Decremental dynamic connectivity

Mikkel Thorup

May 20, 1996

Abstract

We consider Las Vegas randomized dynamic algorithms for on-line connectivity problems with deletions only. In particular, we show that starting from a graph with m edges and n nodes, we can maintain a spanning forest during m deletions in $O(\min\{n^2, m\log n\} + \sqrt{nm}\log^{2.5} n)$ expected total time. This is amortized constant time per operation if we start with a complete graph. The deletions may be interspersed with connectivity queries, each of which is answered in constant time. The previous best bound was $O(m\log^2 n)$ by Henzinger and Thorup (1996), which covered both insertions and deletions. Our bound is stronger for $m/n = \omega(\log n)$.

The result is based on a general randomized reduction of many deletions-only queries to few deletions and insertions queries. Similar results are thus derived for 2-edge-connectivity, bipartiteness, and q-weights minimum spanning tree. For the decremental dynamic k-edge-connectivity problem of deleting the edges of a graph starting with m edges and n nodes, we get a total running time of $O(k^2n^2\text{polylog n})$. The previous best bound was $O(kmn\text{polylog n})$.

Also improved running times are achieved for the static consensus tree problem, with applications to computational biology and relational data bases.

1 Introduction

Dynamic connectivity

A *dynamic graph problem* is the problem of maintaining a graph $G = (V, E)$ where G may be updated by insertion and deletion of edges, and where queries concerning certain properties of G may be asked. For the dynamic connectivity problem, say, the queries are of the form: is G connected? or are the vertices v and w connected in G?. Unless otherwise stated, all our dynamic graph problems are understood to be on-line meaning that each update/query has to be processed without any knowledge of later updates/queries. A dynamic graph problem is called *incremental* if no deletions are allowed and *decremental* if no insertions are allowed. If both insertions and deletions are allowed, the problem is said to be *fully* dynamic. This paper concerns decremental dynamic graph problems. For decremental connectivity we show

Theorem 1 *There is a Las Vegas type randomized algorithm that starting from a graph with m edges and n nodes, maintains a spanning forest during m edge-deletions in expected total time $O(\min\{n^2, m\log n\} + \sqrt{nm}\log^{2.5} n,)$. The deletions may be interspersed with connectivity queries, each of which is answered in constant time.*

Dynamic connectivity is an intensively studied problem (see [9] for an extensive list of references). Incremental connectivity is the union-find problem. For the union-find problem we have an amortized cost of $\alpha(n)$ per operation by Tarjan [16] with a matching lower-bound by Fredman and Saks [7].

Up till recently, deletions (with or without insertions) were considered much harder, with an operation cost of $O(\sqrt{n})$ for fully dynamic connectivity by Eppstein, Galil, and Italiano [4,5]. All the above upper bounds were based on deterministic algorithms. However, using a randomized method Henzinger and King got the cost down to an amortized expected $O(\log^3 n)$ per operation [9]. This bound was further improved to $O(\log^2 n)$ by Henzinger and Thorup. A lower bound of $\Omega(\log n/\log\log n)$ for fully dynamic connectivity was proved independently by Fredman and Henzinger [6] and Miltersen, Subramanian, Vitter, and Tamassia [14].

The decremental connectivity problem has been considered of independent interest by Even and Shiloach [3], but their bounds have been surpassed by the above mentioned fully dynamic algorithms. Recently, the following deterministic bound was obtained by Henzinger, King, and Warnow [10]: starting from a graph with n nodes and m edges, suppose that the edges are deleted in batches (with no queries in between), and let b_0 be the number of batches that do not result in a new component. Then the whole deletion process may be supported in time $O(n^2\log n + b_0\min\{n^2, m\log n\})$. As follows from Theorem 1, we can now support the whole deletion process in expected time $O(\min\{n^2, m\log n\} + \sqrt{nm}\log^{2.5} n)$. For $m/n = \omega(\log n)$, this also improves over the $O(m\log^2 n)$ bound we get if we use the above mentioned randomized fully dynamic algorithm from [8]. In particular, Theorem 1 shows that if we go to the extreme opposite of union-find, with deletions only starting with a complete graph, then an expected constant cost per operation is possible.

A general reduction of dense deletions only connectivity problems

Theorem 1 is based on a general randomized reduction of many deletions-only queries to few deletions and insertions queries. The reduction works for general k-edge-connectivity problems. The *k-edge-connectivity problem* is like the connectivity problem except that the queries to the graph $G = (V, E)$ take the form: is G k-edge-connected? or are the vertices v and w k-edge-connected? To facilitate the reduction, we include two extra requirements. First we need a query of the form: is the component of the vertex v k-edge-connected? and if not, find an edge from an edge-cut of size $< k$ of the component. Second a spanning forest should be maintained, so as to make the vertex set of any component accessible in linear time. The spanning forest may, of course, be maintained independently at operation cost $O(\log^2 n)$ using the fully dynamic algorithm from [8].

