Fully-dynamic min-cut

Mikkel Thorup
AT&T Labs - Research
180 Park Avenue
Florham Park, NJ 07932, USA
mthorup@research.att.com

ABSTRACT

We show that we can maintain up to polylogarithmic edge connectivity for a fully-dynamic graph in $\tilde{O}(\sqrt{n})$ 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 min-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

In this paper, we consider a fully-dynamic graph $G = (V, E)$, $n = |V|$, $m = |E|$, which is updated by edge insertions and deletions. Our goal is to maintain the *edge connectivity* λ denoting the minimum number of edges whose removal disconnect the graph. These edges define a *min-cut*, i.e. a partitioning of the vertex sets into two sets, called *sides*, connected by the smallest possible number of edges. We will also want to maintain a min-cut with a pointer to a tree spanning one of the sides, and with ability to list the cut edges.

Edge connectivity is a basic sensitivity measure for graphs, and maintaining a min-cut, allows us to point to a place where the graph is closest to falling apart.

For 1-edge connectivity and 2-edge connectivity, the best bounds are $O(\sqrt{n})$ per update [5, 7, 8]. For 3-edge and 4-edge connectivity, the best bound are $O(n^{2/3})$ and $O(n\alpha(n))$ [5]. For the special case of planar graphs, however, we can do 3-edge and 4-edge connectivity [6] in $O(\sqrt{n})$ amortized time per operation. For general edge connectivity λ, combining the sparsification in [5], the certificates in [12], and the Monto Carlo type randomized static min-cut algorithm in [11] gives an $\tilde{O}(\lambda n)$ time bound per update. Finally, [17] implies a Monto Carlo type randomized factor $2 + o(1)$ approximation 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 $1 + o(1)$. The algorithms also maintain a concrete (approximate) min-cut. However, in the general approximate case, to list the cut edges in $O(\log n)$ time per edge, the update time increases to $\tilde{O}(\sqrt{m})$.

We note that the result from [17] is stated in terms of polylogarithmic amortized bounds. The results of the current paper do not provide any better fully-dynamic amortized time bounds, and we leave this as a major open problem.

Concerning partially dynamic algorithms, the strongest result is that we maintain 5-edge connectivity incrementally, ending with m edges, in $O(m+n\log^2 n)$ total time [4]. Using general reductions from [16], our result immediately implies that we can do decremental (incremental) connectivity starting (ending) with m edges in $\tilde{O}(n^{3/2} + m)$ total time.

1.1 Techniques

The main obstacle in dealing with edge connectivity > 2 is that an arbitrary spanning tree may cross a min-cut many times. Karger [11] has pointed out that a certain Lagrangian greedy tree packing technique studied implicitly in [14, 19] 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, 5, 7, 8, 17].

1.2 Notation

$G = (V, E)$, $V(G) = V$, $E(G) = E$, $\lambda(G) = \lambda$. If $\mathcal{P}$ partitions V, $G/\mathcal{P}$ denotes G with each set of $\mathcal{P}$ contracted. By a $\mathcal{P}$-cut, we mean a cut whose one side 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 it. The *relative load* is then $\ell^{\mathcal{T}}(e) = L^e(\mathcal{T})/|\mathcal{T}|$. The value of a packing is:

$$\text{pack-val}(\mathcal{T}) = 1/\max_{e \in E(G)} \ell^{\mathcal{T}}(e)$$

We let τ denote the maximal value of a tree packing.

For each partition $\mathcal{P}$ of $V(G)$, we define its *cut value* as

$$\text{cut-val}(\mathcal{P}) = \frac{|E(G/\mathcal{P})|}{|\mathcal{P}| - 1}$$

Since any spanning tree T has $|\mathcal{P}| - 1$ edges between the sets $\mathcal{P}$,

$$\begin{aligned}\text{pack-val}(\mathcal{T}) &\leq \frac{|\mathcal{T}|}{\sum_{e \in E(G/\mathcal{P})} \ell(e, \mathcal{T})/|E(G/\mathcal{P})|} \\ &\leq \text{cut-val}(\mathcal{P})\end{aligned}$$

In fact $\mathcal{T}$ and $\mathcal{P}$ can be picked so we get equality:

THEOREM 1 (TUTTE, NASH-WILLIAMS, 1961 [18, 13]). *The maximum value τ of a tree packing equals the minimum cut value of a partition*[1].

COROLLARY 2 (KARGER [11]). $\lambda/2 < \tau \leq \lambda$.

PROOF. By Theorem 1, we may choose $\mathcal{P}$ with $\text{cut-val}(\mathcal{P}) = \tau$. Then

$$\tau > |E(G/\mathcal{P})|/|\mathcal{P}| \geq \lambda/2 \qquad (1)$$

The last relation follows because $2|E(G/\mathcal{P})|/|\mathcal{P}|$ is the average value of a $\mathcal{P}$-cut with one side being a set of $\mathcal{P}$. $\square$

2.1 Greedy tree packings

A tree packing $\mathcal{T} = \{T_1, ..., T_k\}$ is *greedy* if each T_i is a minimal spanning tree with respect to the loads of $\{T_1, ..., T_{i-1}\}$. As pointed out in [17], the work in [14, 19] implies

LEMMA 3 (YOUNG). *A greedy tree packing $\mathcal{T}$ with $\geq 3\lambda \ln m/\varepsilon^2$ trees has* $\text{pack-val}(\mathcal{T}) \geq (1 - \varepsilon)\tau$.

