Improved Sampling with Applications to Dynamic Graph Algorithms

Monika Rauch Henzinger *
Mikkel Thorup †

December 15, 1995

Abstract

We state a new sampling lemma and use it to improve the running time of dynamic graph algorithms.

For the dynamic connectivity problem the previously best randomized algorithm takes expected time $O(\log^3 n)$ per update, amortized over $\Omega(m)$ updates. Using the new sampling lemma, we improve its running time to $O(\log^2 n)$. There exists a lower bound in the cell probe model for the time per operation of $\Omega(\log n / \log\log n)$ for this problem.

Similarly improved running times are achieved for the following dynamic problems: (1) $O(\log^3 n)$ to maintain the bridges in a graph (the 2-edge connectivity problem); (2) $O(k \log^2 n)$ to maintain a minimum spanning tree in a graph with k different weights (the k-weight minimum spanning tree problem); (3) $O(\log^2 n \log U/\epsilon')$ to maintain a spanning tree whose weight is a $(1+\epsilon')$-approximation of the weight of the minimum spanning tree, where U is the maximum weight in the graph (the $(1+\epsilon')$-approximate minimum spanning tree problem); and (4) $O(\log^2 n)$ to test if the graph is bipartite (the bipartiteness-testing problem).

1 Introduction

We present the sampling lemma below, and use it to improve the running times of various dynamic graph algorithms.

Lemma 1 *Let R be a subset of a set S, and let $r, c \in \Re_{>1}$. Set $s = |S|$. Then there is an algorithm with one of two outcomes:*

*Department of Computer Science, Cornell University, Ithaca, NY. (*mhr@cs.cornell.edu.*) Author's Maiden Name: Monika H. Rauch. This research was supported by an NSF CAREER Award, Grant No. CCR-9501712.

†Department of Computer Science, University of Copenhagen, Universitetsparken 1, 2100 Kbh. Ø, Denmark (*mthorup@diku.dk, http://www.diku.dk/~mthorup*).

(i) *It returns an element from R after having sampled an expected number of $O(r)$ random elements from S and having tested them for membership of R.*

(ii) *Having sampled and tested $O(s/c)$ random elements from S, it states that $|R|/|S| > 1/r$ with probability $< \exp(-s/rc)$.*

1.1 Dynamic graph algorithms

Let $G = (V, E)$ be a graph with n nodes and m edges. A graph property $\mathcal{P}$ is a function that (a) maps every graph G to *true* or *false* or (b) that maps every tuple (G, u, v) to *true* or *false*, where $G = (V, E)$ is a graph and $u, v \in V$. An example for Case (a) is a function that maps every bipartite graph to *true* and every non-bipartite graph to *false*. An example for Case (b) is *connectivity* that returns *true* if u and v are connected in G and *false* otherwise.

A dynamic graph algorithm is a data structure that maintains any graph G and a graph property $\mathcal{P}$ under an arbitrary sequence of the following operations.

- $Insert(u, v)$: Add the edge (u, v) to G.
- $Delete(u, v)$: Remove the edge (u, v) from G if it exists.
- $Query(u, v)$: Return *yes* if $\mathcal{P}$ holds for u and v in G and *false* otherwise.

In this paper we study the following graph properties: connectivity, two-edge connectivity, k-weight minimum spanning tree, $(1+\epsilon')$-approximate minimum spanning tree, and bipartiteness-testing.

1.2 Previous Work

Dynamic graph algorithms are compared using the (amortized or worst-case) time per operation. The best deterministic algorithms for the above graph properties take time $O(\sqrt{n})$ per update operation and $O(1)$ or $O(\log n)$ per query [3, 4]. Recently [6], Henzinger and King gave algorithms with polylogarithmic amortized time per operation using (Las-Vegas type) randomization. Their algorithms achieve the following running times:

1. $O(\log^3 n)$ to maintain a spanning tree in a graph (the connectivity problem;
2. $O(\log^4 n)$ to maintain the bridges in a graph (the 2-edge connectivity problem);
3. $O(k \log^3 n)$ to maintain a minimum spanning tree in a graph with k different weights (k-weight minimum spanning tree problem);
4. $O(\log^3 n \log U/\epsilon')$ to maintain a spanning tree whose weight is a $(1+\epsilon')$ approximation of the weight of the minimum spanning tree, where U is the maximum weight in the graph (the $(1+\epsilon')$-approximate minimum spanning tree problem);
5. $O(\log^3 n)$ to test if the graph is bipartite (the bipartiteness-testing problem).

