Randomized Minimum Spanning Tree Algorithms Using Exponentially Fewer Random Bits

SETH PETTIE
University of Michigan
and
VIJAYA RAMACHANDRAN
The University of Texas at Austin

For many fundamental problems there exist randomized algorithms that are asymptotically optimal and are superior to the best known deterministic algorithm. Among these are the minimum spanning tree (MST) problem, the MST sensitivity analysis problem, the parallel connected components and parallel minimum spanning tree problems, and the local sorting and set maxima problems. (For the first two problems there are provably optimal deterministic algorithms with unknown, and possibly superlinear running times.) One downside of the randomized methods for solving these problems is that they use a number of random bits linear in the size of the input. In this paper we develop some general methods for reducing exponentially the consumption of random bits in comparison based algorithms. In some cases we are able to reduce the number of random bits from linear to *nearly* constant without affecting the expected running time.

Most of our results are obtained by adjusting or reorganizing existing randomized algorithms to work well with a pairwise or $O(1)$-wise independent sampler. The prominent exception — and the main focus of this paper — is a linear-time randomized minimum spanning tree algorithm that is *not* derived from the well known Karger-Klein-Tarjan algorithm. In many ways it resembles more closely the *deterministic* minimum spanning tree algorithms based on Soft Heaps. Further, using our algorithm as a guide, we present a unified view of the existing "non-greedy" minimum spanning tree algorithms. Concepts from the Karger-Klein-Tarjan algorithm, such as F-lightness, MST verification, and sampled graphs, are related to the concepts of edge corruption, subgraph contractibility, and Soft Heaps, which are the basis of the deterministic MST algorithms of Chazelle and Pettie-Ramachandran.

Categories and Subject Descriptors: G.2.2 [**Graph Theory**]: Graph algorithms; F.2.0 [**Analysis of Algorithms and Problem Complexity**]: General; G.3 [**Probability and Statistics**]: Probabilistic algorithms

General Terms: Graph algorithms, minimum spanning trees, random sampling

Additional Key Words and Phrases:

This work was supported by Texas Advanced Research Program Grant 003658-0029-1999 and NSF Grant CCR-9988160. The first author was partly supported by an Alexander von Humboldt Postdoctoral Fellowship. A preliminary version of this paper [Pettie and Ramachandran 2002a] appeared as: Minimizing randomness in minimum spanning tree, parallel connectivity, and set maxima algorithms, Proceedings 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 713–722, 2002. Authors' addresses: Seth Pettie, Department of EECS, University of Michigan, Ann Arbor, MI 48109. Email: pettie@umich.edu. Vijaya Ramachandran, Department of Computer Sciences, The University of Texas at Austin, Austin, TX 78712. Email: vlr@cs.utexas.edu.

1. INTRODUCTION

In the mid-1990s researchers discovered two theoretically efficient methods for solving the fundamental minimum spanning tree (MST) problem that moved away from the traditional greedy approach. One, developed by Karger, Klein, and Tarjan [1995], involves random sampling and makes heavy use of Komlós's [1985] linear time minimum spanning tree verification algorithm. The other, developed by Chazelle [2000a] and further by Pettie and Ramachandran [2002b], is based on the Soft Heap [Chazelle 2000b], a kind of priority queue that is allowed to make many well behaved errors. Both approaches are "non-greedy" inasmuch as they construct approximately optimal spanning trees in the process of finding the minimum spanning tree. However, except for this quality they are apparently completely different.

In this paper we present a new minimum spanning tree algorithm running in expected linear time. One noticeable feature of our algorithm is how it bridges the gap between the randomized approach to the problem [Karger et al. 1995] and the deterministic ones based on Soft Heaps [Chazelle 2000a; Pettie and Ramachandran 2002b]. By unifying the *language* used to describe non-greedy MST algorithms we believe our algorithm might shed some light on their underlying nature.

The original motivation for our work was reducing the number of *random bits* used to solve several problems in optimal expected time. Besides the minimum spanning tree problem these include the parallel connected components and parallel MST problems, the set maxima and local sorting problems, and MST sensitivity analysis. Derandomization is still the primary technical focus of this paper: for all of these problems we present optimal (or optimal-work) randomized algorithms that use at most a polylogarithmic number of random bits.

In the rest of this section we give some background on the problems listed above (in Section 1.1) and we summarize our results (in Section 1.2).

1.1 About the Problems

1.1.1 *Minimum Spanning Trees.* We can only review the last few developments in the history of the MST problem. More background and references can be found in Graham and Hell's survey [1985] and in [Pettie 2003].

Between the late 1920s and 1980s all minimum spanning tree algorithms could be construed as different implementations of a generic greedy algorithm. The culmination of this greedy approach is an algorithm of Gabow et al. [1986], which runs in time $O(m(1+\log^* n-\log^* \frac{m}{n}))$. Although it might still be possible to improve this result within the greedy paradigm, all subsequent research on the MST problem has pursued non-greedy approaches.

Karger, Klein, and Tarjan [1995] presented a *randomized* MST algorithm running in expected linear time. Their algorithm computes (using random sampling of edges and recursion) a spanning tree T that is probably nearly optimal, meaning that there are very few edges $e \notin T$ such that for some $f \in T$, $T\backslash\{f\} \cup \{e\}$ results in a lighter spanning tree. Such edges might still be in the minimum spanning tree; all others can be identified and discarded with one of the linear time MST verification algorithms [Komlós 1985; Dixon et al. 1992; King 1997; Buchsbaum et al. 1998]. The Karger-Klein-Tarjan algorithm [1995] is optimal in one sense but leaves a couple of things to be desired. First, it uses a linear number of random bits, which as we will see is unnecessarily large. Second, it sheds less light on the MST problem than one might hope—the real mechanics of the algorithm are hidden within recursive calls and a black box minimum spanning tree verification procedure.

Chazelle developed a deterministic non-greedy minimum spanning tree algorithm that seemed to have no direct relationship with that of Karger et al. That is, it did not simply replace portions of the Karger et al. algorithm by equivalent deterministic procedures. Chazelle's algorithm [2000a] runs in $O(m\alpha(m,n))$ time, where α is the slowly growing inverse-Ackermann function.[1] Building on Chazelle's algorithm, Pettie and Ramachandran [2002b] presented an explicit, provably optimal MST algorithm whose running time lies somewhere between $\Omega(m)$ and $O(m\alpha(m,n))$, but is currently unknown. This result proved that the uniform and non-uniform complexities of the MST problem are asymptotically equivalent[2]; that is, the running time of the (uniform) algorithm is equivalent to the (non-uniform) decision-tree complexity of the MST problem itself.

1.1.2 *Parallel Connectivity & MSTs.* The connected components problem is trivial to solve in linear time on a RAM or any similar model of computation. However, the parallel complexity of the problem is still open. To date there is no deterministic PRAM [Karp and Ramachandran 1990] algorithm that runs in polylogarithmic time and uses linear work.

Gazit [1991] and Halperin and Zwick [1996; 2001] presented *randomized* logarithmic-time linear-work connectivity algorithms, for the CRCW and EREW PRAM models respectively. Deterministic logarithmic-time algorithms were discovered by Cole and Vishkin [1991] (CRCW) and then Chong, Han, and Lam [2001] (EREW), though neither uses linear-work. The most work-efficient deterministic parallel algorithm is Cole and Vishkin's [1991], which performs $O(m\alpha(m,n))$ work on the CRCW PRAM.

The situation is very similar for the parallel minimum spanning tree problem, except that there is no *trivial* linear-time sequential algorithm. Cole et al. [1996] presented a randomized linear-work CRCW algorithm, which was followed by Pettie and Ramachandran's [2002c] randomized time-work optimal EREW algorithm. In the deterministic setting, Chong, Han, and Lam [2001] presented a deterministic EREW algorithm that is time (but not work) optimal. The most work-efficient deterministic parallel algorithm is due to Cole and Vishkin [1999], which takes $O(m \log\log\log n)$ work on the CRCW PRAM.

All of the randomized algorithms mentioned above use a linear number of random bits.

1.1.3 *Local Sorting, Set Maxima, and MST Sensitivity.* Given a set system (χ, S), where χ is a set of n weighted elements and $S = \{S_1, S_2, \ldots, S_m\}$ is a

[1]An alternative exposition of Chazelle's algorithm appears in [Pettie 1999].
[2]We use the terms "asymptotically optimal" and "asymptotically equivalent" to mean optimal and equivalent up to leading constant factors.

set of m subsets of χ, the set maxima problem is to compute the maximum weight element in each S_i. A number of well-studied problems are special cases of set maxima or easily reducible to it. The local sorting, MST verification, and MST sensitivity analysis problems are a few; see Section 6.1 for other examples and more details.

Graham et al. [1980] noted two simple algorithms for set maxima, which are the best deterministic solutions to date. Sorting the elements takes $O(n \log n)$ comparisons, which is optimal for $m = n^{1+\Omega(1)}$, and computing the maxima obliviously takes $O(n+m2^m)$ comparisons, which is optimal for $m \leq \log n - \log\log n + O(1)$ (see Section 6.1 for more details). Goddard et al. [1993] demonstrated that set maxima could be solved in a randomized fashion with an expected $O(\min\{n \log(2 + \frac{m}{n}), n \log n\})$ comparisons, which is optimal for all m. However their algorithm uses a linear number of random bits.

1.2 Organization & Summary of Our Results

In many randomized algorithms one encounters a step of the form: *sample each element of some universe independently with probability p.* A step of this kind requires a number of random bits linear in the size of the universe, which may be infeasible. A standard trick is to approximate the fully independent sample by selecting pn elements uniformly at random. This alternative requires $O(pn \log n)$ random bits, which may still be prohibitive. In Section 2 we show that if the universe is a set of totally ordered elements, an $O(1)$-wise independent sampler possesses (or approximates) some key properties of a totally independent sampler while consuming only a logarithmic number of random bits. We also describe in Section 2 an efficient pairwise-independent sampler that is particularly effective in parallel algorithms. Many of the parameters in our algorithms are designed with a pairwise or $O(1)$-wise independent sampler in mind.

The rest of the paper presents our reduced-randomness algorithms, most of which make use of the results in Section 2. In Section 3 we present a new minimum spanning tree algorithm that runs in linear expected time and uses a polylogarithmic number of random bits. In Section 4 we show that this algorithm is effectively parallelizable. Our parallel MST algorithm, which also solves the simpler connected components problem, uses a polylogarithmic number of random bits and runs in expected polylogarithmic time with linear work. In Section 5 we discuss the relationship between the non-greedy MST algorithms [Karger et al. 1995; Chazelle 2000a; Pettie and Ramachandran 2002b], using our randomized MST algorithm to illustrate related concepts. In Section 6 we give other low-randomness algorithms: Section 6.1 gives algorithms for set maxima and local sorting that use a polylogarithmic number of random bits. In Section 6.2 we give a general scheme for reducing the number of random bits required to solve problems that possess a certain decomposition property. Using this scheme we show that the MST and MST sensitivity analysis problems can be solved in expected linear time using $o(\log^* n)$ random bits. (This low-randomness MST algorithm uses fewer random bits than the one from Section 3. However, it sheds little light on the MST problem and is not parallelizable.)

2. LIMITED INDEPENDENCE SAMPLING

In this section we examine the properties of a limited independence sampler when sampling from a totally ordered set. In Section 2.1 we show that a 4-wise independent sampler is just as good as a totally independent one, for the purpose of choosing some element whose expected rank is close to a desired rank. Under the assumption of pairwise independence we prove a somewhat weaker guarantee. In Section 2.2 we give a k-wise independent sampler based on Joffe's [1974] construction of k-wise independent variables. We show, further, that our pairwise independent sampler can be effectively parallelized. It yields not only a work-optimal sampler but an ideal scheme for assigning sampled elements to processors.