In combination with Corollary 2, we get

$$(1 - \varepsilon)\lambda/2 < (1 - \varepsilon)\tau \leq \text{pack-val}(\mathcal{T}) \leq \tau \leq \lambda. \qquad (2)$$

Previous applications. Karger used greedy tree packings to find a tree crossing a min-cut at most twice. Also, Thorup and Karger [17] have shown that we can maintain greedy tree packings dynamically, thus getting a factor $2 + o(1)$ approximation to the edge connectivity of a graph.

[1]Really they talk about edge disjoint trees, but if we give each edge multiplicity $n!$, we get the stated version.

3. OUR NEW CONTRIBUTION

Our main technical contribution is a purely combinatorial result:

THEOREM 4. *A greedy tree packing with $\omega(\lambda^7 \log^3 m)$ trees contains a tree crossing some min-cut only once.*

In our case, we are only interested in λ of polylogarithmic size, Before going further, we stipulate that our *Theorem 4 is not implied by Lemma 3*. More precisely, Lemma 3 will only guarantee an error in the order of $1/(\lambda \log n)^{\Theta(1)}$. To see that this does not in itself suffice, consider a cycle over n vertices where one edge has multiplicity d. The single path P consisting of all single edges is a tree packing of value 1. The partitioning making all vertices singletons except for the two end-points of the multiplied edge has cut value $\frac{n-1}{n-2}$, so our tree packing $\{P\}$ is only a factor $1/(n-2)$ from being optimal. Nevertheless, all cuts crossed once by P have value $d + 1$ whereas the min-cut has value $\lambda = 2$.

By combination of ideas and techniques from [1, 5, 7, 8, 17], we then show

LEMMA 5. *In $\tilde{O}(\sqrt{n})$ time per edge insertion or deletion in a fully-dynamic graph, we can maintain a greedy tree packing $\mathcal{T}$ of polylogarithmic size plus a minimum cut among all cuts crossed once by some tree in $\mathcal{T}$.*

This immediately implies

THEOREM 6. *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.*

PROOF. We are looking to maintain min-cuts up to some maximal value $\lambda_{\max} = \log^{O(1)} n$. Using Lemma 5, we maintain a tree packing $\mathcal{T}$ with $\lambda_{\max}^7 \log^4 n$ trees. If $\lambda \leq \lambda_{\max}$, Theorem 4 ascertains that we find a min-cut. Otherwise, the cut found will be of size above $\lambda_{\max}$, in which case we report that the edge connectivity is above $\lambda_{\max}$. $\square$

As pointed out in [17], the sampling based sparsification from [10] implies that if we can maintain min-cuts of polylogarithmic size, we can also maintain near-minimum cuts of arbitrary size. That is,

COROLLARY 7. *In $\tilde{O}(\sqrt{n})$ time per edge insertion or deletion in a fully-dynamic graph, we can maintain a cut of size within a factor $(1 + o(1))$ of the edge connectivity. To list the cut edges in $O(\log n)$ time per edge, the update time increases to $\tilde{O}(\sqrt{m})$.*

4. MAINTAINING EDGE-CONNECTIVITY

In this section, as brief warm-up, we show a comparatively simple way of determining the edge connectivity without identifying the min-cut, not using technically hard Theorem 4.

Set $\varepsilon = 1/(5\lambda)$ and make a greedy tree packing of size $3\lambda \ln m/\varepsilon^2$ as in Lemma 3.

Let λ_1 by $2\text{pack-val}(\mathcal{T})$ rounded to the nearest integer. Further, let λ_2 by the minimal size of a cut crossed once by a tree in $\mathcal{T}$.

LEMMA 8. $\lambda = \min\{\lambda_1, \lambda_2\}$.

PROOF. From (2) we have that $\lambda < 2\text{pack-val}(\mathcal{T})/(1 - 1/(5\lambda)) \leq 2\text{pack-val}(\mathcal{T}) + 1/2$. Consequently, $\lambda \leq \lambda_1$. Now, if $\text{pack-val}(\mathcal{T}) > \lambda/2$, the average tree in $\mathcal{T}$ crosses each min-cut strictly less than twice, so $\lambda_2 = \lambda$. Otherwise $2\text{pack-val}(\mathcal{T}) \leq \lambda$, and since λ is integer, this implies $\lambda_1 \leq \lambda$. $\square$

Thus, all we need to maintain polylogarithmic edge connectivity is Lemma 3 and Lemma 5. However, if the answer is $\lambda_1 < \lambda_2$, we do not get any information about were the min-cut is. Theorem 4 states that if we pack some more trees, eventually we are guaranteed $\lambda_2 = \lambda$.

5. PROOF OF THEOREM 4

We want to show that a sufficiently large tree packing is going to contain a tree crossing some min-cut only once. To illustrate our basic idea, we first study and optimal tree packing $\mathcal{T}^*$ together with an optimal partition $\mathcal{P}^*$, i.e. with $\tau = \text{cut-val}(\mathcal{P}^*) = \text{pack-val}(\mathcal{T}^*)$. By (2), we have $\tau > \lambda/2$, and this on its own implies that any min-cut on the average is cut strictly less than twice by trees in $\mathcal{T}^*$, as desired. However, the gap between τ and $\lambda/2$ can be $O(1/n)$ is illustrated in §3. We need to base our argument on a more substantial gap.