Fredman and Henzinger showed lower bounds of $\Omega(\log n / \log\log n)$ in the cell probe model for the first four of these problems [5] (see also [8]).

1.3 New Results

We show in this paper the following improved running times:

1. $O(\log^2 n)$ for connectivity;
2. $O(\log^3 n)$ for 2-edge connectivity;
3. $O(k \log^2 n)$ for the k-weight minimum spanning tree problem;
4. $O(\log^2 n (\log U)/\epsilon')$ for the $(1+\epsilon')$ approximate minimum spanning tree problem, where U is the maximum weight in the graph;
5. $O(\log^2 n)$ for bipartiteness testing.

1.4 Improved sampling in dynamic graph algorithms

Our improvements are achieved by locally improving a certain bottleneck in the approach by Henzinger and King [6], henceforth referred to as the *HK-approach*. Rather than repeating their whole construction, we will confine ourselves to a reasonably self-contained description of this bottleneck. Our techniques for the bottleneck are of a general flavor and we expect them to be applicable in other contexts.

Let T be a spanning tree of some graph $G = (V, E)$. In the HK-approach, G is only one of many sub-graphs of the real graph. If some tree edge e is removed from T, we get two sub-trees T_1, T_2. Consider the *cut* C_e of non-tree edges with end-points in both T_1 and T_2. Any cut edge $f \in C_e$ can replace e in the sense that $T \cup \{f\} \setminus \{e\}$ is a spanning tree of G. Our general goal is to find such a cut edge f. Alternatively it is acceptable to discover that the cut C_e is sparse as defined below.

For each vertex $v \in T$, we have the set $N(v)$ of non-tree edges incident to T. Let $w(v) = |N(v)|$. For any sub-tree U of T, set $N(U) = \bigcup_{v \in V(U)} N(v)$ and $w(U) = \sum_{v \in V(U)} w(v)$. Note that $w(U)$ may be bigger than $|N(U)|$ because edges with both end-point in U are counted twice. Assume that T_1 contains no more nodes than T_2. We say that the cut C_e is *sparse* if $8 \log_2 n |C_e| < w(T_1)$. Otherwise C_e is said to be *dense*. If the cut is sparse, a cost of $O(w(T_1))$ may be attributed other operations due to an amortization in the HK-approach.

We store all edges of $N(T_1)$ in the leaves of a balanced search tree. This allows us to pick in time $O(\log n)$ a random edge from $N(T_1)$ (edges with both end-points in T_1 are picked with twice the probability of edges with one end-point in T_1) and check if its other end-point is in T_2. This is the desired approach for dense cuts. Alternatively, in time $O(w(T_1))$, we may scan all of $N(T_1)$, identifying all the edges in C_e. This is the desired approach for sparse cuts where the $O(w(T_1))$ is paid for via amortization. Unfortunately, we do not know in advance whether C_e is sparse or dense.

In the HT-approach, in time $O(\log^3 n)$, they sample $16 \log_2^2 n$ random edges from $N(T_1)$. If the sampling successfully finds an edge from C_e, this edge is returned. Otherwise, in time $O(w(T_1))$, they make a complete scan. If C_e is sparse, the scan is attributed to the amortization. The probability of C_e not being sparse is the probability of the sampling not being successful for a dense cut, which is $\leq (1 - 1/(8 \log_2 n))^{16 \log_2 n} < 1/n^2 = O(1/w(T_1))$. Hence the expected cost of an unduly scan (i.e. a scan even though the cut is dense) is

$O(w(T_1)/w(T_1)) = O(1)$. Thus, the total expected cost is $O(\log^3 n)$. This cost remains a bottle-neck for the HK-approach as long as the time per operation is $\Omega(\log^2 n)$.

