SIAM J. COMPUT.
Vol. 39, No. 3, pp. 904–922

ESTIMATING THE WEIGHT OF METRIC MINIMUM SPANNING TREES IN SUBLINEAR TIME*

ARTUR CZUMAJ† AND CHRISTIAN SOHLER‡

Abstract. In this paper we present a sublinear-time $(1+\varepsilon)$-approximation randomized algorithm to estimate the *weight* of the minimum spanning tree of an n-point metric space. The running time of the algorithm is $\widetilde{\mathcal{O}}(n/\varepsilon^{\mathcal{O}(1)})$. Since the full description of an n-point metric space is of size $\Theta(n^2)$, the complexity of our algorithm is *sublinear* with respect to the input size. Our algorithm is almost optimal as it is not possible to approximate in $o(n)$ time the weight of the minimum spanning tree to within any factor. We also show that no deterministic algorithm can achieve a B-approximation in $o(n^2/B^3)$ time. Furthermore, it has been previously shown that no $o(n^2)$ algorithm exists that *returns a spanning tree* whose weight is within a constant times the optimum.

Key words. randomized algorithms, approximation algorithms, sublinear-time algorithms

AMS subject classifications. 68W20, 68W25, 68W40

DOI. 10.1137/060672121

1. Introduction.

Despite extensive investigations over the last few decades, the complexity of the minimum spanning tree problem is still not completely understood. Although an optimal deterministic algorithm is known [22], no tight bounds on the running time of this algorithm could be obtained. The best upper bound on the running time for a deterministic algorithm was obtained by Chazelle [5], who presented an algorithm that achieves a running time of $\mathcal{O}(|V| + |E|\,\alpha(|E|, |V|))$, where α is the functional inverse of Ackermann's function. In turn, Karger, Klein, and Tarjan [19] gave an optimal $\mathcal{O}(|V| + |E|)$-time randomized algorithm. Much research has been devoted to studying the minimum spanning problem for various classes of graphs and for variants of the problem. For example, the problem of computing the minimum spanning tree of a set of points in a Euclidean space and related problems have been intensively studied (see [10] for a summary of results). Despite this effort, the fastest algorithm to compute such a minimum spanning tree in the $\mathbb{R}^d$ requires $\mathcal{O}(n^{2-2/(\lceil d/2\rceil+1)+\varepsilon})$ time for an arbitrarily small constant ε.

In this paper we present another important step toward understanding the minimum spanning tree problem. We consider the classical variant of the *minimum spanning tree problem for metric spaces* or, equivalently, in graphs with *weights satisfying the triangle inequality*. The input to the problem consists of an n-point metric space (P, d), and the goal is to estimate the *weight* of the minimum spanning tree of P. In this paper, we show that even though the full description of an n-point metric space is of size $\Theta(n^2)$, there exists an algorithm that approximates the weight of the minimum spanning tree of P to within a $(1+\varepsilon)$-factor in time $\widetilde{\mathcal{O}}(n/\varepsilon^{\mathcal{O}(1)})$ (we

*Received by the editors October 12, 2006; accepted for publication (in revised form) May 19, 2009; published electronically August 26, 2009. This research was supported in part by NSF grants CCR-0313219 and CCR-0105701, DFG grant Me 872/8-3, EPSRC grant EP/D063191/1, and by the Centre for Discrete Mathematics and its Applications (DIMAP), University of Warwick. A preliminary version of this paper appeared in *Proceedings of the 36th ACM Symposium on Theory of Computing (STOC)*, ACM Press, New York, 2004, pp. 175–183.
http://www.siam.org/journals/sicomp/39-3/67212.html

†Department of Computer Science and Centre for Discrete Mathematics and its Applications, University of Warwick, Coventry, CV4 7AL, United Kingdom (A.Czumaj@warwick.ac.uk).

‡Department of Computer Science, Technical University of Dortmund, 44221 Dortmund, Germany (christian.sohler@tu-dortmund.de).

use $\widetilde{\mathcal{O}}$ to hide polylogarithmic factors). This is the first *sublinear-time* algorithm for this problem. Our algorithm is randomized, and it achieves the promised approximation guarantee with the probability of at least 3/4 (using standard techniques, the probability can be amplified if needed). Furthermore, it was previously shown in [18] that no $o(n^2)$ algorithm exists that *returns a spanning tree* whose weight is within any constant times the weight of the minimum spanning tree of P. Therefore, our result shows that one can approximate the *weight* of the minimum spanning tree, but there is no hope of finding a *witness* for that approximation in sublinear time. Our running time is essentially asymptotically optimal, because it is easy to show that no $o(n)$-time algorithm exists that approximates the weight of the minimum spanning tree within *any* factor. Also, randomization is essential for the algorithm. We show that no deterministic B-approximation algorithm with $o(n^2/B^3)$ running time exists.

By the well-known relationship between minimum spanning trees, travelling salesman tours, and minimum Steiner trees (see, e.g., [23]), our algorithm for estimating the weight of the minimum spanning tree immediately yields *sublinear-time* $(2+\varepsilon)$-approximation algorithms for two other classical problems in metric spaces (or in graphs satisfying the triangle inequality): *estimating the weight of the travelling salesman tour* and *the minimum Steiner tree*. No $o(n^2)$-time algorithms were known for these problems before. We believe that besides being interesting by themselves, these approximation results may find applications in bounding the quality of solutions in subproblems used in branch and cut (bound) algorithms to compute the exact solution to these problems.

Our algorithms enlarge the ever growing list of problems solvable in sublinear time that are often required for the analysis of massive data sets. Due to the tremendous increase in computational power and interconnectivity during the last decade, more and more often we have to deal with *massive* data sets, which are sets of size in the range of several Gigabytes or more. Examples for such massive data sets are Internet traffic logs, clickstream patterns, sales logs, and call-detail data records in the telecommunications industry. Massive data sets typically cannot be processed by algorithms requiring more than linear time, and often even linear-time algorithms may be too slow. This leads to a natural question of which problems of interest (if any) can be solved in *sublinear time*. Some simple results indicating that it is sometimes possible to solve certain approximation problems in sublinear time are well known, for example, approximating the median or the average value of a set of n numbers. More sophisticated results have been obtained in the last few years for a number of more complex problems, including clustering problems in metric spaces [1, 6, 17, 18, 20], graph problems [7, 12, 14, 15, 21], geometric problems [6, 8], matrix approximation [13], and string problems [2, 3, 11]; see also the recent survey in [9].

1.1. Related work. The problem of approximating the weight of the minimum spanning tree in sublinear time was first addressed for arbitrary graphs in *adjacency list* representation in a recent paper by Chazelle, Rubinfeld, and Trevisan [7]. In [7], a sublinear algorithm is given, whose running time is independent of the size of the input graph: If the maximum (or average) degree is D and if all edge weights are known to be in the interval $[1, W]$, then the algorithm approximates the weight of the minimum spanning tree to within a $(1+\varepsilon)$-factor in time $\widetilde{\mathcal{O}}(D \cdot W \cdot \varepsilon^{-3})$ with probability at least $\frac{3}{4}$. However, if $W = \Omega(n)$, then no sublinear algorithm is known.

If we apply their algorithm to the metric version of the problem, then the running time is $\mathcal{O}(n \cdot W \cdot \varepsilon^{-3})$ because $D = n-1$. Therefore, in our setting, their algorithm is sublinear only if the ratio between the longest and the shortest edge W is sublinear

in n, which certainly does not have to be the case in general (for example, even in the case when (P,d) corresponds to the set of points P in a Euclidean plane, it is known that W must be at least $\Omega(\sqrt{n})$ and hence the running time is $\Omega(n^{1.5}\cdot\varepsilon^{-3})$).

In [8], the authors consider the problem of estimating the weight of the Euclidean minimum spanning tree of a set P of n points in the $\mathbb{R}^d$. They consider the model in which the input point set is stored in a sophisticated data structure that supports two types of access operations, namely, (i) emptiness queries for axis parallel squares and (ii) approximate nearest neighbor queries for a set of prespecified cones. Additionally, it is assumed that a smallest axis parallel bounding box of P is given. In this model the authors give an algorithm that approximates the weight of the Euclidean minimum spanning tree of P within a relative error of ε. The algorithm has a running time of $\widetilde{\mathcal{O}}(\sqrt{n}/\varepsilon^{\mathcal{O}(1)})$ assuming that the dimension is a constant and assuming that all access operations to the data structure can be performed in $\log^{O(1)} n$ time. The algorithm uses $\widetilde{\mathcal{O}}(\sqrt{n}/\varepsilon^{\mathcal{O}(1)})$ queries of types (i) and (ii).

1.2. New contribution. In contrast to both of the aforementioned algorithms, our algorithm does not make any assumption about the input other than that we can evaluate the distance between any two points in the metric space in constant time.

The high level approach of our algorithm is similar to that in [7]. We regard the metric space as a complete graph and we express the weight of its minimum spanning tree by a formula depending on the number of connected components in certain auxiliary subgraphs. In contrast to that of [7], our approach builds on geometric rather than arithmetic progression. We will assume that the longest edge in (P,d) has length $W=(1+\varepsilon)^r$ for $r=\lceil\log_{1+\varepsilon}(4n/\varepsilon)\rceil$ (as we will see later, this assumption does not affect the complexity of the problem). We denote by $G^{(i)}=(P,E^{(i)})$ the graph that contains an edge between $p,q\in P$ if $d(p,q)\le(1+\varepsilon)^i$. We use a randomized procedure to approximate the number $c^{(i)}$ of connected components in each subgraph $G^{(i)}$. Using the approximation $\text{MST}\approx n-W+\varepsilon\cdot\sum_{i=0}^{\log_{1+\varepsilon}W-1}(1+\varepsilon)^i\cdot c^{(i)}$ (see Corollary 2.2), we obtain an estimator for the weight of the minimum spanning tree of P.