Theorem 2 *With a Las Vegas randomized reduction with overhead* $O(\min\{n^2, m\log n\})$*, we can reduce the decremental dynamic* k*-edge-connectivity problem with* m *deletions from a graph with* m *edges and* n *nodes, to a fully dynamic* k*-edge-connectivity problem with an expected number of* $O(\min\{\sqrt{mn\ell}, n(\log n)^2\ell\})$*,* $\ell = \max\{k, \log n\}$*, insertions and deletions to a graph over the same vertex set starting with an empty edge set.*

Thus, Theorem 1 is a corollary of Theorem 2 based on the $O(\log^2 n)$ connectivity algorithm from [8]. Theorem 2 allows us to derive new bounds for several of decremental dynamic connectivity problems, for which the previous best bounds where based on a fully dynamic solution. Below, for some different graph properties, we present the new and the old total cost of deleting the edges of from a graph $G = (V, E)$ with n nodes and m edges.

Graph Property	New Bound	Old Bound
Connectivity	$O(\min\{n^2, m\log n\} + \sqrt{nm}\log^{2.5} n)$	$O(m(\log n)^2)$ [8]
2-edge-connectivity	$O(\min\{n^2, m\log n\} + \sqrt{nm}\log^{3.5} n)$	$O(m(\log n)^3)$ [8]
Bipartiteness	$O(\min\{n^2, m\log n\} + \sqrt{nm}\log^{2.5} n)$	$O(m(\log n)^2)$ [8]
q-weights MST	$O(\min\{qn^2, m\log n\} + q\sqrt{nm}\log^{2.5} n)$	$O(qm(\log n)^2)$ [8]
k-edge-connectivity	$O(k^2n^2\text{polylog n})$	$O(kmn\text{polylog n})$ [4,13,15]

For connectivity, the query time is constant while for 2-edge connectivity it is $O(\log n)$. The q-weights MST is the problem of maintaining a minimum spanning tree for a graph with q different weights. For k-edge-connectivity, we are only considering queries of the form: is G k-connected? and if not, what is the connectivity of G? These queries are supported in constant time.

The consensus tree problem

An application of dynamic deletions only connectivity is the static *consensus tree problem.* We are given a set A of leaves and a set X of triples $((a, b), c) \in A^3$. The problem is then, if possible, to find a tree T with leaf set A such that for every $((a, b), c) \in X$, the least common ancestor of a and b is a strict descendant of the least common ancestor of b and c. This problem was raised in 1981 by Aho, Sagiv, Szymanski, and Ullman in the context of optimizing relational expressions [1]. Since then the consensus problem has found applications in computational biology [10,11].

In [1], Aho, Sagiv, Szymanski, and Ullman gave a simple deterministic algorithm for the consensus problem with running time $O(nm)$ where $n = |A|$ and $m = |X|$. Here we assume that $m = O(n^2)$. If not, the additional cost is linear in m. In [10] Henzinger, King, and Warnow reduced the problem to a deletions only connectivity problem with n vertices and m edges. In fact, they only need the above mentioned batched deletions, where at most one batch does not result in a new component, that is, $b_0 = 1$. Consequently, they get an $O(\min\{m\sqrt{n}, n^2\log n)$ deterministic algorithm for the consensus problem and an $O(m\log^3 n)$ randomized algorithm. Our new deletions only algorithm from Theorem 1 provides a randomized $O(\min\{n^2, m\log n\} + \sqrt{nm}\log^{2.5} n)$ algorithm. This is strictly faster than all the previous algorithms for $m/n = \omega(\log n)$.

Outline

Our reduction is based on a result of Karger [12] that, loosely speaking, states that a sparse random sub-graph H of a dense graph G represents most of the connectivity in G. We make an initial choice of such a sparse random sub-graph H of G. Initially, the information in H allows us to rule out most of the edges of G from being relevant for connectivity. As we delete edges and G becomes sparser, H becomes less and less representative, and more and more of the remaining edges of G become relevant. Nevertheless, guided by H, we will maintain a dynamic graph G/H with all relevant edges and which only need to be updated few times.

The paper is divided as follows. In section 2 we present the deletions only connectivity algorithm from Theorem 1. In sections 3 and 4, we generalize our reduction to the k-edge-connectivity problem, as described in Theorem 2. The consensus tree problem is briefly discussed in Section 5. Finally we have some concluding remarks on practicality in Section 6.

Notation

For any graph $G = (V, E)$, $\mathcal{V}(G) = V$ and $\mathcal{E}(G) = E$. The transitive closure of G is denoted G^*. For any vertex $v \in V$, $[v]_G$ denotes the component of G containing v. Thus, $\mathcal{V}([v]_G)$ is the set of vertices reachable from v in G.

For graphs with labeled edges, we will use the notation $(v, w : l)$ for an edge between v and w with label l.

2 Deletions only connectivity

In this section, we prove Theorem 1:

> *There is a Las Vegas type randomized algorithm that starting from a graph with m edges and n nodes, maintains a spanning forest during m deletions in expected total time $O(\min\{n^2, m \log n\} + \sqrt{nm} \log^{2.5} n\})$. The deletions may be interspersed with connectivity queries, each of which is answered in constant worst case time.*

by reduction to:

Theorem 3 (Henzinger, Thorup [8]) *Starting from a graph with n isolated vertices, we can maintain a spanning forest on-line during m insertions and deletions in total time $O(m \log^2 n)$.*

We shall refer to the algorithm of Theorem 3 as the HTK-algorithm, as it is based on the algorithm of Henzinger and King [9]. First we have

Lemma 4 (Even Shiloach [3]) *If a spanning forest is maintained during edge deletions, at an extra total cost of $O(n \log n)$ for the deletions, we can support connectivity queries in constant time.*

Thus, we can restrict our attention to the maintenance of a spanning tree.

