SIAM J. Comput.
Vol. 34, No. 6, pp. 1370–1379

APPROXIMATING THE MINIMUM SPANNING TREE WEIGHT IN SUBLINEAR TIME[*]

BERNARD CHAZELLE[†], RONITT RUBINFELD[‡], and LUCA TREVISAN[§]

Abstract. We present a probabilistic algorithm that, given a connected graph G (represented by adjacency lists) of average degree d, with edge weights in the set $\{1, \ldots, w\}$, and given a parameter $0 < \varepsilon < 1/2$, estimates in time $O(dw\varepsilon^{-2} \log \frac{dw}{\varepsilon})$ the weight of the minimum spanning tree (MST) 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 $\Omega(dw\varepsilon^{-2})$ 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 $O(d\varepsilon^{-2} \log \frac{d}{\varepsilon})$ the number of connected components of an unweighted graph to within an additive error of εn. (This becomes $O(\varepsilon^{-2} \log \frac{1}{\varepsilon})$ for $d = O(1)$.) The time bound is shown to be tight up to within the $\log \frac{d}{\varepsilon}$ factor. Our connected-components algorithm picks $O(1/\varepsilon^2)$ 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.

Key words. minimum spanning tree, sublinear time algorithms, randomized algorithms, approximation algorithms

AMS subject classifications. 68W20, 68W25, 68R10

DOI. 10.1137/S0097539702403244

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 [16, 8], which relaxes the standard notion of a decision problem. Property testing algorithms distinguish between inputs that have a certain property and 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 [15] 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, 5, 6, 1]). For example, one can approximate the

[*]Received by the editors February 27, 2002; accepted for publication (in revised form) November 9, 2004; published electronically July 26, 2005. A preliminary version of this work appeared in *Automata, Languages and Programming*, Lecture Notes in Comput. Sci. 2076, Springer-Verlag, Berlin, 2001, pp. 190–200.
http://www.siam.org/journals/sicomp/34-6/40324.html

[†]Department of Computer Science, Princeton University, Princeton, NJ, 08544 (chazelle@cs.princeton.edu). Part of this research was supported by NSF grant CCR-99817 and ARO grant DAAH04-96-1-0181.

[‡]Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA, 02139 (ronitt@csail.mit.edu). Part of this work was done while the author was at the NEC Research Institute.

[§]Computer Science Division, University of California–Berkeley, Berkeley, CA 94720 (luca@eecs.berkeley.edu).

value of a maximum cut in a dense graph in time $2^{O(\varepsilon^{-3}\log 1/\varepsilon)}$, with relative error at most ε, by looking at only $O(\varepsilon^{-7}\log 1/\varepsilon)$ locations in the adjacency matrix [8]. Other sublinear time approximation schemes have been applied to dense instances of graph bisection, general partitioning problems, quadratic assignment, minimum linear arrangement, and maximum acyclic subgraph and constraint satisfaction [8, 5], as well as clustering [1, 14]. Note that typically such schemes approximate the value of the optimal solution, for example, 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, distinguished 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. The randomized algorithm of Karger, Klein, and Tarjan [11] runs in linear expected time (see also [4, 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 $1+\varepsilon$ and runs in time $O(dw\varepsilon^{-2}\log\frac{dw}{\varepsilon})$ for any G with average degree d and edge weights in the set $\{1,\dots,w\}$. The algorithm requires no prior information about the graph besides w and n; in particular, the average degree is assumed to be unknown. 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 running time 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 $O(1/\varepsilon^2)$ 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 in turn 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 $O(d\varepsilon^{-2}\log\frac{d}{\varepsilon})$—or $O(\varepsilon^{-2}\log\frac{1}{\varepsilon})$ for $d=O(1)$—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 $\Omega(dw/\varepsilon^2)$ 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 $1+\varepsilon$ times the MST weight of the graphs in the support of the other distribution. We show that any algorithm that reads $o(dw/\varepsilon^2)$ weights from the input graph is unlikely to distinguish between graphs from the two distributions. We also prove a lower bound of $\Omega(d/\varepsilon^2)$ on the running time of any approximation algorithm for counting connected components. The above lower bounds apply to the class of graphs which may contain self-loops and multiple edges.