FACT 9. *If $\tau = \text{cut-val}(\mathcal{P}^*) = \text{pack-val}(\mathcal{T}^*)$, all edges e in $G/\mathcal{P}^*$ have $\ell^{\mathcal{T}^*}(e) = 1/\tau$. 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 is at least $(|\mathcal{P}^*|-1)/|E(G/\mathcal{P}^*)| = 1/\tau$, 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 $E(G/\mathcal{P}^*)$. $\square$

If all $\mathcal{P}^*$-cut are min-cuts, we take any tree $T \in \mathcal{T}^*$. By Fact 9, $T/\mathcal{P}^*$ is a tree over $E(G/\mathcal{P}^*)$, and any leaf edge of $T/\mathcal{P}^*$ is the unique edge crossing the $\mathcal{P}^*$-cut defined by the leaf. Otherwise, some $\mathcal{P}^*$-cut is not a min-cut. Since $T/\mathcal{P}^*$ is a tree for all $T \in \mathcal{T}$, on the average each $\mathcal{P}^*$-cut is crossed less than twice. Suppose for a contradiction that all min-cuts are crossed at least twice. To get an average crossing below twice, there must be some $\mathcal{P}^*$-cut B which is non-minimal, and which is crossed less than twice on the average. This, in turns, means that the average relative edge load in B is $< 2/(\lambda+1)$. Hence some edge $f \in B$ has $\ell^{\mathcal{T}^*}(f) < 2/(\lambda+1)$. However, if we take any min-cut C, since all tree cross it twice, there is an edge $e \in C$ with $\ell^{\mathcal{T}^*}(e) \geq 2/\lambda$. Now, by Fact 9, all edges in $E(G/\mathcal{P}^*)$, including e and f, should have the same load, contradicting $\ell^{\mathcal{T}^*}(f) < 2/(\lambda+1)$ while $\ell^{\mathcal{T}^*}(e) \geq 2/\lambda$.

Our idea is to exploit the contradicting gap between $2/(\lambda+1)$ and $2/\lambda$ by identifying a sufficiently good partitioning with sufficiently even distribution of loads.

5.1 The analysis

The rest of this section presents a proof of Theorem 4. Throughout, we will assume all trees in our greedy tree packings cross all min-cuts at least twice, and based on this we will arrive at a contradiction for sufficiently large greedy tree packings.

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 $\text{pack-val}(\mathcal{P}^*, G) = \tau(G)$ (c.f. Theorem 1)
2. For all $e \in E(G/\mathcal{P}^*)$, set $\ell^*(e) = 1/\tau(G)$
3. For each $S \in \mathcal{P}^*$, recurse on the subgraph $G|S$ induced by S.

LEMMA 10. *There is a distribution Π of spanning trees such that for each e, $\Pr_{T\in\Pi}(e \in T) = \ell^*(e)$*

PROOF. The distribution Π is constructed in conjunction with the above recursive algorithm. Let $\mathcal{T}^*$ be an optimal tree packing for G. To pick a random tree from $\Pi(G)$, we pick a uniformly random tree $U \in \mathcal{T}^*$, plus, for each $S \in \mathcal{P}^*$, a random $T_S \in \Pi(G|S)$, and then we return $T = U/\mathcal{P}^* \cup \bigcup_{S\in\mathcal{P}^*} T_S$. Here we note $T/\mathcal{P}^*$ is a tree by Fact 9, so U is indeed a spanning tree. Further, for $e \in E(G/\mathcal{P})$, $\Pr(e \in U) = \ell^{\mathcal{T}}(e)^* = 1/\tau = \ell^*(e)$. as desired, and the correctness of the probability/loads in the subgraphs $G|S$ follows inductively. $\square$

LEMMA 11. *The values of $\tau(G)$ are non-increasing in the sense that for each $S \in \mathcal{P}$, $\tau(G|S) \geq \tau(G)$.*

PROOF. Suppose for a contradiction that we for some $G|S$ have a partition $\mathcal{P}_S$ with $\text{cut-val}(\mathcal{P}_S, G|S) < \tau(G)$. then $\mathcal{P}' = (\mathcal{P} \setminus \{S\}) \cup \mathcal{P}_S$ would have $\text{cut-val}(\mathcal{P}', G) < \tau(G)$. $\square$

For $\circ = <, >, \leq, \geq, =$ and $x = \mathcal{T}, *$, define

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

We will now prove a natural generalization of Lemma 3

LEMMA 12. *A greedy tree packing $\mathcal{T}$ with $\geq 6\lambda \ln m/\varepsilon^2$ trees has $|\ell^{\mathcal{T}}(e) - \ell^*(e)| \leq \varepsilon/\lambda$ for all $e \in E(G)$.*

PROOF. The proof is modeled over the use of conservative estimators in Young [19]. The proof idea is rather subtle, and we refer the reader to [19] for more intuition.

Consider an arbitrary threshold $\ell \leq 1/\tau$. We want to show

$$\max_{e\in E^*_{\leq\ell}} \ell^{\mathcal{T}}(e) \leq \ell + \varepsilon/\lambda \tag{3}$$

Let $t = |\mathcal{T}|$ and let $\mathcal{T}[i]$ be the first i trees in $\mathcal{T}$. For any tree packing $\mathcal{S}$ and $r \geq |S|$, define the random variable $X(\mathcal{S}, r)$ by $\Pr(X(\mathcal{S}, r) > x) = \sum_{e\in E^*_{\leq\ell}} \Pr(L^{\mathcal{S}}(e) + B(r - |\mathcal{S}|, \ell) > x)$. Then, $X(\mathcal{T}[t], t) = X(\mathcal{T}, t) = \max_{e\in E^*_{\leq\ell}} t\ell^{\mathcal{T}}(e)$. For the other extreme, $\Pr(X(\mathcal{T}[0], t) > x) = m\Pr(B(t, \ell) > x)$. Since $\ell \leq 1/\tau < 2/\lambda$ and $t \geq 6\lambda \ln m/\varepsilon^2$, standard Chernoff bounds (see e.g. [3]) give $E(X[0], t) \leq t\ell + t\varepsilon/\lambda$. Hence (3) follows if for $i = 0, ..., p-1$, we can show that $E(X(\mathcal{T}[i+1], t)) \leq E(X(\mathcal{T}[i], t)$.