Description A: The components of a data structure for maintaining a spanning forest of dynamic graph $G = (V, E)$ under edge-deletions.

A.1. Initially we have n nodes and m edges. Fix $p = \sqrt{(n/m) \ln n}$.

A.2. Let H be a random subgraph of G obtained by including each edge of G with probability p. The sampling of H is done initially before any edges are deleted.

A.3. Let $G/H = (G \setminus H^*) \cup H$. That is, G/H is obtained from G by removing all edges (v, w) from $G \setminus \mathcal{E}(H)$ where v and w are connected in H.

A.4. A spanning forest F of G/H is maintained using the HTK-algorithm. Since $(G/H)^* = G^*$, F is also the desired spanning forest of G.

We are now ready to implement the deletion of an edge from G.

Algorithm B: Delete(v, w)

B.1. Delete (v, w) from G/H, using the HTK-algorithm updating F.

B.2. If (v, w) was in $H \cap F$ and no replacement edge was found in G/H,

B.2.1. Let A be the set of edges from $G \setminus (G/H)$ between $[v]_{H\setminus\{(v,w)\}}$ and $[w]_{H\setminus\{(v,w)\}}$. That is, A is the set of possible replacement edges for (v, w) in the full graph G.

B.2.2. Insert all the edges from A in G/H, using the HTK-algorithm updating F.

B.3. Set $G := G \setminus \{(v, w)\}$ and $H := H \setminus \{(v, w)\}$.

Correctness: Note that if (v, w) is a bridge in G/H and (v, w) is not in H, then (v, w) is also a bridge in G. Hence it is correct that we in step B.2 only look for replacement edges outside G/H when deleting bridges in H. ■

To prove the efficiency of the above algorithm, we need:

Theorem 5 (Karger [12]) *Let G be a c-edge-connected graph. Let H be a random subgraph of G obtained by including each edge of G with probability p. If $pc \geq 8(d + 2) \ln n$, with probability $1 - O(1/n^d)$, H is $(pc/2)$-edge-connected.*

In fact, we shall use Theorem 5 in the following form:

Lemma 6 *Let G be a graph where each component is c-edge-connected. Let H be a random subgraph of G obtained by including each edge of G with probability p. If $pc \geq 8(d + 2) \ln n$, with probability $1 - O(1/n^d)$, the components of H are $(pc/2)$-edge-connected and coincide with those of G.*

Proof: Basically, we apply Theorem 5 to each component. First note that $d = pc/8 \ln n - 2$ gives equality in $pc \geq 8(d+2) \ln n$. With this optimal choice of d, the probability of H being $(pc/2)$-edge-connected in Theorem 5 becomes:

$$1 - O(1/n^d) = 1 - O(1/\exp(pc/8 - 2 \ln n)) = 1 - O(n^2/\exp(pc/8))$$

Now consider the case where G is divided into c-edge-connected components $n_1, \ldots, n_l$ where $\sum_{i=1}^{l} n_i = n$. Then the probability that all these components are $(pc/2)$-edge-connected in H is at least

$$1 - \sum_{i=1}^{l} O(n_i^2 / \exp(pc/8)) = 1 - O((\sum_{i=1}^{l} n_i^2)/\exp(pc/8)) \geq 1 - O(n^2/\exp(pc/8)).$$

That is, the division into components can only increase the desired probability. ■

For the the analysis, let $e_1, \ldots, e_m$ denote the sequence of deleted edges, let $G_0, \ldots, G_m$ and $H_0, \ldots, H_m$ denote corresponding sequences of instances of G and H. Thus, G_0 and H_0 denote the original graphs, and for $i = 1, \ldots, m$, $G_i = G_{i-1} \setminus \{e_i\}$ and $H_i = H_{i-1} \setminus \{e_i\}$. Let A_0 be the set of edges in G_0 between components of H_0. Then $A_0 = (G_0/H_0) \setminus H_0$. For $i = 1, \ldots, m$, let A_i denote the set A from step B.2.1 in the ith deletion. If step B.2.1 is not entered during the ith insertion, $A_i = \emptyset$. It follows that including an initial building up of G_0/H_0 from the empty graph, the set of edges added to G/H is $\mathcal{E}(H_0) \cup \bigcup_{i=0}^{m} A_i$.

Lemma 7 *With probability at least* $1 - 1/n$, $|\bigcup_{i=0}^{m} A_i| = O((n/p)\log n)$.

Proof: Note that if $(v, w) \in A_i$, then v and w are in different components of H_i. We shall use this fact to bound $|\bigcup_{i=0}^{m} A_i|$.

Fix $c = (40/p)\ln n$. For $i = 0, \ldots, m$, let C_i denote c-edge-connected induced sub-graphs of G_i. Then, for $i > 0$, C_i is also the c-edge-connected induced sub-graphs of $C_i \setminus \{e_i\}$. Let B_0 be the edges between different c-edge-connected induced sub-graphs in G_0, i.e. $B_0 = \mathcal{E}(G_0) \setminus \mathcal{E}(C_0)$, and for $i = 1, \ldots, m$, let B_i be the edges between different c-edge-connected induced sub-graphs in $C_{i-1} \setminus \{e_i\}$, i.e. $B_i = \mathcal{E}(C_{i-1}) \setminus (\{e_i\} \cup \mathcal{E}(C_i))$.