2. Estimating the number of connected components.

We begin with the problem of estimating the number of components in an arbitrary graph G. For no-

```
approx-number-connected-components($G, \varepsilon, W, d^*$)
  uniformly choose $r = O(1/\varepsilon^2)$ vertices $u_1, \dots, u_r$
  for each vertex $u_i$,
    set $\beta_i = 0$
    take the first step of a BFS from $u_i$
    (*) flip a coin
    if (heads) & (# vertices visited in BFS $< W$)
             & (no visited vertex has degree $> d^*$)
      then {resume BFS to double number of visited edges
            if this allows BFS to complete
              then {if $m_{u_i} = 0$ set $\beta_i = 2$
                    else set $\beta_i = d_{u_i} 2^{\#\text{coin flips}}/\#\text{edges}$ visited in BFS }
              else go to (*) }
  output $\hat{c} = \frac{n}{2r}\sum_{i=1}^{r} \beta_i$
```

FIG. 1. *Estimating the number of connected components; see main text for precise definition of BFS.*

tational convenience, we may assume that $d \geq 1$: This can always be achieved by implicitly adding a fictitious self-loop to each vertex, which does not change the number of connected components. We present an algorithm that gives an additive estimate of the number of components in G to within εn in $O(d\varepsilon^{-2} \log \frac{d}{\varepsilon})$ 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. We use the following notation: Given a vertex u, d_u is the number of edges incident upon it (including self-loops), and m_u is the number of edges in u's component in G. Finally, c denotes the number of connected components. Our algorithm is built around the following simple observation.

FACT 1. *Given a graph with vertex set V, for every connected component $I \subseteq V$, $\sum_{u \in I} \frac{1}{2} d_u/m_u = 1$ and $\sum_{u \in V} \frac{1}{2} d_u/m_u = c$.*

To handle isolated vertices, we must make the convention that $d_u/m_u = 2$ if $m_u = 0$. Our strategy is to estimate c by approximating each d_u/m_u. Computing them directly could take linear time, so we construct an estimator of the quantity d_u/m_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 $4/\varepsilon$ for counting connected components and somewhat higher for MST weight estimation. We also use an estimate d^* of the average degree d, which we compute separately in $O(d/\varepsilon)$ expected time (see Lemma 4). This approximation ensures that $d^* = O(d/\varepsilon)$ and that at most $\varepsilon n/4$ vertices have degree higher than d^*.

In the algorithm, doubling the number of edges does not include duplicate visits to the same edges; in other words, at each phase the number of new edges visited is supposed to match the number of distinct edges 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 and all its d_{u_i} incident edges. That is, unless $d_{u_i} > d^*$, in which case we abort the BFS.

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 $\lceil \log(m_{u_i}/d_{u_i}) \rceil$, and the number of distinct edges visited is m_{u_i}. Let S denote the set of vertices that lie in components with fewer than W vertices all of which are of degree at most d^*. If $u_i \notin S$, then $\beta_i = 0$; otherwise, it is $2^{\lceil \log(m_{u_i}/d_{u_i}) \rceil} d_{u_i}/m_{u_i}$ with probability

$2^{-\lceil \log(m_{u_i}/d_{u_i}) \rceil}$ (and 2 if $m_{u_i} = 0$) and 0 otherwise. Since $\beta_i \le 2$, the variance of β_i is

$$\mathbf{var}\, \beta_i \le \mathbf{E}\, \beta_i^2 \le 2\mathbf{E}\, \beta_i = \frac{2}{n} \sum_{u \in S} \frac{d_u}{m_u} \le \frac{4c}{n}\,.$$

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

$$\mathbf{var}\, \hat{c} = \mathbf{var} \left(\frac{n}{2r} \sum_i \beta_i \right) = \frac{n^2}{4r^2} \cdot r \cdot \mathbf{var}\, \beta_i \le \frac{nc}{r}. \tag{1}$$

By our choice of $W = 4/\varepsilon$ and d^*, there are at most $\varepsilon n/2$ components with vertices not in S, and so

$$c - \frac{\varepsilon n}{2} \le \mathbf{E}\, \hat{c} \le c\,. \tag{2}$$

Furthermore, by Chebyshev,

$$\mathrm{Prob}[\, |\hat{c} - \mathbf{E}\, \hat{c}| > \varepsilon n/2\,] < \frac{\mathbf{var}\, \hat{c}}{(\varepsilon n/2)^2} \le \frac{4c}{\varepsilon^2 rn}\,. \tag{3}$$