2.1 Sampling from a Total Order

Suppose that we must find the rank t element from some total order but cannot bother to examine every element. This is clearly impossible. However, one can sample the elements independently with probability p and choose the rank $\lceil pt \rceil$ sampled element. This simple trick produces an element whose expected rank is within p^{-1} of t, which is good enough in many algorithmic applications. In this section we bound the expected accuracy of this procedure when a k-wise independent sampler is substituted for a totally independent one. Assuming a pairwise independent sampler, we show the rank of the returned element is $O(t+p^{-1}(1+\log(n/t)))$, and that with a 4-wise independent sampler, it is $O(t+p^{-1})$. Higher orders of independence naturally lead to a distribution more concentrated around its mean.

Definition 2.1. *Random variables $X_1,\dots,X_n$ are k-wise independent if for any distinct indices $i_1,\dots,i_k$ and values $x_1,\dots,x_k$:*

$$\Pr[X_{i_1}=x_1\wedge\cdots\wedge X_{i_k}=x_k] \quad = \quad \Pr[X_{i_1}=x_1]\times\cdots\times\Pr[X_{i_k}=x_k]$$

Theorem 2.2 is a slightly simplified version of Theorem 4(I) appearing in Schmidt et al. [1995]. It is based on an analysis of the kth moment inequality $\Pr[|X-\mathbb{E}[X]|\geq T]\leq \mathbb{E}\left[(X-\mathbb{E}[X])^k\right]/T^k$, which holds for even k.

Theorem 2.2. *(Schmidt et al. [1995]) Let $X_1,\dots,X_n$ be a sequence of random k-wise independent $0/1$ variables with $X=\sum_{i=1}^n X_i$. If $k\geq 2$ is even and $C\geq\mathbb{E}[X]$ then:*

$$\Pr[|X-\mathbb{E}[X]|\geq T]\leq\left[\sqrt{2}\cosh\left(\sqrt{k^3/36C}\right)\right]\cdot\left(\frac{kC}{eT^2}\right)^{k/2}$$

Remark. The factor in square brackets in Theorem 2.2 becomes unwieldy when C is very small. We will only employ Theorem 2.2 when $C=\mathbb{E}[X]\geq 1$. In this case the bracketed factor is less than 2 for $k=2$ and less than 3 for $k=4$.

The main lemma of this section is given below.

Lemma 2.3. *Let χ be a set of n totally ordered elements and χ_p be a subset of χ derived by sampling each element with probability p using a k-wise independent sampler. Let Z be the number of unsampled elements less than $\min\chi_p$. Then*

$$\mathbb{E}[Z]\leq\begin{cases} p^{-1}(4\ln(pn)/e+1) & \text{for } k\geq 2\\[2ex] p^{-1}(8(\pi/e)^2+1) & \text{for } k\geq 4\end{cases}$$

PROOF. Let $X_i = 1$ if the element of χ with rank i is sampled, and 0 otherwise. So $\mathbb{E}[X_i] = p$ and for any distinct indices $i_1, \ldots, i_k$, $X_{i_1}, \ldots, X_{i_k}$ are independent. Let $S_\ell = \sum_{i=1}^{\ell} X_i$ count the number of ones in $X_1, \ldots, X_\ell$. The expectation of S_ℓ is clearly $p\ell$, implying:

$$\Pr[Z \geq \ell] = \Pr[S_\ell = 0] \leq \Pr[|S_\ell - \mathbb{E}(S_\ell)| \geq p\ell]$$

We bound $\mathbb{E}[Z]$ using Theorem 2.2 as follows:

$$\begin{aligned}
\mathbb{E}[Z] &= \sum_{\ell=1}^{\infty} \Pr[Z \geq \ell] \\
&\leq p^{-1} + \sum_{\ell=p^{-1}}^{n} \Pr[|S_\ell - \mathbb{E}[S_\ell]| \geq p\ell] \\
&\leq p^{-1} + c \sum_{\ell=p^{-1}}^{n} \left(\frac{k}{ep\ell}\right)^{k/2} \\
&\leq p^{-1} + c(k/ep)^{k/2} \sum_{\ell=p^{-1}}^{n} \ell^{-k/2} \\
(\text{for } k=2, c=2) &\leq p^{-1} + 4\ln(pn)/ep \;=\; p^{-1}(1 + 4\ln(pn)/e) \\
(\text{for } k=4, c=3) &\leq p^{-1}\left(1 + 48/pe^2 \cdot \sum_{\ell=p^{-1}}^{n} \ell^{-2}\right) \\
&\leq p^{-1}\left(1 + 48/p^2e^2 \sum_{\ell=1}^{\infty} (p^{-1}\ell)^{-2}\right) \\
&\leq p^{-1}(1 + 8\pi^2/e^2)
\end{aligned}$$

□

The proof of the following Lemma 2.4 follows the same lines as Lemma 2.3.

LEMMA 2.4. *Let χ be a set of n totally ordered elements and χ_p be a subset of χ derived by sampling each element with probability p using a k-wise independent sampler. Let x_t be the element of χ_p with rank t and let Z_t be the number of elements in χ less than x_t. Then*

$$\mathbb{E}[Z_t] = \begin{cases} O(t/p + \log(pn/t)/p) & \text{for } k \geq 2 \\ O(t/p) & \text{for } k \geq 4 \end{cases}$$

PROOF. Let X_i and S_i be as defined in Lemma 2.3. Using the same arguments as in Lemma 2.3 we have:

$$\mathbb{E}[Z_t] = \sum_i \Pr[Z_t \geq i]$$

$$
\begin{aligned}
&\leq\ 2tp^{-1} + p^{-1} \cdot \sum_{j=2t}^{pn} \Pr[Z_t \geq j/p] \\
&\leq\ 2tp^{-1} + p^{-1} \cdot \sum_{j=2t}^{pn} \Pr[|S_{j/p} - \mathbb{E}[S_{j/p}]| \geq j - t] \\
&\leq\ 2tp^{-1} + c_1 p^{-1} \cdot \sum_{j=2t}^{pn} \left(\frac{kj}{e(j-t)^2}\right)^{k/2} \\
&\leq\ 2tp^{-1} + c_2 p^{-1} \cdot \sum_{j=2t}^{pn} j^{-k/2} \\
\text{for } k \geq 2 &=\ O(t/p + \log(pn/t)/p) \\
\text{for } k \geq 4 &=\ O(t/p)
\end{aligned}
$$

The constants c_1 and c_2 were introduced to simplify the presentation of the proof. □

2.2 Pairwise Independent Sampling in Parallel

The analyses of our algorithms are generally independent of the pairwise independent sampler used, so long as it possesses a couple key properties. It is important that the sampling algorithm takes time linear in the size of the sample, not the universe. For our parallel MST and connectivity algorithms it is also necessary that the sampler take constant time and linear work, including the cost of fairly allocating sampled elements to processors. In this section we give a pairwise independent sampler with these properties that is based on Joffe's [1974] construction of k-wise independent variables.

Definition 2.5. *A* k-wise independent sample of χ *is a random set* S *such that for any* $\{x_1, \ldots, x_k\} \subseteq \chi$, $\Pr[\{x_1, x_2, \ldots, x_{k-i}\} \subseteq S \wedge \{x_{k-i+1}, \ldots, x_k\} \cap S = \emptyset] = p^{k-i}(1-p)^i$*, where* p *is the sampling probability.*

Lemma 2.6. *(Joffe [1974]) Let* q *be prime,* $a_0, a_1, \ldots, a_{k-1}$ *be chosen uniformly at random from* $\mathbb{Z}_q$*, and* $X(i) = \sum_{j=0}^{k-1} a_j \cdot i^j \pmod{q}$*. Then* $X(0), \ldots, X(q-1)$ *are uniformly distributed over* $\mathbb{Z}_q$ *and* k*-wise independent.*

From Lemma 2.6 we have the useful fact that for generating pairwise independent variables we require only two random coefficients, a_0 and a_1. We use this to sample a set of size q as follows. Let p be the desired sampling probability.[3] If $X(i) \in [0, \lceil pq \rceil - 1]$ then element i is sampled; otherwise it is not. Evaluating the polynomial X on q points is too expensive because the number of sampled elements could be sublinear in q. Under the assumption that $a_1 \neq 0$ we can generate the sampled set by generating all solutions to $i = (j - a_0)a_1^{-1} \pmod{q}$ for $j \in [0, \lceil pq \rceil - 1]$. This leads to a simple scheme for assigning sampled elements to processors on an EREW PRAM — refer to Figure 1. (The sequential version of the procedure is obtained when the number of processors $P = 1$.)

[3]Due to rounding the actual sampling probability will be $\lceil pq \rceil / q$, not p.

Pairwise-Sampler:

Specs: There are P EREW processors and q elements, where q is prime. We must find a pairwise independent sample (sampling probability p) and distribute sampled elements among the P processors.
We assume a_0, a_1 have been selected uniformly at random from $\mathbb{Z}_q$, and that each processor knows p, q, a_0, a_1, a_1^{-1} and its processor ID.

Case 1.. If $a_1 = 0$ and $a_0 \geq \lceil pq \rceil$ then $X(i) = a_0$. No elements are sampled.
Case 2.. If $a_1 = 0$ and $a_0 < \lceil pq \rceil$ then all elements are sampled. Processor k is assigned elements $\lceil \frac{q}{P} \rceil k$ through $\lceil \frac{q}{P} \rceil (k+1) - 1$.
Case 3.. If $a_1 \neq 0$, then processor k is assigned elements of the form

$$(j - a_0)a_1^{-1} \pmod q$$

for all j from $\lceil \frac{pq}{P} \rceil k$ through $\lceil \frac{pq}{P} \rceil (k+1) - 1$.

Fig. 1. A combination pairwise-independent sampler and processor allocation scheme.

Lemma 2.7. *Suppose a parallel algorithm requires s pairwise independent samples from a set of size q (a prime), with perhaps different sampling probability for each sample. On an EREW PRAM the samples can be generated and fairly allocated to processors using $O(s \log q)$ random bits, in $O(s + \log q)$-time, and with work linear in the size of all samples.*

Proof. Our EREW sampling algorithm is given in Figure 1. For it to work, every processor must know two random elements $a_0, a_1 \in Z_q$ and a_1^{-1} (if $a_1 \neq 0$) We assign s processors to generate one (a_0, a_1) pair and compute a_1^{-1}. This takes $O(\log q)$ time. In $O(s + \min\{\log P, \log q\})$ time we distribute the (a_0, a_1, a_1^{-1}) tuples to the first $\min\{P, q\}$ processors All other processors do not participate. □

In situations where pairwise independent sampling suffices, our processor allocation scheme is significantly more desirable than other randomized allocation schemes [Halperin and Zwick 2001; 1996]. It is quick, uses minimal communication, and distributes the sampled set perfectly: every processor's load is less than the average load plus 1.

One problem that remains is finding a prime q in parallel for use with Joffe's construction. This is not too hard: we simply run the randomized Miller-Rabin primality test [Miller 1976; Rabin 1980] on a sequence of integers known to contain a prime. Baker and Harman [Bach and Shallit 1996, p. 225] showed that if p_n is the nth prime, then $p_n - p_{n-1} \leq n^{.535+o(1)}$.