CLAIM 7A $|\bigcup_{i=0}^{m} B_i| \leq (c-1)(n-1) = O(n/p \log n)$.

PROOF: The graph C_0 is obtained from G_0 by repeatedly removing edge-cuts of components of size at most $c-1$. Likewise, for $i > 0$, C_i is obtained from $C_{i-1} \setminus \{e_i\}$ by the removal of edge-cuts of size $c-1$. The set $\bigcup_{i=0}^{m} B_i$ is the union of all edge-cuts removed this way. Since each edge-cut increases the number of components, there are at most $n-1$ edge-cuts. □

By Claim 7A, Lemma 7 follows if we can show that $\bigcup_{i=0}^{m} A_i \subseteq \bigcup_{i=0}^{m} B_i$ with high probability. To this end, we need

CLAIM 7B *For* $i = 0, \ldots, m$, *with probability* $1 - 1/n^3$, *no edge in* C_i *connects different components of* H_i.

PROOF: The sequence $C_0, C_1, \ldots, C_n$ depends on the edge-deletion sequence, but is independent of the initial random sub-graph H_0 of G_0. Thus, for each i, we may regard $H_0 \cap C_i$ as a random sub-graph of C_i where each edge is picked with probability p.

Each component of C_i is c-edge-connected. Moreover, for $d = 3$, $pc \geq 8(d+2)\ln n$. Hence, by Lemma 6, the components of $H_0 \cap C_i$ coincide with those of C_i. Since $H_0 \cap C_i \supseteq H_0 \cap G_i = H_i$, it follows that no edge in C_i can connect different components of H_i. □

As the edges in A_i all connect different components of H_i, Claim 7B immediately implies that $A_i \cap C_i = \emptyset$ with high probability. But $A_i \subseteq G_i \subseteq C_i \cup \bigcup_{j=0}^{i} B_j$, so $A_i \cap C_i = \emptyset$ implies $A_i \subseteq \bigcup_{j=0}^{i} B_j$. That is, for every i, with probability $1 - 1/n^3$, $A_i \subseteq \bigcup_{j=0}^{i} B_j \subseteq \bigcup_{i=0}^{m} B_i$. Consequently, $\bigcup_{i=0}^{m} A_i \subseteq \bigcup_{i=0}^{m} B_i$ with probability $1 - m/n^3 \geq 1 - 1/n$. Together with Claim 7A, this settles Lemma 7. ■

Lemma 8 *The expected number of updates to F/H is $O(mp + n/p \log n) = O(\sqrt{mn \log n})$ if $p = \sqrt{(n/m) \log n}$.*

Proof: If we include the initial build-up of G_0/H_0, the number of insertions in G/H is $|\mathcal{E}(H_0) \cup \bigcup_{i=0}^{m} A_i|$. Here $EXP(|\mathcal{E}(H_0)|) = mp$, and by Lemma 7, $EXP(|\bigcup_{i=0}^{m} A_i|) = O(n/p \log p + m/n) = O(n/p \log p)$. That is, the expected number of insertions is $O(mp + n/p \log n)$, as desired, and clearly we cannot delete more edges than we insert. ■

We still need to make clear how the sets A are found in step B.2.1. To this end, we will maintain the forest $F \cap H$. Given any vertex v, the forest $F \cap H$ allows us to identify $V([v]_H) = V([v]_{H \cap F})$ in linear time. Also, we will maintain the incidence lists of $G \setminus G/H$ such that for any vertex v, we can identify $\{(v,w))|(v,w) \in G \setminus G/H\}$ in linear time.

Algorithm C: Finds the set A in Step B.2.

C.1. Let $x \in \{v, w\}$ minimize $|V([x]_{F \cap H \setminus \{v,w\}})|$.

C.2. Let $N = \{(y,z)|y \in V([x]_{F \cap H \setminus \{v,w\}}), (y,z) \in G \setminus G/H\}$.

C.3. Let A be the set of edges in N with only one end-point in $V([x]_{F \cap H \setminus \{v,w\}})$.

Correctness: Let $\{y\} = \{v, w\} \setminus \{x\}$, so $\{x, y\} = \{v, w\}$. We are looking for the edges from $G \setminus G/H$ connecting $[x]_{F \cap H \setminus \{v,w\}}$ with $[y]_{F \cap H \setminus \{v,w\}}$. Clearly this set is contained in N from step C.2. Since $G \setminus G/H \subseteq H^*$, it follows that any edge from N with only one end-point in $[x]_{F \cap H \setminus \{v,w\}}$ must have its other end-point in $[y]_{F \cap H \setminus \{v,w\}}$. ■

Lemma 9 *Algorithm C takes time $O(|V([x]_{F \cap H \setminus \{v,w\}})| + |N|)$.*

Proof: In step C.1, we count $|V([v]_{F \cap H \setminus \{v,w\}})|$ and $|V([w]_{F \cap H \setminus \{v,w\}})|$ in parallel, stopping as soon as the smaller is found. Thus we identify x and $V([x]_{F \cap H \setminus \{v,w\}})$ in $O(|V([x]_{F \cap H \setminus \{v,w\}}|)$ time. Using the incidence lists of $G \setminus G/H$, the set N of step C.2 is found in time $O(|N|)$. Finally, since $V([x]_{F \cap H \setminus \{v,w\}})$ is identified, the subset A of N from step C.3 is found in time $O(|N|)$ by a linear time scan of N. ■

