Minimizing Randomness in Minimum Spanning Tree, Parallel Connectivity, and Set Maxima Algorithms*

Seth Pettie and Vijaya Ramachandran
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712
seth@cs.utexas.edu, vlr@cs.utexas.edu

TR-01-25

July 13, 2001

Abstract

There are several fundamental problems whose deterministic complexity remains unresolved, but for which there exist randomized algorithms whose complexity is equal to known lower bounds. Among such problems are the minimum spanning tree problem, the set maxima problem, the problem of computing connected components and (minimum spanning trees in parallel, and the problem of performing sensitivity analysis on shortest path trees and minimum spanning trees. However, while each of these problems has a randomized algorithm whose performance meets a known lower bound, all of these randomized algorithms use a number of random bits which is linear in the number of operations they perform.

We address the issue of reducing the number of random bits used in these randomized algorithms. For each of the problems listed above, we present randomized algorithms that have optimal performance but use only a polylogarithmic number of random bits; for some of the problems our optimal algorithms use only \( \log^* n \) random bits. Our results represent an exponential savings in the amount of randomness used to achieve the same optimal performance as in the earlier algorithms. Our techniques are general and could likely be applied to other problems.

*This work was supported by Texas Advanced Research Program Grant 003658-0029-1999 and NSF Grant CCR-9088160. Seth Pettie was also supported by an MCD Fellowship.
1 Introduction

For many fundamental algorithmic problems there is a discrepancy between what we know of their deterministic complexity versus their established randomized complexity. In this paper we examine some well-studied problems fitting this description: the minimum spanning tree problem, the parallel connectivity and parallel minimum spanning tree problems, the set maxima problem, and the sensitivity analysis problem on shortest paths trees and minimum spanning trees. We propose new algorithms for these problems which have optimal performance but which use significantly fewer random bits than earlier algorithms.

Randomized algorithms are frequently simpler, and, in the abstract, faster than their deterministic counterparts (e.g. testing primality [Rab80], median finding [BFP+72, FR75], computing minimum spanning trees [KKT95]). However, they make use of a commodity that is scarce in reality: a stream of perfectly random bits. In practice a stream of “random” bits is produced by a pseudo-random function which has been seeded with a little non-determinism\(^1\), say the least significant digits from the computer’s clock. It is not clear if the seed is truly random, and even less clear how good the commonly used pseudo-random functions are. Bach [Bac91] studied a few number theoretic algorithms under the assumption of a truly random seed and a commonly used pseudo-random function, and showed them to have a good probability of success, though not as good as guaranteed by using totally random bits. Karloff and Raghavan [KR93] assumed the same model and showed that Quicksort can have much poorer performance than predicted. Other peculiarities of certain pseudo-random functions were noted in [FLW92] for Monte Carlo simulations and in [HR96, Hsu97] for parallel implementations of basic graph algorithms. Although debunking commonly used pseudo-random functions is worthwhile, the lesson here is to design algorithms which demand much less randomness, and more important, to analyze them in a realistic abstract environment.

There has been considerable amount of work on derandomizing randomized algorithms. A common technique to reducing randomness is to use \(k\)-wise independent random variables rather than totally independent ones. The generation of \(k\)-wise independent and approximately \(k\)-wise independent random variables has been well-studied [Jof74, CG89, NN93, EGL+98, CRS00]. Several results are known on derandomizing randomized algorithms that use \(k\)-wise independence to obtain deterministic algorithms (see [KW85, Lub86, AB186, Lub93, BR91]). Very recently Klivans and Spielman [KS01] gave a randomness-efficient method for testing if a multivariate polynomial is identically zero. In all of these algorithms a reduction in randomness is traded for an acceptable increase in the running time\(^2\).

1.1 Our Results

In this paper we address the issue of reducing randomness but with an additional twist: we require our algorithms to perform an optimal number of operations (to within a constant factor), and we focus on reducing randomness subject to this primary goal. In the limit we would like to use no random bits at all, and obtain optimal deterministic algorithms. But in the absence of this ultimate result, it is a worthwhile goal to reduce our dependence on such a scarce resource as randomness.

\(^1\)We note that there are now web servers (e.g. HotBits, http://www.fourmilab.ch/hotbits/) which will provide a modest number of bits upon request. The bits, apparently random, are derived by measuring some unpredictable physical process, such as the decay of a radioactive substance.

\(^2\)One exception is an intermediate result in [Lub86] where a parallel MIS algorithm that uses a logarithmic number of random bits is given that has the same resource bounds as the original algorithm that used a linear number of random bits, but neither of these algorithms perform optimal work.
We propose optimal algorithms using a reduced number of random bits for all the problems given in the Introduction. In our algorithms, at most a polylogarithmic-sized random seed is assumed, and methods for deriving random variables from that seed are analyzed explicitly. Our results represent an exponential reduction over earlier optimal algorithms in the number of random bits used.

We describe our results below. A summary is given in Table 1. Our results are obtained using two main techniques – (1) using the properties of an arbitrary $k$-wise independent sampler, and (2) re-using random bits.

1.1.1 Parallel MST & Connectivity

The best deterministic parallel MST and connectivity algorithms [CV86, CV91, CHL99] run in logarithmic-time yet they all use superlinear work. There are somewhat simpler logarithmic-time linear expected work randomized MST and connectivity algorithms [Gaz91, CKT96, HZ96, PR99, HZ01], but each uses a linear number of random bits.

We present a new randomized MST algorithm which requires only a pairwise independent sampler. Our parallel implementation, which runs on the EREW PRAM [KR92], takes expected linear work using $O(\log^3 n \log^* n)$ random bits. Our sampling approach differs from [KKT95] and previous parallel MST algorithms [CKT96, PR99]; it is conceptually simpler but not as easily parallelizable, resulting in a running time of $O(\log^2 n \log^* n)$.

A simpler version of our parallel algorithm also solves parallel connected components with the same resources, improving upon [HZ96, HZ01, Gaz91] in terms of the number of random bits used.

1.1.2 Set Maxima and Local Sorting

In the set maxima problem we are given a set system $(\chi, S)$ where $\chi$ is a set of $n$ totally ordered elements and $S = \{S_1, \ldots, S_m\}$ is a collection of subsets of $\chi$, and asked to determine the maximum element in each set $S_1, \ldots, S_m$. The goal is to obtain the solution with an algorithm that uses the minimum number of comparisons between elements in $\chi$.

This intriguing problem seems to have been introduced by Graham, Yao and Yao [GYY80] who noted the trivial solution – just sort $\chi$ – and gave a simple $O(n + m2^m)$ time algorithm, which is optimal for very small $m$. 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; this was later shown to be asymptotically tight in [GKKS93]. Liberatore [Lib98] has shown the set maxima problem to be precisely the problem of verifying the optimal base of an arbitrary matroid, and Karger [Kar93] has demonstrated the usefulness of set maxima in actually finding an optimal base. Many other concrete problems are instances of set maxima (or are reducible to it). These include verifying a partial order [KMK89], sensitivity analysis (including verification) of minimum spanning trees and shortest path trees [Tar82, Kom85], and orienting the edges of an undirected, node-weighted graph from the lesser to greater endpoint. This last problem was dubbed local sorting by Goddard et al. [GKK99].