Lemma 2.8. *Let q be the smallest prime such that $q \geq m$. Then with probability at least $1 - m^{-2c+1}$, q can be found on the EREW PRAM in $O(\log m)$ time, using $m^{.54}$ processors and $c \log^2 m$ random bits.*

Proof. We run the Miller-Rabin primality test $c \log m$ times on each integer in the interval $[m, m + b(m)]$, where $b(m) = m^{.535+o(1)}$, reusing the same random bits for each number tested. The probability that Miller-Rabin reports the wrong answer for any of the numbers is $\leq b(m) \left(\frac{1}{4}\right)^{c \log m} \leq m^{-2c+1}$. Each test uses $\log m$ random bits and takes time $O(\log m)$, hence finding the first prime $\geq m$ takes $c \log^2 m$ random bits and $O(\log m)$ time using $b(m) \cdot c \log m$ processors. □

Remark. In our MST algorithm m represents the number of edges still in play. Since this quantity is gradually reduced over successive phases we require a different prime for each phase. By reusing random bits in the procedure above we can compute up to $\Theta(m/b(m))$ primes, which is more than enough, in $O(m)$ work, $O(\log m)$ time, and with $O(c \log^2 m)$ random bits.
Remark. The number of random bits claimed in Lemma 2.8 can actually be reduced to $O(c \log m)$ using the method developed by Ajtai et al. [1987] and further in [Impagliazzo and Zuckerman 1989; Cohen and Wigderson 1989]. This algorithm involves associating the $O(\log m)$-bit random inputs with the vertices of an expander graph and taking a random walk of length $O(c \log m)$. The algorithm (Miller-Rabin in our case) is re-run at each vertex encountered.

3. A RANDOMIZED MINIMUM SPANNING TREE ALGORITHM

In this section we present a randomized MST algorithm that runs in expected linear time using only polylog random bits. In Section 4 we show that this algorithm can be effectively parallelized and in Section 5 we discuss it relative to the other non-greedy MST algorithms. Later, in Section 6.2, we give another MST algorithm running in expected linear time and using $o(\log^* n)$ random bits; however this algorithm is not parallelizable and contributes little to the discussion of non-greedy MST algorithms.

The goal of our algorithm, as in [Karger et al. 1995; Chazelle 2000a; Pettie and Ramachandran 2002b], is to construct a sequence of approximately minimum spanning trees, each more accurate than the last, until the actual minimum spanning tree is obtained. The accuracy of any spanning tree T is measured by the number of edges e for which $T = MST(T \cup \{e\})$.

The fundamental operations of our algorithm are standard *Borůvka steps*, *approximate* Borůvka steps, which will be discussed in Section 3.1, and MST verification.

3.1 Approximate Borůvka Steps

In a *Borůvka step* we identify and contract the lightest edge incident to each vertex. By the *cut property* of minimum spanning trees all edges contracted are in the MST. Since each Borůvka step reduces the number of (non-isolated) vertices by half and runs in linear time, the overall complexity of Borůvka's algorithm is $O(m \log n)$.[4] Our algorithm is based on an *approximate* version of the standard Borůvka step that requires only sublinear time but still provides useful information about the minimum spanning tree.

An approximate Borůvka step is essentially a Borůvka step applied to a subset $\hat{E}$ of the edges, which in our case will always be obtained by random sampling. Each vertex selects and contracts the lightest incident edge that is both in $\hat{E}$ and *untainted* — more on this soon. Clearly the edges contracted in an approximate Borůvka step are not necessarily in the MST. Indeed, some edges may *bear witness* to the fact that their end points selected a suboptimal edge. If particular, if the edge (u, v) is lighter than either the edge chosen by u or the edge chosen by v then (u, v) is tainted and cannot participate in further approximate Borůvka steps. For technical reasons we will taint (u, v) if either of u or v fails to choose an edge, or

[4]It is possible to implement Borůvka's algorithm faster; see [Gabow et al. 1989].

if (u, v) is the edge selected by u or v. Definition 3.1 defines tainted edges more precisely.

Definition 3.1. *Let $G = (V, E)$ be a graph in which some edges in E are labeled as being tainted, and let $\hat{E} \subseteq E$ be an arbitrary subset of the edges. Let e_v be the lightest incident edge on vertex v that is in $\hat{E}$, not a self-loop, and not tainted. In an* approximate Borůvka step *w.r.t. $\hat{E}$ the edges $\{e_v\}_v$ are identified and contracted. An edge $(x, y) \in E$, where $x \neq y$, becomes* tainted *if $w(x, y) \leq \max\{w(e_x), w(e_y)\}$, where $w(e_x) = \infty$ if e_x does not exist. (Note that all edges in $\{e_v\}_v$ become tainted.)*

Definition 3.2. *To* retract *the last (approximate) Borůvka step means to undo the edge contractions performed in that step and to mark untainted those edges tainted in that step.*

Throughout the paper the phrase "(approximate) Borůvka step" means either a standard or approximate Borůvka step. For The pseudocode for an approximate Borůvka step is given in Figure 2.

```
Approx-Borůvka-Step($G, \hat{E}, E_t$)   :   graph $G$, $\hat{E}, E_t \subseteq E(G)$
$E_t$ are the previously tainted edges

1.    Let $e_v$ be lightest edge in $(\{v\} \times V(G)) \cap \hat{E} \backslash E_t$
2.    For each edge $(u, v) \in E(G)$,
3.        If $e_u$ or $e_v$ do not exist or $w(u, v) \leq \max\{w(e_u), w(e_v)\}$
4.            $E_t := E_t \cup \{(u, v)\}$
5.    $G' = G$ after contracting $\{e_v\}_{v \in V(G)}$
6.    Return $G'$, $E_t$        New graph, new set of tainted edges
```

Fig. 2. Approximate Borůvka Step

Suppose that we apply a sequence of approximate Borůvka steps until all edges become self-loops (due to contractions) or tainted. We will show that any untainted edge cannot be in the MST of the original graph. Furthermore, if the sampling probabilities are chosen carefully, most edges will never be tainted.

Lemmas 3.3–3.6, given below, constitute the salient features of approximate Borůvka steps. Informally, they are: (1) not that many edges become tainted in any step; (2) the time required for each step is reasonable, though superlinear in $|\hat{E}|$; and (3) any untainted edge that becomes a self-loop (due to edge contractions) is not in the minimum spanning tree. Parts (1) and (2), corresponding to Lemmas 3.3 and 3.4, respectively, are used to bound the overall running time of our algorithm. Part (3), given in Lemma 3.6, proves the correctness of our algorithm.

Lemma 3.3. *Consider a graph with edge set E where some of the edges may be tainted. Let $E_p \subseteq E$ be obtained by sampling E with probability p using a k-wise independent sampler. The expected number of edges that become tainted in an approximate Borůvka step w.r.t. E_p is $O(n \log(2m/n)/p)$ for $k \geq 2$ and $O(n/p)$ for $k \geq 4$, where m and n are the number of edges and vertices, respectively.*

Proof. Let L_v be the list of eligible edges incident on v (must be in E_p, untainted, not self-loops) and let L'_v be those edges tainted by the Borůvka step w.r.t.

E_p. By Lemma 2.3 $\mathbb{E}[|L'_v|]$ is $O(p^{-1}\log(p|L_v|))$ for $k \geq 2$ and $O(p^{-1})$ for $k \geq 4$. The expected number of tainted edges is then $O(\sum_v \mathbb{E}[|L'_v|])$ by linearity of expectation. For $k \geq 4$ this bound is $O(n/p)$ and for $k \geq 2$ the bound is, by the concavity of the log function, $O(np^{-1}\log(2m/n))$. $\square$

Lemma 3.4. *Let G be a graph with no tainted edges and let H be derived from G by ℓ (approximate) Borůvka steps. An approximate Borůvka step on H w.r.t. $\hat{E} \subseteq E(H)$ can be executed in time $O((\ell+1)\cdot|\hat{E}|)$ time.*

We omit a proof of Lemma 3.4; it follows from Lemma 4.3 given in Section 4.

Definition 3.5. *(I) Let C be a subgraph of G. G/C represents the graph derived by contracting C and removing any self-loops. (II) C is* contractible *w.r.t. G if for any two edges e, f, each with a distinct endpoint in C, there exists a path $P \subseteq C$ connecting e to f such that either e or f is heavier than all edges in P.*

Lemma 3.6. *Let H be a graph derived from G after some number of (approximate) Borůvka steps and let U be those edges still untainted. Let C be the induced subgraph of G corresponding to some vertex in H. Then C is contractible w.r.t. $C \cup U$.*

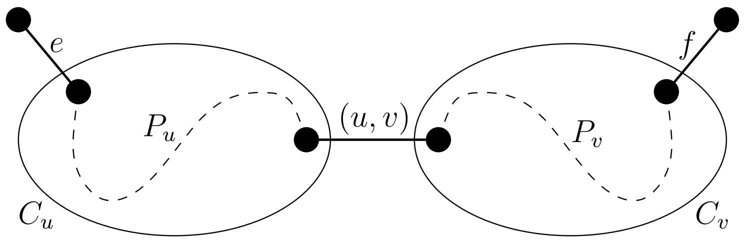

Fig. 3. The edges e, f, and (u,v) are untainted when (u,v) is selected by C_u.

Proof. Enumerate the edge contractions performed by a sequence of approximate Borůvka steps. Let (u,v) be one such edge, selected by u for contraction in some Borůvka step, and assume inductively the Lemma holds just before it is contracted. Let C_u and C_v be the subgraphs in G corresponding to u and v, respectively, and $C = C_u \cup C_v$. Let e and f be two arbitrary edges with one endpoint in C. If e and f remain untainted after the current Borůvka step (when (u,v) is selected) we show that there is a path P from e to f, all of whose edges are lighter than either e or f. The case when both e and f are incident to C_u or C_v is covered by the inductive assumption; assume that e is incident to C_u and f to C_v. Let P_u (resp., P_v) be the path from e to (u,v) (resp., f to (u,v)) guaranteed by the contractibility of C_u and C_v — see Figure 3 for a diagram. If (u,v) is to be contracted in the current Borůvka step it must be untainted. Since e remains untainted after the current Borůvka step, we know e is heavier than (u,v) and therefore heavier than every edge in P_u. Similarly, every edge in P_v is lighter than either (u,v) or f. Thus every edge in $P_u \cup \{(u,v)\} \cup P_v$ is lighter than either e or f, proving the contractibility of C w.r.t. $C \cup U$. $\square$

Corollary 3.7. *If, after some number of approximate Borůvka steps, an edge is both untainted and a self-loop, then it is not in the minimum spanning tree of the original graph.*

Corollary 3.7 follows from Lemma 3.6 and Definition 3.5.