Now, if we pick a random tree U from Π from Lemma 10, for each $e \in E^*_{\leq\ell}$, we know $\Pr(e \in U) = \ell^*(e) \leq \ell$. Hence $E_{U\in\Pi}(X(\mathcal{T}[i]\bar{U}, t)) \leq E(X(\mathcal{T}[i], t))$.

By definition, if $U \in \Pi$, U spans each component of $E^*_{\leq\ell}$. Conditioned on this spanning, we minimize $E_{U\in\Pi}(X(\mathcal{T}[i]U, t))$ with a tree U that contains a minimum spanning tree of each component of $E^*_{\leq \ell}$ with respect to the loads of $\mathcal{T}[i]$.

Consider now the $i+1$th tree T in $\mathcal{T}$. We note that T may not span the components of $E^*_{\leq \ell}$, but for every component C, $T \cap C$ is contained in a subgraph of a minimum spanning tree of the component. Hence $E(X(\mathcal{T}[i+1], t)) \leq E_{U \in \Pi}(X(\mathcal{T}[i]U, t) \leq E(X(\mathcal{T}[i], t)$, as desired. This completes the proof of (3).

A nearly symmetric proof is given for

$$\min_{e \in E^*_{\geq \ell}} \ell^{\mathcal{T}}(e) \leq \ell - \varepsilon/\lambda \qquad (4)$$

This time, we define the random variable $Y(\mathcal{S}, r)$ by $\Pr(Y(\mathcal{S}, r) < x) = \sum_{e \in E^*_{\geq \ell}} \Pr(L^{\mathcal{S}}(e) + B(r - |\mathcal{S}|, \ell) < x)$. Again, $Y(\mathcal{T}[t], t) = \max_{e \in E^*_{\geq \ell}} t\ell^{\mathcal{T}}(e)$, and Chernoff bounds give $E(Y[0], t) \geq t\ell - t\varepsilon/\lambda$.

Also, for each $e \in E^*_{\geq \ell}$, $\Pr_{U \in \Pi}(e \in U) = \ell^*(e) \geq \ell$. Hence $E_{U \in \Pi}(Y(\mathcal{T}[i]U, t)) \geq E(Y(\mathcal{T}[i], t))$.

Finally, we need to show that the greedy tree packing does at least as good a job as $\mathcal{P}$ in maximizing the minimal load. By definition, if $U \in \Pi$, $U/E^*_{<\ell}$ is a tree. Conditioned on this, a minimum spanning tree is best possible. The definition of greedy implies for the next tree T that $T/E^*_{<\ell}$ contains a minimum spanning tree. Hence $E(Y(\mathcal{T}[i+1], t) \geq E_{U \in \Pi}(Y(\mathcal{T}[i]U, t)) \geq E(Y(\mathcal{T}[i], t))$, so (4) follows, completing the proof of the lemma. $\square$

Below $\varepsilon, \alpha > 0$ will be fixed later. We are considering a tree packing $\mathcal{T}$ with at least $6\lambda \ln m/\varepsilon^2$ trees, as in Lemma 12. We will refer to the values $\ell^{\mathcal{T}}(e)$ as *concrete* relative loads as opposed to the *ideal* relative loads $\ell^*(e)$.

Let ℓ_0 be the minimal value such $E^*_{\geq \ell_0} \supseteq MC$, where MC is the set of all edges in minimum cuts. For $i > 1, 2, ...$, let $\ell_i = \ell_0 - i\varepsilon/\lambda$. By Lemma 12, for $i \geq 0$,

$$E^*_{\geq \ell_i} \subseteq E^{\mathcal{T}}_{\geq \ell_{i+1}} \subseteq E^*_{\geq \ell_{i+2}} \qquad (5)$$

By counting, there has to be some $i \leq 2\log_{1+\alpha} m$ such that $|E^*_{\geq \ell_{i+2}} \setminus MC| \leq (1+\alpha)|E^*_{\geq \ell_i} \setminus MC|$, and then

$$|E^{\mathcal{T}}_{\geq \ell_{i+1}} \setminus MC| \leq (1+\alpha)|E^*_{\geq \ell_i} \setminus MC|. \qquad (6)$$

Note that (5) and (6) holds regardless of the concrete tree packing $\mathcal{T}$, as long as it has at least $6\lambda \ln m/\varepsilon^2$ trees from Lemma 12. Hence, (5) and (6) will be preserved under later expansions of $\mathcal{T}$.