Choosing $r = O(1/\varepsilon^2)$ 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 edges visited in a given iteration of the "for loop" is $O(d_{u_i} \log M)$, where M is the maximum number of edges visited, which is at most $Wd^* = O(d/\varepsilon^2)$. Therefore, the expected running time of the entire algorithm is

$$\frac{O(r)}{n} \sum_{u \in V} d_u \log(Wd^*) = O(dr \log(Wd^*)) = O\left(d\varepsilon^{-2} \log \frac{d}{\varepsilon} \right)\,, \tag{4}$$

not counting the $O(d/\varepsilon)$ time needed for computing d^*.

As stated, the algorithm's running time is randomized. If d is known, however, we can get a deterministic running time bound by stopping the algorithm after $Cd\varepsilon^{-2} \log \frac{d}{\varepsilon}$ 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 $O(d\varepsilon^{-2} \log \frac{d}{\varepsilon})$ and with probability at least $3/4$ outputs $\hat{c}$ such that $|c - \hat{c}| \le \varepsilon n$.*

Finetuning the algorithm. If we proceed in two stages, first estimating c within a constant factor, and then in a second pass using this value to optimize the size of the sample, we can lower the running time to $O((\varepsilon + c/n)d\varepsilon^{-2} \log \frac{d}{\varepsilon})$. This is a substantial improvement for small values of c. First, run the algorithm for $r = O(1/\varepsilon)$. By Chebyshev and (1, 2),

$$\mathrm{Prob}\left[|\hat{c} - \mathbf{E}\, \hat{c}| > \frac{\mathbf{E}\, \hat{c} + \varepsilon n}{2} \right] < \frac{4nc}{r(c + \varepsilon n/2)^2} \le \frac{4n}{r(c + \varepsilon n/2)},$$

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),

$$\mathrm{Prob}[\,|c^* - \mathbf{E}\,c^*| > \varepsilon n/2\,] < \frac{4c}{\varepsilon^2 rn} \leq \frac{8c}{A\varepsilon n + A\mathbf{E}\,\hat{c}} \leq \frac{8}{A},$$

and so, with overwhelming probability, our second estimate c^* of the number of connected components deviates from c by at most εn. So we have the following theorem.

THEOREM 3. *Let c be the number of components in a graph with n vertices. Then there is an algorithm that runs in time $O(d\varepsilon^{-2}\log\frac{d}{\varepsilon})$ and with probability at least $3/4$ outputs $\hat{c}$ such that $|c - \hat{c}| \leq \varepsilon n$.*

Approximating the degree. We show how to compute the desired estimate d^* of the average degree d. Pick C/ε vertices of G at random, for some large constant C, and set d^* to be the maximum degree among them. To find the degree of any one of them takes $O(d)$ time on average, and so the expected running time is $O(d/\varepsilon)$. Imagine the vertex degrees sorted in nonincreasing order, and let ρ be the rank of d^*. With high probability, $\rho = \Theta(\varepsilon n)$. To see why, we easily bound the probability that ρ exceeds εn by $(1-\varepsilon)^{C/\varepsilon} \leq e^{-C}$. On the other hand, observe that the probability that $\rho > \varepsilon n/C^2$ is at least $(1-\varepsilon/C^2)^{C/\varepsilon} \geq e^{-2/C} > 1 - 2/C$.

LEMMA 4. *In $O(d/\varepsilon)$ expected time, we can compute a vertex degree d^* that, with high probability, is the kth largest vertex degree for some $k = \Theta(\varepsilon n)$.*

Note that $k = \Omega(\varepsilon n)$ alone implies that $d^* = O(d/\varepsilon)$, and so, if we scale ε by the proper constant, we can ensure that at most $\varepsilon n/4$ vertices have degree higher than d^*, and thus conform to the requirements of approx-number-connected-components.