Besides the simple set maxima algorithm of [GYY80] and the trivial algorithm, there are really only two results to speak of for the general set maxima problem. Bar-ney et al. [BNM92] gave a deterministic algorithm that uses $O(n)$ expected comparisons when the $m = n$ sets are chosen randomly. Goddard et al. gave an elegant randomized algorithm for set maxima which makes an optimal $O(n \log \frac{m+n}{n})$ expected comparisons using the same number of random bits.
We apply our $k$-wise independence result to improve Goddard et al.’s [GKKS93] local sorting and general set maxima algorithms as follows. We give an optimal local sorting algorithm which uses $O(\log n \log \log \log n)$ random bits, and an optimal set maxima algorithm which uses $O(\log n \log^{(3)} n 2^{\Theta(\beta(m,n))})$ random bits, where \(^3\beta(m,n) = \log^* n - \log^* \frac{m}{n}\). Both algorithms make an expected $O(n \log \frac{m+n}{n})$ comparisons, which is optimal [GYY80, GKKS93].

### 1.1.3 Reusing Random Bits

Using the simple technique of re-using random bits, we obtain a dramatic reduction in the number of random bits used to find a minimum spanning tree and to perform sensitivity analysis on MSTs and shortest path trees. Sensitivity analysis subsumes the simpler MST/SSSP verification problem. For each of these problems there exist optimal deterministic algorithms with unknown complexities (see [PR00] for MST and [DRT92] for MST/SSSP sensitivity analysis), deterministic algorithms which take time $O(m\alpha(m,n))$, where $\alpha(m,n)$ is Tarjan’s inverse-Ackermann function (see [Cha00a] for MST and [Tar82] for MST/SSSP sensitivity analysis) and expected linear-time algorithms which use a linear number of random bits [KKT95, DRT92, GKKS93]. For both problems we give expected linear-time algorithms which use just $\log^* n$ random bits.

### 1.2 Organization

The rest of the paper is organized as follows. In Section 2 we give a fairly general lemma on the expected behavior of a $k$-wise independent sampler and show how pairwise independent sampling can be performed efficiently on the EREW PRAM. In Section 3 we give a new parallel MST/connectivity algorithm requiring only pairwise independence. In Section 4 we observe that any 4-wise independent sampler works in a previous set maxima algorithm, and we give improved set maxima and local sorting algorithms which reduce the required number of random bits to polylogarithmic. Finally, in Section 5 we use the technique of re-using random bits to give a simple linear expected time MST algorithm (based on the optimal MST algorithm in [PR00]) and an expected linear-time algorithm for sensitivity analysis of MST and shortest path trees, both of which use only $\log^* n$ random bits.

### 2 Limited Independence Sampling

In this section we establish a fairly general result on $k$-wise independent sampling which suggests that $O(1)$-wise independence is nearly as good in situations common to many randomized sorting-type algorithms. The situation is this: we have a set of elements from a total order $\chi$ and wish to find an element on the cheap whose rank is close to some desired rank $t$. If we have an abundance of randomness, we can simply select each element of $\chi$ independently with probability $p$ and take the rank $tp$ sampled element as a decent approximation of the actual rank $t$ element. A tradeoff between the efficiency and accuracy of this scheme can be had by manipulating the sampling probability $p$.

We show that by using just pairwise independence the expected rank (w.r.t. $\chi$) of the rank $t$ sampled element is $O\left(\frac{t}{p} \log n\right)$, and using a $2k$-wise independence, $k > 1$, its expectation is $O\left(\frac{t}{p}\right)$. There is then a natural tradeoff between $k$ and the concentration of the distribution of the $t^{th}$ sampled element around its mean.

The following Lemma is just an extension of Chebyshev’s inequality for 0/1 random variables. A more complex proof of this result appears in [SSS95]; our proof is elementary. We note that a similar, though incorrect, lemma appears in [Cha00c, p. 424].

---

\(^3\)This is the same $\beta$ function as the one defined in Fredman & Tarjan’s minimum spanning tree algorithm [FT87]
<table>
<thead>
<tr>
<th>Problem</th>
<th>Deterministic Bound</th>
<th>Probabilistic Bound</th>
<th>This paper</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel $NC$ graph connectivity (work)</td>
<td>$O(ma(m,n))$ [CV91]</td>
<td>$O(m)$ random bits $O(m)$ (EREW) $O(m)$ random bits $o(\log^{3+\varepsilon} n)$ random bits</td>
<td></td>
</tr>
<tr>
<td>Parallel $NC$ minimum spanning trees (work)</td>
<td>$O(m \log^3 n)$ [CV86]</td>
<td>$O(m)$ random bits $O(m)$ (EREW) $O(m)$ random bits $o(\log^{3+\varepsilon} n)$ random bits</td>
<td></td>
</tr>
<tr>
<td>Local sorting (comparisons)</td>
<td>$O(n \log n)$ (trivial)</td>
<td>$O(n \log \frac{m+n}{n})$ random bits $O(n \log \frac{m+n}{n})$ random bits $o(\log^{1+\varepsilon} n)$ random bits</td>
<td></td>
</tr>
<tr>
<td>Set maxima (comparisons)</td>
<td>$O(n \log n)$ (trivial)</td>
<td>$O(n \log \frac{m+n}{n})$ random bits $O(n \log \frac{m+n}{n})$ random bits $o(\log^{1+\varepsilon} n)$ random bits</td>
<td></td>
</tr>
<tr>
<td>Minimum spanning trees (time)</td>
<td>$O(ma(m,n))$ [Cha00a]</td>
<td>$O(m)$ random bits $O(m)$ (EREW) $O(m)$ random bits $O(\log^* n)$ random bits</td>
<td></td>
</tr>
<tr>
<td>MST/SSSP sensitivity analysis (time)</td>
<td>$O(ma(m,n))$ [Tar82]</td>
<td>$O(m)$ random bits $O(m)$ (EREW) $O(m)$ random bits $O(\log^* n)$ random bits</td>
<td></td>
</tr>
</tbody>
</table>

Abbreviations: EREW and CRCW are, resp., the Exclusive Read Exclusive Write and Concurrent Read Concurrent Write parallel RAMs. Every EREW algorithm is a CRCW algorithm. Optimal$(m,n)$ is the decision-tree complexity of the respective problem. We denote by $\varepsilon$ an arbitrarily small constant. With the exception of Set Maxima, $m = |E|$ is the number of edges and $n = |V|$ is the number of vertices; for Set Maxima $m$ is the number of sets and $n$ the number of elements.

**Table 1.** Summary of our results.
Lemma 2.1 Let $X_1, \ldots, X_n$ be $2k$-wise independent 0/1 random variables, each with mean $\mu$. For $X = \sum_i X_i$ we have $\mu_X = \mu$ and

$$\Pr [ |X - \mu_X| \geq t ] < \left( \frac{4k \mu_X}{t^2} \right)^k$$

Proof: Along the lines of Chebyshev’s inequality we have $\Pr [ |X - \mu_X| \geq t ] = \Pr [ (X - \mu_X)^{2k} \geq t^{2k} ]$ which is $\leq \frac{\mathbb{E} (X - \mu)^{2k}}{t^{2k}}$ by Markov’s inequality. The numerator can be expanded into an expression of the form $\mathbb{E} \sum_i \prod (X_i - \mu)$ — the expectation of a sum of products. By identifying duplicate factors in each product we can simplify them to be of the form $\prod_i (X_i - \mu)^{a_i}$ where $\sum_i a_i = 2k$. Notice that because the $X_i$’s are $2k$-wise independent the factors of each product are also independent. We may then rewrite the numerator in the form $\sum \prod \mathbb{E} (X_i - \mu)^{a_i}$ — a sum of products of expectations. Observe that in any term, if some $a_i = 1$ then $\mathbb{E} (X_i - \mu)^{a_i} = 0$ and the term disappears. We bound the numerator by first bounding a single term then bounding the number of non-zero terms. For the first, note that $\mathbb{E} (X_i - \mu)^{a_i} = \mu (1 - \mu) [(1 - \mu)^{a_i - 1} - (-\mu)^{a_i - 1}] \in (-\mu, \mu)$, hence each term is bounded by $\mu^k$.