LEMMA 13. *Assuming that all $T \in \mathcal{T}$ cross each min-cut twice, $\ell_i \geq 2/\lambda - \frac{O(\varepsilon/\alpha \, \log n)}{\lambda}$.*

PROOF. First, we argue that ℓ_0 is close to $1/\tau$. Let C be a min-cut intersecting $E_{=\ell_0}$. Clearly $C \cap E_{=\ell_0}$ is a cut of a component of $E_{\leq \ell_0}$, so if $|C \cap E_{=\ell_0}| \leq \lambda/2 < \tau$, this would contradict Lemma 11. Consequently, the average over $T \in \mathcal{T}$ of the number of crossings of C is

$$< (\ell_0 + \varepsilon/\lambda)\lambda/2 + (1/\tau + \varepsilon/\lambda)\lambda/2$$

By assumption, this number is ≥ 2, and $\tau > \lambda/2$, so

$$(\ell_0 + \varepsilon/\lambda)\lambda/2 + (2/\lambda + \varepsilon/\lambda)\lambda/2 \geq 2.$$

This simplifies to $\ell_0 \geq (1-\varepsilon)2/\lambda$, and hence $\ell_i \geq 2/\lambda - (2+i)\varepsilon/\lambda = 2/\lambda - \frac{O(\varepsilon/\alpha \, \log n)}{\lambda}$. $\square$

Let $\mathcal{P}$ be the partition whose sets are the components of $E^*_{<\ell_i}$.

Assume that some $\mathcal{P}$-cut is not a min-cut.. We are going to consider a greedy tree packing $\mathcal{T}'$ expanding $\mathcal{T}$.

LEMMA 14. *There exists an edge $f \in E^*_{\geq \ell_i}$ with $\ell^{\mathcal{T}' \setminus \mathcal{T}}(f) \leq 2/(\lambda+1) + \alpha$.*

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

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

The sum of crossings by T of all non-minimal $\mathcal{P}$-cuts is

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

On the other hand, the sum of the number of edges over all all non-minimal $\mathcal{P}$-cuts is $\geq (\lambda+1)2(p-q)$ and $\geq |E^*_{\geq \ell_i} \setminus MC|$. Thus, counting edges in two non-minimal $\mathcal{P}$-cuts twice, on the average, T uses an edge in a non-minimal $\mathcal{P}$-cut $\leq 2/(\lambda+1) + 2\alpha$ times. Since this average is guaranteed by all trees T added to $\mathcal{T}'$, starting from $\mathcal{T}$, there must exists an edge $f \in E^*_{\geq \ell_i}$ in some non-minimal $\mathcal{P}$-cut with $\ell^{\mathcal{T}' \setminus \mathcal{T}}(f) \leq 2/(\lambda+1) + \alpha$. $\square$

Let $\mathcal{T}'$ be the continuation to thrice the size of the tree packing $\mathcal{T}$, which in terms was of size at least $6\lambda \ln m/\varepsilon^2$. Then, for any edge $e \in E$, Lemma 12 implies $|\ell^{\mathcal{T}' \setminus \mathcal{T}}(e) - \ell^*(e)| \leq |\ell^{\mathcal{T}}(e) - \ell^*(e)| + |\ell^{\mathcal{T}'}(e) - \ell^*(e)| \leq 1/3\varepsilon/\lambda + \sqrt{1/3}\varepsilon/\lambda \leq \varepsilon/\lambda$. In particular, we have

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

for the $f \in E^*_{\geq \ell_i}$ from Lemma 14, which had $\ell^{\mathcal{T}' \setminus \mathcal{T}}(f) \leq 2/(\lambda+1) + \alpha$. However, $\ell^*(f) \geq \ell_i$, and by Lemma 13, $\ell_i \geq 2/\lambda - \frac{O(\varepsilon/\alpha \, \log n)}{\lambda}$. Thus we have

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

To balance, we set $\alpha = \frac{\varepsilon/\alpha \, \log n}{\lambda}$, and to reach a contradiction, we set $\alpha = o(1/\lambda^2)$. That is, we get $\varepsilon = \lambda\alpha^2/\log n = o(1/(\lambda^3 \log n)$.

Plugging $\varepsilon = o(1/(\lambda^3 \log n)$ back in Lemma 12, we see that the contradiction is reached when $\mathcal{T}$ and $\mathcal{T}'$ have $\omega(\lambda^7 \log^3 n)$ trees. This completes the proof of Theorem 4 for the case where some $\mathcal{P}$-cut is not a min-cut.

Assume that all $\mathcal{P}$-cuts are min-cuts.. In this case, $MC = E^*_{\geq \ell_0}$. If $E^*_{\geq \ell_2} = MC$ as well, $E^{\mathcal{T}}_{\geq \ell_1} = MC$, we just consider one tree T added to $\mathcal{T}$. Then $T/\mathcal{P}$ is a tree spanning $G/\mathcal{P}$, and any leaf edge in $T/\mathcal{P}$, is the only edge crossing the $\mathcal{P}$-cut whose one side is defined by the leaf. Since all $\mathcal{P}$-cuts are assumed to be min-cuts, we are done.

Now, if $E^*_{\geq \ell_2} \neq MC$, then we change $\mathcal{P}$ to be the partition defined by the components of $E^*_{<\ell_2}$. Now, some $\mathcal{P}$-cut is not a min-cut, and we can reuse our previous argument for this case. This completes the proof of Theorem 4.

6. MAINTAINING THE TREES

In this section, we will prove Lemma 5, thus showing how to maintain a tree packing of polylogarithmic size, including the minimum cut crossed once by some tree in the packing.

For a dynamic graph, we maintain a greedy tree packing of size k as a list of k dynamic minimum spanning trees structures, each with weights being the loads over the previous spanning trees. Thorup and Karger [17] have shown that each edge insertion or deletions translates into at most k^2 load changes for the spanning trees.