Lemma 10 *The total time spent in Algorithm C is $O(\min\{m \log n, n^2\})$.*

Proof: We view the bound of Lemma 9 as a cost of 1 for every vertex in $V([x]_{F \cap H \setminus \{v,w\}})$ and a cost of 1 for every edge in N. Note that edges in A are transferred to G/H so they will not again appear in A. Since x is chosen so as to minimize $|V([x]_{F \cap H \setminus \{v,w\}})|$, between

every to times that a vertex or edge appears in $V([x]_{F\cap H\setminus\{v,w\}})$ or N, the size of $|V([x]_{F\cap H})|$ is halved. Hence each vertex and edge contributes at most $\log_2 n$ to the total cost, giving us a bound of $O((n+m)\log n) = O(m \log n)$.

To get the $O(n^2)$ bound, note that $O(|V([x]_{F\cap H\setminus\{v,w\}})| + |N|) = O(|V([x]_{F\cap H\setminus\{v,w\}})| \times |V([x]_{F\cap H})|)$. That is, we may calculate the cost by attributing a cost of $O(|V([x]_{F\cap H})|)$ to each vertex in $V([x]_{F\cap H\setminus\{v,w\}})$. Since $|V([x]_{F\cap H})|$ is halved between any to times that a vertex u is found in $V([x]_{F\cap H\setminus\{v,w\}})$, and since $|V([x]_{F\cap H})|$ starts at most n, the total cost is bounded by $O(n^2)$. ■

Proof of Theorem 1: Combining Theorems 3, Lemma 8, and Lemma 10, we get that we can maintain the spanning forest of G in time

$$O(\sqrt{mn\log n}\log^2 n + \min\{n^2, (m\log n)\}) = O(\min\{n^2, m\log n\} + \sqrt{nm}\log^{2.5} n)\}.$$

This combined with Observation 4 proved settles Theorem 1. ■

3 A general reduction

In this section, we show how to modify the construction of the previous section, in order to prove the $O(\sqrt{mn\ell})$ bound of Theorem 2:

Theorem 11 *With a Las Vegas randomized reduction with overhead $O(\min\{n^2, m\log n\})$, we can reduce the decremental dynamic k-edge-connectivity problem with m deletions from a graph with m edges and n nodes, to a fully dynamic k-edge-connectivity problem with an expected number of $O(\sqrt{mn\ell})$, $\ell = \max\{k, \log n\}$, insertions and deletions to a graph over the same vertex set starting with an empty edge set.*

Recall our special requirements for dynamic k-edge-connectivity: First there should be a query of the form: is the component of the vertex v k-edge-connected? and if not, find an edge from an edge-cut of size $< k$ of the component. Second a spanning forest should be maintained.

Description D: The components of a data structure for maintaining k-edge-connectivity of a dynamic graph $G = (V, E)$ under edge-deletions.

D.1. Initially we have n nodes and m edges. Fix $p = \sqrt{(n/m)\max\{k, \log n\}}$.

D.2. Let R be a random subgraph of G obtained by including each edge of G with probability p. The sampling of R is done initially before any edges are deleted.

D.3. We will maintain the graph H of the k-edge-connected induced sub-graphs of R. Note that $H = R$ if $k = 1$. A fully dynamic k-edge-connectivity algorithm will tell us if an edge-deletion from H creates a $< k$-edge-connected component.

D.4. Moreover, we maintain $G/H = (G \setminus H^*) \cup H$. That is, G/H is obtained from G by removing all edges (v,w) from $G \setminus \mathcal{E}(H)$ where v and w are in the same components of H. A fully-dynamic k-edge-connectivity algorithm will answer k-edge-connectivity queries for G/H, and maintain a spanning forest F for G/H. Since all the components of H are k-edge-connected, all k-edge-connectivity queries to G may be addressed to our k-edge-connectivity algorithm for G/H. Similarly, the spanning forest F of G/H is also a spanning forest of G.

We are now ready to implement the deletion of an edge from G.

Algorithm E: Delete(v,w)

E.1. Delete (v,w) from H and G/H and G.

E.2. While there is a component in H which is $< k$-edge-connected,

E.2.1. Let (v',w') be any edge from an edge-cut of size $< k$ of a component in H.

E.2.2. Delete (v',w') from H and G/H.

E.2.3. If (v',w') was a bridge in H,

E.2.3.1. Using Algorithm C, identify the set A of edges from $G \setminus (G/H)$ between the two new components of H.

E.2.3.2. Insert the edges of A in G/H.

For the efficiency of Algorithm E, first note that Algorithm C is used as in the previous section, and hence that Lemma 9 and 10 still hold true. That is, the total time spent on identifying the sets a in step E.2.3.1 is $O(\min\{m \log n, n^2\})$.

