Sampling to Provide or to Bound: With Applications to Fully Dynamic Graph Algorithms*

Monika R. Henzinger †
Mikkel Thorup ‡

Abstract

In dynamic graph algorithms the following *provide-or-bound* problem has to be solved quickly: Given a set S containing a subset R and a way of generating random elements from S testing for membership in R, either (i) provide an element of R or (ii) give a (small) upper bound on the size of R that holds with high probability. We give an optimal algorithm for this problem.

This algorithm improves the time per operation for various dyamic graph algorithms by a factor of $O(\log n)$. For example, it improves the time per update for fully dynamic connectivity from $O(\log^3 n)$ to $O(\log^2 n)$.

1 Introduction

In this paper we present a new sampling lemma, and use it to improve the running times of various fully dynamic graph algorithms.

We consider the following *provide-or-bound* problem: Let S be a set with a subset $R \subseteq S$. Given (1) an oracle to uniformly sample elements in S, (2) an oracle to test membership in R, and (3) the parameters $s = |S|$ and $\theta > 1$ either

(i) provide an element of R, or

(ii) guarantee with high probability that the ratio $|R|/|S|$ is less than $1/\theta$, that is that $\theta|R| < |S|$.

This problem arises in the currently fastest fully dynamic graph algorithms for various problems on graphs, such as connectivity, two-edge connectivity, k-weight minimum spanning tree, $(1+\epsilon)$-approximate minimum spanning tree, and bipartiteness-testing [6]. The connection is made specific in Section 2.

---

*A preliminary version of this paper was presented at ICALP'96.

†Digital System Research Center, 130 Lytton Ave, Palo Alto, CA (*monika@pa.dec.com, http://www.research.digital.com/SRC/staff/monika*). On leave from: Department of Computer Science, Cornell University, Ithaca, NY. 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*).

In [6], Henzinger and King solve the problem by sampling $O(\theta \log |S|)$ elements from S, returning any element found from R. This gives an Monte-Carlo algorithm whose type (ii) answer is false with probability $1/|S|^{\Theta(1)}$. In this paper, we give a randomized Monte-Carlo algorithm for which the expected number of random samples from S is $O(\theta)$. To be precise, we show the following lemma.

Sampling Lemma *Let $\theta, c \in \mathbb{R}_{>1}$ and let R be a subset of a nonempty set S of size s. Given that we can uniformly sample from S and test membership of R, there is an algorithm with one of two outcomes:*

Case (i) Provide: *It returns an element from R.*

Case (ii) Bound: *It outputs the (possibly false) statement "$|R|/|S| < 1/\theta$".*

The statement returned in Case (ii) is false with error probability less than $\exp(-s/(\theta c))$. *The expected number of samples associated with a type (i) outcome is $O(\theta)$, and the worst-case number of samples associated with a type (ii) outcome is $O(s/c)$.*

It should be noted that the expected number of samples in Case (i) is not conditioned on ending in Case (i). If, say, $|R|/|S| = o(1/\theta)$, the expected cost of finding an element from R as in Case (i) is $\omega(\theta)$, but then the probability of ending in Case (i) is $o(1)$.

The bounds in case (i) and case (ii) are asymptotically optimal. Case (i) is optimal since it covers the case $|R|/|S| = 1/\theta$. For case (ii), note that if x elements from S are sampled randomly and no element of R is found, then the probability that $|R|/|S| \leq 1/\theta$ is approximately $\exp(-x/\theta)$. Thus, picking $O(s/c)$ random elements is asymptotically optimal for achieving a bound of $\exp(-s/\theta c)$ on the error probability.

We prove the sampling lemma in Section 3, in which we first prove a simpler lemma achieving an expected cost of $O(\log\log |S| \cdot \theta)$ for case (i). This is already a substantial improvement over the $O(\log |S| \cdot \theta)$ obtained in [6]. We then bootstrap the technique, giving the desired cost of $O(\theta)$.

1.1 Applications

The prime application of our sampling lemma is to speed up fully dynamic graph connectivity, which is the problem of maintaining a graph under edge insertions and deletions. Queries on the connectivity between specified vertices should be answered efficiently.