3.2 The Algorithm

Our randomized MST algorithm consists of a succession of phases. At the beginning of each phase we apply a certain number of standard Borůvka steps, thereby identifying many MST edges and reducing the number of vertices in the graph. We then proceed to compute an approximately minimum spanning tree, using approximate Borůvka steps on sampled subgraphs. By approximately minimum we mean that the spanning tree T found is such that $T = MST(T \cup \{e\})$ for most edges $e \notin T$ still under consideration. The set of these edges can be identified in linear time with an MST verification algorithm [Komlós 1985; Dixon et al. 1992; King 1997; Buchsbaum et al. 1998] and discarded. We then prepare for the next phase by retracting all the approximate Borůvka steps performed in the phase. The pseudocode of our algorithm is given in Figure 4. The parameters λ_i and p_j determine, respectively, the number of standard Borůvka steps performed at the beginning of the ith phase and the sampling probability for the jth approximate Borůvka step performed in any phase.

```
Randomized-MST($G, \lambda, p$)

  $E_1 := E(G)$
  Execute Phases $1, 2, \cdots$ until the graph becomes trivial.
  The MST consists of all edges contracted in Line 1.
  Initially all edges are untainted.


  Phase $i$:

1.    Perform $\lambda_i - \lambda_{i-1}$ exact Borůvka steps
2.    For $j = \lambda_i + 1, \lambda_i + 2, \ldots$ do:
3.        Let $S_j$ be an edge-set sampled from $E_i$ with prob. $p_j$
4.        Perform an approximate Borůvka step w.r.t. $S_j$
5.    Until all edges have become tainted or self-loops
6.    $E_{i+1} := E_i - \{$untainted edges in $E_i\}$
7.    Retract all approximate Borůvka steps made in Phase $i$ (Line 4)
      (All edges tainted in those steps become untainted.)
```

Fig. 4. A randomized minimum spanning tree algorithm based on approximate Borůvka steps.

The correctness of the algorithm follows from a couple of observations. By Corollary 3.7 every edge discarded by the algorithm in Line 6 is not in the minimum spanning tree of G. Thus, the exact Borůvka steps performed in Line 1 contract only minimum spanning tree edges.

3.3 Analysis of the Algorithm

In our analysis we assume only a pairwise independent sampler and appeal to Lemma 2.3 to bound the expected running time.

A vertex is *isolated* if all its incident edges are tainted. In our analysis we assume the expected number of non-isolated vertices after j (approximate) Borůvka steps is at most $n_j = n/\gamma^j$, for some constant $\gamma > 1$. This holds with $\gamma = 2$ for normal (approximate) Borůvka steps. However, the *binary* Borůvka step (which will be introduced in Section 4.1) requires a smaller value for γ, hence we will use γ instead of 2 in the analysis here. The number of exact Borůvka steps performed at the beginning of each phase is given by the λ values, which depend on γ:

$$\lambda_0 = 0, \;\; \lambda_1 = C, \;\; \lambda_{i+1} = \gamma^{\lambda_i/4}$$

where C is a constant depending on γ. The sampling probability for the jth approximate Borůvka step is $p_j = \gamma^{-j/4}$.

The total cost of the algorithm is linear in the costs of the exact and approximate Borůvka steps performed in Lines 1 and 4, respectively. We argue in Section 4.2 that a linear-time MST verification algorithm can be used in Line 6 to detect untainted edges. Thus, in each phase the time taken at Line 6 is always dominated by that at Line 1. By Lemma 3.4 the expected running time of the algorithm is proportional to:

$$\sum_i \left[m_i(\lambda_i)^2 + \sum_{j=\lambda_i+1} j p_j m_i \right] \tag{1}$$

where $m_i = \mathbb{E}[|E_i|]$ is the expected number of edges at the beginning of Phase i. Before analyzing (1) we bound m_i. Assume inductively that $m_i \leq c_1 m \log \lambda_i/(\lambda_i)^3$ for some constant c_1.

$$\begin{aligned}
m_{i+1} &\leq \sum_{j=\lambda_i+1}^{\infty} c_2 n_j p_j^{-1} \log(m_i/n_j) \qquad \{c_2 \text{ from Lemma 3.3}\} \\
&\overset{\text{ind}}{\leq} c_2 n \sum_{j=\lambda_i+1}^{\infty} \gamma^{-3j/4} \left[\log(m/n) + \log(c_1 \gamma^j \log \lambda_i/(\lambda_i)^3)\right] \\
&\leq \frac{c_2 c_3 n \log(m/n)}{(\lambda_{i+1})^3} + \frac{c_2 c_3 n \log(c_1(\lambda_{i+1})^4 \log \lambda_i/(\lambda_i)^3)}{(\lambda_{i+1})^3} \\
&\leq c_1 m \log \lambda_{i+1}/(\lambda_{i+1})^3 \qquad \{c_1 > c_2 c_3(1 + 4\log c_1)\}
\end{aligned}$$

Plugging this bound on m_i into (1) we have:

$$[\text{Eq. } 1] \leq \sum_i \left[c_1 m \log \lambda_i/\lambda_i + c_1 m \log \lambda_i/(\lambda_i)^3 \sum_{j=\lambda_i+1}^{\infty} j\gamma^{-j/4} \right]$$

$$\leq \sum_i \left[c_1 m \log \lambda_i/\lambda_i + c_1 c_4 m \log \lambda_i/\lambda_{i+1}(\lambda_i)^2\right]$$
$$= O(m)$$

In the next Section we will prove that the number of random bits used by this algorithm is $O(\log^2 n \log^* n)$, which gives the following Theorem.

THEOREM 3.8. *The minimum spanning tree of a graph can be computed in expected linear time using $O(\log^2 n \log^* n)$ random bits.*

In the next Section we show that our algorithm can be parallelized to find MSTs and connected components using polylog random bits, polylog time, and expected linear work. A stronger version of Theorem 3.8 appears in Section 6.2, where the number of random bits is reduced to $o(\log^* n)$ without affecting the running time.

4. A PARALLEL MST ALGORITHM

The algorithm presented in Section 3 is based on Borůvka steps, a operation that is relatively easy to parallelize. We have, therefore, very few obstacles to designing a randomness efficient parallel MST algorithm.

We assume, without loss of generality, that edge weights are distinct and that the graph has degree at most 3. Any vertex with higher degree can be expanded into a chain of degree-3 vertices; edges on the chain are given very low weight.

4.1 Binary Borůvka Steps

To simplify the implementation of our algorithm on the EREW PRAM [Karp and Ramachandran 1990] it will be convenient if all Borůvka steps contract vertices in pairs — binary Borůvka steps. We give in Figure 5 a procedure Pair-Up that reduces the number of non-isolated vertices in the graph by a constant factor. The correctness of Pair-Up depends on a certain property of minimum spanning trees that says the endpoints of an edge can be relocated without affecting the minimum spanning tree. More specifically:

LEMMA 4.1. *(Endpoint relocation property) Suppose that $w(x, y) > w(y, z)$, where $(x, y), (y, z)$ are edges in a weighted graph G. Let G' be derived from G by reassigning the endpoints of (x, y) to (x, z); call this edge e, whether it appears in G or G'. Then $e \in MST(G)$ if and only if $e \in MST(G')$.*

PROOF. Suppose that e is the heaviest edge on a cycle Q in G, that is, $e \notin MST(G)$. Then in G', e is the heaviest edge on the unique cycle in $Q \cup \{(y, z)\}$, since $w(e) > w(y, z)$. Note that the edge (y, z) may or may not be in the cycle in $Q \cup \{(y, z)\}$. The reverse direction is proved in an identical manner, switching the roles of G and G'. □

The edges selected in any (approximate) Borůvka step induce a forest F of at least $n'/2$ trees, where n' is the number of non-isolated vertices. If we consider the edges to be directed, where $(u, v) \in F$ represents the edge chosen by u, then each tree in F has two roots, corresponding to an edge that is chosen by both endpoints. We remove one of these directed edges and consider F to be a forest of rooted trees. Pair-Up relocates the endpoints of certain edges of F consistent with Lemma 4.1 then finds a sizable set $F' \subseteq F$ of nonadjacent edges. The interesting features of Pair-Up are given in Lemma 4.2; the particulars of the procedure are given in Figure 5.

LEMMA 4.2. *Let F be a set of rooted trees and $n = |V(F)|$. Pair-Up(F) returns a set of independent edges F', such that $\mathbb{E}[|F'|] \geq n/4$, even if the random bits required of the algorithm are only pairwise independent. Moreover, all edge relocations made by Pair-Up are minimum spanning tree preserving [see Lemma 4.1].*

PROOF. By Lemma 4.1 all edge relocations do not affect which edges are in the MST. Each relocated edge in F' removes from further consideration at most 4 vertices, i.e., at Line 8 of Pair-Up $|V(F - V(F'))| \geq n - 4|F'|$. At Line 8 $F - V(F')$ consists of directed paths. Pair-Up sets the head and tail of each path to be heads and tails, resp., and flips a fair coin for the other vertices. Therefore, in a path of length k in $F - V(F')$, we expect that at least $\frac{1}{4}(k-2) + \frac{1}{2}2 > \frac{1}{4}(k+1)$ of its edges will be added to F' in Line 13, even if the coin flips of Line 9 are only pairwise independent. Summing over all paths in $F - V(F')$ we see that the size of F' returned at Line 14 is at least $n/4$. □

By Lemma 4.2 and linearity of expectation, the expected number of non-isolated vertices after j (approximate) Borůvka steps is no more than $n(\frac{3}{4})^j$. Therefore the analysis of our MST algorithm in Section 3 is correct for $\gamma = 4/3$.

```
Pair-Up($F$)   :   $F$ is a forest of rooted trees
1.    $F' := \emptyset$
2.    For each vertex $v$:
3.        Let $v_1, v_2, \ldots, v_k$ be the children of $v$ in $F$
4.        For $i := 1..\lfloor \frac{k}{2} \rfloor$
5.            Assume w.l.o.g. that $w(v_{2i}, v) > w(v_{2i-1}, v)$
6.            Relocate $(v_{2i}, v)$ to $(v_{2i-1}, v_{2i})$
7.            $F' := F' \cup \{(v_{2i-1}, v_{2i})\}$
8.    ($F - V(F')$ now consists of a collection of directed paths)
9.    Each vertex with in- and out-degree 1 in $F - V(F')$ flips a fair coin
10.   Vertices with out-degree (resp., in-degree) 1 in $F - V(F')$ pick heads (resp., tails)
11.   For each edge $(u, v) \in F - V(F')$
12.       If $u$ picked heads and $v$ picked tails,
13.           $F' := F' \cup \{(u, v)\}$
14.   Return($F'$)
```

Fig. 5. Pair-Up(F) returns a set of independent edges from F, after performing some MST-respecting edge relocations.

4.2 Contraction Trees

We model the Borůvka steps performed by our algorithm with a *contraction tree.* This structure was introduced by King [1997] as part of her MST verification algorithm. After b (approximate) Borůvka steps the contraction tree consists of b levels. The vertices at height i correspond to the graph vertices after i steps and the parent of a vertex v in the contraction tree, denoted $p(v)$, is the vertex representing v after performing one Borůvka step. (We use the same notation for both graph vertices and their equivalent contraction tree vertices.) The weight of an edge $(v, p(v))$ is exactly the weight of the edge chosen by v in the appropriate Borůvka step.

If we perform only binary Borůvka steps (using the Pair-Up procedure from Section 4.1) then we can make some additional claims about the structure of the contraction tree. Obviously every vertex in the tree has at most two children. Some vertices have only one child though they are not necessarily isolated. They may simply have been unpaired by Pair-Up. By Lemma 4.2 the expected number of non-isolated vertices in the tree is $O(n)$; however the number of isolated vertices could be $\Omega(bn)$ after b Borůvka steps. For this reason we do not explicitly represent a vertex $p(v)$ if v is isolated.

We use the contraction tree to perform approximate Borůvka steps and to detect untainted edges after each phase of the algorithm. Lemmas 4.3 and 4.4 give, respectively, bounds on the time for one (approximate) Borůvka step and a characterization of the untainted edges w.r.t. the contraction tree.