We will now apply the sampling from Lemma 1 with $R = C_e$, $S = N(T_1)$, $w(T_1)/2 \le s \le w(T_1) = O(n^2)$, $r = 8\log_2 n$, and $c = O(\log n)$. Moreover, the cost of sampling and testing is $O(\log n)$. Then, in case (i), we find an element from C_e in expected time $O(\log n \cdot 8\log_2) = O(\log^2 n)$. In case (ii), the cost is $O(\log n \cdot w(T_1)/\log n) = O(w(T_1))$ matching the cost of the sub-sequent scanning. If the cut turns out to be sparse this cost is attributed to the amortization. In case (ii) the probability of a dense cut is $\exp(-s/rc) = \exp(-w(T_1)/O(\log^2 n))$, so the expected contribution from unduly scanning is $O(w(T_1)\exp(-w(T_1)/O(\log^2 n))) = O(\log^2 n)$. Thus, our expected cost is $O(\log^2 n)$, as opposed to the $O(\log^3 n)$ cost achieved by the HK-approach.

The removal of a factor $O(\log n)$ explains our improvements.

2 Proving the sampling lemma

In this section, we will prove Lemma 1 constructively, presenting a concrete algorithm. First recall the statement of the lemma:

> *Let R be a subset of a set S, and let $r, c \in \Re_{>1}$. Set $s = |S|$. Then there is an algorithm with one of two outcomes:*
>
> **(i)** *It returns an element from R after having sampled an expected number of $O(r)$ random elements from S and having tested them for membership of R.*
>
> **(ii)** *Having sampled and tested $O(s/c)$ random elements from S, it states that $|R|/|S| > 1/r$ with probability $< \exp(-s/rc)$.*

Proof: Let the increasing sequence $n_0, \ldots, n_k \cdots$ be defined such that $n_0 = 26^4$ and for $i > 0$, $n_i = \exp(n_{i-1}^{1/4})$. Let the decreasing sequence $r_0, \ldots, r_k \cdots$ be defined such that $r_0 = 2r(1 + 2n_0^{-1/4}) = 28/13 \cdot r < 3r$ and for $i > 0$, $r_i = r_{i-1}/(1 + n_{i-1}^{-1/4})$.

Claim 1A *For all $i \ge 0$,*

(a) $2n_i < n_{i+1}$.

(b) $2n_i^{1/4} < n_{i+1}^{1/4}$.

(c) $2r < r_i < 3r$.

Proof: Both (a) and (b) are easily verified by insertion. The r_i are decreasing, so $r_i \le r_0 < 3r$. Finally, $r_i = 2r(1 + 2n_0^{-1/4})/\prod_{j=1}^{i-1}(1 + n_j^{-1/4}) \ge 2r\exp(2n_0^{-1/4} - \sum_{j=1}^{i-1} n_j^{-1/4}) > 2r$. The last inequality uses (b). □

Algorithm A: Does the task described in Lemma 1.

A.1. $i := 0$;

A.2. While $r_i n_i < 8s/c$:

A.2.1. Let S_i be a random subset of S of size $r_i n_i$.

A.2.2. $R_i := S_i \cap R$.

A.2.3. If $|R_i| \geq n_i$, then return $x \in S_i \cap R$

A.2.4. $i := i + 1$;

A.3. Let S_i be a random subset of S of size $8s/c$.

A.4. $R_i := S_i \cap R$.

A.5. If $|R_i| \geq 8s/(cr_i)$, then return $x \in S_i \cap R$.

A.6. Return "$|R|/|S| > 1/r$ with probability $< \exp(-s/rc)$."

We show next a bound on the number of sampled edges (Claim 1B) and on the probability that the algorithm return an element from R in round i (Claim 1C). Afterwards we prove that the Algorithm A satisfies the conditions of Lemma 1.