In the literature, fully dynamic graph algorithms are compared using the cost per insert, delete, and query operation. The best deterministic algorithms for fully dynamic graph connectivity take time $O(\sqrt{n})$ per update operation and $O(1)$ per query [2, 3, 5]. Recently, Henzinger and King gave a fully dynamic connectivity algorithm with $O(\log^3 n)$ expected amortized time per operation using Las-Vegas randomization [6]. This should be compared with a lower bound of $\Omega(\log n / \log\log n)$ in the cell probe model [4, 8].

In this papter, we prove a sampling lemma and use it to reduce the bound above to $O(\log^2 n)$.

Henzinger and King show that their approach applies to several other fully dynamic graph problems, for which we also get improved running times. Thus we get

- $O(k \log^2 n)$ to maintain a minimum spanning tree in a graph with k different weights;
- $O(\log^2 n \log U/\epsilon)$ 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,
- $O(\log^2 n)$ to test if the graph is bipartite, and
- $O(\log^2 n)$ to test if whether two edges are cycle-equivalent.

2 Improved sampling in fully dynamic graph connectivity

Our results for fully dynamic graph algorithms are achieved by locally improving a certain sampling 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 self-contained description of this bottleneck, focussing on connectivity. Our technique for the bottleneck is of a general flavor and we expect it to be applicable in other contexts.

Consider the problem of maintaining a spanning tree T of some connected graph $G = (V, E)$, $n = |V|$. If some tree edge e is deleted from T, we get two sub-trees T_1 and T_2. Let R be the set of non-tree edges with end-points in both T_1 and T_2. Then R is exactly the set of edges f that 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 replacement edge $f \in R$. Alternatively, it is acceptable to discover that R is sparse in the following sense: Let S be the set of non-tree edges incident to T_1. Then $R \subseteq S$, and we say that R is *sparse* if

$$\theta|R| < |S|, \text{ where } \theta = \Theta(\log n).$$

Otherwise R is said to be *dense*.

Given an algorithm that either (a) provides a replacement edge at expected cost $t(n)$, or (b) discovers that R is sparse at cost $O(t(n) + |S|)$, the amortized expected operation cost of Henzinger and King's fully dynamic connectivity algorithm is $O(t(n) + \log^2 n)$.

Using the data structures from the HK-approach, edges from S can be sampled and tested for membership in R in time $O(\log n)$. Also, in time $O(|S|)$, we can scan all of S, identifying all the edges in R.

The HK-approach achieves $t(n) = O(\log^3 n)$ as follows. First, $2\theta \ln n$ random edges from S are sampled. If the sampling successfully finds an edge from R, this edge is returned, as in (a). Otherwise, hoping for (b), in time $O(|S|)$, a complete scan of S is performed, identifying all edges of R. If it turns out, however, that R is dense, an edge from R is returned as in (a). The probability of this "mistake" is the probability of not finding a replacement edge in $2\theta \ln n$ samples despite R being dense, which is

$$\leq (1 - 1/\theta))^{2\theta \ln n} < 1/n^2 = O(1/|S|),$$

Thus, the expected cost of a mistake is $O((\log^3 n + |S|)/|S|)$. Adding up, Henzinger and King get $t(n) = O(\log^3 n)$, which is hence the expected amortized operation cost for their fully dynamic connectivity algorithm.

We achieve $t(n) = \log^2 n$ by applying our sampling lemma with $c = \ln n$ and $\theta = O(\log n)$. Then, in case (i) of the lemma, we find an element from R at expected cost $O(\log^2 n)$. In case (ii), the cost is $O(\log n \cdot |S| / \log n) = O(|S|)$, matching the cost of a subsequent scanning. According to the lemma, the probability that R turns out to be dense is $\exp(-|S|/\theta c) = \exp(-|S|/O(\log^2 n))$, so the expected contribution from such a mistaken scan is

$$O(|S| \exp(-|S|/O(\log^2 n))) = O(\log^2 n).$$

Thus, we get $t(n) = O(\log^2 n)$, which is hence the new expected amortized operation cost for fully dynamic connectivity.

All our other results for fully dynamic graph algorithms are achieved by the same local improvement.