LEMMA 4.3. *Suppose that G' is a graph derived from b (approximate) binary Borůvka steps. Given a contraction-tree representing those steps, the next (approximate) binary Borůvka step can be executed in $O(b+1)$ time and $O((b+1)m)$ work on an EREW PRAM. Here m is the number of edges participating in the step.*

PROOF. We assume that the participating edges have been fairly allocated to processors. Our description proceeds as if two virtual processors were assigned to each edge. An edge (u, v) (where u and v are nodes in the original graph) is eligible if $\text{root}(u) \neq \text{root}(v)$, where $\text{root}(u)$ is the ancestor of u at height b in the contraction tree, and if $w(u, v)$ is strictly greater than the weights along the paths from u to $\text{root}(u)$ and from v to $\text{root}(v)$. The two processors assigned to (u, v) attempt to crawl up the contraction tree starting from u and v, recording the heaviest edge along the way. If several processors meet at a vertex x only one continues; after it discovers the heaviest edge on the path from x to $\text{root}(x)$ it relates this information to the stalled processors. We find the lightest eligible edge incident on each vertex in a similar manner. Each of the $O(b+1)$ steps of these procedures can be implemented in constant time on an EREW PRAM because the contention at each node of the contraction tree is constant. At most three processors access a leaf in one step (the graph is degree at most three) and at most two processors access any internal node. We omit a description of the parallelized Pair-Up procedure. It runs in $O(b + 1)$ steps using the same tree-crawling technique. □

LEMMA 4.4. *Let G be a graph and let F be the contraction tree resulting from some number of (approximate) Borůvka steps, where the leaves of F are identified with the vertices of G. An edge (u, v) is untainted either if (a) u and v are connected in F and (u, v) is the unique heaviest edge on the cycle in $F \cup \{(u, v)\}$, or (b) (u, v) is the unique heaviest edge on the path from root(u) to root(v) in $F \cup \{(u, v)\}$ and both root(u) and root(v) are non-isolated.*

PROOF. Follows directly from Definition 3.1. □

One can now see how an MST verification algorithm [Komlós 1985; Dixon et al. 1992; King 1997; Buchsbaum et al. 1998] could be used to detect untainted edges in Line 6 of our algorithm presented in Section 3. Since all roots of F are isolated at

Line 6, the untainted edges are precisely those satisfying condition (a) in Lemma 4.4. Condition (a), in turn, is equivalent to the condition that (u, v) is not in any minimum spanning forest of $F \cup \{(u, v)\}$.

4.3 Analysis

THEOREM 4.5. *The minimum spanning tree of a graph can be computed on an EREW PRAM in expected linear work and $O(\log^2 n \log^* n)$ time while using only $O(\log^2 n \log^* n)$ random bits.*

COROLLARY 4.6. *The connected components of a graph can be computed on an EREW PRAM with expected linear work, $O(\log^2 n \log^* n)$ time, and $O(\log^2 n \log^* n)$ random bits.*

PROOF. We will show that there are at most $2 \log^* n$ phases of the algorithm presented in Sections 3 and 4, and that each phase requires both $O(\log^2 n)$ random bits and time.

The expected number of phases is $\min\{i : \lambda_i \geq \log_\gamma n\}$. Recall that the λ values are $\lambda_1 = C$, $\lambda_{i+1} = 2^{\lambda_i/c}$, where c is a constant and C depends on c. For a $C \geq c^2$ we prove inductively that $\lambda_{i+2} \geq 2^{\lambda_i}$, which implies $\min\{i : \lambda_i \geq \log_\gamma n\} \leq 2 \log^* n$. Assume that $\lambda_i \geq C$. Then $\lambda_{i+2} \geq 2^{2^{\lambda_i/c}/c} \geq 2^{2^{\lambda_i^{1/2} - \log \lambda_i/2}}$. Thus, for $C \geq c^2$ such that $C^{1/2} \geq (3/2) \log C$, it follows that $\lambda_{i+2} \geq 2^{\lambda_i}$.

The expected number of (approximate) Borůvka steps in each phase is $\log_\gamma n$. By Lemma 4.3 each step takes $O(\log n)$ time and requires $2(\log n + \log m)$ random bits: $2 \log m$ for sampling the edges (if it is an approximate Borůvka step) and $2 \log n$ to perform the random mating in Pair-Up. The number of random bits required to find all the primes used in our sampling procedure is $O(\log^2 n)$ — see Lemma 2.8. The time required to perform MST verification is $O(\log n)$ per phase [King et al. 1997] and is not significant overall. The expected work of our algorithm is linear; see Section 3 for the analysis. □

5. DISCUSSION

In this section we will attempt to unify the language and concepts introduced in the recent non-greedy minimum spanning tree algorithms. We argue that our algorithm in Section 3 serves as a bridge between the randomized and deterministic approaches because it combines ideas from both in a very natural way.

Non-Greedy Algorithms.. The recent MST algorithms [Karger et al. 1995; Chazelle 2000a; Pettie and Ramachandran 2002b], are all said to be non-greedy, but they are not really alike. Karger et al.'s algorithm [1995] uses random sampling and MST verification, while the deterministic MST algorithms of Chazelle [2000a] and Pettie-Ramachandran [2002b] are based on the Soft Heap [Chazelle 2000b]. The reduced-randomness MST algorithm we presented in Section 3 reveals some of the commonality between the two approaches. Like the Karger et al. it uses random sampling and MST verification. However one should imagine our approximate Borůvka steps as implementing a certain kind of soft heap, in particular, one where the tradeoff between *running time* and *error rate* changes depending on the sampling probability, which can be set arbitrarily. We elaborate more on this below.

Abstract Soft Heaps.. The central concept of non-greedy MST algorithms can be viewed as that of an *abstract soft heap*, not to be confused with the concrete Soft Heap data structure invented by Chazelle [2000b]. We consider an abstract soft heap to be any data structure supporting the usual heap operations (make-empty-heap, insert, delete, findmin, meld) but without the obligation of answering all findmin queries correctly. Any heap element that *bears witness* to the fact that a previous findmin was answered incorrectly is called *corrupted.* (Chazelle's [2000b] definition is different but equivalent. His Soft Heap *always* answers findmin queries correctly. However, it is free to increase the key of any element in order to ensure correctness. The corrupted elements are those with increased keys.) We use the lower case *soft heap* to denote any implementation of an abstract soft heap.

Chazelle's Soft Heap is deterministic and provides an accuracy-time tradeoff that meets an information theoretic lower bound. There are, however, many other conceivable ways to design abstract soft heaps. For instance, the approximate Borůvka steps from our algorithm implement a coordinated system of soft heaps, with one soft heap associated with each vertex of the contracted graph. Tainted edges would then correspond to corrupted elements in some abstract soft heap. An important subtlety here is that since each element appears in two soft heaps, corresponding to the associated edge's endpoints, if it is corrupted in one soft heap it also becomes corrupted in the other. (The MST algorithm of Karger et al. [1995] can also be reinterpreted as a soft heap-based algorithm, with *F-lightness* taking the role of taintedness/corruptedness. However doing so is very unnatural. As the algorithm traverses its recursion tree we would need to imagine the "clock" of the system of abstract soft heaps wiggling forward and backward in time.)

The accuracy-time tradeoff of a random sampling-based soft heap can be incomparable to the tradeoff of Chazelle's Soft Heap. The Soft Heap performs all operations in $O(\log \epsilon^{-1})$ time while guaranteeing that an ϵ fraction of the heap is corrupted. By comparison the tth approximate Borůvka step performs $n' = O(n/2^t)$ deletions on n' abstract soft heaps (and between $n'/2$ and $n' - 1$ melds) while corrupting an expected kn' elements (edges), at a total cost of $O(tm/k)$. Approximate Borůvka steps (viewed as a soft heap) could be construed as less efficient than the Soft Heap. However an approximate Borůvka step possesses several useful properties beyond the reach of Soft Heaps. First, the error rate (i.e., sampling probability) can be chosen at will. Second, approximate Borůvka steps are guaranteed to return only uncorrupted elements. Finally, approximate Borůvka steps corrupt edges in different heaps without difficulty.

Abstract Soft Heap Verification.. Approximate Borůvka steps have many strengths but one glaring weakness: they cannot easily distinguish between the corrupted and uncorrupted elements, a property that the deterministic Soft Heap-based algorithms [Chazelle 2000a; Pettie and Ramachandran 2002b] depend on heavily. In other words, a sampling-based soft heap cannot stand alone. It must be used in conjunction with a *soft heap verification* algorithm that performs a post mortem analysis of the transcript of heap operations in order to decide which elements were ever corrupted. In Section 4.2 we argued that an MST verification algorithm could be used to decide which edges were tainted/corrupted by the approximate Borůvka steps in one phase of our algorithm. In fact, the MST verification problem is *entirely* *equivalent* to the soft heap verification problem. Below we give one direction of the reduction, from soft heap verification to MST verification. The input is a transcript of heap operations that includes the answers produced by the findmin operation. We model the history of a system of soft heaps by a forest of rooted trees. A root node corresponds to an extant soft heap and internal nodes corresponds to extinct ones. A make-empty-heap operation creates a new root node representing the new soft heap. A meld(H, H') operation creates a new (root) node that is made the parent of the nodes representing H and H', and a findmin(H) operation makes the node representing H the child of a new parent node. Edges introduced by melds have weight $-\infty$, while the edge of a findmin has weight equal to the answer of that findmin. This tree T represents the purported minimum spanning tree in the corresponding instance of MST verification. Consider a soft heap element e that was inserted into a soft heap H and deleted from the soft heap H', which was derived from H by a sequence of inserts, deletes, and melds. Viewing H and H' as vertices in T, the element e corresponds to a non-tree edge $e = (H, H')$ with weight equal to e's original key. It may be the case that e is assigned the same weight as an edge $e' \in T$. We break such a tie by letting e (the non-tree edge) have the heavier weight. For an element e that was inserted into soft heap H and not deleted, we place a non-tree edge from the vertex representing H to the root of its tree, again with weight equal to the original key of e.

Notice that for any non-tree edge e, the unique cycle in $T \cup \{e\}$ contains precisely those soft heaps (vertices) that contained the element e. Clearly, if $T = MST(T \cup \{e\})$ then e cannot bear witness to any findmin queries that were answered incorrectly, i.e., it remains uncorrupted. On the other hand, if $T \neq MST(T \cup \{e\})$ then for some vertex H on the cycle in $T \cup \{e\}$ the query findmin(H) must have returned an element heavier than e, corrupting e. This concludes the reduction from soft heap verification to MST verification. The reverse reduction follows similar lines. We would orient the purported MST and replace each non-tree edge (u, v) by two edges connecting u and v to their least common ancestor. Converting such an instance of MST verification into a transcript of abstract soft heap operations is then straightforward.[5]

[5] *Begin Digression.* The relationship between MST-related problems and heap-related problems is stronger than generally acknowledged, the MST verification/soft heap verification connection being just one. We demonstrate here one more. Consider the Offline Heap problem, where we are given a sequence of heap operations (make-empty-heap, insert, delete, find-min, meld). The arguments to the operations are given; the problem is to compute the correct answer to each find-min query. We argue that the Offline Heap problem is equivalent to the MST Sensitivity Analysis [Pettie 2005] problem: given the MST of a graph, decide how much each MST edge can be increased without affecting the identity of the MST. We model the Offline Heap instance as a rooted tree using the exact same transformation given for abstract soft heap verification. However in this case it is the non-tree edges that are weighted; tree edges have weight $-\infty$. One can verify that an answer to the MST Sensitivity Analysis problem on this graph gives an answer to the Offline Heap instance. In particular, if $e \in MST$ is an edge representing a find-min, the maximum possible weight of that edge is precisely the key-value returned by the corresponding find-min. The reverse direction, reducing MST Sensitivity Analysis to Offline Heap, is equally straightforward. One application of an Offline Heap algorithm is monitoring a purportedly correct heap data structure. The Offline Heap algorithm would periodically check a transcript of the input & output of the heap and detect and correct any errors. *End of Digression.*

Contractibility. The discussion above does not answer the question of *why* abstract soft heaps are such a natural means to obtain fast MST algorithms. Consider the definition of contractibility (Definition 3.5) from Section 3. It is not difficult to show that at any stage in the classical minimum spanning tree algorithms of Kruskal [1956], Dijkstra-Jarník-Prim [Dijkstra 1959; Jarník 1930; Prim 1957], and Borůvka [1926], every maximal tree of MST edges already found represents a contractible component. These greedy algorithms are generally implemented using (non-soft) heaps. It is an interesting exercise to show that by substituting a soft heap for a standard one, the components computed by the standard algorithms are contractible *with respect to uncorrupted edges.* This statement applies in a straightforward manner to Kruskal's algorithm and the Dijkstra-Jarník-Prim algorithm; see [Pettie and Ramachandran 2002b]. However there are some complications in Borůvka's algorithm because elements (edges) normally appear in two soft heaps simultaneously. To ensure contractibility these two copies must be corrupted together, i.e., have their weights increased by the same amount. This slight twist helps explain why Chazelle's Soft Heap [2000b] cannot be merely plugged into Borůvka's algorithm, but why, say, it can be used with the algorithms of Kruskal or Dijkstra-Jarník-Prim, which employ only one heap.

