Sublinear-time algorithms

Artur Czumaj* Christian Sohler†

Abstract

In this paper we survey recent advances in the area of sublinear-time algorithms.

1 Introduction

The area of *sublinear-time algorithms* is a new rapidly emerging area of computer science. It has its roots in the study of massive data sets that occur more and more frequently in various applications. Financial transactions with billions of input data and Internet traffic analyses (Internet traffic logs, clickstreams, web data) are examples of modern data sets that show unprecedented scale. Managing and analyzing such data sets forces us to reconsider the traditional notions of efficient algorithms: processing such massive data sets in more than linear time is by far too expensive and often even linear time algorithms may be too slow. Hence, there is the desire to develop algorithms whose running times are not only polynomial, but in fact are *sublinear* in n.

Constructing a sublinear time algorithm may seem to be an impossible task since it allows one to read only a small fraction of the input. However, in recent years, we have seen development of sublinear time algorithms for optimization problems arising in such diverse areas as graph theory, geometry, algebraic computations, and computer graphics. Initially, the main research focus has been on designing efficient algorithms in the framework of *property testing* (for excellent surveys, see [26, 30, 31, 40, 50]), which is an alternative notion of approximation for decision problems. But more recently, we have seen some major progress in sublinear-time algorithms in the classical model of randomized and approximation algorithms. In this paper, we survey some of the recent advances in this area. Our main focus is on sublinear-time algorithms for combinatorial problems, especially for graph problems and optimization problems in metric spaces.

Our goal is to give a flavor of the area of sublinear-time algorithms. We focus on in our opinion the most representative results in the area and we aim to illustrate main techniques used to design sublinear-time algorithms. Still, many of the details of the presented results are omitted and we recommend the readers to follow the original works. We also do not aim to cover the entire area of sublinear-time algorithms, and in particular, we do not discuss property testing algorithms [26, 30, 31, 40, 50], even though this area is very closely related to the research presented in this survey.

*Department of Computer Science and Centre for Discrete Mathematics and its Applications (DIMAP), University of Warwick. Email: A.Czumaj@warwick.ac.uk.

†Department of Computer Science, TU Dortmund, Email: christian.sohler@tu-dortmund.de

Organization. We begin with an introduction to the area and then we give some sublinear-time algorithms for a basic problem in computational geometry [14]. Next, we present recent sublinear-time algorithms for basic graph problems: approximating the average degree in a graph [25, 34], estimating the cost of a minimum spanning tree [15] and approximating the size of a maximum matching [48, 53]. Then, we discuss sublinear-time algorithms for optimization problems in metric spaces. We present the main ideas behind recent algorithms for estimating the cost of minimum spanning tree [19] and facility location [10], and then we discuss the quality of random sampling to obtain sublinear-time algorithms for clustering problems [20, 46]. We finish with some conclusions.

2 Basic Sublinear Algorithms

The concept of sublinear-time algorithms has been known for a very long time, but initially it has been used to denote "pseudo-sublinear-time" algorithms, where after an appropriate *preprocessing*, an algorithm solves the problem in sublinear-time. For example, if we have a set of n numbers, then after an $\mathcal{O}(n \log n)$ preprocessing (sorting), we can trivially solve a number of problems involving the input elements. And so, if the after the preprocessing the elements are put in a sorted array, then in $\mathcal{O}(1)$ time we can find the kth smallest element, in $\mathcal{O}(\log n)$ time we can test if the input contains a given element x, and also in $\mathcal{O}(\log n)$ time we can return the number of elements equal to a given element x. Even though all these results are folklore, this is not what we call nowadays a sublinear-time algorithm.

In this survey, our goal is to study algorithms for which the input is taken to be in any standard representation and with no extra assumptions. Then, an algorithm does not have to read the entire input but it may determine the output by checking only a subset of the input elements. It is easy to see that for many natural problems it is impossible to give any reasonable answer if not all or almost all input elements are checked. But still, for some number of problems we can obtain good algorithms that do not have to look at the entire input. Typically, these algorithms are *randomized* (because most of the problems have a trivial linear-time deterministic lower bound) and they return only an *approximate* solution rather than the exact one (because usually, without looking at the whole input we cannot determine the exact solution). In this survey, we present recently developed sublinear-time algorithm for some combinatorial optimization problems.