3 The sampling lemma

The HK-approach solves the provide-or-bound problem as follows:

Algorithm A:

A.1. Let S_0 be a random subset of S of size $\theta \ln s$.

A.2. $R_0 := S_0 \cap R$.

A.3. If $R_0 \neq \emptyset$, then return $x \in R_0$.

A.4. Print "$|R|/|S| < 1/\theta$ with probability $> 1 - 1/s$."

Thus, the algorithm provides the first element of R that it finds. Only if it does not find one, does it give a bound on the size of R. Recall from the sampling lemma that we are willing to pay more for a bound on the size of R than for an element of R. Suppose that we have made many samples from S and that we have only found one or a few elements from R. Even if our sample size is not big enough for the desired high probability bound on R, it may still be fair to hypothesize that R is small. Instead of just returning the element from R, based on the hypothesis, we should rather continue sampling until we reach a sample size big enough for the desired probability bound on R. The probability that the continued sampling contradicts our hypothesis that R is small should be low, so that the expected cost of such a mistake is low.

We approximate this approach using a step function: To demonstrate a simplified version of our technique, we first show a weaker lemma in Section 3.1 using an algorithm with two rounds of sampling and bounding. In Section 3.2 we use $\log^* s$ rounds to prove the sampling lemma.

In this section, we make repeated use of the following Chernoff bounds (see [1], for example): Let $B(n, p)$ be a random variable that has a binomial distribution with parameters n and p. Then for $\delta \leq 1$,

$$\Pr(|B(n,p)| \geq (1+\delta)E(|B(n,p)|)) \leq e^{-\delta^2 E(|B(n,p)|)/3} \tag{1}$$

$$\Pr(|B(n,p)| \leq (1-\delta)E(|B(n,p)|)) \leq e^{-\delta^2 E(|B(n,p)|)/2} \tag{2}$$

3.1 Sampling in two rounds

Lemma 1 *Let $\theta, c \in \mathbb{R}_{>1}$ and let R be a subset of a nonempty set S of size s. Given that we can uniformly sample from S and test membership of R, there is an algorithm with one of two outcomes:*

Case (i) Provide: *It returns an element from R.*

Case (ii) Bound: *It outputs the (possibly false) statement "$|R|/|S| < 1/\theta$".*

The statement returned in Case (ii) is false with error probability less than $1/s$. The expected number of samples associated with a type (i) outcome is $4\theta(\ln\ln s + 2)$, and the worst-case number of samples associated with a type (ii) outcome is $8\theta \ln s + 4\theta \ln\ln s$.

Proof: The idea is the following: Instead of just sampling $O(\theta \log s)$ elements returning any element from R, we first make an *initial round*, where we sample $O(\theta \log\log s)$ elements. If an element from R is found, we just return it; otherwise, we believe that R is sparse, in other words that $|R|/|S| < 1/\theta$. In fact, with appropriately chosen constants, we conclude with error probability $O(1/\log s)$, that $|R|/|S| < 1/(4\theta)$. We now have a *confirming round*, where we sample $O(\theta \log s)$ elements. If the proportion of elements from R in this sample is $< 1/(2\theta)$, then using Chernoff bounds, we conclude that $|R|/|S| < 1/\theta$ with error probability $< 1/s$. We have a contradiction to the hypothesis that R is sparse otherwise and we return one of the elements of R found in the confirming round. However, using Chernoff bounds, we can show that the probability of entering the confirming round and finding a ratio $\geq 1/(2\theta)$ is $O(1/\log s)$, giving an expected cost of $O(\theta)$ for contradicting the confirming round.

We are now ready to formally present an algorithm with the properties described in Lemma 1.

Algorithm B:

B.1. Let S_0 be a random subset of S of size $4\theta \ln\ln s$.

B.2. $R_0 := S_0 \cap R$.

B.3. If $R_0 \neq \emptyset$, then return $x \in R_0$.

B.4. Let S_1 be a random subset of S of size $8\theta \ln s$.

B.5. $R_1 := S_1 \cap R$.

B.6. If $|R_1| \geq 4 \ln s$, then return any $x \in R_1$.

B.7. Print "$|R|/|S| < 1/\theta$ with probability $> 1 - 1/s$."

