Approximating the Minimum Spanning Tree Weight in Sublinear Time

Bernard Chazelle *   Ronitt Rubinfeld †   Luca Trevisan ‡

Abstract

We present a probabilistic algorithm that, given a connected graph G (represented by adjacency lists) of maximum degree d, with edge weights in the set $\{1, \ldots, w\}$, and given a parameter $0 < \varepsilon < 1/2$, estimates in time $O(dw\epsilon^{-2} \log \frac{w}{\epsilon})$ the weight of the minimum spanning tree of G with a relative error of at most ε. Note that the running time does *not* depend on the number of vertices in G. We also prove a nearly matching lower bound of $\Omega(dw\epsilon^{-2})$ on the probe and time complexity of any approximation algorithm for MST weight.

The essential component of our algorithm is a procedure for estimating in time $O(d\varepsilon^{-2} \log \varepsilon^{-1})$ the number of connected components of an unweighted graph to within an additive error of ϵn. The time bound is shown to be tight up to within the $\log \varepsilon^{-1}$ factor. Our connected-components algorithm picks $O(1/\epsilon^2)$ vertices in the graph and then grows "local spanning trees" whose sizes are specified by a stochastic process. From the local information collected in this way, the algorithm is able to infer, with high confidence, an estimate of the number of connected components. We then show how estimates on the number of components in various subgraphs of G can be used to estimate the weight of its MST.

1 Introduction

Traditionally, a linear time algorithm has been held as the gold standard of efficiency. In a wide variety of settings, however, large data sets have become increasingly common, and it is often desirable and sometimes necessary to find very fast algorithms which can assert nontrivial properties of the data in *sublinear* time.

One direction of research that has been suggested is that of property testing [15, 8], which relaxes the standard notion of a decision problem. Property testing algorithms distinguish between inputs that have a certain property and

* chazelle@cs.princeton.edu. Princeton University and NEC Research Institute, Princeton, NJ. Part of this research was supported by NSF grant CCR-99817 and ARO Grant DAAH04-96-1-0181.

† ronitt@research.nj.nec.com. NEC Research Institute, Princeton, NJ.

‡ luca@eecs.berkeley.edu. U.C. Berkeley, Berkeley, CA.

those that are far (in terms of Hamming distance, or some other natural distance) from having the property. Sublinear and even constant time algorithms have been designed for testing various algebraic and combinatorial properties (see [14] for a survey). Property testing can be viewed as a natural type of approximation problem and, in fact, many of the property testers have led to very fast, even constant time, approximation schemes for the associated problem (cf. [8, 6, 7, 1]). For example, one can approximate the value of a maximum cut in a dense graph in time $2^{O(\epsilon^{-3}\log 1/\epsilon)}$, with relative error at most ε, by looking at only $O(\epsilon^{-7}\log 1/\epsilon)$ locations in the adjacency matrix [8]. Note that typically such schemes approximate the value of the optimal solution, here the size of a maxcut, without computing the structure that achieves it, i.e., the actual cut. Sometimes, however, a solution can also be constructed in linear or near-linear time.

In this paper, we consider the problem of finding the weight of the minimum spanning tree (MST) of a graph. Finding the MST of a graph has a long and interesting history [3, 10, 12]. Currently the best known deterministic algorithm of Chazelle [2] runs in $O(m\alpha(m,n))$ time, where n (resp. m) is the number of vertices (resp. edges) and α is inverse-Ackermann, and the randomized algorithm of Karger, Klein and Tarjan [11] runs in linear expected time (see also [5, 13] for alternative models).

In this paper, we show that there are conditions under which it is possible to approximate the weight of the MST of a connected graph in time sublinear in the number of edges. We give an algorithm which approximates the MST of a graph G to within a multiplicative factor of $1+\epsilon$ and runs in time $O(dw\epsilon^{-2}\log\frac{w}{\epsilon})$ for any G with max degree d and edge weights in the set $\{1,\ldots,w\}$. The relative error ε ($0<\varepsilon<1/2$) is specified as an input parameter. Note that if d and ε are constant and the ratios of the edge weights are bounded, then the algorithm runs in constant time. We also extend our algorithm to the case where G has nonintegral weights in the range $[1,w]$, achieving a comparable runtime with a somewhat worse dependence on ϵ.