The advantage of geometric progression is that we have to estimate only $\mathcal{O}(\log W)$ times the number of the connected components of a certain threshold graph in contrast to W times in [7]. We achieve this at the cost of an increased variance of the estimator (which makes it impossible to apply this approach to arbitrary graphs considered in [7]). Instead of approximating the number of connected components within an additive error of $\varepsilon\, n$, as in [7], we obtain an approximation with additive error of $\varepsilon\cdot\text{weight}(\text{MST})$; i.e., we relate the error directly to the weight of the minimum spanning tree.

Our analysis explores the triangle inequality to ensure trade-offs between the running time, the number of connected components, the vertex degrees, and the size of the minimum spanning tree.

2. Preliminaries. We consider the problem of *estimating the weight of the minimum spanning tree in a metric space*: Given access to the $n\times n$ distance matrix of a metric space (P,d), $|P|=n$, the goal is to approximate the weight of the minimum spanning tree of P. Throughout the paper, we denote by MST the weight of the minimum spanning tree of P. Our main contribution is an algorithm that in $\widetilde{\mathcal{O}}(n/\varepsilon^7)$ time computes a $(1+\varepsilon)$-approximation of MST, i.e., outputs a value M such that $\text{MST}\le M\le(1+\varepsilon)\cdot\text{MST}$ with probability at least $\frac{3}{4}$. For convenience, in the remainder of the paper we will only require the output value M to be in the interval $[(1-\varepsilon)\cdot\text{MST},\,(1+\varepsilon)\cdot\text{MST}]$. We can always achieve the former bound by multiply-

ing the output by $1/(1-\varepsilon)$ and replacing ε by $\varepsilon' = \min\{\varepsilon/4, \frac{1}{2}\}$. Because of this transformation, we will refer to an algorithm that computes a value M in the interval $[(1-\varepsilon)\cdot \text{MST}, (1+\varepsilon)\cdot \text{MST}]$ as a $(1+\varepsilon)$-approximation algorithm for the weight of the minimum spanning tree. We will use ε as the approximation parameter throughout the paper.

Our result holds for arbitrary metrics (P, d) and assumes only that a constant-time access to the distance oracle is provided. For our analysis we will also assume that for every pair $p, q \in P$ we have $d(p,q) \in [1, (1+\varepsilon)^r]$, where $r = \lceil \log_{1+\varepsilon}(4n/\varepsilon) \rceil$; i.e., the distance between any pair of points is at most $4n/\varepsilon$ rounded up to the nearest power of $(1+\varepsilon)$. As we will show below, such an assumption may introduce an additive approximation error of at most $\frac{\varepsilon}{2}\cdot \text{MST}$, which will not affect the final result. We use a result of Indyk [18], who showed that in $\mathcal{O}(n)$ time one can approximate to within a factor $\frac{1}{2}$ the longest distance in a metric space. (The algorithm picks an arbitrary vertex and takes the longest incident edge. By the triangle inequality, this edge is at least half as long as the longest edge in the metric space.) Once we have such an approximation W^*, we can rescale the distances such that $W^* = 2\cdot n/\varepsilon$. Since W^* is a $\frac{1}{2}$-approximation of the largest distance, after the scaling all distances are in $[0, 4\,n/\varepsilon]$. Furthermore, by the triangle inequality the weight of a minimum spanning tree of a metric space is at least as large as the weight of the longest distance, and since the longest distance is at least W^*, we have $\text{MST} \geq \frac{2n}{\varepsilon}$. Next, we observe that rounding up every distance smaller than 1 to the distance 1 will change MST by an additive term of at most $n-1$ while preserving the triangle inequality. Since $n-1 \leq \frac{\varepsilon}{2}\cdot \text{MST}$, in the so modified metric the weight of a minimum spanning tree is a $(1+\frac{\varepsilon}{2})$-approximation of the weight of a minimum spanning tree in the original metric. Hence, we can assume that all edges are in $[1, W^*]$. To simplify the exposition we increase the upper bound W^* to the nearest power of $(1+\varepsilon)$; i.e., we use $W = (1+\varepsilon)^r$. In the remainder of the paper, we will show how to obtain a $(1+\varepsilon)$-approximation algorithm under our assumption. Replacing ε by $\varepsilon' = \varepsilon/4$, this gives an algorithm with approximation guarantee $(1+\varepsilon/2)\cdot(1+\varepsilon/4) \leq (1+\varepsilon)$ without changing the asymptotic running time of our algorithm.

2.1. Approximating MST via counting connected components in auxiliary graphs. Our high level approach to approximating the weight of the minimum spanning tree is similar to the one used in [7]. We express the weight of the minimum spanning tree in terms of the number of connected components in certain auxiliary graphs. While the formula from [7] uses arithmetic progression, our formula will be based on geometric progression. Then we show how to approximate the number of connected components.

For a given parameter $i \in \mathbb{N}$, we define a *threshold graph* $G^{(i)} = (P, E^{(i)})$, where $(p,q) \in E^{(i)}$ if and only if $d(p,q) \leq (1+\varepsilon)^i$. We call the connected components of $G^{(i)}$ the *i-connected components* and denote their number by $c^{(i)}$.

Let us first consider the case that the lengths of all edges in the metric space are powers of $(1+\varepsilon)$. Then the next lemma can be obtained in a similar way as in [7] by replacing arithmetic progression with geometric progression.

LEMMA 2.1. *Let (P,d) be an n-point metric space such that for any pair of points $p, q \in P$ we have $d(p,q) = (1+\varepsilon)^i$ for some $i \in \{0, \ldots, r\}$. Let $W = (1+\varepsilon)^r$. Then we can write*

$$\text{MST} \;=\; n - W + \varepsilon \cdot \sum_{i=0}^{r-1} (1+\varepsilon)^i \cdot c^{(i)}. \tag{2.1}$$

Proof. Let us denote by $n(j)$ the number of edges of weight $(1+\varepsilon)^j$ in a minimum spanning tree. Then, we can write the weight of the minimum spanning tree as

$$\begin{aligned}
\text{MST} &= \sum_{j=0}^{r} (1+\varepsilon)^j \cdot n(j) \\
&= \sum_{j=0}^{r} \left(1 + \varepsilon \cdot \frac{(1+\varepsilon)^j - 1}{\varepsilon}\right) \cdot n(j) \\
&= (n-1) + \varepsilon \cdot \sum_{j=0}^{r} \frac{(1+\varepsilon)^j - 1}{\varepsilon} \cdot n(j) \\
&= (n-1) + \varepsilon \cdot \sum_{j=1}^{r} \frac{(1+\varepsilon)^j - 1}{\varepsilon} \cdot n(j) \\
&= (n-1) + \varepsilon \cdot \sum_{j=1}^{r} \sum_{i=0}^{j-1} (1+\varepsilon)^i \cdot n(j) \\
&= (n-1) + \varepsilon \cdot \sum_{i=0}^{r-1} \sum_{j=i+1}^{r} (1+\varepsilon)^i \cdot n(j) \\
&= (n-1) + \varepsilon \cdot \sum_{i=0}^{r-1} (1+\varepsilon)^i \cdot \sum_{j=i+1}^{r} n(j).
\end{aligned}$$

Now we use the observation that $\sum_{j=i+1}^{r} n(j) = c^{(i)} - 1$. Hence,

$$\begin{aligned}
\text{MST} &= (n-1) + \varepsilon \cdot \sum_{i=0}^{r-1} (1+\varepsilon)^i \cdot (c^{(i)} - 1) \\
&= (n-1) + \varepsilon \cdot \sum_{i=0}^{r-1} (1+\varepsilon)^i \cdot c^{(i)} - \varepsilon \cdot \sum_{i=0}^{r-1} (1+\varepsilon)^i \\
&= (n-1) + \varepsilon \cdot \sum_{i=0}^{r-1} (1+\varepsilon)^i \cdot c^{(i)} - ((1+\varepsilon)^r - 1) \\
&= n - W + \varepsilon \cdot \sum_{i=0}^{r-1} (1+\varepsilon)^i \cdot c^{(i)}. \qquad \square
\end{aligned}$$

COROLLARY 2.2. *Let (P,d) be an n-point metric space such that all pairwise distances are in the interval $[1,W]$, where $W = (1+\varepsilon)^r$. Then we have*

$$\text{MST} \le n - W + \varepsilon \cdot \sum_{i=0}^{r-1} (1+\varepsilon)^i \cdot c^{(i)} \le (1+\varepsilon) \cdot \text{MST}. \tag{2.2}$$