We will now analyse the number of updates to H and G/H. Since any update to H is also done to G/H, we may restrict our attention to updates to G/H. Let e_i, G_i, and H_i be defined as in the last section. That is, $i = 0$ refers to the initial values, while $i = 1, \ldots, m$ refers to the values after the ith deletion with Algorithm E. Here we are not considering the local deletions in step E.2.2. To get an appropriate definition of the A_i, let A_0 be the set of edges in G_0 between components of H_0. Then $A_0 = (G_0/H_0) \setminus H_0$. For $i = 1, \ldots, m$, let A_i denote the union of the sets A found in step E.2.3.1 in the while-loop during the ith deletion. Again it follows that including an initial building up of G_0/H_0 from the empty graph, the set of edges added to G/H is $\mathcal{E}(H_0) \cup \bigcup_{i=0}^{m} A_i$. Since H_0 is a subgraph of the initial random subgraph R the expected value of $|\mathcal{E}(H_0)|$ is bounded by pm.

Lemma 12 *With probability at least* $1 - 1/n$, $|\bigcup_{i=0}^{m} A_i| = O((n/p)\min\{k, \log n\})$.

Proof: The proof goes as the proof of Lemma 7 except with $c = \max\{40 \log n, 2k\}/p$. Again the C_i are the c-edge-connected induced sub-graphs of G_i. We exploit that by Lemma 6, $R_0 \cap C_i = H_i$ inherits $(pc/2)$-edge-connectedness of C_i where now $pc/2 \geq k$. ∎

Since the number of deletions to G/H is bounded by the number of insertions, and since the number of updates to H is bounded by the number of updates to G/H, we conclude that the expected total number of updates to H and G/H is bounded by

$$O(mp + (n/p)\min\{k, \log n\}) = O(\sqrt{nm \min\{k, \log n\}}).$$

Thus Theorem 11 follows.

4 A recursive reduction

In this section, we complete the proof of Theorem 2 by showing

Theorem 13 *With a Las Vegas randomized reduction with overhead $O(\min\{n^2, m \log n\})$, we can reduce the decremental dynamic k-edge-connectivity problem with m deletions from a graph with m edges and n nodes, to a fully dynamic k-edge-connectivity problem with an expected number of $O(n(\log n)^2\ell)$, $\ell = \max\{k, \log n\}$, insertions and deletions to a graph over the same vertex set starting with an empty edge set.*

Our starting point is the data structure from last section. Again we have the random sub-graph H of G where all components of H are k-edge-connected. However, instead of maintaining the graph G/H containing both H and the edges from G between components in G, we maintain the graph $G//H$ where all components of H have been contracted. Clearly H and $G//H$ together provide us with all information on k-edge-connectivity. The point now is that no edges are inserted in H, and hence H may be maintained recursively. The price we pay is that whenever a component of H is disconnected, we need to change the edges of $G//H$ according to the new components of H.

Description F: The components of a data structure for maintaining k-edge-connectivity of a dynamic graph $G = (V, E)$ under edge-deletions.

F.1. Initially we have n nodes and m edges. Fix $p = 1/2$.

F.2. Let R be a random subgraph of G obtained by including each edge of G with probability p. The sampling of R is done initially before any edges are deleted.

F.3. As in the previous section, we will maintain the graph H of the k-edge-connected induced sub-graphs of R. This time, a recursively defined deletions-only dynamic k-edge-connectivity algorithm for H will tell us if an edge-deletion from H creates a $< k$-edge-connected induced sub-graph. Also, the recursive deletions only algorithm provides us with a spanning tree F_H for H

F.4. With each vertex v, we associate an identifier $c(v)$ for the component H that v belongs to.

F.5. Moreover, we maintain the graph $G//H$ obtained from G by contraction of all the components of H. The graph $G//H$ may have parallel edges. Every edge e in the contracted graph $G//H$ is labeled with the original edges from G. Thus, the vertex set of $G//H$ is the set $\{c(v)|v \in V\}$. For each edge (v,w) between different components, i.e. $c(v) \neq c(w)$, we have an edge $(c(v),c(w) : (v,w))$ in $G//H$. A fully-dynamic k-edge-connectivity algorithm will answer k-edge-connectivity queries for $G//H$, and maintain a spanning forest $F_{G//H}$ for $G//H$.
Since all the components of H are k-edge-connected, any $\leq k$-edge-connectivity query to vertices v and w in G may be addressed to $c(v)$ and $c(w)$ in $G//H$. Also a spanning forest F of G may be found as

$$F_H \cup \{(v,w)|(c(v),c(w) : (v,w)) \in \mathcal{E}(F_{G//H})\}.$$

We are now ready to implement the deletion of an edge from G.

Algorithm G: Delete(v,w)

G.1. Set $G := G \setminus \{(v,w)\}$, $H := H \setminus \{(v,w)\}$, and $G//H := G//H \setminus \{(c(v),c(w) : (v,w))\}$.

G.2. While there is a component in H which is $< k$-edge-connected,

G.2.1. Let (v',w') be any edge from an edge-cut of size $< k$ of a component in H.

G.2.2. Delete (v',w') from H.

G.2.3. If (v',w') was a bridge in H,

G.2.3.1. Let $\{x,y\} = \{v',w'\}$ be such that $|\mathcal{V}([x]_{F_H})|$ is minimized.

G.2.3.2. Let i be a new component identifier.

G.2.3.3. Set $j := c(y)$.

G.2.3.4. For all vertices $a \in \mathcal{V}([x]_{F_H})$, set $c(a) := i$.