We show next a bound on the probability p that the algorithm returns an element from R in B.6 (Claim 1A), that is the initial guess of sparsity is *not* confirmed. Afterwards we prove that the Algorithm B satisfies the conditions of Lemma 1.

Claim 1A *The probability p that the algorithm returns an element from R in B.3 is $\leq 1/\ln s$.*

PROOF: We consider two cases:

Case 1: $|R|/|S| > 1/(4\theta)$. The algorithm can only reach B.6 if it did not return in B.3, and this happens with probability

$$p < (1 - 1/(4\theta))^{4\theta \ln \ln s} \leq e^{-\ln \ln s} = 1/\ln s$$

Case 2: $|R|/|S| \leq 1/(4\theta)$. Then the expected value of $|R_1|$ is at most $2 \ln s$. But

$$p \leq \Pr(|R_1| \geq 4 \ln s) \leq \Pr(|R_1| \geq 2E(|R_1|)) \leq e^{-E(|R_1|)/3} < 1/\ln s$$

The second inequality follows by Chernoff bound (1). The last inequality is trivially satisfied for $\ln s \leq 1$. Otherwise, since $x/\ln x \geq e$ for any real $x > 1$, we have $2(\ln s)/3 \geq 2e(\ln \ln s)/3 > \ln \ln s$.

□

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

Case (i) First, we determine the expected number of samples if the algorithm returns an element from R. By Claim 1A, the probability p that the algorithm returns an element from R in Step B.6 is bounded by $1/\ln s$. Thus, the expected number of samples is at most

$$4\theta \ln \ln s + 8\theta \ln s/\ln s = 4\theta(\ln \ln s + 2)$$

Case (ii) Second, we consider the case when the algorithm does not return an element from R, in other words when the conditions in Steps B.3 and B.6 are not satisfied. We want to show that the probability of this case is at most $1/s$.

Suppose $|R|/|S| \geq 1/\theta$. We did not return an element from R in Step B.6, so $|R_1| < 4 \ln s$. However, the expected value of $|R_1|$ is at least $8 \ln s$. Thus, by Chernoff bound (2),

$$\Pr(|R_1| < E(|R_1|)/2) \leq e^{-E(|R_1|)/8} = e^{-8 \ln s/8} = 1/s$$

■

In the next section we show the general sampling lemma.

3.2 Sampling in many rounds

In this section, we will prove the sampling lemma restated below.

Lemma 2 *Let $\theta, c \in \mathbb{R}_{>1}$ and let R be a subset of a nonempty set S of size s. Given that we can uniformly sample from S and test membership of R, there is an algorithm with one of two outcomes:*

Case (i) Provide: *It returns an element from R.*

Case (ii) Bound: *It outputs the (possibly false) statement "$|R|/|S| < 1/\theta$" .*

The statement returned in Case (ii) is false with error probability less than $\exp(-s/(\theta c))$. The expected number of samples associated with a type (i) outcome is $O(\theta)$, and the worst-case number of samples associated with a type (ii) outcome is $O(s/c)$.

Proof: We will now generalize the construction from the previous section to work with a sequence of confirming rounds, $i = 1, \ldots, \Theta(\log^* s)$. In round i, we will pick $r_i n_i$ random elements from S, and if at least n_i elements from R are found, one of these is returned. For the initial round 0, $n_0 = 1$, that is, any element from R is returned. In the subsequent confirming rounds, the numbers n_i of elements of R increase in order to increase our confidence. At the same time, the thresholds $1/r_i$ are increased, in order to minimize the probability that the threshold is passed in a later round. The concrete values of the n_i and r_i are fine tuned relative to the Chernoff bounds that we use to calculate our probabilities.

Let the increasing sequence $n_0, n_1 \ldots$ be defined such that $n_0 = 1$ and for $i > 0$, $n_i = a4^i(i+3)$, where $a = 64 \ln 16 < 178$. Let the decreasing sequence $r_0, r_1, \ldots$ be defined such that $r_0 = \ln(2n_1)2e\theta \approx 47\theta$ and for $i > 0$, $r_i = 2e\theta / \prod_{j=1}^{i}(1+1/2^j)$. Since $e > \prod_{j=1}^{\infty}(1+1/2^j)$,

$$r_i = 2e\theta / \prod_{j=1}^{i}(1+1/2^j) > 2\theta \prod_{j=i+1}^{\infty}(1+1/2^j) > 2\theta.$$

We are now ready to present an algorithm satisfying the conditions of Lemma 2.