Bounding the number of non-zero terms is equivalent to a balls-and-bins problem: how many ways are there to put $2k$ balls in $n$ bins (order counts!) such that all bins have zero or $\geq 2$ balls? Let $N$ be the number of non-zero terms, we have that

$$N \leq \sum_{i=1}^{k} \binom{n}{i} \binom{2k - i - 1}{i - 1} \frac{(2k)!}{2^k}$$

Here $i$ represents the number of non-empty bins, $\binom{n}{i}$ the number of ways of selecting such bins, and $\binom{2k - i - 1}{i - 1}$ the number of ways to distribute the $2k - 2i$ balls still unaccounted for. Each such distribution of balls-to-bins can be realized by a number of distinct orderings, which is certainly no more than $(2k)!/2^k$. Simplifying the above expression,

$$N \leq \sum_{i=1}^{k} \frac{n^i}{i!} \cdot (2k - i - 1)^{i - 1} \cdot \frac{(2k)!}{2^k}$$

$$\leq \frac{4}{3} \cdot \frac{n^k}{k!} \cdot \frac{(2k)!}{k!} \cdot \frac{(2k)!}{2^k} \cdot \frac{1}{(i - 1)!} \cdot \{i^{th} \text{ term } \leq \frac{1}{i} \cdot \{i + 1\}^{th} \text{ term.} \}$$

$$\leq \frac{4}{3} \cdot \frac{1 + 2k - 1}{\sqrt{2k \pi}} \cdot \left( \frac{ne}{e} \right)^k \cdot \left( \frac{2k e}{k} \right)^k \cdot \left( \frac{2k + 2}{e^2} \right)^k \cdot \{\text{Stirling’s approximation} \}$$

$$\leq (4nk)^k$$

We conclude that $\mathbb{E} (X - \mu_X)^{2k} \leq (4\mu_X k)^k$.

This bound is reasonably tight. For a lower bound consider just those terms with exactly $k$ distinct factors (each repeated twice).

$$N \geq \binom{n}{k} \cdot \frac{(2k)!}{2^k}$$

$$\geq \frac{n^k}{k!} \cdot \frac{(2k)!}{4^k} \cdot \{k \leq n/2 \}$$

$$\geq n^k \left( \frac{e}{k} \right)^k \frac{4k^2}{e^2} \cdot 4^{-k}$$
\[ \geq \left( \frac{nk}{e} \right)^k \]

We conclude, using the fact that \( E(X_i - \mu)^2 = \mu(1-\mu) \), that \( \mathbb{E}(X - \mu_x)^{2k} = \Omega \left( (nk\mu(1 - \mu)e^{-1})^k \right) \)

\[ \square \]

The main lemma of this section is given below.

**Lemma 2.2** Let \( \chi \) be a set of totally ordered elements and \( \chi_s \) be a subset of \( \chi \) derived by sampling each element with probability \( p \) using a 2\( k \)-wise independent sampler. Let \( Y \) be the number of unsampled elements less than \( \min \chi_s \). Then

\[
\mathbb{E}(Y) \leq \begin{cases} 
\frac{4 \ln(np) + O(1)}{p} & \text{for } k = 1 \\
\frac{21}{p} & \text{for } k > 1
\end{cases}
\]

and

\[
\mathbb{P}[Y \geq \ell] \leq \min_{\nu \leq k} \left\{ \left( \frac{4 \nu}{p \ell} \right)^\nu \right\}
\]

**Proof:** Let \( X_i = 1 \) if the element of \( \chi \) with rank \( i \) is sampled, and 0 otherwise. So \( \mathbb{E}(X_i) = p \) and for any distinct indices \( i_1, \ldots, i_{2k}, \ X_i_1, \ldots, X_i_{2k} \) are independent.

Let \( S_\ell = \sum_{i=1}^{\ell} X_i \) count the number of ones in \( X_1, \ldots, X_\ell \). We have that \( \mathbb{E}(S_\ell) = p\ell \) and

\[
\mathbb{P}[Y \geq \ell] = \mathbb{P}[S_\ell = 0] \leq \mathbb{P}[|S_\ell - \mathbb{E}(S_\ell)| \geq p\ell]
\]

Using Lemma 2.1 we can bound \( \mathbb{P}[Y \geq \ell] \) as follows.

\[
\mathbb{P}[|S_\ell - \mathbb{E}(S_\ell)| \geq p\ell] \leq \frac{\mathbb{E}(S_\ell - E(S_\ell))^{2k}}{(p\ell)^{2k}} < \left( \frac{4p\ell k}{(p\ell)^2} \right)^k \quad \{ \text{Lemma 2.1} \} = \left( \frac{4k}{p\ell} \right)^k
\]

The second part of the Lemma follows from the simple observation that any \( 2k \)-wise independent distribution is also \( 2\nu \)-wise independent for \( \nu \leq k \).

To bound \( \mathbb{E}(Y) \) we use a variation on a familiar identity. For a random variable \( Z \) taking on values from the naturals it is easy to show that \( \mathbb{E}(Z) = \sum_{i=1}^{\infty} \mathbb{P}[Z \geq i] \) (a similar expression can be used for real \( Z \)). Plugging our best bound on \( \mathbb{P}[Y \geq \ell] \) into this identity gives a weak bound on the expectation of \( Y \). We have \( \mathbb{E}(Y) \leq \left( 4kp^{-1} \right)^k \cdot \sum_{i} i^{-k} \). Before we give a tighter analysis, consider the following bound on \( \mathbb{E}(Z) \) for any natural r.v. \( Z \).

\[
\mathbb{E}(Z) = \sum_{i=1}^{\infty} i \cdot \mathbb{P}[Z = i] \leq \delta + \sum_{i=\delta+1}^{\infty} \mathbb{P}[Z \geq i]
\]

\[
\leq \delta + \Delta \sum_{i=0}^{\infty} \mathbb{P}[Z \geq \delta + 1 + \Delta \cdot i]
\]
We will now bound $E(Y)$ using this inequality. Assume w.l.o.g. that $k$ is 1 or 2. Letting $\delta = \Delta = \frac{\lambda}{p}$, we have that

$$E(Y) \leq \frac{\beta}{p} + \frac{\beta}{p} \sum_{i=0}^{\infty} \Pr[Y \geq \frac{\beta(i+1)}{p}]$$

$$\leq \frac{\beta}{p} \left( 1 + \frac{4k}{\beta} \sum_{i=1}^{k} \frac{i^{-k}}{i^{-k}} \right)$$

\{For $k = 2$ and $\beta = 10$\} $\leq \frac{21}{p}$

\{For $k = 1$ and $\beta = 1$\} $\leq \frac{4\ln(np) + O(1)}{p}$

We omit the proof of the following Lemma; it is similar to that of Lemma 2.2.