Our algorithm considers several auxiliary graphs: If G is the weighted graph, let us denote by $G^{(i)}$ the subgraph of G that contains only edges of weight at most i. We estimate the number of connected components in each $G^{(i)}$. To do so, we sample uniformly at random $O(1/\epsilon^2)$ vertices in $G^{(i)}$, and then estimate the size of the component that contains each sampled vertex by constructing "local trees" of some appropriate size defined by a random process. Based on information about these local trees, we can produce a good approximation for the weight of the MST of G. Our algorithm for estimating the number of connected components in a graph runs in time $O(d\varepsilon^{-2}\log\varepsilon^{-1})$ and produces an estimate that is within an additive error of ϵn of the true count. The method is based on a similar principle as the property tester for graph connectivity given by Goldreich and Ron [9].

We give a lower bound of $\Omega(dw/\epsilon^2)$ on the time complexity of any algorithm which approximates the MST weight. In order to prove the lower bound, we give two distributions on weighted graphs, where the support set of one distribution contains graphs with MST weight at least $1+\epsilon$ times the MST weight of the graphs in the support of the other distribution. We show that any algorithm that reads $o(dw/\epsilon^2)$ weights from the input graph is unlikely to distinguish between graphs from the two distributions. We also prove a lower bound of $O(d/\varepsilon^2)$ on the running time of any approximation algorithm for counting connected components.

2 Estimating the Number of Connected Components

We begin with the problem of estimating the number of components in an arbitrary graph G. We present an algorithm which gives an additive estimate of the number of components in G to within ϵn in $O(d\epsilon^{-2}\log\epsilon^{-1})$ time, for any $0<\varepsilon<1/2$. We later show how to use the ideas from our algorithm to aid in estimating the weight of the MST of a graph.

Let c be the number of connected components in G. Let n_u be the number of vertices in u's component in G. Our algorithm is built around a simple observation:

Fact 1 *Given a graph with vertex set V, for every connected component $I \subseteq V$, $\sum_{u\in I}\frac{1}{n_u}=1$ and $\sum_{u\in V}\frac{1}{n_u}=c$.*

Our strategy is to estimate c by approximating each summand $1/n_u$. Computing n_u directly can take linear time, so we construct an estimator of the quantity $1/n_u$ that has the same expected value. We approximate the number of connected components via the algorithm given in Figure 1. The parameter W is a threshold value, which is set to $2/\varepsilon$ for counting connected components and somewhat higher for its use in MST weight estimation.

In the algorithm, doubling the number of vertices does not include duplicate visits to the same vertices; in other words, at each step the number of new vertices visited is supposed to match the number of vertices already visited. In our terminology, the first step of the BFS (shorthand for breadth first search) involves the visit of the single vertex u_i. We now bound the expectation and variance of the estimator β_i for a fixed i. If the BFS from u_i completes, the number of coin flips associated with it is $\lceil \log n_{u_i}\rceil$ and the number of distinct vertices visited is n_{u_i}. Let S denote the set of vertices in components of size $<W$. If $u_i\notin S$, then $\beta_i=0$; otherwise, it is $2^{\lceil \log n_{u_i}\rceil}/n_{u_i}$ with probability $2^{-\lceil \log n_{u_i}\rceil}$ and 0 otherwise. Since $\beta_i<2$, the variance of β_i is:

$$\mathbf{var}\,\beta_i \le \mathbf{E}\,\beta_i^2 \le 2\mathbf{E}\,\beta_i = \frac{2}{n}\sum_{u\in S}\frac{1}{n_u} \le \frac{2c}{n}\,.$$

Then the variance of $\hat{c}$ is bounded by

$$\mathbf{var}\,\hat{c} = \mathbf{var}\left(\frac{n}{r}\sum_i \beta_i\right) = \frac{n^2}{r^2}\cdot r\cdot \mathbf{var}\,\beta_i \le \frac{2nc}{r}. \qquad (1)$$

```
approx-number-connected-components(G, ε, W)
  uniformly choose r = O(1/ε^2) vertices u_1, ..., u_r
  for each vertex u_i,
    set β_i = 0
    take the first step of a BFS from u_i
    (*) flip a coin
        if heads and number of vertices visited in BFS < W
          then resume BFS to double number of visited vertices
            if this allows BFS to complete
              then set β_i = 2^{#coinflips}/#vertices visited in BFS
              else go to (*)
  output ĉ = (n/r) Σ_{i=1}^{r} β_i
```