In [17] they used the minimum spanning tree structure from [9] with polylogarithmic amortized time bounds. Here instead we use minimum spanning tree data structure from [5, 7] with updates in $O(\sqrt{n})$ worst case time. Thus, we get

LEMMA 15 ([5, 7, 17]). *We can maintain a greedy tree packing of polylogarithmic for a fully-dynamic graph in $\tilde{O}(\sqrt{n})$ time per edge insertion or deletion.*

Our remaining problem is now, independently for each 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 a dynamic trees.

6.1 Top Trees

Top trees were introduced by Alstrup et al. [1]. It is a variant of Frederickson's topology trees [8] with a simpler interface for dynamic trees of unbounded degree. The version used here essentially stems from [2]. The edges from G that are not part of the dynamic tree T are called non-tree edges.

A top tree is defined based on a pair consisting of a tree T and a set ∂T of at most 2 nodes from T, called *external boundary nodes*. Given $(T, \partial T)$, any connected subtree C of T has a set $\partial_{(T,\partial T)}C$ of *boundary nodes* which are the nodes of C that are either in ∂T or incident to an edge in T leaving C. The subtree C is called a *cluster* of $(T, \partial T)$ if it has at most two boundary nodes. Then T is itself a cluster with $\partial_{(T,\partial T)}T = \partial T$. Also, if A is a subtree of C, $\partial_{(C,\partial_{(T,\partial T)}C)}A = \partial_{(T,\partial T)}A$, so A is a cluster of $(C, \partial_{(T,\partial T)}C)$ if and only if A is a cluster of $(T, \partial T)$. Since $\partial_{(T,\partial T)}$ is a canonical generalization of ∂ from T to all subtrees of T, we will use ∂ as a shorthand for $\partial_{(T,\partial T)}$ in the rest of the paper.

A *top tree* $\mathcal{T}$ over $(T, \partial T)$ is a binary tree such that:

1. The nodes of $\mathcal{T}$ are clusters of $(T, \partial T)$.
2. The leaves of $\mathcal{T}$ are clusters with $\leq \sqrt{m}$ vertices and incident non-tree edges from G. If an end-point of a non-tree edge is a boundary node of a leaf cluster, one such leaf cluster is chosen as the designated owner. Hence, each non-tree edge is viewed as having end-points in at most 2 leaf clusters.
3. If C is an internal node of $\mathcal{T}$ with children A and B, then $C = A \cup B$ and A and B are *neighbors*, that is they share a single node (see Figure 1).
4. The root of $\mathcal{T}$ is T itself.

The top trees over the trees in our forest are maintained under the following operations:

Link(v, w) where v and w are in different trees, links these trees by adding the edge (v, w) to our dynamic forest.

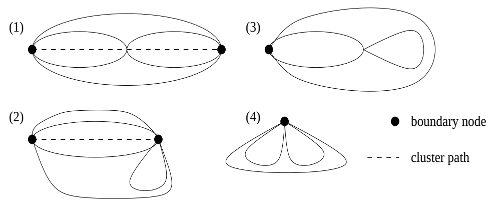

Figure 1: Combining two clusters in one. The boundary nodes and cluster paths in the figure are for the resulting cluster.

Cut(e) removes the edge e from our dynamic forest.

Expose(v, w) where v and w are in the same tree T, makes v and w the external boundary nodes of T. Moreover, Expose returns the new root cluster of the top tree over T.

Expose can also be called with one or zero nodes as arguments if we want less than two external boundary nodes.

In general, Link and Cut makes the set of external boundary nodes for the resulting trees empty. Every update of the top tree can be implemented as a sequence of the following two operations:

Create(L) where L is a cluster with $\leq \sqrt{m}$ vertices and incident non-tree edges, creates a new top tree with L the only cluster.

Merge(A, B) where A and B are the root-clusters of two top trees $\mathcal{T}_A$ and $\mathcal{T}_B$. Creates a new cluster $C = A \cup B$ and makes it the common root of A and B, thus turning $\mathcal{T}_A$ and $\mathcal{T}_B$ into a single new top tree $\mathcal{T}_C$. Finally, the new root cluster C is returned.

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 $\mathcal{T}_B$.

Destroy(L) eliminates the top tree consisting of the cluster L.

Recall that n denotes the size of the trees involved in a given update operation.

LEMMA 16 ([1, 7]). *For a dynamic forest we can maintain top trees of height $O(\log n)$ and with $O(\sqrt{m})$ cluster nodes, supporting each Link, Cut, or Expose with a sequence of $O(\log n)$ Merge and Split and $O(1)$ Create and Destroy. Here the sequence itself is identified in $O(\log n)$ time.*

Notation. If v and w are vertices in our tree T, $v \cdots w$ denotes the unique path from v to w. If a cluster C has two boundary nodes a and b, we call $a \cdots b$ the *cluster path* of C, denoted $\pi(C)$. If $|\partial(C)| < 2$, $\pi(C) = \emptyset$. Note that if A is a child cluster of C and A shares an edge with $\pi(C)$, then $\pi(A) \subseteq \pi(C)$, and then we call A a *path child* of C. In terms of boundary nodes, if C has children A and B, A is a path child of A if and only if $|\partial C| = 2$ and either $\partial A = \partial C$ (Fig. 1 (2)) or $\partial C \subset \partial A \cup \partial B$ (Fig. 1 (1)).

6.2 Maintaining cuts crossed once

We now return to our original problem of maintaining the minimum cut crossed once by our dynamic tree T. Since we are only interested in polylogarithmic min-cuts, we can apply the sparsification from [5], and hence assume $m = \tilde{O}(n)$.