Proof. Let us consider an arbitrary pair of points p, q with $(1+\varepsilon)^i < d(p,q) \le (1+\varepsilon)^{i+1}$. Then we have $(p,q) \notin E^{(k)}$ for $k \in \mathbb{N}$ with $k \le i$, and $(p,q) \in E^{(j)}$ for $j \in \mathbb{N}$ with $j > i$. However, the same property would also hold if the edge weight $d(p,q)$ were exactly $(1+\varepsilon)^{i+1}$. Hence, by rounding up every edge weight to the nearest power of $(1+\varepsilon)$, we do not change the sets of edges in any of the threshold graphs. Thus $c^{(i)}$ is also not affected by this transformation. Let G_P be the weighted graph

obtained by rounding up the edge weights to the nearest power of $(1+\varepsilon)$ (G_P is not necessarily a metric space, since the triangle inequality may become invalid after the rounding). By Lemma 2.1, the weight of the minimum spanning tree of G_P is exactly $n - W + \varepsilon \cdot \sum_{i=0}^{r-1} (1+\varepsilon)^i \cdot c^{(i)}$. Since we increased only edge weights, the weight of the minimum spanning tree of G_P is larger than that of (P, d), and it follows that $\text{MST} \leq n - W + \varepsilon \cdot \sum_{i=0}^{r-1} (1+\varepsilon)^i \cdot c^{(i)}$. Since we increased every edge weight by a factor of at most $(1+\varepsilon)$, the weight of the minimum spanning tree is increased by at most a factor of $(1+\varepsilon)$. This yields $n - W + \varepsilon \cdot \sum_{i=0}^{r-1} (1+\varepsilon)^i \cdot c^{(i)} \leq (1+\varepsilon) \cdot \text{MST}$, which proves the corollary. ∎

Our algorithm is based on computing a randomized estimator $\hat{c}^{(i)}$ for each $c^{(i)}$. Using this estimator, we can now phrase our randomized algorithm.

```
METRIC-MST-APPROXIMATION (P, ε)
    for i = 0 to r − 1 do
        Compute estimator ĉ^(i) for c^(i)
    Output M = n − W + ε · Σ_{i=0}^{r−1} (1 + ε)^i · c^(i)
```

Our main contribution is a sublinear-time randomized algorithm that outputs an estimator $\hat{c}^{(i)}$ that with probability at least $1 - \frac{1}{4r}$ satisfies the following property:

$$c^{(i+1)} - \frac{1}{r} \cdot \frac{\text{MST}}{(1+\varepsilon)^i} \;\leq\; \hat{c}^{(i)} \;\leq\; c^{(i)} + \frac{1}{r} \cdot \frac{\text{MST}}{(1+\varepsilon)^i}. \tag{2.3}$$

It follows that with probability at least $\frac{3}{4}$ all estimators $\hat{c}^{(i)}$ simultaneously satisfy inequality (2.3). Observe now that we have

$$\begin{aligned}
M &= n - W + \varepsilon \cdot \sum_{i=0}^{r-1} (1+\varepsilon)^i \cdot \hat{c}^{(i)} \\
&\geq n - W + \varepsilon \cdot \sum_{i=0}^{r-1} (1+\varepsilon)^i \cdot \left(c^{(i+1)} - \frac{1}{r} \cdot \frac{\text{MST}}{(1+\varepsilon)^i} \right) \\
&= n - W - \varepsilon \cdot \text{MST} + \varepsilon \cdot \sum_{i=0}^{r-1} (1+\varepsilon)^i \cdot c^{(i+1)} \\
&\geq n - W - \varepsilon \cdot \text{MST} + \frac{1}{1+\varepsilon} \cdot \varepsilon \cdot \sum_{i=1}^{r} (1+\varepsilon)^i \cdot c^{(i)}.
\end{aligned}$$

Since $c^{(0)} \leq n$, we obtain

$$\begin{aligned}
M &\geq \left(1 - \frac{\varepsilon}{1+\varepsilon}\right) \cdot n - W - \varepsilon \cdot \text{MST} + \frac{1}{1+\varepsilon} \cdot \varepsilon \cdot \sum_{i=0}^{r-1} (1+\varepsilon)^i \cdot c^{(i)} \\
&\geq \left(1 - \frac{\varepsilon}{1+\varepsilon}\right) \cdot \left(n - W + \varepsilon \cdot \sum_{i=0}^{r-1} (1+\varepsilon)^i \cdot c^{(i)} \right) - \varepsilon \cdot \text{MST} \\
&\geq (1 - 2\varepsilon) \cdot \text{MST}.
\end{aligned}$$

From Corollary 2.2, we obtain

$$M \leq n - W + \varepsilon \cdot \sum_{i=0}^{r-1} \left((1+\varepsilon)^i \cdot \left(c^{(i)} + \frac{1}{r} \cdot \frac{\text{MST}}{(1+\varepsilon)^i} \right) \right)$$

$$\leq n - W + \varepsilon \cdot \text{MST} + \varepsilon \cdot \sum_{i=0}^{r-1} (1+\varepsilon)^i \cdot c^{(i)}$$
$$\leq (1+2\varepsilon) \cdot \text{MST}.$$

Hence, we have with probability at least 3/4

$$(1-2\varepsilon) \cdot \text{MST} \;\leq\; M \;\leq\; (1+2\varepsilon) \cdot \text{MST}.$$

In sections 3–7 we describe details of our randomized algorithm that in $\widetilde{\mathcal{O}}(n/\varepsilon^6)$ time computes the estimator $\hat{c}^{(i)}$ that satisfies inequality (2.3). Replacing ε by $\varepsilon/2$ will conclude the proof of our main theorem.

THEOREM 2.3. *Let $0 < \varepsilon < 1$ be an approximation parameter. Given access to the $n \times n$ distance matrix of a metric space (P, d), $|P| = n$, algorithm* METRIC-MST-APPROXIMATION *computes in $\widetilde{\mathcal{O}}(n/\varepsilon^7)$ time a value M such that, with probability $\frac{3}{4}$,*

$$(1-\varepsilon) \cdot \text{MST} \;\leq\; M \;\leq\; (1+\varepsilon) \cdot \text{MST}.$$

We remark that this result is almost optimal since it is easy to see that any constant-factor algorithm requires time $\Omega(n)$ even in a randomized setting (see Theorem 8.1 in section 8). Also, no deterministic algorithm can compute a constant-factor approximation for the weight of a metric minimum spanning tree in $o(n^2)$ time (see section 8), and no algorithm can compute a spanning tree that approximates the weight of the minimum spanning tree within a constant factor in $o(n^2)$ time [18].

3. Estimating the number $c^{(i)}$ of i-connected components: Main ideas. In this section we recall the approach taken by [7] to approximate the number of connected components in a graph (not necessarily a metric space). The algorithm repeats the following procedure until a certain threshold value is reached, to ensure that the estimation of the number of i-connected components is with high probability close to $c^{(i)}$:

- Pick a starting vertex $p \in P$ uniformly at random.
- Choose a random integer number X according to the probability distribution $\mathbf{Pr}[X \geq k] = 1/k$.
- Verify whether the connected component in $G^{(i)}$ containing vertex p has at most X vertices or has more than X vertices.

With the exception of a minor modification in the probability distribution of X, the scheme above has been proposed by Chazelle, Rubinfeld, and Trevisan [7]. We will run this procedure multiple times, and in each repetition of this procedure we output β_j, that is, the indicator random variable of the event that in the jth trial, the connected component has at most X vertices. That is, if we denote by $n_p^{(i)}$ *the size of the connected component in $G^{(i)}$ containing vertex p*, then $\beta_j = 1$ if $n_p^{(i)} \leq X$ and $\beta_j = 0$ otherwise. Notice that

$$\mathbf{E}[\beta_j] = \sum_{\text{connected component } C \text{ in } G^{(i)}} \mathbf{Pr}[p \in C] \cdot \mathbf{Pr}[X \geq |C|]$$
$$= \sum_{\text{connected component } C \text{ in } G^{(i)}} \frac{|C|}{n} \cdot \frac{1}{|C|} \;=\; \frac{c^{(i)}}{n}.$$

Therefore, if there are s repetitions of the procedure above, then we define

$$\hat{c}^{(i)} \;=\; \frac{n}{s} \cdot \sum_{j=1}^{s} \beta_j.$$

Since by the arguments above $\mathbf{E}[\hat{c}^{(i)}] = c^{(i)}$, this motivates the use of $\hat{c}^{(i)}$ as an estimator of the number of connected components; see [7]. The challenging part of completing the analysis is to show that the random variable $\hat{c}^{(i)}$ is sharply concentrated around its expectation *and* to show that it can be computed efficiently. In particular, our algorithm will use a nonstandard graph traversal that exploits the triangle inequality combined with some randomized algorithm to check whether the degree of the starting vertex is higher than X (and hence the connected component contains more than X vertices). Unfortunately, the expected value of our estimator is not exactly $c^{(i)}$. However, we will show that it is sufficiently close to $c^{(i)}$. Moreover, we will develop some concentration bounds for $\hat{c}^{(i)}$ that satisfy the bounds in (2.3).

4. The Clique-Tree-Traversal. Our method for verifying whether a given connected component in $G^{(i)}$ has at most X vertices is to traverse the graph $G^{(i)}$ starting at vertex p. Chazelle, Rubinfeld, and Trevisan [7] used the standard breadth-first search (BFS) traversal algorithm for this purpose. However, in our setting a BFS will not suffice to ensure both near-linear running time *and* sufficient concentration. We have to develop a new traversal algorithm that exploits the triangle inequality.

Our graph traversal will not necessarily explore the connected components in $G^{(i)}$. If we denote by $V_p^{(i)}$ the set of vertices in the connected component of p in graph $G^{(i)}$, then the graph traversal starting at p will visit a set of vertices V_{exp} such that $V_p^{(i)} \subseteq V_{exp} \subseteq V_p^{(i+1)}$. We will see later in the analysis that this does not significantly affect our estimator. We now describe our graph traversal algorithm in detail.

At the beginning all vertices are *unexplored*. Then the starting vertex p is marked as *explored* and *representative*. In the next step, all neighbors of p that are in distance less than $\varepsilon \cdot (1+\varepsilon)^i$ are marked as *explored*. Then we proceed similarly to Prim's algorithm for the computation of minimum spanning trees: We choose the shortest of the edges in $E^{(i+1)}$ that connect a representative vertex with an unexplored vertex. This leads us to a new vertex that is again chosen to be *explored* and *representative*. Then, we repeat all steps above until no further point can be explored. We call this graph traversal the Clique-Tree-Traversal.