Algorithm C:

C.1. $i := 0$; $S_{-1} := \emptyset$;

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

C.2.1. Let $S_i \subseteq S$, $|S_i| = n_i r_i$ be a random extension of S_{i-1}, i.e. S_i is obtained by adding $n_i r_i - |S_{i-1}|$ random elements from S to S_{i-1}.

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

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

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

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

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

C.5. If $|R_i| \geq 4s/(c\theta)$, then return $x \in S_i \cap R$.

C.6. Print "$|R|/|S| \leq 1/\theta$ with probability $> 1 - \exp(-s/\theta c)$."

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 C.2.3 or C.5.

Claim 2A *For all $i \geq 1$, the probability p_i that the algorithm returns an element from R is at least $1/(n_i 2^i)$.*

Proof: First note that

$$1/(n_i 2^i) = 1/(a4^i(i+3)2^i) = 1/(a8^i(i+3)) > 1/(178 \cdot 8^i(i+3))$$

We consider two cases:

Case 1: $|R|/|S| > (1+1/2^{i+1})/((1+1/2^i)r_i)$*:* First consider the case where $i = 1$. Then $|R|/|S| > (1+1/2^2)/((1+1/2)r_1) = (1+1/4)/2e\theta > 1/2e\theta$. Let b be $\ln(2n_1)$. We did not find any element from R in any of the $2be\theta$ samples in round 0, so

$$p_1 \leq (1-|R|/|S|)^{2be\theta} < (1-1/2e\theta)^{2be\theta} < e^{-b} = 1/(2n_1)$$

Now suppose $i > 1$. Then $|R|/|S| > (1+1/2^{i+1})/((1+1/2^i)r_i) = (1+1/2^{i+1})/r_{i-1}$. In round $i-1$ we did not return, so $x = |R_{i-1}|$ is less than n_{i-1}. However, the expected value of $|R_{i-1}|$ is

$$\mu = r_{i-1}n_{i-1}|R|/|S| > n_{i-1}(1+1/2^{i+1})$$

By Chernoff bound (2),

$$p_i \leq \Pr(|R_{i-1}| \leq x) \leq e^{-(\mu-x)^2/(2\mu)}$$

For $\mu \geq n_{i-1}(1+1/2^{i+1})$ and $x \leq n_{i-1}$, we get

$$\begin{aligned} p_i &\leq \exp\left(\tfrac{-(n_{i-1}/2^{i+1})^2}{2n_{i-1}(1+1/2^{i+1})}\right) \\ &< \exp(-n_{i-1}/2^{2i+4}) \\ &= 16^{-64\cdot 4^{i-1}\cdot(i-1+3)/2^{2i+4}} \\ &= 16^{-(i+2)} < 1/(178\cdot 8^i(i+3)) < 1/(n_i 2^i) \qquad (\forall i \geq 2) \end{aligned}$$

Case 2: $|R|/|S| \leq (1+1/2^{i+1})/((1+1/2^i)r_i)$*:* In order to avoid calculating the probabilities conditioned on the algorithm not having returned in any round $< i$, consider the following simplified algorithm D_i: Pick $r_i n_i$ random elements from S. If at least n_i elements belong to R, return one of them, otherwise do not return any element.

We show below that here in Case 2, the probability p_{D_i} that algorithm D_i returns an element of R is at most $1/(n_i 2^i)$. Now notice that the probability p_i that algorithm C returns an element in round i is at most p_{D_i}, since this event happens only if S_i contains at least n_i elements from R *and* none of the previous rounds returned an element of R. Thus, the claim follows if $p_{D_i} \leq 1/(n_i 2^i)$.