G.2.3.5. For all edges $(a,b) \in E$ with $c(a) = i$,

G.2.3.5.1. If $c(b) = j$, add $(i,j : (a,b))$ to $G//H$.

G.2.3.5.2. If $c(b) \notin \{i,j\}$, change $(j,c(b) : (a,b))$ to $(i,c(b) : (a,b))$.

Lemma 14 *With probability $1 - 1/n$, the total number of edges added to $G//H$ via step G.2.3.5.1 is $O((n/p)\min\{k,\log n\})$.*

Proof: The edges (a,b) with $c(b) = j$ selected in step G.2.3.5.1 are those those between newly created components of H. Hence these edges coincide with those from A in step E.2.3.1. Hence, by Lemma 12, with probability $1 - 1/n$, the total number of edges in the sets A is $O((n/p)\min\{k,\log n\})$. ■

Lemma 15 *With probability $1-1/n$, the total number of changes to $G//H$ via step G.2.3.5.2 is $O((n/p)(\log n)\min\{k,\log n\})$.*

Proof: Between every two times an edge in $G//H$ is changed in step G.2.3.5.2, the component size in H of one of its end points is halved. Hence the total number of changes is at most $2\log_2 n$ times the total number of edges inserted in $G//H$. That is, the total number of changes is $O(n(\log n)\max\{k, \log n\})$. ■

Lemma 16 *If we exclude updates to H and $G//H$, the total cost of Algorithm G is bounded by $O(\min\{n^2, m\log n\})$.*

Proof: In connection with the for-loop G.2.3.5, the cost of considering an edge (a, b) with $c(b) \notin \{i, j\}$ is attributed to the update to $G//H$ in step G.2.3.5.2. However, all other edges considered are also considered in the set N in Algorithm C, step C.2. Hence the lemma follows from Lemma 10. ■

Combining Lemmas 14, 15, and 16, we have now shown

Proposition 17 *For any $p \in (0;1)$, with an overhead of $O(\min\{n^2, m\log n\})$, we can reduce the decremental dynamic on-line k-edge-connectivity problem where we delete the edges from a graph with m edges and n nodes to the problems of*

- *decremental dynamic on-line k-edge-connectivity problem where we delete the edges from a graph H with an expected number of at most mp edges and n nodes.*
- *fully dynamic on-line k-edge-connectivity problem where we make an expected number of $O(n(\log n)\max\{k, \log n\}/p)$ updates to a graph with n nodes starting with an empty vertex set.* ■

Let $D(m)$ be the cost of the decremental dynamic on-line k-edge-connectivity problem where we delete the edges from a graph with m edges and n nodes. Also, let q denote the operation cost of a fully-dynamic k-edge-connectivity a graph with n nodes starting with no edges. Finally set $\ell = \max\{k, \log n\}$. We can then summarize Proposition 17 in the recurrence:

$$D(m) \leq D(mp) + O(\min\{n^2, m\log n\}) + q \times O(n(\log n)\ell). \tag{1}$$

Theorem 11 is expressed as

$$D(m) \leq O(\min\{n^2, m\log n\}) + q \times O(\sqrt{mn\ell}). \tag{2}$$

Note that $\sqrt{mn\ell} = n(\log n)\ell \iff m = n(\log n)^2\ell$. Using this instantiation of m in (2), we get

$$D(n(\log n)^2\ell) \leq O(\min\{n^2, m\log n\}) + q \times O(n(\log n)\ell). \tag{3}$$

Using (1) with $p = 2$, we derive

$$\begin{aligned} \forall m \geq 2n(\log n)\ell: \quad D(m) &\leq O(m\log n) + q \times O(n(\log(m/(n\ell\log n)))(\log n)\ell) && (4)\\ &\leq O(m\log n) + q \times O(n(\log n)^2\ell) && (5) \end{aligned}$$

For the case where $m = n^2$, we first apply (1) with $p = 1/\log n$, and then we apply (5), so

$$D(n^2) \leq O(n^2) + q \times O(n(\log n)^2 \ell), \tag{6}$$

Now (5) and (6) together provides the statement of Theorem 13. With reference to (4), we are actually proving something stronger for sparse graphs. The bound in (4) is strictly stronger than (2) from Theorem 11 for any $m = \omega(n(\log n)^2 \ell)$. It is the bound we get from the algorithm that swaps at $m = n(\log n)^2 \ell$ from the recursive construction in Proposition 17 with $p = 2$, to the construction in Theorem 11.

5 The consensus tree problem

We will now be a little bit more precise about the connectivity problem to which Henzinger, King, and Warnow reduce the consensus tree problem [10, p. 338]. They are considering a consensus tree problem with a set A of n leaves and a set X of t triples $((a,b),c) \in A^3$. Set $m = \min\{n^2, 2t\}$. In their reduction, they are operating on a dynamic graph G with both red and yellow edges. The vertex set is the set A of leaves. Initially, for each triple $((a,b),c)$, there are two edges (a,b) and (b,c) in G. The edges of different triples may coincide, so there may be less than m edges. Their algorithm finds an initial coloring of the edges with red and yellow. They are interested in the components of the yellow graph. Every time new yellow components are found, they want to identify the red edges between them. On the basis of these red edges, some yellow edges will be colored red, corresponding to deletions in the yellow graph. The red edges may be found using Algorithm C with the red graph in place of $G \setminus G/H$. Hence it follows that the total time spent on identifying red edges is $O(\min\{n^2, m \log n\})$. Given the red edges, the total time spent on identifying the yellow edges to be turned red is $O(t)$. That is, their overhead in reducing the consensus tree to a deletions only connectivity problem for a graph with n nodes and m edges is $O(\min\{n^2, m \log n\}+t)$. Thus, by Theorem 1, there is a Las Vegas type randomized algorithm for the consensus tree problem with n leaves, t triples, and $m = \min\{t, n^2\}$, with an expected running time of $O(\min\{n^2, m \log n\} + \sqrt{nm} \log^{2.5} n + t)$.