We give a pseudocode for the Clique-Tree-Traversal below. The sets V_{exp}, V_{unexp}, and V_{rep} denote the sets of explored, unexplored, and representative vertices, respectively.

Clique-Tree-Traversal $(P,\ p,\ i,\ \varepsilon)$
$V_{rep} = \{p\}$; $V_{exp} = \{p\}$; $V_{unexp} = P \setminus \{p\}$
while there is an edge $e = (p,q) \in E^{(i+1)}$ with $p \in V_{rep}$ and $q \in V_{unexp}$ **do**
  let (p,q) be the shortest such edge
  $V_{exp} = V_{exp} \cup \{q\}$; $V_{unexp} = V_{unexp} \setminus \{q\}$
  if $d(p,q) \geq \varepsilon\,(1+\varepsilon)^i$ **then** $V_{rep} = V_{rep} \cup \{q\}$

Properties of the Clique-Tree-Traversal. First, it is easy to see that the Clique-Tree-Traversal algorithm can be implemented to run in $\widetilde{O}(n \cdot |V_{rep}|)$ time, where here and in the following we use V_{rep} and V_{exp} to refer to their final value in an execution of the Clique-Tree-Traversal. Also, we have that (for $\varepsilon \leq \frac{1}{2}$) the points that are assigned to the same representative vertex form a clique in $G^{(i)}$. This property will be important in the analysis of our algorithm. It may be helpful to think of the set of vertices explored by the Clique-Tree-Traversal as being the

connected component in $G^{(i)}$ that contains p (as we have already discussed, this is only approximately true, but it gives a good intuition).

Since we are considering edges incident to V_{rep} in increasing order of length, we make sure that all vertices in V_{rep} have pairwise distance at least $\varepsilon \cdot (1+\varepsilon)^i$. This will be used later to obtain a lower bound on the cost of the minimum spanning tree of G.

Finally, we show that $V_p^{(i)} \subseteq V_{exp} \subseteq V_p^{(i+1)}$. $V_{exp} \subseteq V_p^{(i+1)}$ follows immediately from the fact that the algorithm uses only edges from $E^{(i+1)}$ to explore the graph. To show $V_p^{(i)} \subseteq V_{exp}$, let us consider an arbitrary vertex $q \in V_p^{(i)}$ and let us assume that q is not explored by the algorithm. Since q is in $V_p^{(i)}$, it is connected to p by a sequence of edges $p = p_0, p_1, \ldots, p_k = q$ such that $d(p_j, p_{j+1}) \leq (1+\varepsilon)^i$ for all $0 \leq j < k$. Let ℓ denote the smallest index such that p_ℓ is not explored by the algorithm. Thus, we know that $p_{\ell-1}$ is explored. If $p_{\ell-1}$ is a representative vertex, then p_ℓ is explored from $p_{\ell-1}$, which is a contradiction to the choice of ℓ. Thus, $p_{\ell-1}$ is no representative vertex. But then there is a representative vertex w with $d(w, p_{\ell-1}) < \varepsilon(1+\varepsilon)^i$. By the triangle inequality we have $d(w, p_\ell) \leq d(w, p_{\ell-1}) + d(p_{\ell-1}, p_\ell) < \varepsilon(1+\varepsilon)^i + (1+\varepsilon)^i = (1+\varepsilon)^{i+1}$. But this implies that p_ℓ is explored from w, which contradicts the choice of ℓ and hence our assumption that q is not explored.

We summarize our discussion in the following lemma.

LEMMA 4.1. *The algorithm* CLIQUE-TREE-TRAVERSAL *satisfies the following properties. V_{rep} and V_{exp} refer to the final value of these sets, i.e., their value at the end of an execution of the* CLIQUE-TREE-TRAVERSAL.

(1) *The algorithm can be implemented to run in time $\mathcal{O}(n \cdot \log n \cdot |V_{rep}|)$.*
(2) *For $\varepsilon < \frac{1}{2}$, the points assigned to the same representative point form a clique in $G^{(i)}$.*
(3) *$V_p^{(i)} \subseteq V_{exp} \subseteq V_p^{(i+1)}$.*
(4) *The vertices in V_{rep} have pairwise distance at least $\varepsilon \cdot (1+\varepsilon)^i$.* ∎

In the analysis of our algorithm we will also use the notion of *graph dispersion*. To define this notion, let us first extend the CLIQUE-TREE-TRAVERSAL to a full graph traversal in the following natural way: We start with an arbitrary vertex p and run the CLIQUE-TREE-TRAVERSAL with parameters (P, p, i, ε). If not all vertices are explored at the end of this traversal, we start the CLIQUE-TREE-TRAVERSAL from one of the unexplored vertices (we never start at the same connected component more than once). We do this until every vertex has been explored. We call this process the *full* CLIQUE-TREE-TRAVERSAL.

It is easy to see that the number of representative vertices computed by the full CLIQUE-TREE-TRAVERSAL may depend on the starting vertices. Let $U_{rep}^{(i)}$ be a maximum cardinality set of representative vertices computed by the full CLIQUE-TREE-TRAVERSAL for given P, i, and ε (the maximum is taken over all possible vertex orderings). One parameter of particular interest for our analysis is the *dispersion* of the graph $G^{(i)}$, which is defined as $\mathcal{L}(G^{(i)}) = |U_{rep}^{(i)}|$; i.e., $\mathcal{L}(G^{(i)})$ is the maximum number of representative vertices computed by the full CLIQUE-TREE-TRAVERSAL for given P, i, and ε. We will use the dispersion of $G^{(i)}$ together with property (2) to obtain bounds on the density of $G^{(i)}$. Furthermore, our main use of $\mathcal{L}(G^{(i)})$ is to obtain a lower bound for MST as stated in the next lemma.

LEMMA 4.2. $\text{MST} \geq \varepsilon \cdot (1+\varepsilon)^i \cdot \mathcal{L}(G^{(i)})/4$.

Proof. By our initial transformation we have $\text{MST} \geq 2n/\varepsilon$. Also, we have $i \leq r = \lceil \log_{1+\varepsilon}(4n/\varepsilon) \rceil$. If $\mathcal{L}(G^{(i)}) = 1$, then we have $\varepsilon \cdot (1+\varepsilon)^i/4 \leq \varepsilon \cdot (1+\varepsilon)^r/4 \leq \varepsilon(1+\varepsilon) \cdot 4n/(4\varepsilon) \leq \varepsilon \cdot (1+\varepsilon) \cdot \text{MST}/2 \leq \text{MST}$ for $\varepsilon \leq 1$.

Next we consider the case $\mathcal{L}(G^{(i)}) > 1$. Representative vertices in distinct connected components have distance more than $(1+\varepsilon)^{i+1}$. This, together with Lemma 4.1(4), implies that all vertices in $U_{rep}^{(i)}$ have pairwise distance at least $\varepsilon \cdot (1+\varepsilon)^i$. Hence, a minimum spanning tree of the subgraph of G induced by $U_{rep}^{(i)}$ has cost at least $(\varepsilon \cdot (1+\varepsilon)^i) \cdot (|U_{rep}^{(i)}| - 1) \geq (\varepsilon \cdot (1+\varepsilon)^i) \cdot \mathcal{L}(G^{(i)})/2$ if $\mathcal{L}(G^{(i)}) > 1$. It is known that this minimum spanning tree is a factor 2 approximation of the minimum Steiner tree of $U_{rep}^{(i)}$ (cf. [23]). Since a minimum Steiner tree of $U_{rep}^{(i)}$ has cost that is at most the cost of a minimum spanning tree of G, the lemma follows. □

5. Estimating the number of connected components: First attempt. We now discuss a first version of our algorithm Number-of-Connected-Components $(P,\ i,\ \varepsilon)$. The algorithm combines the sampling approach from [7] with the graph traversal algorithm Clique-Tree-Traversal described in section 4. It is identical to our final algorithm except for two modifications which with sufficiently high probability affect only the running time of the algorithm. Also, in this version we assume that we know the dispersion of the graph in order to determine the number of required iterations to achieve a sharp concentration of the estimator. This assumption will be removed in the final algorithm. The algorithm uses the value $r = \lceil \log_{1+\varepsilon}(4n/\varepsilon) \rceil$ defined earlier as a bound on the logarithm of the maximum edge length. It also requires a bound T on the number of iterations.

Number-of-Connected-Components-Version-1 $(P,\ i,\ T,\ \varepsilon)$
$s = 0$
while $s \leq T$ **do**
  $s = s + 1$; $\beta_s = 0$
  choose a vertex p_s independently and uniformly at random
  choose a random integer X according to $\mathbf{Pr}[X \geq k] = 1/k$
  run Clique-Tree-Traversal (P, p_s, i, ε) until one of the following events happens:
    (1) more than X vertices are explored
    (2) more than $\frac{4r}{\varepsilon}$ representative vertices are explored
    (3) the entire connected component in $G^{(i)}$ containing p_s is explored
  if event (3) happens **then** $\beta_s = 1$
output $\hat{c}^{(i)} = \frac{n}{s} \cdot \sum_{j=1}^{s} \beta_j$

We first show that the expected value of $\hat{c}^{(i)}$ is close to $c^{(i)}$.