Contractible components are useful because they represent sub-problems that can be solved recursively (as in [Chazelle 2000a]) or with minimum-depth decision trees (as in [Pettie and Ramachandran 2002b]) or with another MST algorithm (as in Section 6.2 of this paper). Furthermore, if multiple uncorrupted edges join two contractible components, all but at most one must be non-MST edges. In [Karger et al. 1995] and Section 3 of this paper, these non-MST edges can be identified and discarded using an MST verification/soft heap verification algorithm. One way or another, the non-greedy algorithms use contractible components to reduce the size of the problem geometrically, in linear or near-linear time.

The observations made above illustrate some of the common structure of all non-greedy MST algorithms[6] and explain, to a limited extent, why they are the way they are. To summarize, the existing algorithms use Soft Heaps as in [Chazelle 2000a; Pettie and Ramachandran 2002b] or can be construed as using an abstract soft heap as in [Karger et al. 1995] and Section 3 of this paper. Abstract soft heaps allow the algorithm to find contractible components w.r.t. the set of edges uncorrupted/untainted by the heap, and these components let the algorithm reduce the problem size by a constant factor, either by simply filtering out uncorrupted non-MST edges between contractible components (as in [Karger et al. 1995] and in Section 3 of this paper) or by solving the subproblems induced by those components as in [Chazelle 2000a; Pettie and Ramachandran 2002b].

6. FURTHER RESULTS

In this section we present optimal randomized algorithms using reduced number of random bits for other problems concerning total orders. See Table I for a summary of our results. In Section 6.1 we demonstrate how the techniques developed in

[6]We should point out that the optimal MST algorithm of Pettie and Ramachandran [2002b] uses minimum-depth MST decision trees, for which we cannot claim any resemblance to existing non-greedy MST algorithms.

Problem	**Deterministic Bound**	**Probabilistic Bound**	
		Best previous	**This paper**
Minimum spanning trees	$O(m\alpha(m,n))$ $O(\text{Optimal}(m,n))$	$O(m)$ $O(m)$ rand. bits	$O(m)$ $o(\log^* n)$ random bits
Parallel $\mathcal{NC}$ graph conn. (work)	$O(m\alpha(m,n))$	$O(m)$ $O(m)$ rand. bits	$O(m)$ $o(\log^{2+\epsilon} n)$ rand. bits
Parallel $\mathcal{NC}$ MST (work)	$O(m\log^{(3)} n)$	$O(m)$ $O(m)$ rand. bits	$O(m)$ $o(\log^{2+\epsilon} n)$ rand. bits
MST/SSSP sensitivity analysis	$O(m\log\alpha(m,n))$ $O(\text{Optimal}(m,n))$	$O(m)$ $O(m)$ rand. bits	$O(m)$ $o(\log^* n)$ rand. bits
Local sorting (comparisons)	$O(\min\{m, n\log n\})$ (trivial)	$O(n\log\frac{m+n}{n})$ $\omega(n\log\frac{m+n}{n})$ r.b.	$O(n\log\frac{m+n}{n})$ $o(\log^{1+\epsilon} n)$ rand. bits
Set maxima (comparisons)	$O(\min\{n+m2^m, n\log n\})$ (trivial)	$O(n\log\frac{m+n}{n})$ $\omega(n\log\frac{m+n}{n})$ r.b.	$O(n\log\frac{m+n}{n})$ $o(\log^{1+\epsilon} n)$ rand. bits

Notes: Optimal(m,n) is the decision-tree complexity of the respective problem and ϵ is an arbitrarily small constant. With the exception of Set Maxima, $m = |E(G)|$ is the number of edges and $n = |V(G)|$ is the number of vertices. For Set Maxima m is the number of sets and n the number of elements. See [Chazelle 2000a; Pettie and Ramachandran 2002b; Karger et al. 1995] for sequential minimum spanning tree algorithms, [Cole and Vishkin 1991; Gazit 1991; Halperin and Zwick 2001] for parallel connected components algorithms, [Cole and Vishkin 1986; Pettie and Ramachandran 2002c; Cole et al. 1996] for parallel minimum spanning tree algorithms, [Pettie 2005; Dixon et al. 1992] for minimum spanning tree and shortest path sensitivity analysis algorithms, and [Goddard et al. 1993] for set maxima and local sorting.

Table I. A summary of our results

Sections 2 and 3 can be used to adapt the *local sorting* and *set maxima* algorithms in [Goddard et al. 1993] to use $O((\log n)^{1+o(1)})$ random bits. In Section 6.2 we describe how a simple technique — reusing random bits — can give optimal randomized algorithms that use a *nearly* constant number of random bits. This technique is applied to the minimum spanning tree and MST sensitivity analysis problems. In Section 6.3 we show that selection can be performed (non-uniformly) with $1.5n + o(n)$ comparisons using only $O(\log\log n)$ random bits.

6.1 Local Sorting and Set Maxima

In the *set maxima* problem we are given a set system $(\chi, \mathcal{S})$ where χ is a set of n totally ordered elements and $\mathcal{S} = \{S_1, \ldots, S_m\}$ is a collection of subsets of χ. The problem is to determine $\{\max S_i\}_{1\le i\le m}$. This intriguing problem seems to have been introduced by Graham, Yao, and Yao [1980] who noted the trivial $O(n\log n)$ time solution. A bound of Fredman appears in the same paper showing that an instance of set maxima can have no more than $\binom{m+n-1}{n-1}$ distinct solutions, implying any information-theoretic lower bound must be $O(n\log(1+\frac{m+n}{n}))$. This bound was shown to be tight in [Goddard et al. 1993] for randomized algorithms. Liberatore [1998] has shown the set maxima problem to be precisely the problem of verifying the optimal base of an arbitrary matroid, and Karger [1993] has demonstrated the usefulness of set maxima in actually finding an optimal base. Many more concrete problems are instances of set maxima (or are reducible to it), such as verifying a given partial order [Kenyon-Mathieu and King 1989], sensitivity analysis and verification of minimum spanning trees and shortest path trees [Tarjan 1982; Komlós 1985; Pettie 2005], and orienting the edges of an undirected, node-weighted graph from the lesser to greater endpoint. This last problem is just set maxima when all sets have two elements; it was dubbed *local sorting* by Goddard et al. [1993].

Besides the trivial $O(n\log n)$ set maxima algorithm there are really only three results to speak of. Graham, Yao, and Yao [1980] observed that set maxima could be solved with $O(n+m2^m)$ comparisons by dividing the elements into 2^m groups based on their set memberships. Bar-noy et al. [1992] gave a deterministic algorithm that uses $O(n)$ expected comparisons when the $m=n$ sets are chosen *randomly.* Goddard et al. [1993] gave an elegant randomized algorithm for set maxima that makes an optimal $O(n\log(1+\frac{m+n}{n}))$ expected comparisons.

We first make an observation on the sampling requirements of the algorithms from [Goddard et al. 1993].

Observation 6.1. *The optimal local sorting and set maxima algorithms of [Goddard et al. 1993] use only properties of random samples that are guaranteed by Lemma 2.3 or Lemma 2.4 using a 4-wise independent sampler.*

Thus the number of random bits required by these algorithms is $4\log n$ times the number of sampling events, $\sqrt{\log n}$ in the case of local sorting and $2^{O(\log_t n)}$ in the case of set maxima, for any constant t. Note that this does *not* lead to a polylogarithmic number of random bits for set maxima.

In this section we modify the local sorting algorithm in order to accommodate a pairwise independent sampler, then show how the set maxima algorithm can be made to use a polylog number of random bits using a 4-wise independent sampler. (The time required to find the primes used in our sampler from Section 2.2 is small, and moreover, unimportant. In this subsection we only consider comparison complexity.)

Theorem 6.2. *A graph on n vertices and $m\geq n$ edges can be locally sorted with a pairwise independent sampler using an expected $O(n\log\frac{m+n}{n})$ comparisons and $(2+\epsilon)\log n\log\log\log n$ random bits, for any $\epsilon>0$.*

Proof. We will not describe the algorithm of [Goddard et al. 1993] in detail but go straight to the analysis. The algorithm consists of a number of phases, each having one sampling event. We have total freedom in choosing the sampling probabilities. Let p_i be the sampling probability for the i^{th} phase, where p_0 is fixed at 1, and let $D=\frac{m+n}{n}$. Using a *pairwise* independent sampler, the expected number of comparisons in the i^{th} phase is then on the order of

$$n \log(D) \left[2^{2i} p_{i-1} + 2^{2i} p_{i-1}^2 / p_i\right]$$

Notice that this quantity is larger than in [Goddard et al. 1993] due to the pairwise independent sampler. We let $p_i = 1/2^{2^{i/c}}$ for $i > 0$ and $c = 1 + \frac{\epsilon}{2}$. Summing over all phases the total number of comparisons is

$$\begin{aligned} n \log(D) \sum_{i>0} 2^{2i} \left(\frac{1}{2^{2^{(i-1)/c}}} + \frac{2^{2^{i/c}}}{2^{2^{(i+c-1)/c}}} \right) &= O(n \log D) \cdot \sum_{i>0} \frac{2^{2i} \cdot 2^{2^{i/c}}}{2^{2^{(i+c-1)/c}}} \\ &\quad \{2^{(i-1)/c} > 3i \text{ for } i \text{ sufficiently large}\} \\ &= O(n \log D) \end{aligned}$$

The last line follows from the fact that $(2^{2i} \cdot 2^{2^{i/c}})/2^{2^{(i+c-1)/c}} < 2^{-i}$ for i sufficiently large (recall $c > 1$).

As observed by Goddard et al. [1993], if $p_i \leq 1/\log n$ then i phases are sufficient. Therefore our algorithm has $c \log\log\log n$ sampling events, each requiring $2 \log n$ random bits to generate the pairwise independent random variables. The hidden constants in this scheme are a bit large. Smaller constants can be obtained by setting $p_i = 1/2^{3i}$, leading to an algorithm using $O(\log n \log\log n)$ random bits. □

A generalization of set maxima is t-maxima [Goddard et al. 1993], where we are asked to identify and sort the largest t elements in each set.