Searching in a sorted *list*. It is well-known that if we can store the input in a sorted array, then we can solve various problems on the input very efficiently. However, the assumption that the input array is sorted is not natural in typical applications. Let us now consider a variant of this problem, where our goal is to *search* for an element x in a linked sorted list containing n *distinct* elements[1]. Here, we assume that the n elements are stored in a doubly-linked, each list element has access to the next and preceding element in the list, and the list is sorted (that is, if x follows y in the list, then $y < x$). We also assume that we have access to all elements in the list, which for example,

[1]The assumption that the input elements are *distinct* is important. If we allow multiple elements to have the same key, then the search problem requires $\Omega(n)$ time. To see this, consider the input in which about a half of the elements has key 1, another half has key 3, and there is a single element with key 2. Then, searching for 2 requires $\Omega(n)$ time.

can correspond to the situation that all n list elements are stored in an array (but the array is not sorted and we do not impose any order for the array elements). How can we find whether a given number x is in our input or is not?

On the first glace, it seems that since we do not have direct access to the rank of any element in the list, this problem requires $\Omega(n)$ time. And indeed, if our goal is to design a deterministic algorithm, then it is impossible to do the search in $o(n)$ time. However, if we allow randomization, then we can complete the search in $\mathcal{O}(\sqrt{n})$ expected time (and this bound is asymptotically tight).

Let us first sample uniformly at random a set S of $\Theta(\sqrt{n})$ elements from the input. Since we have access to all elements in the list, we can select the set S in $\mathcal{O}(\sqrt{n})$ time. Next, we scan all the elements in S and in $\mathcal{O}(\sqrt{n})$ time we can find two elements in S, p and q, such that $p \leq x < q$, and there is no element in S that is between p and q. Observe that since the input consist of n distinct numbers, p and q are uniquely defined. Next, we traverse the input list containing all the input elements starting at p until we find either the sought key x or we find element q.

Lemma 1 *The algorithm above completes the search in expected $\mathcal{O}(\sqrt{n})$ time. Moreover, no algorithm can solve this problem in $o(\sqrt{n})$ expected time.*

Proof. The running time of the algorithm if equal to $\mathcal{O}(\sqrt{n})$ plus the number of the input elements between p and q. Since S contains $\Theta(\sqrt{n})$ elements, the expected number of input elements between p and q is $\mathcal{O}(n/|S|) = \mathcal{O}(\sqrt{n})$. This implies that the expected running time of the algorithm is $\mathcal{O}(\sqrt{n})$.

For a proof of a lower bound of $\Omega(\sqrt{n})$ expected time, see, e.g., [14]. □

2.1 Geometry: Intersection of Two Polygons

Let us consider a related problem but this time in a geometric setting. Given two convex polygons A and B in $\mathbb{R}^2$, each with n vertices, determine if they intersect, and if so, then find a point in their intersection.

It is well known that this problem can be solved in $\mathcal{O}(n)$ time, for example, by observing that it can be described as a linear programming instance in 2-dimensions, a problem which is known to have a linear-time algorithm (cf. [24]). In fact, within the same time one can either find a point that is in the intersection of A and B, or find a line $\mathcal{L}$ that separates A from B (actually, one can even find a bitangent separating line $\mathcal{L}$, i.e., a line separating A and B which intersects with each of A and B in exactly one point). The question is whether we can obtain a better running time.

The complexity of this problem depends on the input representation. In the most powerful model, if the vertices of both polygons are stored in an array in cyclic order, Chazelle and Dobkin [13] showed that the intersection of the polygons can be determined in logarithmic time. However, a standard geometric representation assumes that the input is not stored in an array but rather A and B are given by their doubly-linked lists of vertices such that each vertex has as its successor the next vertex of the polygon in the clockwise order. Can we then test if A and B intersect?

Chazelle et al. [14] gave an $\mathcal{O}(\sqrt{n})$-time algorithm that reuses the approach discussed above for searching in a sorted list. Let us first sample uniformly at random $\Theta(\sqrt{n})$ vertices from each A and B, and let C_A and C_B be the convex hulls of the sample point sets for the polygons A and

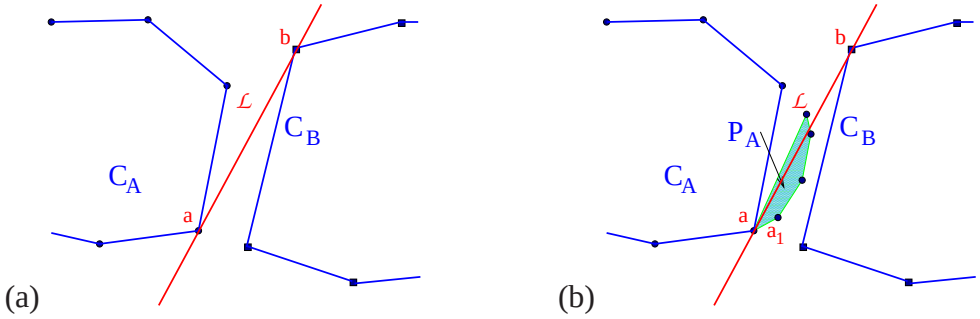

Figure 1: *(a) Bitangent line $\mathcal{L}$ separating C_A and C_B, and (b) the polygon P_A.*

