **48(4)** (2001), 723–760. Announced at STOC'98.
[14] D. R. Karger: Random sampling in cut, flow, and network design problems; *Math. Oper. Res.* **24(2)** (1999), 383–413.
[15] D. R. Karger: Minimum cuts in near-linear time, *J. ACM* **47(1)** (2000), 46–76.
[16] H. Nagamochi and T. Ibaraki: Linear time algorithms for finding a sparse $k$-connected spanning subgraph of a $k$-connected graph, *Algorithmica* **7** (1992), 583–596.
[17] C. St. J. A. Nash-Williams: Edge disjoint spanning trees of finite graphs, *J. London Math. Soc.* **36(144)** (1961), 445–450.
[18] J. Petersen: Die Theorie der regulären Graphs, *Acta Mathematica* **15** (1891), 193–220.
[19] S. A. Plotkin, D. B. Shmoys and É. Tardos: Fast approximation algorithms for fractional packing and covering problems, *Mathematics of Operations Research* **20** (1995), 257–301.
[20] D. D. Sleator and R. E. Tarjan: A data structure for dynamic trees, *J. Comput. Syst. Sc.* **26(3)** (1983), 362–391. Announced at STOC'81.
[21] M. Thorup: Decremental dynamic connectivity, *J. Algorithms* **33** (1999), 229–243.
[22] M. Thorup: Fully-dynamic min-cut, in *Proc. 33rd STOC*, pages 224–230, 2001.
[23] M. Thorup and D. Karger: Dynamic graph algorithms with applications (invited talk), in *Proc. 7th Scandinavian Workshop Algorithms Theory, LNCS 1851*, pages 1–9, 2000.
[24] W. T. Tutte: On the problem of decomposing a graph into $n$ connected factors, *J. London Math. Soc.* **36** (1961), 221–230.
[25] N. Young: Randomized rounding without solving the linear program, in *Proc. 6th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 170–178, 1995.

Mikkel Thorup

*AT&T Labs – Research*
*Shannon Laboratory*
*180 Park Avenue*
*Florham Park, NJ 07932*
*USA*
mthorup@research.att.com