Lemma 5.1 (expectation bound). *The random variable $\hat{c}^{(i)}$ computed in algorithm* Number-of-Connected-Components-Version-1 *satisfies the following:*

$$c^{(i)} \ \geq\ \mathbf{E}\big[\hat{c}^{(i)}\big] \ \geq\ c^{(i+1)} - \frac{1}{2r} \cdot \frac{\text{MST}}{(1+\varepsilon)^i}.$$

Proof. The analysis from section 3 says that if we ignore events of type (2) and if Clique-Tree-Traversal (P, p, i, ε) always explores exactly $V_p^{(i)}$ (like a BFS), then the expected value of $\hat{c}^{(i)}$ is exactly $c^{(i)}$. Now we observe that the fact that the Clique-Tree-Traversal may explore not only vertices in $V_p^{(i)}$ but possibly other vertices (see Lemma 4.1) can only lower the expected value of the β_j and hence that of $\hat{c}^{(i)}$. Also, the events of type (2) only reduce the expected value of $\hat{c}^{(i)}$. Hence, the inequality $c^{(i)} \geq \hat{c}^{(i)}$ follows.

Now, we prove the second inequality, namely, $\mathbf{E}[\hat{c}^{(i)}] \geq c^{(i+1)} - \frac{1}{2\,r} \cdot \frac{\text{MST}}{(1+\varepsilon)^i}$. We partition the $(i+1)$-connected components in P into two types. An $(i+1)$-connected component C is of type (I) if there *exists* a vertex $p \in C$ such that the Clique-Tree-Traversal with starting vertex p stops with more than $4r/\varepsilon$ representative vertices. Otherwise, an $(i+1)$-connected component is of type (II).

Let K be the number of connected components of type (I). Then $\text{MST} \geq K \cdot \frac{\varepsilon\,(1+\varepsilon)^i}{2} \cdot \frac{4r}{\varepsilon} = 2\,K\,r\,(1+\varepsilon)^i$, and hence $K \leq \frac{1}{2\,r} \cdot \frac{\text{MST}}{(1+\varepsilon)^i}$. Now observe that, given an arbitrary $(i+1)$-connected component of type (II), if X is at least the number of vertices in that component, then our algorithm will always output $\beta_j = 1$. This implies

$$\mathbf{E}[\beta_j] \;\geq \sum_{\text{type (II) connected component } C} \mathbf{Pr}[p_i \in C] \cdot \mathbf{Pr}[X \geq |C|] \;=\; \frac{c^{(i+1)} - K}{n}.$$

Hence,

$$\mathbf{E}[\hat{c}^{(i)}] \;\geq\; c^{(i+1)} - K \;\geq\; c^{(i+1)} - \frac{\text{MST}}{2\,r\,(1+\varepsilon)^i}. \qquad \square$$

Our next step is to show that the number of iterations of the algorithm suffices to achieve sharp concentration around the expectation of $\hat{c}^{(i)}$. We parametrize the additive error in terms of $\mathcal{L}(G^{(i)})$. Using the lower bound on MST in terms of $\mathcal{L}(G^{(i)})$ (cf. Lemma 4.2), this bound can be easily transformed into a relative error in terms of the cost of the minimum spanning tree.

Lemma 5.2 (concentration bound). *If $T \geq \frac{2^{10} \cdot n \cdot r^2}{\varepsilon^2 \cdot \mathcal{L}(G^{(i)})}$, then for the random variable $\hat{c}^{(i)}$ computed by algorithm* Number-of-Connected-Components-Version-1 *the following bound holds:*

$$\mathbf{Pr}\left[\left|\hat{c}^{(i)} - \mathbf{E}\big[\hat{c}^{(i)}\big]\right| \leq \frac{\varepsilon}{8\,r} \cdot \mathcal{L}(G^{(i)})\right] \;\geq\; \frac{15}{16}.$$

Proof. Similarly to [7], we can provide an upper bound for the variance of any single β_i as follows:

$$\mathbf{Var}[\beta_i] \;\leq\; \mathbf{E}\big[\beta_i^2\big] \;\leq\; \mathbf{E}\big[\beta_i\big] \;\leq\; \frac{c^{(i)}}{n},$$

where the first inequality uses the fact that $0 \leq \beta_i \leq 1$ and the last inequality follows from our analysis in the proof of Lemma 5.1 and from section 3. Therefore, we obtain the following inequality:

$$\mathbf{Var}[\hat{c}^{(i)}] \;=\; \left(\frac{n}{T}\right)^2 \cdot \sum_{1 \leq i \leq T} \mathbf{Var}[\beta_i] \;\leq\; \left(\frac{n}{T}\right)^2 \cdot T \cdot \frac{c^{(i)}}{n} \;=\; \frac{n\,c^{(i)}}{T}.$$

Next, by Chebyshev's inequality we obtain

$$\mathbf{Pr}\left[\left|\hat{c}^{(i)} - \mathbf{E}\big[\hat{c}^{(i)}\big]\right| \geq \frac{\varepsilon}{8\,r}\mathcal{L}(G^{(i)})\right] \leq \frac{64 \cdot n \cdot c^{(i)} \cdot r^2}{T \cdot \varepsilon^2 \cdot \mathcal{L}(G^{(i)})^2} \;\leq\; \frac{64 \cdot n \cdot r^2}{T \cdot \varepsilon^2 \cdot \mathcal{L}(G^{(i)})},$$

where the last inequality follows from $\mathcal{L}(G^{(i)}) \geq c^{(i)}$, which trivially holds for $\varepsilon < \frac{1}{2}$.

With this, we obtain for $T \geq \frac{2^{10} \cdot n \cdot r^2}{\varepsilon^2 \cdot \mathcal{L}(G^{(i)})}$ the bound

$$\mathbf{Pr}\left[\left|\hat{c}^{(i)} - \mathbf{E}\big[\hat{c}^{(i)}\big]\right| \geq \frac{\varepsilon}{8r} \cdot \mathcal{L}(G^{(i)})\right] \;\leq\; \frac{1}{16},$$

which in turn yields

$$\mathbf{Pr}\left[\left|\hat{c}^{(i)} - \mathbf{E}\left[\hat{c}^{(i)}\right]\right| \leq \frac{\varepsilon}{8r} \cdot \mathcal{L}(G^{(i)})\right] \geq \frac{15}{16}. \qquad \square$$

Let us summarize the obtained results and outline how to proceed further. Right now, we know that if the algorithm performs at least $\frac{2^{10} \cdot n \cdot r^2}{\varepsilon^2 \cdot \mathcal{L}(G^{(i)})}$ iterations, then our estimator for the number of connected components is sharply concentrated. Ignoring the running time and the problem that the number of iteration depends on $\mathcal{L}(G^{(i)})$, one can show that plugging this approximation into our initial approach will yield a $(1+\varepsilon)$-approximation of the weight of the minimum spanning tree.

Therefore, we have to deal only with the number of iterations and the running time. Right now, the running time of a single iteration of the algorithm is $\mathcal{O}(nr/\varepsilon)$. Hence, in the case that $\mathcal{L}(G^{(i)}) = \Omega(n)$, we can use our algorithm as it is and obtain linear running time. However, if $\mathcal{L}(G^{(i)}) \ll n$, then we have to improve the (expected) running time of a single iteration of the algorithm. This can be achieved as follows. Recall that the vertices assigned to the sample representative point in $G^{(i)}$ form a clique. Since the number of representative points is at most $\mathcal{L}(G^{(i)})$, we know that we have at least $\mathcal{L}(G^{(i)})$ such cliques in $G^{(i)}$. This has certain impact on the degree distribution in the graph. For example, it implies that the average degree of $G^{(i)}$ is at least $n/\mathcal{L}(G^{(i)})$. In general, the smaller $\mathcal{L}(G^{(i)})$ is, the more vertices of high degree are in $G^{(i)}$. This can be used to speed up the algorithm. Recall that our algorithm uses a random number X as a stopping value in the exploration of connected components, i.e., the exploration of the graph stops, when the connected component has more than X vertices. A simple condition to stop the exploration is that the starting vertex already has more than X neighbors. But in the case when $\mathcal{L}(G^{(i)})$ is small, there are many such vertices in the graph. So, we just need a way to detect that the starting vertex has large degree. This can be done using a simple random sampling approach, whose analysis follows from Chernoff bounds and that is given in section 6. It will turn out that using this algorithm as a filter to immediately stop exploration of connected components whose starting vertex has degree larger than X will reduce the expected running time of a single iteration of the algorithm to $\mathcal{O}(\log^2 n \cdot r \cdot \mathcal{L}(G^{(i)})/\varepsilon)$. Finally, we can replace the bound on the number of iterations by a bound on the running time that ensures that with sufficiently high probability we have enough iterations. This will lead to our ultimate algorithm, which will be discussed in detail in section 7.

6. Estimating degrees of vertices: DEGREE-ESTIMATE algorithm. In order to test whether a given vertex p has degree larger than X, we perform a more general task and approximate the degree $\deg_i(p)$ of vertex p in $G^{(i)}$ within a constant factor. Note that in our setting finding $\deg_i(p)$ *exactly* requires $\Omega(n)$ time. Our estimator will run in time $\mathcal{O}(n \log n/(1+\deg_i(p)))$; i.e., it will be much faster if the degree of the vertex is large. Let c be a sufficiently large constant.

```
DEGREE-ESTIMATE (P, i, p)
  ℓ = 1
  repeat
     ℓ = 2ℓ
     pick a multiset S of N = c · log n · ℓ vertices uniformly at random with
     replacement
     let Y be the number of vertices in S that are adjacent to p in G^(i)
   until Y ≥ c log n or ℓ > n
  return D̂(p) = Yn/N
```

The algorithm increases the size of a sample set until we see at least $c \cdot \log n$ neighbors of p in our sample set. If this is the case, we can be sure that our sample set is a good approximation.