6 Discussion

In this section, we discuss the actual constants involved in our reductions, focusing on the case of connectivity from Section 2.

Concerning the overhead $O(\min\{n^2, m \log n\})$ in our reductions, it should be noted that it stems from the simple scanning in Algorithm C. Hence the constants involved in the overhead are expected to be small. Consequently, the practicality of our approach hinges on the practicality of the fully dynamic connectivity algorithms reduced to. Consider the case from Section 2. Recently there has been given experimental evidence that the fully dynamic 1-connectivity algorithm from [9] is fast, and the improvement in [8] is simply reducing the

work. Thus our algorithm should be fast in practice. The critical question is, however, whether we for practical values of m and n actually reduce the number of up-dates relative to if we just used a fully-dynamic algorithm directly.

We will now consider the constants in our analysis in Section 2. The initial build-up of a dynamic data structure is comparatively cheap, so we are only interested in updates after the first deletion. This means that edges in H are only deleted. All other edges in G/H may still be both inserted and deleted. Re-examining our analysis, we now get an expected total of $\leq 2 \times (40/p) \ln n + pm$ updates to G/H. Choosing $p = \sqrt{(80n/m) \ln n}$ gives us an expected total number of $2\sqrt{80mn \ln n}$ updates. If we used the fully dynamic connectivity algorithm directly, we would only need m deletions. Thus we expect to reduce the number updates if

$$\sqrt{340mn \ln n} < m \iff m/n > 340 \ln n,$$

which is the case for $m/n > 3,000$. However, 3,000 is only the upper-bound on the threshold that follows from our analysis. The true threshold is likely much smaller.

References

[1] A.V. Aho, Y. Sagiv, T.G. Szymanski, and J.D. Ullman. Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM J. on Computing*, 10 (3), 1981, 405–421.

[2] D. Alberts, G. Cattaneo, and G.F. Italiano. An empirical study of dynamic graph algorithms. *Proc. 7th ACM-SIAM Symp. on Discrete Algorithms*, 1996, 192–201.

[3] S. Even and Y. Shiloach. An On-Line Edge-Deletion Problem. *Journal of ACM*, 28(1) 1981, 1–4.

[4] D. Eppstein, Z. Galil, G. F. Italiano, Improved Sparsification. Tech. Report 93-20, Department of Information and Computer Science, University of California, Irvine, CA 92717.

[5] D. Eppstein, Z. Galil, G. F. Italiano, A. Nissenzweig, Sparsification - A Technique for Speeding up Dynamic Graph Algorithms. *Proc. 33rd Symp. on Foundations of Computer Science*, 1992, 60–69.

[6] M. L. Fredman and M. R. Henzinger. Lower Bounds for Fully Dynamic Connectivity Problems in Graphs. Submitted to *Algorithmica*.

[7] M.L. Fredman and M.E. Saks. *Proc. 21th ACM Symp. on Theory of Computing*, 1989, 345–354.

[8] M.R. Henzinger and M. Thorup. Improved sampling with applications to dynamic graph algorithms. Technical Report 95/26, DIKU, 1995. To appear in *Proceedings of the 23rd International Colloquium on Automata Languages, and Programming.*

[9] M. R. Henzinger and V. King. Randomized Dynamic Graph Algorithms with Polylogarithmic Time per Operation. *Proc. 27th ACM Symp. on Theory of Computing*, 1995, 519–527.

[10] M. R. Henzinger, V. King, and T. Warnow. Constructing a tree from homeomorphic subtrees, with applications to computational evolutionary biology. *Proc. 7th ACM-SIAM Symp. on Discrete Algorithms*, 1996, 333–340.

[11] S. Kannan, T. Warnow, and S. Yooseph. Computing the local consensus of trees. *Proc. 6th ACM-SIAM Symp. on Discrete Algorithms*, 1995, 68–77.

[12] D. R. Karger. Random sampling in cut, flow, and network design problems. *Proc. 26th ACM Symp. on Theory of Computing*, 1994, 648–657.

[13] D. R. Karger. Minimum Cuts in Near-Linear Time. To appear in *Proc. 28th ACM Symp. on Theory of Computing*, 1996.

[14] P.B. Miltersen, S. Subramanian, J.S. Vitter, and R. Tamassia. Complexity models for incremental computation. *Theoretical Computer Science,* 130, 1994, 203-236.

[15] H. Nagamochi and T. Ibaraki. Linear time algorithms for finding k-edges connected and k-node connected spanning sub-graphs. *Algorithmica,* 7, 1992, 583–596.

[16] R.E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of ACM*, 22:215–225, 1975.