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 average degree d and with each edge 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 MST 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 $0 \leq \ell \leq w$, 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 MST of G. Using the above quantities, we give an alternate way of computing the value of $M(G)$ in the following claim.

```
approx-MST-weight$(G, \varepsilon)$
  For $i = 1, \dots, w-1$
    $\hat{c}^{(i)} =$ approx-number-connected-components$(G^{(i)}, \varepsilon, 4w/\varepsilon, d^*)$
  output $\hat{v} = n - w + \sum_{i=1}^{w-1} \hat{c}^{(i)}$
```

FIG. 2. *Approximating the weight of the MST.*

CLAIM 5. *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 [7].) Observe that for all $0 \leq \ell \leq w-1$, $\sum_{i>\ell} \alpha_i = c^{(\ell)} - 1$; therefore

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

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. Note that we do not set $W = 4/\varepsilon$ in the call to the connected-components algorithm. For the same reason (to be explained below) we need a different estimate of the degree d^*. We use Lemma 4 just once to compute, in $O(dw/\varepsilon)$ time, an estimate $d^* = O(dw/\varepsilon)$ such that at most $\varepsilon n/4w$ vertices have degree higher than d^*.

In the following, we assume that $w/n < 1/2$, since otherwise we might as well compute the MST explicitly, which can be done in $O(dn)$ time with high probability [11].

THEOREM 6. *Let $w/n < 1/2$. Let v be the weight of the MST of G. Algorithm* approx-MST-weight *runs in time $O(dw\varepsilon^{-2} \log \frac{dw}{\varepsilon})$ and outputs a value $\hat{v}$ that, with probability at least $3/4$, differs from v by at most εv.*

Proof. Let $c = \sum_{i=1}^{w-1} c^{(i)}$. Repeating the previous analysis, we find that (1), (2) become

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

By summing over i, it follows that $c - \varepsilon n/2 \leq \mathbf{E}\,\hat{c} \leq c$ and $\mathbf{var}\,\hat{c} \leq nc/r$, where $\hat{c} = \sum_{i=1}^{w-1} \hat{c}^{(i)}$. Choosing $r\varepsilon^2$ large enough, by Chebyshev we have

$$\text{Prob}[\,|\hat{c} - \mathbf{E}\,\hat{c}| > (n - w + c)\varepsilon/3\,] < \frac{9nc}{r\varepsilon^2(n-w+c)^2},$$

which is arbitrarily small. 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, by (4), the expected running time of each call to approx-number-connected-components is $O(dr\log(Wd^*))$, the total expected running time is $O(dw\varepsilon^{-2}\log\frac{dw}{\varepsilon})$. As before, if we know d, then the running time can be made deterministic by stopping execution of the algorithm after $Cdw\varepsilon^{-2}\log\frac{dw}{\varepsilon}$ 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 $1/\varepsilon$ and round each weight to the nearest integer. Then one can run the above algorithm with error parameter $\varepsilon/2$ and with a new range of weights $[1,\lceil w/\varepsilon\rceil]$ to get a value v. Finally, output εv. The relative error introduced by the rounding is at most $\varepsilon/2$ per edge in the MST and hence $\varepsilon/2$ for the whole MST, which gives a total relative error of at most ε. The running time of the above algorithm is $O(dw\varepsilon^{-3}\log\frac{w}{\varepsilon})$.

4. Lower bounds.

We prove that our algorithms for estimating the MST weight and counting connected components are essentially optimal. Our lower bounds apply to graphs that may contain self-loops and multiple edges.

THEOREM 7. *Any probabilistic algorithm for approximating, with relative error ε, the MST weight of a connected graph with average degree d and weights in $\{1,\dots,w\}$ requires $\Omega(dw\varepsilon^{-2})$ 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 might seem restrictive, but it is not at all. Indeed, by monotonicity on ε, the theorem implies a lower bound of $\Omega(dw(C\sqrt{w/n})^{-2})$ for any $\varepsilon\le C\sqrt{w/n}$. But this is $\Omega(dn)$, which we know is tight. Therefore, the case $C\sqrt{w/n}<\varepsilon$ is the only one that deserves attention.

THEOREM 8. *Given a graph with n vertices and average degree d, any probabilistic algorithm for approximating the number of connected components with an additive error of εn requires $\Omega(d\varepsilon^{-2})$ 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. (For technical reasons, we allow graphs to have self-loops.)