LEMMA 6.1. *Let (P, d) be a metric space with $|P| = n$, and let $G^{(i)}$ be the threshold graph for a given i. Then, with probability at least $1 - \frac{1}{8n^2}$, algorithm* DEGREE-ESTIMATE *runs in time $\mathcal{O}\left(\frac{n \log n}{1+\deg_i(p)}\right)$ and returns value $\widehat{D}(p)$ such that $\frac{1}{2} \cdot \deg_i(p) \le \widehat{D}(p) \le 2 \cdot \deg_i(p)$.*

Proof. For $\deg_i(p) = 0$ the algorithm outputs 0 in time $\mathcal{O}(n \log n)$. Hence, it remains to consider $\deg_i(p) > 0$.

Fix ℓ. Let Y_j denote the indicator random variable for the event that the jth vertex in S is adjacent to p. Clearly, $Y = \sum_{j=1}^{N} Y_j$. Further, we know that $\mathbf{E}[Y] = N \cdot \deg_i(p)/n = c \cdot \log n \cdot \frac{\ell \cdot \deg_i(p)}{n}$. Therefore, for $\ell = \frac{1}{2} \cdot n/\deg_i(p)$ we use a Chernoff bound to obtain

$$\mathbf{Pr}[Y \ge c \cdot \log n] \le \mathbf{Pr}\left[|Y - \mathbf{E}[Y]| \ge \frac{1}{2}\mathbf{E}[Y]\right] \le 2 \cdot e^{\frac{1}{4} \cdot \mathbf{E}[Y]/3} \le \frac{1}{16 \cdot \lceil \log n \rceil \cdot n^2}$$

using our assumption that c is a sufficiently large constant. By majorization, this implies a similar bound for $\ell < \frac{1}{2} \cdot n/\deg_i(p)$. In a similar way, we obtain that for $\ell > 2 \cdot n/\deg_i(p)$ we have

$$\mathbf{Pr}[Y < c \cdot \log n] \le \mathbf{Pr}\left[|Y - \mathbf{E}[Y]| \ge \frac{1}{2}\mathbf{E}[Y]\right] \le 2 \cdot e^{\frac{1}{4} \cdot \mathbf{E}[Y]/3} \le \frac{1}{16 \cdot (\lceil \log n \rceil + 1) \cdot n^2}.$$

Therefore, the probability that the algorithm terminates with $\frac{1}{2} \cdot n/\deg_i(p) \le \ell \le 2 \cdot n/\deg_i(p)$ is at least

$$1 - \sum_{\ell=1}^{\lceil \log n \rceil + 1} \frac{1}{16 \cdot (\lceil \log n \rceil + 1) \cdot n^2} = 1 - \frac{1}{16 \cdot n^2}.$$

Now we consider the case that $\frac{1}{2}n/\deg_i(p) \le \ell \le 2n/\deg_i(p)$. In this case, we have $\mathbf{E}[Y] \ge \frac{c}{2} \log n$. Chernoff bounds imply that

$$\mathbf{Pr}\left[\frac{\mathbf{E}[Y]}{2} \le Y \le 2\mathbf{E}[Y]\right] \le \mathbf{Pr}\left[|Y - \mathbf{E}[Y]| \ge \frac{1}{4}\mathbf{E}[Y]\right] \le 2 \cdot e^{\frac{1}{4} \cdot \mathbf{E}[Y]/3} \le \frac{1}{16 \cdot n^2}$$

for c large enough. Hence, the overall error probability is at most $\frac{1}{8 \cdot n^2}$. If the algorithm makes no error, the running time is clearly $\mathcal{O}(n \cdot \log n/(1 + \deg_i(p)))$, which proves the lemma. ∎

7. A sublinear time algorithm for estimating $c^{(i)}$. In this section we describe and analyze our $\widetilde{\mathcal{O}}(n/\varepsilon^6)$-time algorithm for estimating the number of connected components $c^{(i)}$ in $G^{(i)}$. The algorithm combines the algorithm from section 5 with the sampling algorithm DEGREE-ESTIMATE from section 6 to speed up the running time.

We now present our algorithm NUMBER-OF-CONNECTED-COMPONENTS (P, i, ε).

NUMBER-OF-CONNECTED-COMPONENTS (P, i, ε)
$s = 0$
while the running time is less than $T^* = \widetilde{\mathcal{O}}(n/\varepsilon^6)$ **do**
  $s = s + 1$; $\beta_s = 0$
  choose a vertex p_s independently and uniformly at random
  choose a random integer X according to $\mathbf{Pr}[X \geq k] = 1/k$
  $\widehat{D}(p_s) =$ DEGREE-ESTIMATE (P, i, p_s)
  if $\widehat{D}(p_s) \leq 2X$ **then**
    run CLIQUE-TREE-TRAVERSAL (P, p_s, i, ε) until one of the following events happens:
    (1) more than X vertices are explored
    (2) more than $\frac{4r}{\varepsilon}$ representative vertices are explored
    (3) the entire connected component in $G^{(i)}$ containing p_s is explored
  if event (3) happens **then** $\beta_s = 1$
output $\hat{c}^{(i)} = \frac{n}{s} \cdot \sum_{j=1}^{s} \beta_j$

We say algorithm DEGREE-ESTIMATE (P, i, p) *works properly* if it returns a value $\widehat{D}(p)$ with $\frac{1}{2}\widehat{D}(p) \leq \deg_i(p) \leq 2\widehat{D}(p)$ and its running time is $\mathcal{O}(n \cdot \log n/\deg_i(p))$. Notice that by Lemma 6.1, every run of algorithm DEGREE-ESTIMATE (P, i, p) works properly with probability at least $1 - \frac{1}{8n^2}$. Obviously, we can assume that the overall running time is $o(n^2)$ because otherwise we can simply compute the minimum spanning tree directly. Therefore, with probability at least $\frac{7}{8}$, all runs of algorithm DEGREE-ESTIMATE (P, i, p) incorporated in NUMBER-OF-CONNECTED-COMPONENTS (P, i, ε) work properly. Hence, from now on, we condition on the assumption that all runs of DEGREE-ESTIMATE (P, i, p) work properly.

Before we proceed with the analysis of the algorithm, we first explain our use of algorithm DEGREE-ESTIMATE that is needed to decrease the total running time of the algorithm and has no influence on the output value. If in the jth iteration of algorithm NUMBER-OF-CONNECTED-COMPONENTS procedure DEGREE-ESTIMATE returns a value $\widehat{D}(p) > 2X$, then we know that $\deg_i(p) > X$. If $\deg_i(p) > X$, then we know that $n_p^i > X$. Hence, our procedure stops the CLIQUE-TREE-TRAVERSAL because of event (1) before event (3) can happen. This would cause $\beta_j = 0$. Therefore, we do not have to invoke the CLIQUE-TREE-TRAVERSAL in that case and we can immediately set $\beta_j = 0$.

For the remaining analysis (besides the running time analysis) we can therefore ignore the procedure DEGREE-ESTIMATE. We can assume that for every sampled vertex p_j, we set $\beta_j = 0$ if algorithm CLIQUE-TREE-TRAVERSAL (P, p_j, i, ε) stops because of event (1) or (2), or we set $\beta_j = 1$ otherwise.

Our next step is to prove a bound on the number of iterations of algorithm NUMBER-OF-CONNECTED-COMPONENTS. Here we will make use of the dispersion $\mathcal{L}(G^{(i)})$, and our bound will depend on that value.

LEMMA 7.1 (expected running time of a single iteration). *Let $0 < \varepsilon < \frac{1}{2}$. In a single iteration of* NUMBER-OF-CONNECTED-COMPONENTS*, if the call to algorithm* DEGREE-ESTIMATE *works properly, then the expected running time (over the choice of X) is $\mathcal{O}(r \cdot \log^2 n \cdot \mathcal{L}(G^{(i)})/\varepsilon)$.*

Proof. Let T denote the random variable for the expected running time of a single iteration of the algorithm under the condition that algorithm DEGREE-ESTIMATE works properly.

We start our analysis with a partition of P into $\mathcal{L}(G^{(i)})$ *clusters* according to the full Clique-Tree-Traversal. There is exactly one cluster for each representative vertex. If a vertex p is not a representative vertex itself, then it has been explored from a representative vertex q. In this case we assign p to the cluster containing q. We observe that each cluster forms a clique in $G^{(i)}$, because the distance between any two points in the same cluster is at most $2\,\varepsilon\,(1+\varepsilon)^i$ and $\varepsilon < \frac{1}{2}$. For any vertex p, let C_p denote the cluster that contains p. Notice that $\deg_i(p) \geq |C_p| - 1$.

If all calls to algorithm Degree-Estimate work properly, then the test $\widehat{D}(p) < 2\,X$ rejects every vertex p_s in a cluster of size greater than $4\,X$. In this case, when $|C_{p_s}| > 4\,|X|$, the running time T of the iteration is at most $\alpha \cdot n \log n/(1+\widehat{D}(p_s))$, where α is a constant used to upper bound the constants hidden in the big-Oh notation of the running time of Degree-Estimate. Hence we get

$$T \;\leq\; \alpha \cdot n \cdot \log n/(1+\widehat{D}(p_s)) \;\leq\; 2 \cdot \alpha \cdot n \cdot \log n/(1+\deg_i(p_s)) \;\leq\; 2 \cdot \alpha \cdot n \cdot \log n/|C_{p_s}|.$$

For any vertex that is in a cluster of size smaller than or equal to $4\,X$, we run the Clique-Tree-Traversal until either we find more than $4r/\varepsilon$ representative vertices or some other stopping criterion is matched. By Lemma 4.1, the running time of this call to Clique-Tree-Traversal is $\mathcal{O}(n \log n)$ times the number of representative vertices visited, and so we have $T \leq \alpha \cdot n \cdot \log n \cdot r/\varepsilon$ for any vertex p_s with $|C_{p_s}| \leq 4\,|X|$.