Figure 1: Estimating the number of connected components

Since the number of components with vertices not in S is at most n/W, we have that

$$c - \frac{n}{W} \leq \mathbf{E}\,\hat{c} = \sum_{u \in S} \frac{1}{n_u} \leq c\,.$$

If we set $W = 2/\epsilon$, then

$$c - \frac{\epsilon n}{2} \leq \mathbf{E}\,\hat{c} \leq c \tag{2}$$

and, by Chebyshev,

$$\mathrm{Prob}[\,|\hat{c} - \mathbf{E}\,\hat{c}| > \epsilon n/2\,] < \frac{\mathbf{var}\,\hat{c}}{(\epsilon n/2)^2} \leq \frac{8c}{\epsilon^2 r n}\,. \tag{3}$$

Choosing $r = O(1/\epsilon^2)$ ensures that with constant probability arbitrarily close to 1, our estimate $\hat{c}$ of the number of connected components deviates from the actual value by at most ϵn.

The expected number of vertices visited in a given execution of the "for loop" is $O(\log W)$, and each newly visited vertex incurs a cost of $O(d)$, so the algorithm runs in expected time $O(d\epsilon^{-2}\log W)$. For our setting of W, this is $O(d\epsilon^{-2}\log\epsilon^{-1})$. As stated, the algorithm's running time is randomized. However, one can get a deterministic running time bound by stopping the algorithm after $Cd\epsilon^{-2}\log\epsilon^{-1}$ steps and outputting 0 if the algorithm has not yet terminated. This event occurs with probability at most $O(1/C)$, which is a negligible addition to the error probability. Thus we have the following theorem:

Theorem 2 *Let c be the number of components in a graph with n vertices. Then Algorithm* approx-number-connected-components *runs in time $O(d\epsilon^{-2}\log\epsilon^{-1})$ and with probability at least $3/4$ outputs $\hat{c}$ such that $|c - \hat{c}| \leq \epsilon n$.*

We can improve the running time to $O((\varepsilon + c/n)d\epsilon^{-2}\log\epsilon^{-1})$, which is much better for small values of c. First, run the algorithm for $r = O(1/\varepsilon)$. By Chebyshev and (1, 2),

$$\mathrm{Prob}\Big[\,|\hat{c} - \mathbf{E}\,\hat{c}| > \frac{\mathbf{E}\,\hat{c} + \epsilon n}{2}\,\Big] < \frac{8nc}{r(c + \varepsilon n/2)^2} \le \frac{8n}{r(c + \epsilon n/2)},$$

which is arbitrarily small for $r\varepsilon$ large enough. Next, we use this approximation $\hat{c}$ to "improve" the value of r. We set $r = A/\varepsilon + A\hat{c}/(\varepsilon^2 n)$ for some large enough constant A and we run the algorithm again, with the effect of producing a second estimate c^*. By (2, 3),

$$\mathrm{Prob}[\,|c^* - \mathbf{E}\,c^*| > \epsilon n/2\,] < \frac{8c}{\epsilon^2 rn} \le \frac{16c}{A\varepsilon n + A\mathbf{E}\,\hat{c}} \le \frac{16}{A},$$

and so with overwhelming probability, our second estimate c^* of the number of connected components deviates from c by at most ϵn. The running time of this new algorithm is $O((\varepsilon + c/n)d\epsilon^{-2}\log\epsilon^{-1})$.

3 Approximating the Weight of an MST

In this section we present an algorithm for approximating the value of the MST in bounded weight graphs. We are given a connected graph G with maximum degree d and with each edge is assigned an integer weight between 1 and w. We assume that G is represented by adjacency lists or, for that matter, any representation that allows one to access all edges incident to a given vertex in $O(d)$ time. We show how to approximate the weight of the minimum spanning tree of G with a relative error of at most ϵ.

In Section 3.1 we give a new way to characterize the weight of the MST in terms of the number of connected components in subgraphs of G. In Section 3.2 we give the main algorithm and its analysis. Finally, Section 3.3 addresses how to extend the algorithm to the case where G has nonintegral weights.