First suppose that we are returning in Step C.2.3. Then $x = |R_i|$ is at least n_i. However, the expected value μ of $|R_i|$ is at most

$$n_i r_i(1+1/2^{i+1})/((1+1/2^i)r_i) = n_i(1-1/(2^{i+1}+2)) < n_i(1-1/2^{i+2}) \qquad (\forall i \geq 1)$$

By Chernoff bound (1),

$$p_{D_i} \leq \Pr(|R_i| \geq x) \leq e^{-(x-\mu)^2/(3\mu)}$$

For $\mu \leq n_i(1 - 1/2^{i+2})$ and $x \geq n_i$, we get

$$\begin{aligned} p_{D_i} &\leq \exp\left(\tfrac{-(n_i/2^{i+2})^2}{3n_i(1-1/2^{i+2})}\right) \\ &< \exp(-n_i/(3 \cdot 4^{i+2})) \\ &< 40^{-(i+3)} < 1/(178 \cdot 8^i(i+3)) < 1/(n_i 2^i) \qquad (\forall i \geq 1) \end{aligned}$$

Next suppose that we are returning in Step C.5. Then $x = |R_i|$ is at least $4s/(c\theta)$ and the expected value μ is at most $(8s/c)(1 + 1/2^{i+1})/r_{i-1} \leq x(1 + 1/2^{i+1})2\theta/r_{i-1}$ Recall that the r_j were chosen so that

$$r_{i-1}/2\theta = e/\prod_{j=1}^{i-1}(1 + 1/2^j) > \prod_{j=i}^{\infty}(1 + 1/2^j).$$

Hence

$$\begin{aligned} \mu &\leq x(1 + 1/2^{i+1})2\theta/r_{i-1} \\ &< x(1 + 1/2^{i+1})/((1 + 1/2^i)(1 + 1/2^{i+1})\cdots) \\ &< x/(1 + 1/2^i) \\ &< x(1 - 1/2^{i+1}) \qquad (\forall i \geq 1) \end{aligned}$$

Note that $x > n_{i-1}r_{i-1}/r_i > n_{i-1}$ since $8s/c > r_{i-1}n_{i-1}$ and $r_i > 2\theta$. Thus, we get the desired bound on p_{D_i}.

$$\begin{aligned} p_{D_i} &\leq \exp\left(\tfrac{-(x/2^{i+1})^2}{3x(1-1/2^{i+1}))}\right) \\ &< \exp(-x/(3 \cdot 2^{2i+2})) \\ &\leq 40^{-(i+2)} < 1/(178 \cdot 8^i(i+3)) < 1/(n_i 2^i) \qquad (\forall i \geq 1) \end{aligned}$$

□

We are now ready to show that the Algorithm C satisfies the conditions of Lemma 2.

Case (i) First, we analyze the expected number of samples associated with a type (i) outcome. By Claim 2A, for $i > 0$, the probability p_i that the algorithm returns an element from R in round i is bounded by $1/(n_i 2^i)$. The expected number of samples is thus bounded by 54θ:

$$\begin{aligned} &r_0 + \textstyle\sum_{i=1}^{\infty} p_i r_i n_i \\ &\leq r_0 + \textstyle\sum_{i=1}^{\infty} r_i n_i/(n_i 2^i) \\ &\leq r_0 + \textstyle\sum_{i=1}^{\infty} 2e\theta/2^i \\ &= r_0 + 2e\theta \leq 54\theta \end{aligned}$$

Case (ii) Second, we consider the case that the algorithm does not return an element from R, in other words that the conditions in Steps C.2.3 and C.5 are never satisfied. Then, the total sample size is $8s/c$.

Suppose $|R|/|S| > 1/\theta$. We did not return an element from R in Step C.5, so $x = |R_i|$ is less than $4s/(c\theta)$. However, the expected value μ of $|R_i|$ is at least $8s/(c\theta)$. The probability p is now calculated as in Case 1 of the proof of Claim 2A.

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

■

Acknowledgements

We want to thank Andrei Broder and Lyle Ramshaw for their valuable comments on the presentation of the paper.

References

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

[2] 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.

[3] 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.

[4] M. L. Fredman and M. R. Henzinger. Lower Bounds for Fully Dynamic Connectivity Problems in Graphs. To appear in *Algorithmica.*

[5] G. N. Fredrickson. Data Structures for On-line Updating of Minimum Spanning Trees. *SIAM J. Comput.* (14), 1985, 781–798.

[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] 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.