Theorem 6.3. *The t-maxima problem can be solved optimally, with expected $O(n \log(t(m+n)/n))$ comparisons, while using $O\left(\log n \log\log\log n 2^{\Theta(1+\log^* n - \log^* t)}\right) = o(\log^{1+\epsilon} n)$ random bits. Here n and m are the number of elements and sets, respectively.*

Proof. As above, we will skip a description of the algorithm and jump straight to the analysis. Each recursive invocation of the t-maxima algorithm solves one local sorting problem and makes three recursive calls. The recursion depth of the [Goddard et al. 1993] algorithm can be shown to be $\log_t n$, hence the $3^{\log_t n}$ upper bound on the number of calls to the local sorting algorithm. We adapt their algorithm so the recursion depth is $O(\log^* n)$, thus reducing the number of random bits required from n^ϵ (for any $\epsilon > 0$) to $\log^{1+\epsilon} n$.

Let $T(t, n, m)$ be the expected number of comparisons performed by the t-maxima algorithm. It is shown in [Goddard et al. 1993] that

$$T(t, n, m) \leq c_1 n \log(\tfrac{mt^2}{np^2}) + T(t, pn, m) + T(t_0, pn, n) + T(t, \tfrac{mt}{pt_0}, m)$$

where p is a sampling probability and t_0 can be chosen arbitrarily. It is not difficult to chose p and t_0 such that $T(t, n, m) = O(n \log \frac{mt}{n})$ We reduce the recursion depth as follows. In addition to parameters t, n, and m, we let a recursive call to t-maxima set p and t_0 according to another parameter i. If $i = 0$ we let $p_0 = 1/t$ (whatever t is at the time), otherwise we let $p_i = 2^{-1/4p_{i-1}}$, where p_{i-1} was the sampling probability used by the "parent" recursive call. We set $t_0 = mt/np_ip_{i+1}$. We have then

$$T(i,t,n,m) \leq c_1 n \log(\tfrac{mt^2}{np_i^2}) + T(i+1,t,p_in,m) + T(0,t_0,p_in,n) + T(i+1,t,p_{i+1}n,m)$$

In other words we increment i for the first and third recursive calls, but reset it to zero for the second recursive call. We maintain the invariants that t is at least 24 and non-decreasing in recursive calls, $p_i \leq 1/t$, and that $m/n \geq 2$.

Assuming inductively that $T(i,t,n,m) \leq cn\log(\frac{m}{np_i})$ for some c,

$$\begin{aligned}
T(i,t,n,m) &\leq c_1 n \log\left(\frac{mt^2}{np_i^2}\right) + cp_i n \log\left(\frac{m}{np_ip_{i+1}}\right) + cp_i n \log\left(\frac{mt}{n(p_i)^2 p_{i+1}}\right) \\
&\quad + cp_{i+1} n \log\left(\frac{m}{n(p_{i+1})^2}\right) \\
&\leq 4c_1 n \log\left(\frac{m}{np_i}\right) + \tfrac{c}{10} n \log\left(\frac{m}{np_i}\right) + \tfrac{c}{5} n \log\left(\frac{m}{np_i}\right) + \tfrac{c}{4} n \log\left(\frac{m}{np_i}\right) \\
&= cn \log\left(\frac{m}{np_i}\right) \qquad \{\text{for } c \geq 9c_1, \text{ completing the induction.}\}
\end{aligned}$$

The coefficients $\frac{1}{10}$, $\frac{1}{5}$, and $\frac{1}{4}$ are easily derived using the lower bounds $p_i^{-1}, t \geq 24$ and $p_{i+1}^{-1} \geq 64$. If the algorithm is started with $i = 0$ then the expected total number of comparisons is $O(n \log\left(\frac{mt}{n}\right))$, which is optimal [Goddard et al. 1993]. It is easy to see that if the sampling probability is p_i in one call to t-maxima, then it is no more than p_{i+1} in each recursive call. This is trivial for the first and third recursive calls as i is incremented. For the second, where i is set to zero, the new sampling probability is $1/t_0 = np_ip_{i+1}/mt < p_{i+1}$. Since the algorithm bottoms out when the sampling probability is less than $1/n$ we conclude that the recursion depth is $O(1 + \log^* n - \log^* t)$ and hence there are $3^{O(1+\log^* n - \log^* t)}$ recursive calls. □

Corollary 6.4. *Set Maxima can be solved optimally, with expected $O(n \log \frac{m+n}{n})$ comparisons, while using $\log n \log\log\log n 2^{O(\log^* n)}$ random bits.*

6.2 Reusing Random Bits

In this section we present a simple but general method for drastically reducing the number of random bits required to solve certain problems in optimal expected time. The problems we give as examples are the minimum spanning tree problem and the MST sensitivity analysis problem, though the method works for any problem possessing Properties 6.5(1)–(3).

Property 6.5. *Let $\mathcal{P}$ be an arbitrary problem whose input sizes are measured by pairs (m,n). We assume that any instance of size (m,n) is no harder than one of size $(m' \geq m, n' \geq n)$. Let $f(m,n)$ be a function such that $f(kn,n)$ is non-decreasing in n and non-increasing in k and let $g_i(n) = \min\{k : f(kn,n) \leq i\}$. The following properties hold for problem $\mathcal{P}$:*

(1) *There exists a deterministic algorithm $\mathcal{A}_{det}$ solving $\mathcal{P}$ in $O(m \cdot f(m,n) + n)$ time.*

(2) *There exists a randomized algorithm $\mathcal{A}_{rand}$ solving problem $\mathcal{P}$ in $O(m+n)$ expected time using $O(m)$ random bits.*

(3) *There exists a deterministic algorithm $\mathcal{A}_{decomp}$ that reduces, in $O(m+n)$ time, a size (m,n) instance of $\mathcal{P}$ to a set of instances $\{(m_j,n_j)\}_{0\le j\le \ell}$ such that for any q, $m_0 \le m$, $\sum_{j=1}^{\ell} m_j \le m$, $n_0 \le n/q$, and for $j \ge 1$, $m_j \ge n_j = \Theta(q)$.*

Theorem 6.6. *If a problem $\mathcal{P}$ satisfies Property 6.5 then there exists a randomized algorithm $\mathcal{A}_{hybrid}$ running in $O(im+n)$ expected time using $O(g_i(n) \cdot g_i^{(2)}(n) \log g_i^{(2)}(n))$ random bits.*

Proof. First execute $\mathcal{A}_{\text{decomp}}$ with $q = g_i(n)$. This takes $O(m+n)$ time. Execute $\mathcal{A}_{\text{det}}$ on the problem of size (m_0,n_0) in time $O(m_0 \cdot f(m_0,n_0) + n_0) = O(mf(m,n/q)+n/q) = O(im+n/q)$. For each subproblem of size (m_j,n_j), if $m_j > n_j g_i(n_j) = \Theta(g_i(n) \cdot g_i(g_i(n)))$, solve it with $\mathcal{A}_{\text{det}}$ in $O(im_j)$ time. Otherwise solve it as follows. For at most $O(\log g_i^{(2)}(n))$ iterations, run $\mathcal{A}_{\text{rand}}$ for $O(m_j)$ steps (using $O(m_j)$ random bits) until it returns an answer. If there is no answer after $O(\log g_i^{(2)}(n))$ iterations revert to $\mathcal{A}_{\text{det}}$, which takes $O(m_j \cdot f(m_j,n_j)+n_j) = O(g_i(n) \cdot g_i^{(2)}(n) \cdot f(g_i(n) g_i^{(2)}(n),\, g_i(n))) = O(i \cdot g_i(n) \cdot g_i^{(2)}(n))$ time. The expected running time for the problem of size (m_j,n_j) is therefore $O(m_j + ig_i(n)) = O(im_j)$. By reusing the random bits for each subproblem (m_j,n_j), $j>0$, the total number of random bits used is $O(g_i(n) \cdot g_i^{(2)}(n) \log g_i^{(2)}(n))$ and, by linearity of expectation, the expected running time is $O(m+n)+O(im+n/q)+\sum_{j>0} O(im_j) = O(im)$. □

Define $\lambda_i(n)$ to be the ith row inverse of Ackermann's function. The functions $\{\lambda_i(n)\}_i$ can be defined apart from Ackermann's function as $\lambda_1(n) = \log n$ and $\lambda_{i+1}(n) = \min\{j \,:\, \lambda_i^{(j)}(n) \le 1\}$.

Corollary 6.7. *The minimum spanning tree problem can be solved in $O(im)$ expected time using $\lambda_i(n)$ random bits.*

Proof. The MST problem satisfies Property 6.5 with $f(m,n) = \alpha(m,n)$ [Chazelle 2000a], where α is the inverse-Ackermann function, and $g_{i+1}(n) = \lambda_{i+1}(n)$. We use the Pettie-Ramachandran [2002b] algorithm for $\mathcal{A}_{\text{det}}$. For $\mathcal{A}_{\text{rand}}$ we can use either the Karger-Klein-Tarjan [1995] algorithm or the one presented in Section 3. For a description and analysis of $\mathcal{A}_{\text{decomp}}$ see Lemma 3.3 in [Pettie and Ramachandran 2002b]. By applying Theorem 6.6 the expected running time is $O((i+1)m) = O(im)$ and the number of random bits used is $O(\lambda_{i+1}(n)\, \lambda_{i+1}^{(2)}(n) \log \lambda_{i+1}^{(2)}(n)) = o(\lambda_i(n))$. □

The MST sensitivity analysis problem [Tarjan 1982; Pettie 2005] is, given a graph G and $T = MST(G)$, to decide by how much the weights of edges in T can be increased without affecting the identity $T = MST(G)$.

Corollary 6.8. *The minimum spanning tree sensitivity analysis problem can be solved in $O(im)$ expected time using $\lambda_i(n)$ random bits.*

Proof. In the sensitivity analysis problem the $\mathcal{A}_{\text{decomp}}$ algorithm is nearly trivial [Dixon et al. 1992]. For $\mathcal{A}_{\text{det}}$ we can use the $O(m\alpha(m,n))$ time algorithm of Tarjan [1982] or the faster $O(m \log \alpha(m,n))$ time algorithm of Pettie [2005]. We use the Dixon-Rauch-Tarjan [1992] algorithm for $\mathcal{A}_{\text{rand}}$. □

6.3 Randomized Selection

Blum et al. [1973] showed that with $O(n)$ binary comparisons one can select the kth largest element in a set of n elements from a total order. All subsequent work on this problem has focused on bounding the constant factors involved. Floyd and Rivest [1975] gave a *randomized* algorithm that performs an expected $1.5n + o(n)$ comparisons. This bound was later shown to be tight [Cunto and Munro 1989]. For the deterministic version of the problem somewhat weaker bounds are known. Dor and Zwick [1999], building on the approach of Schonhage et al. [1976], gave a selection algorithm making roughly $2.945n$ comparisons. Dor and Zwick [2001] also showed that any *deterministic* selection algorithm must perform $(2 + \epsilon)n$ comparison, where ϵ is a very small constant. Thus there is at least a $(\frac{1}{2} + \epsilon)n$ separation between the randomized and deterministic complexity of the problem.

Although the Floyd-Rivest algorithm is fast and optimal in terms of comparisons, it uses $\tilde{O}(n^{1/2})$ random bits. Motwani and Raghavan [1995] observed that by using a pairwise independent sampler it is possible to perform selection with $O(\log n)$ random bits and $1.5n + o(n)$ comparisons. We improve this result to show below how the number of random bits can be reduced exponentially, to $O(\log \log n)$.