3.1 MST Weight and Connected Components

We reduce the computation of the MST weight to counting connected components in various subgraphs of G. To motivate the new characterization, consider the special case when G has only edges of weight 1 or 2 (i.e., $w = 2$). Let $G^{(1)}$ be the subgraph of G consisting precisely of the edges of weight 1, and let n_1 be its number of connected components. Then, any MST in G must contain exactly $n_1 - 1$ edges of weight 2, with all the others being of weight 1. Thus, the weight of the MST is exactly $n - 2 + n_1$. We easily generalize this derivation to any w.

For each $0 \le \ell \le w$, let $G^{(\ell)}$ denote the subgraph of G consisting of all the edges of weight at most ℓ. Define $c^{(\ell)}$ to be the number of connected components in $G^{(\ell)}$ (with $c^{(0)}$ defined to be n). By our assumption on the weights, $c^{(w)} = 1$. Let $M(G)$ be the weight of the minimum spanning tree of G. Using the above quantities, we give an alternate way of computing the value of $M(G)$:

```
approx-MST-weight(G, ε)
  For i = 1, ..., w − 1
    ĉ^(i) = approx-number-connected-components(G^(i), ε, 2w/ε)
  output v̂ = n − w + Σ_{i=1}^{w−1} ĉ^(i)
```

Figure 2: Approximating the weight of the MST

Claim 3 *For integer $w \geq 2$,*

$$M(G) = n - w + \sum_{i=1}^{w-1} c^{(i)}\,.$$

Proof: Let α_i be the number of edges of weight i in an MST of G. (Note that α_i is independent of which MST we choose [4].) Observe that for all $0 \leq \ell \leq w-1$, $\sum_{i>\ell} \alpha_i = c^{(\ell)} - 1$, therefore

$$M(G) = \sum_{i=1}^{w} i\alpha_i = \sum_{\ell=0}^{w-1} \sum_{i=\ell+1}^{w} \alpha_i = -w + \sum_{\ell=0}^{w-1} c^{(\ell)} = n - w + \sum_{i=1}^{w-1} c^{(i)}.$$

$\square$

Thus, computing the number of connected components allows us to compute the weight of the MST of G.

3.2 The Main Algorithm

Our algorithm approximates the value of the MST by estimating each of the $c^{(\ell)}$'s. The algorithm is given in Figure 2.

Theorem 4 *Let v be the weight of the MST of G. Algorithm* approx-mst-weight *runs in time $O(dw\epsilon^{-2} \log \frac{w}{\epsilon})$ and outputs a value $\hat{v}$ that, with probability at least 3/4, differs from v by at most εv.*

Proof: Let $c = \sum_{i=1}^{w-1} c^{(i)}$. Since we call approx-number-connected-components with parameter $W = 2w/\epsilon$, (1, 2) become

$$c^{(i)} - \frac{\epsilon n}{2w} \leq \mathbf{E}\,\hat{c}^{(i)} \leq c^{(i)} \qquad \text{and} \qquad \mathbf{var}\,\hat{c}^{(i)} \leq \frac{2nc^{(i)}}{r}\,.$$

By summing over i, it follows that $c - \epsilon n/2 \leq \mathbf{E}\,\hat{c} \leq c$ and $\mathbf{var}\,\hat{c} \leq 2nc/r$. Choosing $r\epsilon^2$ large enough, by Chebyshev we have

$$\mathrm{Prob}[\,|\hat{c} - \mathbf{E}\,\hat{c}| > (n - w + c)\epsilon/3\,] < \frac{18nc}{r\epsilon^2(n - w + c)^2},$$

which is arbitrarily small since we may assume that w/n is sufficiently small (else we might as well compute the MST explicitly, which can be done in $O(dn)$ time [11]). It follows that, with high probability, the error on the estimate satisfies

$$|v - \hat{v}| = |c - \hat{c}| \leq \frac{\varepsilon n}{2} + \frac{\varepsilon(n - w + c)}{3} \leq \varepsilon v.$$

Since the expected running time of each call to approx-number-connected-components is $O(dr \log w/\epsilon)$, the total running time is $O(dw\epsilon^{-2} \log \frac{w}{\epsilon})$. As before, the running time can be made deterministic by stopping execution of the algorithm after $Cdw\epsilon^{-2} \log \frac{w}{\epsilon}$ steps for some appropriately chosen constant C. □

3.3 Nonintegral Weights

Suppose the weights of G are all in the range $[1, w]$, but are not necessarily integral. To extend the algorithm to this case, one can multiply all the weights by $1/\epsilon$ and round each weight to the nearest integer. Then one can run the above algorithm with error parameter $\epsilon/2$ and with a new range of weights $[1, \lceil w/\epsilon \rceil]$ to get a value v. Finally, output ϵv. The relative error introduced by the rounding is at most $\epsilon/2$ per edge in the MST, and hence $\varepsilon/2$ for the whole MST, which gives a total relative error of at most ε. The runtime of the above algorithm is $O(dw\epsilon^{-3} \log \frac{w}{\varepsilon})$.