**Lemma 2.3** Let $\chi$ be a set of totally ordered elements and $\chi_s$ be a subset of $\chi$ derived by sampling each element with probability $p$ using a 2k-wise independent sampler. Let $x_t$ be the element of $\chi_s$ with rank $t$ and let $Y_t$ be the number of elements in $\chi$ less than $x_t$. Then

$$E(Y_t) = \begin{cases} O(tp^{-1}\log(np)) & \text{for } k = 1 \\ O(tp^{-1}) & \text{for } k > 1 \end{cases}$$

### 2.1 Pairwise Independent Sampling on the EREW PRAM

In Section 3 we need a method for generating a set of sampled elements in linear time in the size of the sample. Furthermore, we would like it to work on the EREW PRAM, which is a much more realistic model than the CRCW PRAM. We solve both of these problems using Joffe’s method for generating $k$-wise independent variables, given below.

**Lemma 2.4** (Joffe [Joff94]) 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.

That is, for generating pairwise independent variables we require two random coefficients, $a_0$ and $a_1$. We assume that $m$ (the number of edges) is prime and that all edges are given a unique ID in $\mathbb{Z}_m$; if $m$ is composite we find a prime $q > m$ and include $q - m$ dummy edges. We sample the edges with probability (about) $p$ as follows. If $X(i) = a_1i + a_0 \pmod{m} \in [0..[pm] - 1]$ then edge $i$ is sampled; otherwise it is not. Evaluating the polynomial $X$ on $m$ points is too expensive because the number of sampled elements could be sublinear in $m$. Under the assumption that $a_1 \neq 0$ we can generate the sampled graph by generating all solutions to $i = (j - a_0a_1^{-1}) \pmod{m}$ for $j \in [0..[pm] - 1]$. This leads us to the following scheme for assigning processors to sampled edges. It takes work linear in the size of the sample, usually $O(pm)$.

We assume an EREW PRAM with $P$ processors, each of which knows $m, a_0, a_1, a_1^{-1}$, and its unique processor ID.

If $a_1 = 0$ and $a_0 \geq [pm]$ then $X(\cdot) = a_0$ and no edges are sampled.
If \( a_1 = 0 \) and \( a_0 < \lceil pm \rceil \) then all edges are sampled. Processor \( k \) is assigned edges \( \lceil \frac{m}{p} \rceil k \) through \( \lceil \frac{m}{p} \rceil (k + 1) - 1 \).

If \( a_1 \neq 0 \), then processor \( k \) is assigned edges with IDs of the form \((j - a_0)a_1^{-1} \mod m\), for \( \lceil \frac{m}{p} \rceil k \leq j < \lceil \frac{m}{p} \rceil (k + 1) \).

Notice that with Joffe’s pairwise independent sampler, assigning EREW processors to sampled edges is quite easy, whereas using his 3-wise independent sampler would be more cumbersome. We would need to resolve the inevitable conflicts that occur when more than one processor attempts to claim the same edge.

### 2.2 Finding a Prime in Parallel

Joffe’s [Jof74] method for generating pairwise independent variables relies on having a known prime. Since \( m \) is a relatively small number (w.r.t. the number of processors), we can find the first prime greater than \( m \) very easily. Baker and Harman (see [BS96, p. 225]) showed that if \( p_n \) is the \( n \)th prime, then \( p_n - p_{n-1} \leq n^{535+o(1)} \). We use this bound to find the smallest prime not less than \( m \).

**Lemma 2.5** 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 with \( O(\log m) \) time, \( c \log^2 m \) random bits, and \( b(m) \cdot c \log m \) processors, where \( b(m) = m^{535+o(1)} \).

**Proof:** We run the Miller-Rabin [Mil76, Rab80] primality test \( c \log m \) times on each integer in \([m..m+b(m)]\), 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. \( \square \)

If a better bound on \( b(m) \) is established then finding the next prime \( \geq m \) can be made deterministic. Consider Cramér’s conjecture.

**Conjecture 2.1** (Cramér) Let \( p_n \) be the \( n \)-th prime. Then \( p_n - p_{n-1} = O(\log^2 n) \).

If this conjecture were true we could employ \( O(\sqrt{m} \log m) \) processors to find the first prime \( \geq m \) in \( O(\log m) \) time deterministically.

### 3 A Parallel MST/Connectivity Algorithm

#### 3.1 Overview

In this section we present a new randomized MST algorithm which uses the same type of approach as the one in Karger, Klein and Tarjan [KKT95]. Our algorithm has two desirable attributes. It, like the KKT algorithm, is based on ‘Borůvka steps’ and is therefore parallelizable. Second, we show that our algorithm can tolerate a lower quality random sampler. In particular we use a pairwise independent sampler in lieu of total independence. This allows us to reduce the number of random bits used to polylogarithmic in the input size.

We assume, in this section and in Section 5, a familiarity with the minimum spanning tree problem. See [CLR90] or [Tar82] for a description of MST.
3.2 An Alternate Randomized MST Algorithm

A traditional Borůvka step identifies and contracts at least half of the unidentified MST edges, reducing the number of vertices by at least a factor of two. If implemented in the usual way, each Borůvka step takes linear time and Borůvka’s algorithm takes $O(m \log n)$ time. Below we define inductively an approximate Borůvka step, based upon a normal Borůvka step in a sampled graph.

Definition 3.1 In the $i^{th}$ approximate Borůvka step all edges are sampled with a fixed probability. An edge is eligible to participate in this step if it is sampled, it is not a self-loop, and it was not tainted in the first $i-1$ approximate Borůvka steps — see Definition 3.2.

Definition 3.2 In an approximate Borůvka step each vertex selects and contracts its minimum weight incident edge which is eligible by Definition 3.1. An unsampled edge $(u, v)$ becomes tainted if it is lighter than either one of the edges selected by $u$ or $v$.

Let $G_0 = G$ be the original graph and let $G_i$ be the graph after $i$ approximate Borůvka steps. If for some vertex $v$ no edge incident on $v$ is sampled, we let $v$ choose an imaginary infinite weight edge $(v, \infty)$, thus tainting all unsampled edges incident on $v$. This guarantees that after $\log n$ approximate Borůvka steps the graph will contract to a single vertex.

```
\textbf{$i^{th}$ Approximate Borůvka Step:}
1. Let $G_i$ be derived from $G_{i-1}$ by randomly sampling each edge with probability $p(i)$ and removing self-loops.
2. For each vertex $v$ let $e_v$ be the least weight untainted edge incident on $v$ which appears in $G_i$, and let $F = \{e_v : v \in V(G_i)\}$
3. Let $G_i$ be derived from $G_{i-1}$ by contracting all edges in $F$.
```

After $\log n$ approximate Borůvka steps we have constructed an approximate MST (composed of all edges contracted in Step 3). We then employ a linear-time MST verification algorithm (see [DRT92, Kin97, Kom85]) to filter out those edges never tainted, then reduce the number of vertices in the graph by performing a few exact Borůvka steps. Repeated iterations — approximate Borůvka steps, filtering, and exact Borůvka steps — will eventually reduce the graph to a single vertex. All edges identified in the exact Borůvka steps belong to the MST. The efficiency of this algorithm depends upon how we implement the sampling and ensuring that (most of the time) the number of edges in the graph is reduced by a constant factor in each iteration. Before addressing these matters of efficiency we should prove correctness. Lemma 3.1, proved below, implies that edges never tainted after the $\log n^{th}$ approximate Borůvka step are indeed not in the MST.