Both proofs revolve around the difficulty of distinguishing between two nearby distributions. For any $0<q\le 1/2$ 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 $\{0,1\}^n$ (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 9. *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 $b_1b_2\cdots$ 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 $\geq t \overset{\text{def}}{=} t^*/\alpha$. 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

$$p_{\text{err}} \geq \frac{1}{2} \sum_{k=0}^{t} \min_s p_s(k). \tag{5}$$

By the normal approximation of the binomial distribution,

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

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 $p_{\text{err}} = \Omega(1)$. 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 $1/4$. $\square$

Proof of Theorem 8. 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 $\mathcal{D}_{1/2}^s$. 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 9, this requires $\Omega(\varepsilon^{-2})$ edge probes into G on average. Replacing ε by 2ε proves Theorem 8 for graphs of average degree about 1.

For values of d smaller than one, we may simply build a graph of the previous type on a fraction d of the n vertices and leave the others isolated. The same lower bound still holds as long as $d\varepsilon^2 n$ is bigger than a suitable constant. If $d > 1$, then we may simply add $d \pm O(1)$ self-loops to each vertex in order to bring the average degree up to d. Each linked list thus consists of two "cycle" pointers and about d "loop" pointers. If we place the cycle pointers at random among the loop pointers, 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 $T = \Omega(d\varepsilon^{-2})$. ∎

Proof of Theorem 7. 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 ith 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 $\mathcal{D}^0_q$ and $\mathcal{D}^1_q$ 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 9, this means that $\Omega(w\varepsilon^{-2})$ probes into G are required on average.

In this construction, $d = 2 - 2/n$ (the smallest possible value for a connected graph). For higher values of d, we join each vertex in the cycle to about $d - 2$ others (say, at distance > 2 to avoid introducing multiple edges) to drive the degree up to d. Also, as usual, we 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 for which we are aiming.) 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 $\Omega(dw\varepsilon^{-2})$. ∎

5. Open questions.

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.

Acknowledgments. We wish to thank David Silver for his helpful comments about the paper and for implementing the MST algorithm and showing that it works well in practice. We also thank the referees for many useful suggestions.

REFERENCES

[1] N. Alon, S. Dar, M. Parnas, and D. Ron, *Testing of clustering,* SIAM J. Discrete Math., 16 (2003), pp. 393–417.

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

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

[4] M. L. Fredman and D. E. Willard, *Trans-dichotomous algorithms for minimum spanning trees and shortest paths*, J. Comput. System Sci., 48 (1994), pp. 533–551.

[5] A. Frieze and R. Kannan, *Quick approximation to matrices and applications,* Combinatorica, 19 (1999), pp. 175–220.

[6] A. Frieze, R. Kannan, and S. Vempala, *Fast Monte-Carlo algorithms for finding low-rank approximations*, J. ACM, 51 (2004), pp. 1025–1041.

[7] D. Gale, *Optimal assignments in an ordered set: An application of matroid theory*, J. Combinatorial Theory, 4 (1968), pp. 176–180.

[8] O. Goldreich, S. Goldwasser, and D. Ron, *Property testing and its connection to learning and approximation*, J. ACM, 45 (1998), pp. 653–750.

[9] O. Goldreich and D. Ron, *Property testing in bounded degree graphs*, Algorithmica, 32 (2002), pp. 302–343.

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

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

[12] J. Nešetřil, *A few remarks on the history of MST-problem*, Arch. Math. (Brno), 33 (1997), pp. 15–22.

[13] S. Pettie and V. Ramachandran, *An optimal minimum spanning tree algorithm*, in Automata, Languages and Programming (Geneva, 2000), Lecture Notes in Comput. Sci. 1853, Springer-Verlag, Berlin, 2000, pp. 49–60.

[14] N. Mishra, D. Oblinger, and L. Pitt, *Sublinear time approximate clustering*, in Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2001, pp. 439–447.

[15] D. Ron, *Property testing,* in Handbook of Randomized Computing, Vol. II, S. Rajasekaran, P. M. Pardalos, J. H. Reif, and J. D. P. Rolim, eds., Kluwer Academic Publishers, Dordrecht, The Netherlands, 2001, pp. 597–649.

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