4 Lower Bounds

We prove that our algorithms for estimating the MST weight and counting connected components are essentially optimal.

Theorem 5 *Any probabilistic algorithm for approximating, with relative error ε, the MST weight of a connected graph with max degree d and weights in $\{1, \ldots, w\}$ requires $\Omega(dw\epsilon^{-2})$ edge weight lookups on average. It is assumed that $w > 1$ and $C\sqrt{w/n} < \varepsilon < 1/2$, for some large enough constant C.*

We can obviously assume that $w > 1$, otherwise the MST weight is always $n - 1$ and no work is required. The lower bound on ε is nonrestrictive since we can always compute the MST exactly in $O(dn)$ time, which is $O(dw\epsilon^{-2})$ for $\varepsilon = O(\sqrt{w/n})$.

Theorem 6 *Given a graph with n vertices, any probabilistic algorithm for approximating the number of connected components with an additive error of εn requires $\Omega(d\epsilon^{-2})$ edge lookups on average. It is assumed that $C/\sqrt{n} < \varepsilon < 1/2$, for some large enough constant C.*

Again, note that the lower bound on ε is nonrestrictive since we can always solve the problem exactly in $O(dn)$ time.

Both proofs revolve around the difficulty of distinguishing between two nearby distributions. For any $0 < q < 1/2$ and $s = 0, 1$, let $\mathcal{D}_q^s$ denote the distribution induced by setting a 0/1 random variable to 1 with probability $q_s = q(1+(-1)^s\varepsilon)$. We define a distribution $\mathcal{D}$ on n-bit strings as follows: (1) pick $s=1$ with probability 1/2 (and 0 else); (2) then draw a random string from $\mathcal{D}_q^s$ (by choosing each b_i from $\mathcal{D}_q^s$ independently). Consider a probabilistic algorithm that, given access to such a random bit string, outputs an estimate on the value of s. How well can it do?

Lemma 7 *Any probabilistic algorithm that can guess the value of s with a probability of error below $1/4$ requires $\Omega(\varepsilon^{-2}/q)$ bit lookups on average.*

Proof: By Yao's minimax principle, we may assume that the algorithm is deterministic and that the input is distributed according to $\mathcal{D}$. It is intuitively obvious that any algorithm might as well scan $b_1 b_2 \cdots$ until it decides it has seen enough to produce an estimate of s. In other words, there is no need to be adaptive in the choice of bit indices to probe (but the running time itself can be adaptive). To see why is easy. An algorithm can be modeled as a binary tree with a bit index at each node and a 0/1 label at each edge. An adaptive algorithm may have an arbitrary set of bit indices at the nodes, although we can assume that the same index does not appear twice along any path. Each leaf is naturally associated with a probability, which is that of a random input from $\mathcal{D}$ following the path to that leaf. The performance of the algorithm is entirely determined by these probabilities and the corresponding estimates of s. Because of the independence of the random b_i's, we can relabel the tree so that each path is a prefix of the same sequence of bit probes $b_1 b_2 \cdots$. This oblivious algorithm has the same performance as the adaptive one.

We can go one step further and assume that the running time is the same for all inputs. Let t^* be the expected number of probes, and let $0 < \alpha < 1$ be a small constant. With probability at most α, a random input takes time $\geq t \stackrel{\text{def}}{=} t^*/\alpha$. Suppose that the prefix of bits examined by the algorithm is $b_1 \cdots b_u$. If $u < t$, simply go on probing $b_{u+1} \cdots b_t$ without changing the outcome. If $u > t$, then stop at b_t and output $s = 1$. Thus, by adding α to the probability of error, we can assume that the algorithm consists of looking up $b_1 \cdots b_t$ regardless of the input string.