Definition 3.3 Let $C$ be a subgraph of $G$ and $U \subseteq E(G)$. A subgraph $C$ is contractible w.r.t. $U$ if for any $e_1 = (u_1, v_1), e_2 = (u_2, v_2) \in U$ where $u_1, u_2 \in C$, there exists a path in $C$ between $u_1$ and $u_2$ consisting of edges with weight less than $\max\{w(e_1), w(e_2)\}$.

Lemma 3.1 Let $C_v$ denote the subgraph of $G$ contracted to form $v \in V(G_i)$. Then $C_v$ is contractible w.r.t. all edges still untainted after the $i^{th}$ approximate Borůvka step.

Proof: Note that $C_v$ is the union of some set $\{C_{v_1}, \ldots, C_{v_j}\}$ where $\{v_1, \ldots, v_j\} \subseteq V(G_{i-1})$. We inductively assume that the $\{C_{v_j}\}$ are contractible w.r.t. untainted edges. Consider first the case
where \( j = 2 \), that is \( C_v = C_{v_1} \cup C_{v_2} \), and let \( e_1 \) and \( e_2 \) be arbitrary untainted edges incident on \( C_{v_1} \) and \( C_{v_2} \) resp. We must show there exists a path in \( C_v \) connecting \( e_1 \) to \( e_2 \) satisfying Definition 3.3.

Let \( e_3 = (v_1, v_2) \) be the edge chosen in the \( i^{th} \) Borůvka step, and let \( P_1 \) (resp. \( P_2 \)) be the path connecting \( e_1 \) and \( e_3 \) (resp. \( e_3 \) and \( e_2 \)) which is guaranteed by the contractibility of \( C_{v_1} \) (resp. \( C_{v_2} \)). See Figure 1.

![Figure 1: Components are contractible w.r.t. untainted edges.](image)

We claim all edges on the path \( < P_1, e_3, P_2 > \) are lighter than \( \max \{w(e_1), w(e_2)\} \). W.l.o.g. assume \( e_3 \) was chosen by \( v_1 \) (as opposed to \( v_2 \)) in the \( i^{th} \) Borůvka step. Since \( e_1 \) remains untainted this implies \( w(e_3) < w(e_1) \). By the contractibility of \( C_{v_1} \) edges in \( P_1 \) are lighter than either \( e_1 \) or \( e_3 \); similarly edges in \( P_2 \) are lighter than either \( e_2 \) or \( e_3 \). Hence all edges in \( < P_1, e_3, P_2 > \) are lighter than either \( e_1 \) or \( e_2 \) implying \( C_v \) is contractible w.r.t. untainted edges. We can reduce the case \( j > 2 \) to \( j = 2 \) by simply contracting subgraphs in pairs. \( \square \)

The usefulness of the approximate Borůvka steps does not lie in how many MST edges they find (we assume none) but in how many non-MST edges are filtered. We must therefore show that a significant number of edges are untainted at the end of the computation. The following Lemma assumes a totally independent sampler. We show later how to use a sampler with limited independence.

**Lemma 3.2** The expected number of tainted edges is at most \( \sum_{i=1}^{\infty} \frac{n}{2^{i-1} p(i)} \)

**Proof:** Consider a vertex \( v \in G_{i-1} \). The probability that exactly \( k \) untainted edges incident on \( v \) are tainted in the \( i^{th} \) Borůvka step is \((1 - p(i))^k \cdot p(i)\), hence the expected number of edges tainted by \( v \) in the \( i^{th} \) Borůvka step is bounded by \( \frac{1 - p(i)}{p(i)} \). Since there are \( \leq n/2^{i-1} \) vertices taking part the expected total number of tainted edges is \( \leq \frac{n}{2^{i-1} p(i)} \). \( \square \)

**Corollary 3.1** The randomized MST algorithm of this section makes, with probability \( 1 - e^{-\Omega(m)} \), \( O(m) \) comparisons and runs in expected linear time when \( i^{1+\Theta(1)} \geq p(i) \geq 2^{-i} \cdot i^{1+\Theta(1)} \).

**Proof:** It is straightforward to see that the number of comparisons made by the algorithm is \( O(m) \) with the stated probability when \( p(i) \) is within the stated range. To show that all steps can be implemented in expected linear time, we defer to Section 3.4, where we show the much stronger result that the algorithm runs in polylog time with linear work on an EREW PRAM using only pairwise independent sampling. \( \square \)
3.3 Analysis with Pairwise Independent Sampling

In this Section we prove a variant of Lemma 3.2 assuming a limited independence sampler in each approximate Borůvka step. The samples from different Borůvka steps must be 

totally independent.

**Lemma 3.3** Let $p(i)$ be the sampling probability in the $i^{th}$ approximate Borůvka step, where 

$$i^{1+O(1)} \leq p(i)^{-1} \leq 2^{1-(2+O(1))}$$

Then the total number of sampled edges is $O(m)$ and the expected total number of tainted edges is,

a) $O(n \log \frac{2m}{n})$ using a pairwise independent sampler, and

b) $O(n)$ using a 4-wise independent sampler.

**Proof:** We prove only part (a); part (b) is proved similarly. In the $i^{th}$ approximate Borůvka step there are $n_i \leq n/2^i$ vertices, each with a list of adjacent untainted edges. Let $T_i$ be the edges tainted in the $i^{th}$ Borůvka step and let $T_i^j$ be the tainted edges of $T_i$ incident on vertex $j$. We bound the total number of tainted edges as

$$\mathbb{E}(\sum_i |T_i|) \leq \sum_{i,j} \mathbb{E}(|T_i^j|) \leq \sum_i \sum_j O(p(i)^{-1} \log |T_i^j|) \quad \{\text{Lemma 2.2}\}$$

$$\leq \sum_i O(p(i)^{-1}) n_i \log \frac{2m}{n} \quad \{\text{Concavity of log function}\}$$

$$\leq \sum_i O(p(i)^{-1}) n(i + \log \frac{2m}{n})$$

$$= O(n \log \frac{2m}{n})$$

The expected total number of sampled edges is $m \cdot \sum_i p(i)^{-1} = O(m)$. □

3.4 Implementation on the EREW PRAM

Our randomized MST algorithm presented in Section 3.2 is simpler than that of [KKT95], however, certain problems are introduced in its parallelization which were never encountered by previous parallel connectivity or MST algorithms. The problem of generating sampled graphs in linear time (in the sample size) was addressed in Section 2.1. A subtler question is this: once the set of sampled edges has been generated, how do their associated processors find each other? This was not an issue in [CKT94, CTK96, Gaz91, HZ96, HZ01, PR97, PR99] because the edges of a sampled graph were always kept together, in say, a linked list. Because of our sampling approach — the $i^{th}$ sampled graph being completely independent of the $(i-1)^{th}$ — we need a structure for sampled edges to find each other without stumbling over unsampled ones. We use King’s [Kin97] Borůvka tree structure to deal with this problem, as well as detecting tainted and contracted edges which are ineligible to participate in approximate Borůvka steps. The Borůvka tree and its uses are described below.