Let t be the final value of i - if we return an element from R in Step A.2.3, then i is not subsequently increased.

Claim 1B *For all $t \geq i \geq 0$, $\sum_{j=0}^{i} |S_j| = O(rn_i)$.*

Proof: Note that in Steps A.3–A.5, $|S_i| = 8s/c \leq r_i n_i$. Thus, for all $i \geq 0$,

$$\sum_{j=0}^{i} |S_j| \leq \sum_{j=0}^{i} r_j n_j \leq 3r \sum_{j=0}^{i} n_j = O(rn_i).$$

The last inequality uses Claim 1Aa. □

For $i > 0$, let p_i be the probability that the algorithm returns an element from R in round i. Here the round refers to the value of i in Step A.2.3 or A.5.

Claim 1C *For all $i \geq 1$, $p_i \leq n_i^{-2}$*

Proof: We divide into two cases:

Case 1: $|R|/|S| > (1 + n_{i-1}^{-1/4}/2)/r_{i-1}$**:** In round $i - 1$ we did not return, so $|R_{i-1}|$ is less than $x = n_{i-1}$. However, the expected value μ of $|R_{i-1}|$ is at least $n_{i-1}(1 + n_{i-1}^{-1/4}/2)$. Note that

$$p_i \leq Pr(|R_{i-1}| < (1 - \delta)\mu) \text{ with } \delta = (\mu - x)/\mu.$$

Using the Chernoff bound (according to [1]),

$$Pr(|R_{i-1}| < (1 - \delta)\mu) < e^{-\delta^2 \mu/2} = e^{-(\mu - x)^2/(2\mu)}.$$

For $\mu \geq n_{i-1}(1+n_{i-1}^{-1/4}/2)$ this function is maximized for $\mu = n_{i-1}(1+n_{i-1}^{-1/4}/2)$. Thus,

$$p_i \leq \exp(\frac{-(n_{i-1}^{3/4}/2)^2}{2n_{i-1}(1+n_{i-1}^{-1/4}/2)}) < \exp(\frac{-n_{i-1}^{1/2}}{9}) < \exp(-2n_{i-1}^{1/4}) = n_i^{-2}.$$

The inequalities use that $n_{i-1}^{1/4} \geq n_0^{1/4} > 18 > 16$.

Case 2: $|R|/|S| \leq (1+n_{i-1}^{-1/4}/2)/r_{i-1}$**:** Note that

$$\frac{1+n_{i-1}^{-1/4}/2}{r_{i-1}} = \frac{1+n_{i-1}^{-1/4}/2}{r_i(1+n_{i-1}^{-1/4})} = \frac{1-n_{i-1}^{-1/4}/2(1+n_{i-1}^{-1/4})}{r_i} < \frac{1-n_{i-1}^{-1/4}/2.1}{r_i}.$$

The last inequality uses that $n_{i-1}^{-1/4} \geq n_0^{1/4} > 20$. Thus we have $|R|/|S| < (1 - n_{i-1}^{-1/4}/2.1)/r_i$.

First suppose that we are returning in Step A.2.3. Then $|R_i|$ is at least $x = n_i$. However, the expected value μ of $|R_i|$ is at most $n_i(1-n_{i-1}^{-1/4}/2.1) = n_i(1-1/(2.1\ln n_i))$. Note that

$$p_i \leq Pr(|R_{i-1}| > (1+\delta)\mu) \text{ with } \delta = (x-\mu)/\mu.$$

Using the Chernoff bound (according to [2, 9]),

$$Pr(|R_{i-1}| > (1+\delta)\mu) < e^{-\delta^2\mu/3} = e^{-(x-\mu)^2/(3\mu)}.$$

For $\mu \leq n_i(1-1/(2.1\ln n_i))$ this function is maximized for $\mu = n_i(1-1/(2.1\ln n_i))$. Thus,

$$p_i \leq \exp(\frac{-(n_i/(2.1\ln n_i))^2}{3n_i(1-1/(2.1\ln n_i))}) < \exp(\frac{-n_i}{13(\ln n_i)^2}) \leq n_i^{-2}.$$

For the last inequality, we use that $n_i \geq 26(\ln n_i)^3$ which follows from $\ln n_i \geq n_0^{1/4} = 26$.