Let $p_s(b_1 \cdots b_t)$ be the probability that a random t-bit string chosen from $\mathcal{D}_q^s$ is equal to $b_1 \cdots b_t$. The probability of error satisfies

$$p_{\text{err}} \geq \frac{1}{2} \sum_{b_1 \cdots b_t} \min_s \, p_s(b_1 \cdots b_t).$$

Obviously, $p_s(b_1 \cdots b_t)$ depends only on the number of ones in the string, so if $p_s(k)$ denotes the probability that $b_1 + \cdots + b_t = k$, then

$$p_{\text{err}} \geq \frac{1}{2} \sum_{k=0}^{t} \min_s \, p_s(k). \tag{4}$$

By the normal approximation of the binomial distribution,

$$p_s(k) \rightarrow \frac{1}{\sqrt{2\pi t q_s(1-q_s)}} \, e^{-\frac{(k-tq_s)^2}{2tq_s(1-q_s)}},$$

as $t \to \infty$. This shows that $p_s(k) = \Omega(1/\sqrt{qt}\,)$ over an interval I_s of length $\Omega(\sqrt{qt}\,)$ centered at tq_s. If $qt\varepsilon^2$ is smaller than a suitable constant γ_0, then $|tq_0 - tq_1|$ is small enough that $I_0 \cap I_1$ is itself an interval of length $\Omega(\sqrt{qt}\,)$; therefore $p_{\rm err} = \Omega(1)$. This shows that if the algorithm runs in expected time $\gamma_0\varepsilon^{-2}/q$, for some constant $\gamma_0 > 0$ small enough, then it will fail with probability at least some absolute constant. By setting α small enough, we can make that constant larger than 2α. This means that, prior to uniformizing the running time, the algorithm must still fail with probability α.

Note that by choosing γ_0 small enough, we can always assume that $\alpha > 1/4$. Indeed, suppose by contradiction that even for an extremely small γ_1, there is an algorithm that runs in time at most $\gamma_1\varepsilon^{-2}/q$ and fails with probability $\leq 1/4$. Then run the algorithm many times and take a majority vote. In this way we can bring the failure probability below α for a suitable $\gamma_1 = \gamma_1(\alpha, \gamma_0) < \gamma_0$, and therefore reach a contradiction. This means that an expected time lower than ε^{-2}/q by a large enough constant factor causes a probability of error at least $1/4$. $\Box$

Proof (Theorem 6): Consider the graph G consisting of a simple cycle of n vertices $v_1, \ldots, v_n$. Pick $s \in \{0, 1\}$ at random and take a random n-bit string $b_1 \cdots b_n$ with bits drawn independently from $\mathcal{D}^s_{1/2}$. Next, remove from G any edge $(v_i, v_{i+1 \bmod n})$ if $b_i = 0$. Because $\varepsilon > C/\sqrt{n}$, the standard deviation of the number of components, which is $\Theta(\sqrt{n}\,)$, is sufficiently smaller than εn so that with overwhelming probability any two graphs derived from $\mathcal{D}^0_{1/2}$ and $\mathcal{D}^1_{1/2}$ differ by more than $\varepsilon n/2$ in their numbers of connected components. That means that any probabilistic algorithm that estimates the number of connected components with an additive error of $\varepsilon n/2$ can be used to identify the correct s. By Lemma 7, this requires $\Omega(\varepsilon^{-2})$ edge probes into G on average. Replacing ε by 2ε proves Theorem 6 for graphs of degree $d = 2$. For arbitrary d, we may simply add $d - 2$ loops to each vertex. Each linked list thus consists of two "cycle" pointers and $d-2$ "loop" ones. If we place the cycle pointers at random among the loop ones, then it takes $\Omega(d)$ probes on average to hit a cycle pointer. If we single out the probes involving cycle pointers, it is not hard to argue that the probes involving cycle pointers are, alone, sufficient to solve the connected components problem on the graph deprived of its loops: one expects at most $O(T/d)$ such probes and therefore $T = \Omega(d\varepsilon^{-2})$. $\Box$

Proof (Theorem 5): Again we begin with the case $d = 2$. The input graph G is a simple path of n vertices. Pick $s \in \{0, 1\}$ at random and take a random $(n-1)$-bit string $b_1 \cdots b_{n-1}$ with bits drawn independently from $\mathcal{D}^s_q$, where $q = 1/w$. Assign weight w (resp. 1) to the i-th edge along the path if $b_i = 1$ (resp. 0). The MST of G has weight $n - 1 + (w-1)\sum b_i$, and so its expectation is $\Theta(n)$. Also, note that the difference Δ in expectations between drawing from $\mathcal{D}^0_q$ or $\mathcal{D}^1_q$ is $\Theta(\varepsilon n)$.