3.5 The Borůvka Tree

King’s Borůvka tree [Kin97] represents the edge contractions made during the course of Borůvka’s MST algorithm. Nodes on the $i^{th}$ level of the Borůvka tree correspond to vertices of the (contracted) graph after $i$ Borůvka steps. The children of some level-$i$ node in the Borůvka tree represent those vertices which were united by edge contractions made in the $i^{th}$ Borůvka step. With the application of MST verification in mind, Borůvka tree edges are weighted as follows. If $v$ is a level-$(i-1)$ node,
then the edge \((v, parent(v))\) is given the same weight as the edge selected by \(v\) in the \(i^{th}\) Borůvka step. We use a slightly looser definition of Borůvka tree. First, we allow some of the steps to be approximate Borůvka steps, and secondly, we allow for the case when only a subset of the vertices participate in a Borůvka step. For non-participating vertices \(v\), the edge \((v, parent(v))\) has weight \(-\infty\). The following Lemma is analogous to one proved by King [Kin97] about exact Borůvka steps.

**Lemma 3.4** Let \(B\) be a Borůvka tree constructed using approximate Borůvka steps, and \(B(u, v)\) be the tree path between \(u\) and \(v\) in \(B\). Then an edge \((u, v)\) is untainted iff it is heavier than all edges in \(B(u, v)\).

By Lemma 3.4 Borůvka trees are clearly useful for identifying tainted edges. When used on the EREW PRAM Borůvka trees are most easily managed if all nodes have no more than two children. The following procedure, which can be applied equally well to approximate and exact Borůvka steps, ensures that only pairs of vertices are contracted. It is based on a scheme of [HZ96] used originally for connected components.

**Pairing-Up Procedure:**

1. Let \((u, v)\) be the edge chosen by \(u\) in the (approximate) Borůvka step. The chosen edges form a forest of pseudotrees, each with a cycle of length two. Remove one edge from each cycle, forming a forest of rooted trees.

2. Let the children of \(v\) in the forest be \(\{v_1, \ldots, v_{\ell}\}\). For \(0 < i \leq \lfloor \frac{\ell}{2} \rfloor\), let \(e_i\) be the heavier of \((v_{2i-1}, v)\) and \((v_2i, v)\). Relocate the endpoints of \(e_i\) to be \((v_{2i-1}, v_{2i})\) and contract \(e_i\).

3. All vertices not contracted in Step (2) are arranged in a set of disjoint paths. Using a pairwise independent sampler, sample these vertices with probability \(\frac{1}{2}\) and contract those edges \((v, successor(v))\) where \(v\) was sampled but \(successor(v)\) was not.

Suppose \(n'\) edges were contracted in step (2). Then the expected number of edges contracted in total is at least \(n' + \frac{n - 3n'}{8} \geq \frac{n}{8}\). If, in step (3), we force the first vertex in each path to be sampled and the last to be unsampled, the expected number of edge contractions is at least \(n' + \frac{n - 3n'}{4} \geq \frac{n}{4}\).

**3.5.1 Detecting Ineligible Edges**

Let \(B_i\) be the partially constructed Borůvka tree after \(i\) approximate Borůvka steps and \(root_i(u)\) be the root of the subtree containing \(u\) in \(B_i\). A sampled edge is ineligible to participate in an approximate Borůvka step if it is tainted or if its endpoints are the same, i.e. it has been contracted. In the \(i^{th}\) approximate Borůvka step an edge \((u, v)\) is untainted if \(root_{i-1}(u) = root_{i-1}(v)\) in the Borůvka forest. An uncontracted edge \((u, v)\) is tainted if \(w(u, v)\) is lighter than some weight in the paths \(<u, \ldots, root_{i-1}(u)\>\) or \(<v, \ldots, root_{i-1}(v)\>\). Hence for each sampled edge, we need only traverse the tree paths from its endpoints to the root to determine eligibility. If the sample size conforms to its expectation of about \(p_m\), detecting ineligible edges can easily be accomplished in \(O(i \cdot \lfloor \frac{pm}{p} \rfloor)\) time using no special processor allocation steps. On the other hand, if all the edges are sampled, we can perform an initial processor allocation step in \(O\left(\frac{pm}{p} + \log n\right)\) time and detect ineligible edges in \(O\left(\frac{pm}{p} + i\right)\) time. Using the same type of tree traversal we can also determine the minimum weight sampled edge incident on each vertex on \(G_{i-1}\) within the same time bounds.
3.5.2 Contracting Vertices in Pairs

The edges chosen in the $i$th approximate Borůvka step form a forest $F$ of rooted trees in $G_{i-1}$. Let $v$ be a $G_{i-1}$ vertex and $\{v_1, \ldots, v_p\}$ be its children in $F$. We implement the Pairing-Up procedure by assigning a (virtual) processor to each edge $(v_i, v)$, and having each processor climb the Borůvka tree associated with $v$. When two processors reach the same node they form a pair and cease climbing the tree (see Figure 2). The set of unpaired nodes form a set of chains which can then be paired up using standard techniques. The Pairing-Up procedure — finding pairs and adding new Borůvka tree nodes — can easily be implemented in the same time bounds as detecting ineligible edges. Since exactly $pm$ edges are sampled with probability $1 - \frac{1}{m}$, and $m$ edges are sampled with probability only $\frac{p}{m}$, we conclude with the following Lemma.

**Lemma 3.5** The $i$th approximate Borůvka step can be executed on the EREW PRAM in expected $O(i \cdot \frac{p[i]m}{p})$ time using $4\log m$ random bits. The expected number of contracted edges is at least $\frac{n_{i-1}}{4}$, where $n_{i-1}$ is the number of vertices after $i - 1$ approximate Borůvka steps.

**Proof Sketch:** The time bounds were shown above. The number of random bits is $2\log m$ for each application of Joffe’s pairwise independent sampler. It is used once to generate the sampled edges and once to perform random mating. We note that a simpler sampler suffices for the random mating, since the sampling probability is always exactly $\frac{1}{2}$ (see [LW95]). \(\Box\)

3.6 Putting It All Together

The whole algorithm is given below. We use the fact that after each exact or approximate Borůvka step, the number of vertices shrinks by an expected factor of $\frac{3}{4}$.

Let $\lambda_1 = C, \lambda_{i+1} = \left(\frac{4}{3}\right)^{\lambda_i / 2}$.

Perform no more than $2\log^* n$ phases:

**Phase $k$:**

1. Perform $\lambda_k - \lambda_{k-1}$ exact Borůvka steps. Mark all contracted edges as part of MST.

2. Perform $O(\log n) - \lambda_k$ approximate Borůvka steps, using the sampling probability $p(i) = 1/\lambda^2$, for $i = \lambda_k + 1 \ldots O(\log n)$. Mark all contracted edges as part of approx. MST.

3. Use an MST verification algorithm to determine edges still untainted after Step 2. Discard these edges and retract the approximate Borůvka steps made in Step 2.
The following claim is key to the performance of our algorithm.

**Claim 3.1** After the $k^{th}$ Phase,
(a) The expected number of vertices is no more than \( \frac{n}{\lambda_{k+1}} \), and
(b) The expected number of edges is no more than \( \frac{n}{\lambda_{k+1}^2} \).