We say an edge (v, w) *covers* the path $v \cdots w$ in T between its end-points. The cover $c(e)$ of a tree edge $e \in T$ is the number of non-tree edges covering it. The cut induced by removing e has size $c(e) + 1$, so $\min_{e \in T} c(e) + 1$ is exactly the minimum size of a cut crossed once by T.

In his dynamic 2-edge connectivity algorithm, Frederickson showed how to maintain whether each tree edge was covered or not [8]. Here we will show how to maintain covering up to at least $\sqrt{m}$ in $\tilde{O}(\sqrt{m})$ time per operation. By up to at least $\sqrt{m}$, we mean that if the cover is larger than $\sqrt{m}$, we may report a smaller cover $\geq \sqrt{m}$. Really, we are only interested in covering up to polylogarithmic size, so getting up to $\sqrt{m}$ is more than we need. However, $\sqrt{m}$ turns out to be the natural bound for our algorithm.

In [15] it is shown how given two vertices v and w, in $O(\log n)$ time, we can add a value to all edges between them, the value here being the cover. As described in [2] this can also be done with top trees if we want to keep things unified. It is also easy to maintain the edge in the tree with the minimum value, representing the min-cut. Thus, our main challenge is to convert changes to the dynamic tree into $\tilde{O}(\sqrt{m})$ path cover updates.

For a cluster C, we will maintain the covering up to at least $\sqrt{m}$ of all edges not in the cluster path. The basic observation is that only edges incident to C can cover $C \setminus \pi(C)$, and the way they cover is independent of the structure of $T \setminus C$. Also, we will maintain the covering of $\pi(C)$ by edges with both end-points in C. Thus, the only covering not maintained is that of edges in $\pi(C)$ by edges with one or both end-points outside C.

The above choice of covering information may seem somewhat contrived. It is, however, typical for top tree algorithms [1, 2, 9], that a careful choice of cluster information is the key to simple implementations.

Doing the above covering is easily done in $O(\sqrt{m})$ time for a newly created leaf cluster. Also, if a non-path cluster A gets merged with another cluster (c.f. the right cluster in Figure 1 (2) and (3), and both clusters in (4)), the covering within A is not affected. However, if A is a path cluster, the covering of $\pi(A)$ is affected. To deal with this, we need to store some more information.

For each pair (A, B) of clusters, let $E(A, B)$ denotes the set of non-tree edges between A and B. For each pair (L, M) of leaf clusters, we maintain the set $E(L, M)$ directly. For each pair (A, B) of clusters, we maintain the number $m(A, B) = |E(A, B)|$. So far, the maintenance is easy: Each leaf cluster has at most $\sqrt{m}$ incident edges, each of which goes to $O(\log n)$ clusters, so adding or subtracting them when a leaf cluster is created or destroyed takes $O(\sqrt{m} \log n)$ time. Also, when two clusters are merged $C = A \cup B$, for each of the $\Theta(\sqrt{m})$ other top clusters D, we just set $m(C, D) = m(A, D) + m(B, D)$. Splitting a cluster requires no action. With this information, we can easily list the set $E(C, D)$ of edges between C and D in $O(\log n)$ time per edge, simply by following the counters down the top tree.

We now return to our problem of updating the covering of the cluster path when a path cluster A gets merged with another cluster B. First, we consider the case where the resulting cluster C is a path cluster (c.f. the both clusters in Figure 1 (1) and the left cluster in (2)). We need to update $\pi(A)$ with covering of edges $(v, w) \in E(A, B)$. Viewing $\pi(A)$ as starting in the common boundary node b between A and B, each $(v, w) \in E(A, B)$ covers a possibly empty prefix of $\pi(A)$. If $|E(A, B)| > \sqrt{m}$, we will identify a set $E^-(A, B)$ of between $\sqrt{m}$ and $2\sqrt{m}$ edges from $E(A, B)$ maximizing the length of the covered prefix of $\pi(A)$. Let P is the shortest prefix of $\pi(A)$ covered by an edge in $E^-(A, B)$. For all edges in $\pi(A) \setminus P$, $E^-(A, B)$ gives the same covering as $E(A, B)$ would, and for edges in P, $E^-(A, B)$ gives a covering of at least $\sqrt{m}$. If $E(A.B) < \sqrt{m}$, we set $E^-(A, B) = E(A, B)$. Above we note that the pair (A, B) is ordered in the sense that $E^-(A, B)$ relates to the covering of $\pi(A)$ whereas $E^-(B, A)$ relates to the covering of $\pi(B)$.

Assuming we have identified $E^-(A, B)$, for each $(v, w) \in E^-(A, B)$, we want to increment the cover on $\pi(A) \cap v \cdots w$ by 1. This is easily done by incrementing the cover of $v \cdots w$ and $a \cdots b$ by $1/2$ and decrementing the cover of $v \cdots a$ by $1/2$. The set $E^-(A, B)$ is saved so that if later the cluster $C = A \cup B$ gets split, we can just reverse the above process. 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 only $O(m)$.