Since the number of clusters is $\mathcal{L}(G^{(i)})$, we clearly have at most $4\,X \cdot \mathcal{L}(G^{(i)})$ vertices in all clusters of size at most $4\,X$. Now we observe that in the case $X > n$ the behavior of our algorithm is identical to the case $X = n$. Therefore, if we define a random variable $X^* = X$ for $X < n$ and $X^* = n$ for $X \geq n$, then we get for a fixed value of X^* and for a single iteration of algorithm Number-of-Connected-Components

$$\begin{aligned}
\mathbf{E}\big[T|X^*\big] &\leq \sum_{p\,:\,|C_p|\leq 4X^*} \mathbf{Pr}[p_s = p] \cdot \frac{\alpha \cdot n \cdot \log n \cdot r}{\varepsilon} \\
&\quad + \sum_{p\,:\,|C_p|>4X^*} \mathbf{Pr}[p_s = p] \cdot \frac{2\,\alpha \cdot n \cdot \log n}{|C_p|} \\
&= \sum_{p\,:\,|C_p|\leq 4X^*} \frac{1}{n} \cdot \frac{\alpha \cdot n \cdot \log n \cdot r}{\varepsilon} \;+\; \sum_{p\,:\,|C_p|>4X^*} \frac{1}{n} \cdot \frac{2 \cdot \alpha \cdot n \cdot \log n}{|C_p|} \\
&\leq \frac{4 \cdot \alpha \cdot X^* \cdot r \cdot \mathcal{L}(G^{(i)}) \cdot \log n}{\varepsilon} \;+\; 2 \cdot \alpha \cdot \log n \sum_{p \in P} \frac{1}{|C_p|} \\
&= \frac{4 \cdot \alpha \cdot X^* \cdot r \cdot \mathcal{L}(G^{(i)}) \cdot \log n}{\varepsilon} \;+\; 2 \cdot \alpha \cdot \log n \cdot \mathcal{L}(G^{(i)}) \\
&\leq \frac{6 \cdot \alpha \cdot X^* \cdot r \cdot \mathcal{L}(G^{(i)}) \cdot \log n}{\varepsilon}.
\end{aligned}$$

Since the choice of X^* is independent of the other choices, the inequality $\mathbf{E}[X^*] \leq \log n$ implies that

$$\mathbf{E}\big[T\big] \;\leq\; \frac{6 \cdot \alpha \cdot r \cdot \log^2 n \cdot \mathcal{L}(G^{(i)})}{\varepsilon} \;=\; \mathcal{O}\left(\frac{r \cdot \log^2 n \cdot \mathcal{L}(G^{(i)})}{\varepsilon}\right). \qquad \square$$

Corollary 7.2 (number of iterations). *If in all calls algorithm* Degree-Estimate *works properly, then for certain* $T^* = \mathcal{O}(n \cdot r^3 \cdot \log^2 n \cdot \varepsilon^{-3})$ *the number of iterations* s *of algorithm* Number-of-Connected-Components *is at least* $\frac{2^{10} \cdot n \cdot r^2}{\varepsilon^2 \cdot \mathcal{L}(G^{(i)})}$, *with probability at least* $\frac{15}{16}$.

Proof. By Lemma 7.1, the expected running time of a single iteration is at most $\frac{c\cdot r\cdot\log^2 n\cdot\mathcal{L}(G^{(i)})}{\varepsilon}$ for some constant c. Therefore, the expected running time for j iterations of the algorithm is at most j times the expected running time of a single iteration. Let T' be a random variable for the running time for $\frac{2^{10}\cdot n\cdot r^2}{\varepsilon^2\cdot\mathcal{L}(G^{(i)})}$ iterations of the algorithm. We have

$$\mathbf{E}[T] \le \frac{c'\cdot n\cdot r^3\cdot\log^2 n}{\varepsilon^3}$$

for an appropriate constant c'. We apply the Markov inequality to obtain that

$$\mathbf{Pr}\left[T' \ge 16\cdot\frac{c'\cdot n\cdot r^3\cdot\log^2 n}{\varepsilon^3}\right] \;\le\; \frac{1}{16}.$$

We conclude that for $T^* = 16\cdot\frac{c'\cdot n\cdot r^3\cdot\log^2 n}{\varepsilon^3} = \Theta(n\cdot r^3\cdot\log^2 n\cdot\varepsilon^{-3})$ the number of iterations of the **while**-loop of algorithm Number-of-Connected-Components is at least $2^{10}\cdot n\cdot r^2\cdot\varepsilon^{-2}/\mathcal{L}(G^{(i)})$ with probability at least $\frac{15}{16}$. ∎

A consequence of Corollary 7.2 is that if we run algorithm Number-of-Connected-Components for at least T^* steps, it will perform at least as many iterations as required to apply Lemma 5.2. Hence, it provides a sharply concentrated estimation of the number of connected components.

We can summarize our discussion with the following theorem which immediately implies Theorem 2.3.

Theorem 7.3. *For any given P, i, ε, algorithm* Number-of-Connected-Components *computes in $\widetilde{\mathcal{O}}(n/\varepsilon^6)$ time a value $\hat{c}^{(i)}$ that with probability at least $\frac{3}{4}$ satisfies*

$$c^{(i+1)} - \frac{1}{r}\cdot\frac{\text{MST}}{(1+\varepsilon)^i} \;\le\; \hat{c}^{(i)} \;\le\; c^{(i)} + \frac{1}{r}\cdot\frac{\text{MST}}{(1+\varepsilon)^i}.$$

Proof. Let us first assume that all calls of algorithm Degree-Estimate work properly. Lemma 5.1 gives us

$$c^{(i+1)} - \frac{1}{2r}\cdot\frac{\text{MST}}{(1+\varepsilon)^i} \;\le\; \mathbf{E}\big[\hat{c}^{(i)}\big] \;\le\; c^{(i)}.$$

We use Lemma 4.2 to obtain $\text{MST} \ge \varepsilon\cdot(1+\varepsilon)^i\cdot\mathcal{L}(G^{(i)})/4$. This together with Lemma 5.2 and Corollary 7.4 implies that with probability at least $\frac{7}{8}$ we have

$$\begin{aligned}
\hat{c}^{(i)} &\le \mathbf{E}\big[\hat{c}^{(i)}\big] + \frac{\varepsilon}{8r}\cdot\mathcal{L}(G^{(i)}) \;\le\; c^{(i)} + \frac{1}{r}\cdot\frac{\text{MST}}{(1+\varepsilon)^i},\\
\hat{c}^{(i)} &\ge \mathbf{E}\big[\hat{c}^{(i)}\big] - \frac{\varepsilon}{8r}\cdot\mathcal{L}(G^{(i)}) \;\ge\; c^{(i+1)} - \frac{1}{2r}\cdot\frac{\text{MST}}{(1+\varepsilon)^i} - \frac{\varepsilon}{2r}\cdot\frac{\text{MST}}{\varepsilon\cdot(1+\varepsilon)^r}\\
&\ge c^{(i+1)} - \frac{1}{r}\cdot\frac{\text{MST}}{(1+\varepsilon)^i}.
\end{aligned}$$

Finally, since all calls of algorithm Degree-Estimate work properly with probability at least $\frac{7}{8}$, the theorem follows. ∎

Using standard amplification bounds, we obtain the following corollary.

Corollary 7.4. *There is an algorithm that, for given P, i, ε, computes in $\widetilde{\mathcal{O}}(n\log(\varrho^{-1})/\varepsilon^6)$ time a value $\hat{c}^{(i)}$ that with probability at least $1-\varrho$ satisfies*

$$c^{(i+1)} - \frac{1}{r}\cdot\frac{\text{MST}}{(1+\varepsilon)^i} \;\le\; \hat{c}^{(i)} \;\le\; c^{(i)} + \frac{1}{r}\cdot\frac{\text{MST}}{(1+\varepsilon)^i}.$$

Proof. We run the algorithm $k = 324 \ln(\varrho^{-1})$ times and choose the median of the computed output values. Let X_j denote the random indicator variable for the event that the jth output value is within the bounds from Theorem 7.3. Clearly, the median is always correct if at least $\frac{2}{3}$ of the output values are correct. It follows from Chernoff bounds that the probability that fewer than $\frac{2}{3}$ of the output values are correct is at most

$$\mathbf{Pr}\left[\sum_{j=1}^{k} X_j \le \tfrac{2}{3}k\right] \le \mathbf{Pr}\left[\sum_{j=1}^{k} X_j \le \left(1-\frac{1}{9}\right)\cdot \mathbf{E}\left[\sum_{j=1}^{k} X_j\right]\right] \le e^{-k/324} = 1/\varrho. \qquad \square$$

8. Lower bounds. Let us first observe that no algorithm with $o(n)$ running time can approximate the cost of the minimum spanning tree within *any* factor. To see this, for a given approximation factor B, let us consider two graphs G_1 and G_2. G_1 consists of a clique of $n-1$ vertices having mutual distance 1 and a single outlier with distance $2\,B\,n$ to every other vertex. In graph G_2 the distance between every pair of vertices is 1. Clearly, the minimum spanning tree of graph G_2 has cost $n-1$, while the minimum spanning tree of graph G_1 has cost $n-2+2\,B\,n$. In order to distinguish between the two graphs one has to find the single outlier. This cannot be achieved in time $o(n)$ with constant confidence probability. This yields the following easy claim.