**Proof:** After \( k \) Phases \( \sum_{i=1}^{k} (\lambda_i - \lambda_{i-1}) = \lambda_k \) exact Borůvka steps have been performed, so the expected number of vertices is no more than \( n \left( \frac{3}{4} \right)^{\lambda_k} = \frac{n}{\lambda_{k+1}} \) giving part (a). As for part (b), observe that the expected number of edges tained in Step 2 is no more than

\[
\sum_{i=\lambda_{k+1}}^{\infty} i^3 n \left( \frac{3}{4} \right)^{i} \cdot 4 \ln \left( \frac{\frac{4}{3}^{\frac{i}{2}}}{\lambda_k^{2k}} \right) \leq \sum_{i=\lambda_{k+1}}^{\infty} 1.15 \cdot i^4 n \left( \frac{3}{4} \right)^{i} \\
\leq \frac{n}{\lambda_{k+1}^2} \sum_{i=\lambda_k}^{\infty} 1.15 \cdot i^4 \left( \frac{3}{4} \right)^{i} \\
\leq \frac{n}{\lambda_{k+1}^2} \{ \text{For } \lambda_k \text{ sufficiently large} \}
\]

□

The next claim establishes that \( 2 \log^* n \) Phases are sufficient to compute the MST.

**Claim 3.2** For \( C = \lambda_1 \) sufficiently large, \( \log^* (\lambda_i) \geq \frac{i}{5} \).

**Proof:** Let \( \alpha = 4/3 \). We need only show \( \lambda_{i+2} = \alpha^{\lambda_{i+2}/2} \geq 2^{\lambda_i} \).

\[
\alpha^{\lambda_{i+2}/2} \geq 2^{\log_2 \alpha^{\frac{\lambda_{i+2}/2}{2}}} = 2^{4(\lambda_{i+1}/2 - 2.44)} \geq 2^{4(\lambda_{i+1}/2 - 2.44) - 1.2702} \geq 2^{\lambda_i - 2.28} \geq 2^{\lambda_i} \{ \text{For } \lambda_i \geq 38 \}
\]

Hence the claim holds for \( C \geq 38 \). □

**Theorem 3.1** The minimum spanning tree problem can be solved on a \( P \)-processor EREW PRAM using expected linear work, \( O(\log^3 n \log^* n) \) random bits, and expected \( O \left( \frac{n}{P} + \log^2 n \log^* n \right) \) time.

**Proof:** Let \( m_k \) be the number of edges before the \( k^{th} \) phase. By Claim 3.1, \( \mathbb{E}[m_k] \leq \frac{m}{\lambda_{k+1}^2} \), and hence the \( \lambda_k - \lambda_{k-1} \) exact Borůvka steps take expected time \( O \left( \frac{m}{P \cdot \lambda_{k+1}^2} + \lambda_k \log n \right) \). By summing over \( k \) the expected total time needed for all exact Borůvka steps is \( O \left( \frac{m}{P} + \log^2 n \right) \). By Lemma 3.5 the \( i^{th} \) approximate Borůvka step takes expected time \( O \left( i \cdot \left[ \frac{i}{P} \cdot m \right] \right) = O \left( \frac{m}{P} + \log n \right) \) time. Using the bound from Claim 3.2, the total time required for approximate Borůvka steps is \( \sum_{i=1}^{O(\log n)} 2^{\log^* \left( \frac{m}{P} + \log n \right)} = O \left( \frac{m}{P} + \log^2 n \log^* n \right) \). We use the \( O(\log n) \)-time linear-work EREW MST verification algorithm of [KPRS97] for Step 3.
Finding the prime \( q \) for Joffe’s sampler is done in \( O(\log m) \) time with \( O(\log^2 m) \) random bits (see Lemma 2.5). One processor generates the \( O(\log^2 n \log^* n) \) random coefficients \( (a_0, a_1, a_2, a_3, \ldots) \), sending \( a_i \) to processor \( i \), which then computes \( (a_i^{-1} (a_1^{-1} (a_2^{-1} \ldots (a_i^{-1} (a_{i-1}^{-1} (a_{i-2}^{-1} \ldots (a_0^{-1}) \ldots) \ldots)))) \) in \( O(\log^2 n \log^* n) \) time. Then in \( O(\log^2 n \log^* n) \) time the sequence \( (q, a_0, a_1, a_2, a_3, \ldots) \) is disseminated to all processors. □

**Corollary 3.2** Connected components can be solved on a \( P \)-processor EREW PRAM using expected linear work, \( O(\log^3 n \log^* n) \) random bits, and expected \( O(\frac{m}{P} + \log^2 n \log^* n) \) time.

# 4 Local Sorting and Set Maxima

We first make an observation on the sampling requirements of the algorithms from [GKKS93].

**Observation 4.1** The optimal local sorting and set maxima algorithms of [GKKS93] only use properties of random samples which are guaranteed by Lemma 2.2 or Lemma 2.3 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 \( 2^{O(\log t \cdot n)} \) 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. We now describe better results for both problems.

**Local sorting.** We modify the local sorting algorithm of [GKKS93] in order to accommodate a pairwise independent sampler.

**Theorem 4.1** A graph on \( n \) vertices and \( m \) 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 [GKKS93] 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 [GKKS93] due to the pairwise independent sampler. We let \( p_i = 1/2^{2i/c} \) for \( i > 0 \) and \( c = 1 + \frac{2}{\log n} \). Summing over all phases the total number of comparisons is

\[
n \log D \sum_{i>0} 2^{2i} \left( \frac{1}{2^{2i-1/c}} + \frac{2^{2i/c}}{2^{2i+1/c}} \right) = O(n \log D) \cdot \sum_{i>0} \frac{2^{2i} \cdot 2^{2i/c}}{2^{2i (c+1)/c}} \{2^{(i-1)/c} > 3i \text{ for } i \text{ sufficiently large}\}
\]

\[
= O(n \log D)
\]

The last line follows from the fact that \( (2^{2i} \cdot 2^{2i/c}) / 2^{2i (c+1)/c} < 2^{-i} \) for \( i \) sufficiently large (recall \( c > 1 \)).
As observed by Goddard et al. [GKKS93], if \( p_i \leq 1/\log n \), \( i \) phases are sufficient, hence \( c \log \log \log n \) sampling events, each requiring \( 2 \log n \) random bits to generate the pairwise independent random variables. The hidden “big-Oh” constants in this scheme are somewhat hideous. A more practical choice of \( p_i \) would be \( 1/2^i \), causing the local sorting algorithm to consume \( O(\log n \log \log n) \) random bits. \( \square \)

**Set maxima.** We show that the set maxima algorithm can be made to use a polylog number of random bits using a 4-wise independent sampler. We consider a generalization of set maxima, ‘t-maxima’ [GKKS93], where we are asked to identify and sort the largest \( t \) elements in each set.

**Theorem 4.2** The \( t \)-maxima problem can be solved optimally, with expected \( O(n \log(t(m+n)/n)) \) comparisons, while using \( \log n \log \log n \log^{t \Theta(\log^{*} n - \log^{t} t)} = O(\log^{1+\epsilon} n) \) random bits.