Next suppose that we are returning in Step A.5. Then $|R_i|$ is at least $x = 8s/(cr_i)$ and $\mu \leq (1-1/(2.1\ln n_i))8s/(cr_i)$. Note that $x > n_{i-1}r_{i-1}/r_i > n_{i-1}$, since $8s/c > r_{i-1}n_{i-1}$. As above

$$p_i \leq \exp(\frac{-(x/(2.1n_{i-1}^{1/4}))^2}{3x(1-1/(2.1n_{i-1}^{1/4}))}) < \exp(\frac{-x}{13n_{i-1}^{1/2}}) \leq \exp(\frac{-n_{i-1}^{1/2}}{13}) \leq \exp(-2n_{i-1}^{1/4}) = n_i^{-2}.$$

For the last inequality, we actually require that $n_{i-1}^{1/4} \geq 26$.

□

We are now ready to show that the Algorithm A satisfies the conditions of Lemma 1.

(i) First we find the expected number of samples if the algorithm returns an element from R. By Claim 1C, for $i > 0$, the probability p_i of the algorithm returns an element from R in round i is bounded by n_i^{-2}. Moreover, by Claim 1B, if the algorithm returns in round i, it has sampled $O(rn_i)$ edges. Finally, by Claim 1Aa, $2n_i < n_{i+1}$. The expected number of samples is thus

$$\sum_{i=0}^{\infty} p_i O(rn_i) = O(rn_0 + \sum_{i=1}^{\infty} r/n_i) = O(rn_0 + 2r/n_1) = O(r).$$

(ii) Second we consider the case that the algorithm does not return an element from R, i.e. that the conditions in Steps A.2.3 and A.5 are never satisfied. Using Claim 1B, the total sample size is $\sum_{i=0}^{t} |S_i| = O(rn_{t-1}) + 8s/c = O(s/c)$.

Suppose $|R|/|S| > 1/r$. We did not return an element from R in Step A.5, so $X = |R_t|$ is less than $x = 8s/(cr_i) < 4s/(cr)$ by Claim 1Ac. However, the expected value μ of $|R_t|$ is at least $8s/(cr)$. The probability p is now calculated as in Case 1 of the proof of Claim 1C:

$$p \leq e^{-(\mu-x)^2/(2\mu)} \leq \exp(\frac{-(4s/(cr))^2}{2(8s/(cr))}) \leq \exp(-s/(cr)),$$

as desired. ■

At present, in case (i), we are making an expected number of $\leq 2n_0r_0 = 6 \cdot 26^4 r = O(r)$ samples. The constant can be reduced by adding a round -1, with $n_{-1} = 1$ (meaning that we return if we find just one representative) and $r_{-1} = 3 \cdot 14 = 42$ ($14 > \ln 26^4(1 + 2/24)$). This gives an expected number of $\leq 84r$ samples, which can be further reduced by introducing more preliminary rounds.

References

[1] N. Alon, J. Spencer, P. Erdös, The Probabilistic Method. *Wiley-Interscience Series*, Johan Wiley and Sons, Inc., 1992.

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

[3] 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.

[4] 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.

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

[6] 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.

[7] K. Mehlhorn. Data Structures and Algorithms 1: Sorting and Searching. *EATCS Monographs on Theoretical Computer Science*, Springer-Verlag, 1984.

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

[9] J. P. Schmidt, A. Siegel, A. Srinivasan. Chernoff-Hoeffding Bounds for Limited Independence. *SIAM J. on Discrete Mathematics* 8 (2), 1995, 223-250.

[10] R.E. Tarjan and U. Vishkin. Finding biconnected components and computing tree functions in logarithmic parallel time. *SIAM J. Computing*, 14(4): 862–874, 1985.