To identify the edges for $E^-(A, B)$, we visit the leaf clusters in A in the order indicated in Figure 2, visiting leaf clusters with the same number in an arbitrary order. Formally, the point in the ordering is that if L_1 is before L_2, $v_1 \in L_1$, and $v_2 \in L_2$, $v_1 \cdots b \cap \pi(A) \supseteq v_2 \cdots b \cap \pi(A)$. When we visit a leaf cluster L, we add all of $E(L, B)$ to $E^-(A, B)$, visiting no more leaf clusters if $|E^-(A, B)| \geq \sqrt{m}$. Since no leaf cluster has more than $\sqrt{m}$ incident edges, we end with $\sqrt{m} \leq |E^-(A, B)| < 2\sqrt{m}$, as desired. Since there are only $\sqrt{m}$ leaf clusters all together, and since each edge in $E(L, B)$ can be found in $O(\log n)$ time, including the time for covering, the whole merge takes $O(\sqrt{m} \log n)$ time.

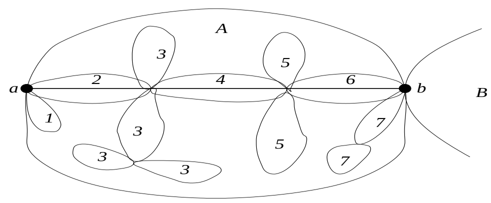

Figure 2: Combining two clusters in one. The boundary nodes and cluster paths in the figure are for the resulting cluster.

We still have one merge case left to consider; namely when a path cluster A gets merged with another cluster B and the resulting cluster C is not a path cluster (c.f. the left cluster in Figure 1 (3)). In this case, we first do exactly as in the previous case where C was a path cluster. Second, we "un-path" C, following the same recipe one more time, as if merging C with $D = V \setminus C$. The cluster $V \setminus C$ is not expected to be in the top tree, but this does not stop us from finding the relevant edges. For example, if a leaf cluster L is contained in C, $|E(L, V \setminus C)| = m - m(L, C)$, and hence, we can find edges in $E(L, V \setminus C)$ in $O(\log n)$ time per edge.

Finally, when a non-tree edge is inserted or deleted, we note that this only affects the information associated with the $O(\log n)$ clusters to which its end-points belong, and for each such cluster we can easily update the information in $O(\sqrt{m})$ time.

To present a min-cut, we just cut a tree edge e with minimal cover. Each of the resulting subtrees span one side of the cut. Moreover, if A and B are the root clusters of these new top trees, $E(A, B) \cup \{e\}$ is the set of cut edges, and we already know how to list the edges in $E(A, B)$ in $O(\log n)$ time per edge.

As mentioned previously, when dealing with polylogarithmic cuts, we can just use sparsification [5] to replace m by $\tilde{O}(n)$ in the above bounds, so together with Lemma 15, this completes the proof of Lemma 5, hence of Theorem 6.

7. OPEN PROBLEMS

Finally we would like to conclude with a list some major open problems:

1. Is it possible to avoid the $(1 \pm o(1))$ approximation factor for general edge connectivity?

2. Is it possible to get polylogarithmic amortized time, for higher edge connectivity? Currently this is done for 2-edge connectivity in [9].

3. Can we provide a better analysis for the convergence of greedy tree packing (c.f. Theorem 4). The current time bounds really end up with something like $O(\sqrt{n}\lambda^{14} \log^6 n) = \tilde{O}(\sqrt{n})$, but it seems likely that the analysis is way of, and that the true bounds are much closer to $O(\sqrt{n})$.

4. Can anything be done for general vertex connectivity?

Acknowledgment. I would like to thank Neal Young for explaining me the intuition behind the analysis from [19].

8. REFERENCES

[1] S. Alstrup, J. Holm, K. de Lichtenberg, and M. Thorup. Minimizing diameters of dynamic trees. In *Proc. 24th Int. Coll. on Automata, Languages, and Programming, LNCS 1256*, pages 270–280, 1997.

[2] S. Alstrup, J. Holm, and M. Thorup. Maintaining center and median in dynamic trees. In *Proc. 7th Scandinavian Workshop Algorithms Theory, LNCS 1851*, pages 46–56, 2000.

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

[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, G. F. Italiano, and A. Nissenzweig. Sparsification — a technique for speeding up dynamic graph algorithms. *J. ACM*, 44(5):669–696, 1997. See also FOCS'92.

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

[7] G. N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM J. Comput.*, 14(4):781–798, 1985. See also STOC'83.

[8] G. N. Frederickson. Ambivalent data structures for dynamic 2-Edge-Connectivity and k smallest spanning trees. *SIAM J. Comput.*, 26(2):484–538, April 1997. See also FOCS'91.

[9] J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. In *Proc. 30th ACM Symp. on Theory of Computing*, pages 79–89, 1998.

[10] D. R. Karger. Using randomized sparsification to approximate minimum cuts. In *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms*, pages 424–432, 1994.

[11] D. R. Karger. Minimum cuts in near-linear time. *J. ACM*, 47(1):46–76, 2000.

[12] H. Nagamochi and T. Ibaraki. Linear time algorithms for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7:583–596, 1992.

[13] C. St. J. A. Nash-Williams. Edge disjoint spanning trees of finite graphs. *J. London Math. Soc.*, 36(144):445–450, 1961.

[14] S. A. Plotkin, D. B. Shmoys, and É. Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 20:257–301, 1995.

[15] D.D. Sleator and R.E. Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sc.*, 26(3):362–391, 1983. See also STOC'81.

[16] M. Thorup. Decremental dynamic connectivity. *J. Algorithms*, 33:229–243, 1999.

[17] 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.

[18] W. T. Tutte. On the problem of decomposing a graph into n connected factors. *J. London Math. Soc.*, 36:221–230, 1961.

[19] N. Young. Randomized rounding without solving the linear program. In *Proc. 6th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 170–178, 1995.