**Proof:** As above, we will skip a description of the algorithm and jump straight to the analysis. Each recursive invocation of \( t \)-maxima solves one local sorting problem and makes three recursive calls. The recursion depth of the [GKKS93] algorithm can be shown to be \( \log_{2} n \), hence the \( 3^{\log_{2} n} \) 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 \( n^e \) (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 [GKKS93] that

\[
T(t,n,m) \leq c_{1} n \log \left( \frac{mt^{2}}{np^{2}} \right) + T(t, pn, m) + T(t, pm, n) + T(t, \frac{mt}{p^{2}}, m)
\]

where \( p \) is the 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 \( p_i = 1/2^{\frac{r-1}{4}} \), where \( p_{r-1} \) was the sampling probability used by the “parent” recursive call. We set \( t_0 = mt/np_{i+1} \).

We have then

\[
T(i, t, n, m) \leq c_{1} n \log \left( \frac{mt^{2}}{np^{2}} \right) + T(i+1, t, p_{i}n, m) + T(0, t_0, p_{i}n, n) + T(i+1, t, p_{i+1}n, m)
\]

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 \( T(i, t, n, m) \leq c n \log \left( \frac{m}{np_i} \right) \) for some \( c, \)

\[
T(i, t, n, m) \leq c_1 n \log \left( \frac{mt^2}{np_i^2} \right) + cp_i n \log \left( \frac{m}{np_{i+1}p_i} \right) + cp_i n \log \left( \frac{mt}{n(p_i)^2p_{i+1}} \right) + cp_{i+1} n \log \left( \frac{m}{np_{i+1}} \right)
\]

\[
\leq 4c_1 n \log \left( \frac{m}{np_i} \right) + c_{10} n \log \left( \frac{m}{np_i} \right) + 2n \log \left( \frac{m}{np_i} \right) + 2n \log \left( \frac{m}{np_{i+1}} \right)
\]

\[
= cn \log \left( \frac{m}{np_i} \right) \quad \{ \text{for } c \geq 9c_1, \text{ completing the induction} \}
\]

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 [GKKS93]. 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 \( p_{k+1} \) 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 \( 1/t_0 = np_{i+1}/mt < p_{i+1} \). Since the algorithm bottoms out when the
sampling probability is less than \( 1/n \) we conclude that the recursion depth is \( O(\log^* n - \log^* t) \) and
hence there are \( 3^O(\log^* n - \log^* t) \) recursive calls. \( \square \)

**Corollary 4.1** 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.

## 5 A Randomness-Efficient MST Algorithm

In this section we use the simple technique of re-using random bits to dramatically decrease the
amount of randomness needed to compute minimum spanning trees (and also sensitivity analysis)
in expected linear time.

The following Lemma is from Pettie & Ramachandran [PR00]. The partitioning procedure it
alludes to is based on a pass of Fredman & Tarjan’s MST algorithm [FT87], substituting Chazelle’s
soft heap [Cha00b] for the usual Fibonacci heap.

**Lemma 5.1** [PR00] A graph \( G \) may be partitioned in linear time into edge-disjoint subgraphs
\( M, C_1, \ldots, C_k \) such that

(a) \( |E(M)| \leq |E(G)|/4 \)

(b) \( t \leq |E(C_i)| \leq 2t \), for any choice of \( t \).

(c) \( \text{MST}(G) \subseteq \bigcup_i \text{MST}(C_i) \cup \text{MST}(G') \cup M \)

where \( G' \) is derived from \( G \) by contracting all the \( C_i \)'s and removing edges in \( M \).

We perform several iterations of the following algorithm, which is very similar to the MST
algorithm of [PR00]. See [PR00] for a detailed proof of correctness. After \( \log \alpha(m,n) \) iterations
the number of edges is reduced to less than \( m/\alpha(m,n) \) and either [Cha00a] or [PR00] may then
be employed to solve this problem in \( O(m) \) time. Let \( R \) be the sequence of random bits available,
\( |R| > \log^* n \).

1. Let \( M, C_1, \ldots, C_k \) be the partition guaranteed by Lemma 5.1 where \( t = \Theta(|R|) \).
2. Compute \( \text{MST}(G') \) in \( O(m \alpha(m,n/\sqrt{t})) \) time.
3. Reusing the same sequence \( R \) of random bits, compute \( \text{MST}(C_i) \) for all \( i \) using the
KKT randomized MST algorithm. If KKT attempts to use more than \( |R| \) random bits,
compute \( \text{MST}(C_i) \) deterministically in \( O(t \alpha(t,t)) \) time.
4. Perform two Borůvka steps on \( \bigcup_i \text{MST}(C_i) \cup \text{MST}(G') \cup M \); contract the MST edges
found.

The **Borůvka steps** performed in Step 4 simply reduce the number of vertices in the graph
by a constant factor by identifying and contracting a constant fraction of the remaining MST
edges. Each Borůvka step takes linear time. Since \( t = \Omega(\log^* n) \) this implies \( \alpha(m,n/\sqrt{t}) = O(1) \)
and therefore Step 2 takes linear time. The other steps of this algorithm take \( O(m) \) time except
perhaps Step 3. It is shown in [KKT95] that the KKT algorithm takes linear time with probability
\( 1 - e^{-\Omega(|E|)} \) where \( |E| \) is the number of edges. The probability that Step 3 takes linear time over
all \( \log \alpha(m,n) \) iterations of the algorithm is just the probability that KKT runs in \( O(t) \) time for all but \( O\left(\frac{m}{t \alpha(t,t)}\right) \) of all the \( C_i \)'s, in all iterations. For the purposes of analysis imagine a matrix whose rows correspond to the \( C_i \)'s and whose columns correspond to choices of \( R \), the random bits. The \( (i,j) \) element is good if the KKT algorithm runs in \( O(t) \) time on \( C_i \) using the random bits \( R = j \), and bad otherwise. A column is bad if more than \( \frac{1}{\alpha(m,n)} \) of its elements are bad. It follows that the proportion of bad elements in the matrix is \( e^{-\Omega(t)} \) and the proportion of bad columns is \( \alpha(t,t) \cdot e^{-\Omega(t)} = e^{-\Omega(t)} \) as \( t = \Omega(\log^* n) \). Hence the probability that Step 3 takes linear time is \( 1 - e^{-\Omega(t)} \).

**Theorem 5.1** For any \( r > \log^* n \), the minimum spanning tree of a graph can be computed in linear time with probability \( 1 - e^{-\Omega(r)} \), where the constant of proportionality does not depend on \( r \). The expected running time is also linear.

Using the same type of approach, the minimum spanning (and shortest paths) tree sensitivity analysis problem can be solved in expected linear time with \( \log^* n \) random bits. In this problem we must compute, for each edge, the largest perturbation of its weight that will not affect the minimum spanning tree (or shortest paths tree) – see [Tar82, DRT92]. As in [DRT92] we decompose the MST into microtrees, but we use microtrees of size \( \approx \log^* n \). We solve each microtree problem using the randomized expected linear time algorithm of [DRT92], re-using the same random bits, and we solve the ‘macrotree’ problem using [Tar82]. By linearity of expectations we obtain the following result.

**Theorem 5.2** Minimum spanning tree sensitivity analysis can be solved in expected linear time using \( \log^* n \) random bits.

The lower limit of \( \log^* n \) is not special. Any function \( f(n) \) for which \( \alpha(f(n) \cdot n, n) = O(1) \) can be substituted for \( \log^* n \) in both Theorems 5.1 and 5.2, though the constant hidden by big-Oh notation may increase.

6 Acknowledgment

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

References