Because $\varepsilon > C\sqrt{w/n}$, the standard deviation of the MST weight, which is $\Theta(\sqrt{nw}\,)$, is sufficiently smaller than Δ that with overwhelming probability any two graphs derived from $\mathcal{D}_q^0$ and $\mathcal{D}_q^1$ differ by more than $\Delta/2$ in MST weight. Therefore, any probabilistic algorithm that estimates the weight with a relative error of ε/D, for some large enough constant D, can be used to identify the correct s. By Lemma 7, this means that $\Omega(w\varepsilon^{-2})$ probes into G are required on average.

For $d > 2$, simply join each vertex in the cycle to $d-2$ others (say, at distance > 2 to avoid introducing multiple edges) and, as usual, randomize the ordering in each linked list. Assign weight $w+1$ to the new edges. (Allowing the maximum weight to be $w+1$ instead of w has no influence on the lower bound we are aiming for.) Clearly none of the new edges are used in the MST, so the problem is the same as before, except that we now have to find our way amidst $d-2$ spurious edges, which takes the complexity to $\Omega(dw\varepsilon^{-2})$. $\square$

5 Open Questions

It is natural to ask what can be done if the max degree restriction is lifted. We have made some progress on the case of graphs of bounded *mean* degree. Our algorithm for the case of nonintegral weights requires extra time. Is this necessary? Can the ideas in this paper be extended to finding maximum weighted independent sets in general matroids? There are now a small number of examples of approximation problems that can be solved in sublinear time; what other problems lend themselves to sublinear approximation schemes? More generally, it would be interesting to gain a more global understanding of what can and cannot be approximated in sublinear time.

References

[1] Alon, N., Dar, S., Parnas, M., Ron, D., *Testing of clustering,* Proc. *FOCS*, 2000.

[2] Chazelle, B., *A minimum spanning tree algorithm with inverse-Ackermann type complexity*, J. ACM, 47 (2000), 1028–1047.

[3] Chazelle, B., *The Discrepancy Method: Randomness and Complexity*, Cambridge University Press, 2000.

[4] Eppstein, D., *Representing all minimum spanning trees with applications to counting and generation,* Tech. Rep. 95-50, ICS, UCI, 1995.

[5] Fredman, M.L., Willard, D.E. *Trans-dichotomous algorithms for minimum spanning trees and shortest paths*, J. Comput. and System Sci., 48 (1993), 424–436.

[6] Frieze, A., Kannan, R. *Quick approximation to matrices and applications,* Combinatorica, 19 (1999).

[7] Frieze, A., Kannan, R., Vempala, S., *Fast monte-carlo algorithms for finding low-rank approximations*, Proc. 39th FOCS (1998).

[8] Goldreich, O., Goldwasser, S., Ron, D., *Property testing and its connection to learning and approximation*, Proc. 37th FOCS (1996), 339–348.

[9] Goldreich, O., Ron, D., *Property testing in bounded degree graphs*, Proc. 29th STOC (1997), 406–415.

[10] Graham, R.L., Hell, P. On the history of the minimum spanning tree problem, *Ann. Hist. Comput.* 7 (1985), 43–57.

[11] Karger, D.R., Klein, P.N, Tarjan, R.E., *A randomized linear-time algorithm to find minimum spanning trees,* J. ACM, 42 (1995), 321–328.

[12] Nešetřil, J. A few remarks on the history of MST-problem, *Archivum Mathematicum, Brno* 33 (1997), 15–22. Prelim. version in *KAM Series,* Charles University, Prague, No. 97–338, 1997.

[13] Pettie, S., Ramachandran, V. *An optimal minimum spanning tree algorithm*, Proc. 27th ICALP (2000).

[14] Ron, D., *Property testing (a tutorial),* to appear in "Handbook on Randomization."

[15] Rubinfeld, R., Sudan, M., *Robust characterizations of polynomials with applications to program testing,* SIAM J. Comput. 25 (1996), 252–271.