THEOREM 8.1. *No $o(n)$-time algorithm can approximate the weight of the minimum spanning tree within* any *factor.* $\square$

Next, we give another lower bound: We prove that no $o(n^2)$-time *deterministic* algorithm can approximate the weight of the minimum spanning tree in a metric space up to a constant factor.

Let $B \ge 1$ be an arbitrary constant, and let λ be a parameter, $0 < \lambda < 1$. We assume that $4B$ divides n. Consider the following two metric spaces. The first metric space, (P, d_1), consists of $\frac{n}{2B}$ cliques of size $2B$ with the pairwise distance within the nodes in each clique being λ; all other distances in (P, d_1) are 1. The second metric, (P, d_2), consists of $\frac{n}{4B}$ cliques of size $2B$, where the distance between any pair of nodes within one clique is λ, and all other distances in (P, d_2) are 1. Clearly, the cost of the minimum spanning tree of (P, d_1) is $\frac{n}{2B} + \lambda n \cdot (1 - \frac{1}{2B}) - 1$, and the cost of the minimum spanning tree of (P, d_2) is $\frac{n}{2} + \frac{n}{4B} + \frac{\lambda n}{2} \cdot (1 - \frac{1}{2B}) - 1$. Thus, for $\lambda = \frac{1}{8B^2}$ and $n \ge 8B$, the ratio between the weight of the minimum spanning tree of (P, d_2) and (P, d_1) is at least B. Hence, if it is not possible to distinguish between these two metric spaces using $o(n^2)$ time, then no sublinear-time factor B-approximation algorithm can exist.

We show that, given an arbitrary deterministic algorithm, we can always answer $\Omega(n^2/B^3)$ queries to the distance function of the input metric space (P, d) such that we can still construct both (P, d_1) and (P, d_2) by filling out the remaining (undetermined) entries in the distance matrix. This immediately implies that no $o(n^2/B^3)$-time factor B-approximation algorithm exists.

We call a point $p \in P$ *unspecified* if fewer than $\frac{n}{4B} - 1$ of its incident edges have lengths known to the algorithm at the current point of time. Otherwise, we call a point *specified*. If a point is specified, we reveal *all* distances to its neighbors according to a process described below. This, in particular, reveals all points that are in the same clique, and so we can reveal all distances from these points as well.

We show that the algorithm has to have at least $\frac{n}{8B} - 1$ specified points to be able to distinguish between metric spaces (P, d_1) and (P, d_2). This will require $\Omega(n^2/B^3)$ distance queries, because any specified point is in a clique of size (at most) $2B$, and

at least $\frac{n}{4B} - 1 - (\frac{n}{8B} - 1) = \frac{n}{8B}$ edges must be queried to make a point specified (we might get some edges for free, because they connect a point to other specified vertices).

In the following we describe how we answer the distance queries if the number of specified points is smaller than $\frac{n}{8B} - 1$. We assume that the algorithm queries every distance at most once. We answer queries in the following way. Since we assume for a specified point that the algorithm has queried all distances to its neighbors, any distance query is always between two unspecified points. We will always answer such a query with distance 1. Eventually, some point p becomes specified. When this happens, we group p and the remaining unspecified vertices in groups of $2B$ points such that no edge between any pair of points from any group has been queried. To show the existence of such a grouping we apply the following lemma by Hajnal and Szemerédi.

LEMMA 8.2 (see [16]). *Every graph with n vertices and maximum vertex degree $\Delta(G) \leq k$ is $(k+1)$-colorable with all color classes of size $\lceil n/(k+1) \rceil$ or $\lfloor n/(k+1) \rfloor$.*

Let U denote the set of all unspecified vertices and p. Clearly, $|U|$ is a multiple of B and $|U| \geq n/2$. Now let us consider the graph $G = (U, E)$, where $E = \{(p, q) : d(p, q)$ has been queried$\}$. Clearly, the degree of G is at most k, where we set $k = \frac{|U|}{2B} - 1$. Then, Lemma 8.2 implies the existence of a k-coloring such that each color class contains exactly $2B$ vertices. This will provide our (preliminary) grouping. We then reveal all distances to p and all distances to the points that are in the same group as p to the algorithm. This may cause other vertices to become specified, and we have to show all distances of these points to the algorithm. We continue this process until no more vertices have to be specified or we have reached $\frac{n}{2B} - 1$ specified vertices. At this point, we still have a valid $(k + 1)$-coloring such that no distances within a color class are specified, and so we can end up with metric space (P, d_1) and also with (P, d_2). This proves our final result.

THEOREM 8.3. *There exists no $o(n^2/B^3)$-time deterministic B-approximation algorithm for the cost of the minimum spanning tree in a metric space given as distance oracle.*

Acknowledgment. We would like to thank the anonymous referees for making many helpful comments that improved the writing of the paper.

REFERENCES

[1] N. ALON, S. DAR, M. PARNAS, AND D. RON, *Testing of clustering*, SIAM J. Discrete Math., 16 (2003), pp. 393–417.

[2] T. BATU, F. ERGÜN, J. KILIAN, A. MAGEN, S. RASKHODNIKOVA, R. RUBINFELD, AND R. SAMI, *A sublinear algorithm for weakly approximating edit distance*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC), 2003, pp. 316–324.

[3] P. BERENBRINK, F. ERGÜN, AND T. FRIEDETZKY, *Finding frequent patterns in a string in sublinear time*, in Proceedings of the 13th Annual European Symposium on Algorithms (ESA), Springer-Verlag, Berlin, 2005, pp. 746–757.

[4] P. B. CALLAHAN AND S. R. KOSARAJU, *A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields*, J. ACM, 42 (1995), pp. 67–90.

[5] B. CHAZELLE, *A minimum spanning tree algorithm with inverse-Ackermann type complexity*, J. ACM, 47 (2000), pp. 1012–1027.

[6] B. CHAZELLE, D. LIU, AND A. MAGEN, *Sublinear geometric algorithms*, SIAM J. Comput., 35 (2006), pp. 627–646.

[7] B. CHAZELLE, R. RUBINFELD, AND L. TREVISAN, *Approximating the minimum spanning tree weight in sublinear time*, SIAM J. Comput., 34 (2005), pp. 1370–1379.

[8] A. Czumaj, F. Ergün, L. Fortnow, A. Magen, I. Newman, R. Rubinfeld, and C. Sohler, *Approximating the weight of the Euclidean minimum spanning tree in sublinear time*, SIAM J. Comput., 35 (2005), pp. 91–109.
[9] A. Czumaj and C. Sohler, *Sublinear-time algorithms*, Bull. Eur. Assoc. Theor. Comput. Sci. EATCS, 89 (2006), pp. 23–47.
[10] D. Eppstein, *Spanning trees and spanners*, in Handbook of Computational Geometry, J.-R. Sack and J. Urrutia, eds., North–Holland, Amsterdam, 2000, pp. 425–461.
[11] F. Ergün, S. Muthukrishnan, and C. Sahinalp, *Sublinear methods for detecting periodic trends in data streams*, in Proceedings of the 6th Latin American Symposium on Theoretical Informatics (LATIN), Springer-Verlag, Berlin, 2004, pp. 16–28.
[12] A. Frieze and R. Kannan, *The regularity lemma and approximation schemes for dense problems*, in Proceedings of the 37th IEEE Symposium on Foundations of Computer Science (FOCS), 1996, pp. 12–20.
[13] A. Frieze, R. Kannan, and S. Vempala, *Fast Monte-Carlo algorithms for finding low-rank approximations*, J. ACM, 51 (2004), pp. 1025–1041.
[14] O. Goldreich, S. Goldwasser, and D. Ron, *Property testing and its connection to learning and approximation*, J. ACM, 45 (1998), pp. 653–750.
[15] O. Goldreich and D. Ron, *Approximating average parameters of graphs*, in Proceedings of the 10th International Workshop on Randomization and Computation (RANDOM), Springer-Verlag, Berlin, Heidelberg, 2006, pp. 363–374.
[16] A. Hajnal and E. Szemerédi, *Proof of a conjecture of P. Erdős*, in Combinatorial Theory and Its Applications, Vol. 2, P. Erdős, A. Rényi, and V. T. Sós, eds., North–Holland, London, 1970, pp. 601–623.
[17] P. Indyk, *A sublinear time approximation scheme for clustering in metric spaces*, in Proceedings of the 39th IEEE Symposium on Foundations of Computer Science (FOCS), 1998, pp. 154–159.
[18] P. Indyk, *Sublinear time algorithms for metric space problems*, in Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC), 1999, pp. 428–434.
[19] 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.
[20] N. Mishra, D. Oblinger, and L. Pitt, *Sublinear time approximate clustering*, in Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2001, pp. 439–447.
[21] M. Parnas and D. Ron, *Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms*, Theoret. Comput. Sci., 381 (2007), pp. 183–196.
[22] S. Pettie and V. Ramachandran, *An optimal minimum spanning tree algorithm*, J. ACM, 49 (2002), pp. 16–34.
[23] V. V. Vazirani, *Approximation Algorithms*, Springer-Verlag, Berlin, 2001.
[24] A. C.-C. Yao, *On constructing minimum spanning trees in k-dimensional spaces and related problems*, SIAM J. Comput., 11 (1982), pp. 721–736.