Theorem 6.9. *There is a randomized Selection algorithm that uses $O(\log \log n)$ random bits and performs $1.5n + o(n)$ comparisons with probability $1 - (\log n)^{-\Omega(1)}$.*

Proof. (Sketch) Consider the following variation on the Floyd-Rivest algorithm to select the element with rank k. Let $g = n/\log^2 n$. We choose a sample S of cardinality g and sort S. We then choose two sampled elements $a, b \in S$ whose ranks are expected to be closest to $k - g$ and $k + g$, respectively. Using an expected $1.5n + o(n)$ comparisons we categorize all elements as less than a, greater than b, or between a and b. If the rank k element lies between a and b we sort this set; otherwise the algorithm fails. If the probability of failure is at most $(\log n)^{-2}$ then we can afford to sort all n elements, at an expected cost of $O(n/\log n) = o(n)$ comparisons.

Rather than choosing S from all $\binom{n}{g}$ subsets we choose it from a small collection $\mathcal{S}_n$ of subsets. Thus the number of random bits used is $\log |\mathcal{S}_n|$. A particular collection of subsets $\mathcal{S}_n$ is *good* if for every input permutation, the probability that the algorithm either fails or fails to use less than $1.5n + O(n/\log n)$ comparisons is no more than $\log^{-2} n$. We show, using the probabilistic method, that there exists a good $\mathcal{S}_n$ with $|\mathcal{S}_n| = s_n \le 4 \log^9 n$.

If the algorithm fails then the chosen subset satisfies one of four conditions; we give two, each has a symmetric condition. Either fewer than $(k - g)/\log^2 n$ elements were included from the first k elements (corresponding to the case when a has rank higher than k) or more than $(k - g)/\log^2 n$ elements were included from the first $k - 2g$ elements (corresponding to the case when too many elements lie between a and b). We bound the probability of failure using the inequality $\Pr[|X - \mathbb{E}(X)| > t] < e^{-t^2/2m}$, where X is the sum of m Bernoulli trials [Cormen et al. 1990]. The probability of any failing condition holding is at most $4e^{-n/2\log^6 n}$. Suppose that $\mathcal{S}_n$ consists of s_n subsets of size g, each chosen uniformly at random. The probability that $\mathcal{S}_n$ is not good is at most $n! \cdot s_n^{s_n/\log^2 n} \cdot 4e^{-ns_n/2\log^8 n}$. Therefore there must exist, for each n, a good collection $\mathcal{S}_n$ of size $s_n = (2 + o(1)) \log^9 n$. □

ACKNOWLEDGMENTS

We would like to thank David Zuckerman for his helpful suggestions.

REFERENCES

Ajtai, M., Komlós, J., and Szemerédi, E. 1987. Deterministic simulation in Logspace. In *Ann. ACM Symposium on the Theory of Computation (STOC'87).* 132–140.

Bach, E. and Shallit, J. 1996. *Algorithmic Number Theory.* The MIT Press.

Bar-Noy, A., Motwani, R., and Naor, J. 1992. A linear time approach to the set maxima problem. *SIAM J. Discr. Math. 5,* 1, 1–9.

Blum, M., Floyd, R. W., Pratt, V., Rivest, R. L., and Tarjan, R. E. 1973. Time bounds for selection. *J. Comput. Syst. Sci. 7,* 4, 448–461.

Borůvka, O. 1926. O jistém problému minimálním. *Práce Moravské Přírodovědecké Společnosti 3*, 37–58. In Czech.

Buchsbaum, A. L., Kaplan, H., Rogers, A., and Westbrook, J. R. 1998. Linear-time pointer-machine algorithms for LCAs, MST verification, and dominators. In *Proc. 30th ACM Symposium on Theory of Computing (STOC).* 279–288.

Chazelle, B. 2000a. A minimum spanning tree algorithm with inverse-Ackermann type complexity. *J. ACM 47,* 6, 1028–1047.

Chazelle, B. 2000b. The soft heap: an approximate priority queue with optimal error rate. *J. ACM 47,* 6, 1012–1027.

Chong, K. W., Han, Y., and Lam, T. W. 2001. Concurrent threads and optimal parallel minimum spanning trees algorithm. *J. ACM 48,* 2, 297–323.

Cohen, A. and Wigderson, A. 1989. Dispersers, deterministic amplification, and weak random sources. In *Proceedings 30th Symposium on Foundations of Computer Science (FOCS).* 14–25.

Cole, R. 1999. Personal communication.

Cole, R., Klein, P. N., and Tarjan, R. E. 1996. Finding minimum spanning forests in logarithmic time and linear work using random sampling. In *Proc. SPAA'96.* 243–250.

Cole, R. and Vishkin, U. 1986. Approximate and exact parallel scheduling with applications to list, tree, and graph problems. In *Proc. FOCS'86.* 478–491.

Cole, R. and Vishkin, U. 1991. Approximate parallel scheduling. II. Applications to logarithmic-time optimal parallel graph algorithms. *Information and Computation 92,* 1, 1–47.

Cormen, T. H., Leiserson, C. E., and Rivest, R. L. 1990. *Introduction to Algorithms.* MIT Press, Cambridge, Mass.

Cunto, W. and Munro, J. I. 1989. Average case selection. *J. ACM 36,* 2, 270–279.

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik 1*, 269–271.

Dixon, B., Rauch, M., and Tarjan, R. E. 1992. Verification and sensitivity analysis of minimum spanning trees in linear time. *SIAM J. Comput. 21,* 6, 1184–1192.

Dor, D. and Zwick, U. 1999. Selecting the median. *SIAM J. Comput. 28,* 5, 1722–1758.

Dor, D. and Zwick, U. 2001. Median selection requires $(2+\epsilon)n$ comparisons. *SIAM J. Discr. Math. 14,* 3, 312–325.

Floyd, R. and Rivest, R. 1975. Expected time bounds for selection. *Comm. ACM 18,* 3, 165–172.

Gabow, H. N., Galil, Z., and Spencer, T. H. 1989. Efficient implementation of graph algorithms using contraction. *J. ACM 36,* 3, 540–572.

Gabow, H. N., Galil, Z., Spencer, T. H., and Tarjan, R. E. 1986. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica 6*, 109–122.

Gazit, H. 1991. An optimal randomized parallel algorithm for finding connected components in a graph. *SIAM J. Comput. 20,* 6, 1046–1067.

Goddard, W., Kenyon, C., King, V., and Schulman, L. 1993. Optimal randomized algorithms for local sorting and set-maxima. *SIAM J. Comput. 22,* 2, 272–283.

GRAHAM, R. L. AND HELL, P. 1985. On the history of the minimum spanning tree problem. *Ann. Hist. Comput. 7,* 1, 43–57.

GRAHAM, R. L., YAO, A. C., AND YAO, F. F. 1980. Information bounds are weak in the shortest distance problem. *J. ACM 27,* 3, 428–444.

HALPERIN, S. AND ZWICK, U. 1996. An optimal randomised logarithmic time connectivity algorithm for the EREW PRAM. *J. Comput. Syst. Sci. 53,* 3, 395–416.

HALPERIN, S. AND ZWICK, U. 2001. Optimal randomized EREW PRAM algorithms for finding spanning forests. *J. Algor. 39,* 1, 1–46.

IMPAGLIAZZO, R. AND ZUCKERMAN, D. 1989. How to recycle random bits. In *30th Ann. Symp. on Foundations of Computer Science (FOCS)*. 248–253.

JARNÍK, V. 1930. O jistém problému minimálním. *Práca Moravské Přírodovědecké Společnosti 6*, 57–63. In Czech.

JOFFE, A. 1974. On a set of almost deterministic k-independent random variables. *Ann. Probability 2,* 1, 161–162.

KARGER, D. R. 1993. Random sampling in matroids, with applications to graph connectivity and minimum spanning trees. In *34th Annual Symposium on Foundations of Computer Science*. 84–93.

KARGER, D. R., KLEIN, P. N., AND TARJAN, R. E. 1995. A randomized linear-time algorithm for finding minimum spanning trees. *J. ACM 42*, 321–329.

KARP, R. M. AND RAMACHANDRAN, V. 1990. Parallel algorithms for shared-memory machines. In *Handbook of Computer Science*. MIT Press/Elsevier, 869–942.

KENYON-MATHIEU, C. AND KING, V. 1989. Verifying partial orders. In *Proc. STOC'89*. 367–374.

KING, V. 1997. A simpler minimum spanning tree verification algorithm. *Algorithmica 18,* 2, 263–270.

KING, V., POON, C. K., RAMACHANDRAN, V., AND SINHA, S. 1997. An optimal EREW PRAM algorithm for minimum spanning tree verification. *Info. Proc. Lett. 62,* 3, 153–159.

KOMLÓS, J. 1985. Linear verification for spanning trees. *Combinatorica 5,* 1, 57–65.

KRUSKAL, J. B. 1956. On the shortest spanning subtree of a graph and the traveling salesman problem. In *Proc. Amer. Math. Soc.* Vol. 7. 48–50.

LIBERATORE, V. 1998. Matroid decomposition methods for the set maxima problem. In *Proc. SODA'98*. 400–409.

MILLER, G. L. 1976. Riemann's hypothesis and tests for primality. *J. Comput. Syst. Sci. 13,* 3, 300–317.

MOTWANI, R. AND RAGHAVAN, P. 1995. *Randomized Algorithms.* Cambridge University Press, New York.

PATERSON, A. S. M. S. AND PIPPENGER, N. 1976. Finding the median. *J. Comput. Syst. Sci. 13*, 184–199.

PETTIE, S. 1999. Finding minimum spanning trees in $O(m\alpha(m,n))$ time. Technical Report CS-TR-99-23, University of Texas, Austin.

PETTIE, S. 2003. Ph.D. thesis. Ph.D. thesis, The University of Texas at Austin. Department of Computer Sciences Technical Report TR-03-35, http://www.cs.utexas.edu/ftp/pub/techreports/tr03-35.ps.gz.

PETTIE, S. 2005. Sensitivity analysis of minimum spanning trees in sub-inverse-Ackermann time. In *Proceedings 16th Int'l Symposium on Algorithms and Computation (ISAAC)*. 964–973.

PETTIE, S. AND RAMACHANDRAN, V. 2002a. Minimizing randomness in minimum spanning tree, parallel connectivity and set maxima algorithms. In *Proc. 13th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA)*. 713–722.

PETTIE, S. AND RAMACHANDRAN, V. 2002b. An optimal minimum spanning tree algorithm. *J. ACM 49,* 1, 16–34.

PETTIE, S. AND RAMACHANDRAN, V. 2002c. A randomized time-work optimal parallel algorithm for finding a minimum spanning forest. *SIAM J. Comput. 31,* 6, 1879–1895.

PRIM, R. C. 1957. Shortest connection networks and some generalizations. *Bell Systems Technical Journal*, 1389–1401.

Rabin, M. O. 1980. Probabilistic algorithm for testing primality. *J. Number Theory 12,* 1, 128–138.

Schmidt, J. P., Siegel, A., and Srinivasan, A. 1995. Chernoff-Hoeffding bounds for applications with limited independence. *SIAM J. Discr. Math. 8,* 2, 223–250.

Tarjan, R. E. 1982. Sensitivity analysis of minimum spanning trees and shortest path problems. *Info. Proc. Lett. 14,* 1, 30–33. See Corrigendum, IPL **23**(4):219.