B, respectively. Using the linear-time algorithm mentioned above, in $\mathcal{O}(\sqrt{n})$ time we can check if C_A and C_B intersects. If they do, then the algorithm will get us a point that lies in the intersection of C_A and C_B, and hence, this point lies also in the intersection of A and B. Otherwise, let $\mathcal{L}$ be the bitangent separating line returned by the algorithm (see Figure 1 (a)).

Let a and b be the points in $\mathcal{L}$ that belong to A and B, respectively. Let a_1 and a_2 be the two vertices adjacent to a in A. We will define now a new polygon P_A. If none of a_1 and a_2 is on the side C_A of $\mathcal{L}$ the we define P_A to be empty. Otherwise, exactly one of a_1 and a_2 is on the side C_A of $\mathcal{L}$; let it be a_1. We define polygon P_A by walking from a to a_1 and then continue walking along the boundary of A until we cross $\mathcal{L}$ again (see Figure 1 (b)). In a similar way we define polygon P_B. Observe that the expected size of each of P_A and P_B is at most $\mathcal{O}(\sqrt{n})$.

It is easy to see that A and B intersects if and only if either A intersects P_B or B intersects P_A. We only consider the case of checking if A intersects P_B. We first determine if C_A intersects P_B. If yes, then we are done. Otherwise, let $\mathcal{L}_A$ be a bitangent separating line that separates C_A from P_B. We use the same construction as above to determine a subpolygon Q_A of A that lies on the P_B side of $\mathcal{L}_A$. Then, A intersects P_B if and only if Q_A intersects P_B. Since Q_A has expected size $\mathcal{O}(\sqrt{n})$ and so does P_B, testing the intersection of these two polygons can be done in $\mathcal{O}(\sqrt{n})$ expected time. Therefore, by our construction above, we have solved the problem of determining if two polygons of size n intersect by reducing it to a constant number of problem instances of determining if two polygons of expected size $\mathcal{O}(\sqrt{n})$ intersect. This leads to the following lemma.

Lemma 2 [14] *The problem of determining whether two convex n-gons intersect can be solved in $\mathcal{O}(\sqrt{n})$ expected time, which is asymptotically optimal.*

Chazelle et al. [14] gave not only this result, but they also showed how to apply a similar approach to design a number of sublinear-time algorithms for some basic geometric problems. For example, one can extend the result discussed above to test the intersection of two convex polyhedra in $\mathbb{R}^3$ with n vertices in $\mathcal{O}(\sqrt{n})$ expected time. One can also approximate the volume of an n-vertex convex polytope to within a relative error $\varepsilon > 0$ in expected time $\mathcal{O}(\sqrt{n}/\varepsilon)$. Or even, for a pair of two points on the boundary of a convex polytope P with n vertices, one can estimate the length of an optimal shortest path outside P between the given points in $\mathcal{O}(\sqrt{n})$ expected time.

In all the results mentioned above, the input objects have been represented by a linked structure: either every point has access to its adjacent vertices in the polygon in $\mathbb{R}^2$, or the polytope is

defined by a doubly-connected edge list, or so. These input representations are standard in computational geometry, but a natural question is whether this is necessary to achieve sublinear-time algorithms — what can we do if the input polygon/polytop is represented by a set of points and no additional structure is provided to the algorithm? In such a scenario, it is easy to see that no $o(n)$-time algorithm can solve exactly any of the problems discussed above. That is, for example, to determine if two polygons with n vertices intersect one needs $\Omega(n)$ time. However, still, we can obtain some approximation to this problem, one which is described in the framework of *property testing*.

Suppose that we relax our task and instead of determining if two (convex) polytopes A and B in $\mathbb{R}^d$ intersects, we just want to distinguish between two cases: either A and B are intersection-free, or one has to "significantly modify" A and B to make them intersection-free. The definition of the notion of "significantly modify" may depend on the application at hand, but the most natural characterization would be to remove at least $\varepsilon\, n$ points in A and B, for an appropriate parameter ε (see [18] for a discussion about other geometric characterization). Czumaj et al. [23] gave a simple algorithm that for any $\varepsilon > 0$, can distinguish between the case when A and B do not intersect, and the case when at least $\varepsilon\, n$ points has to be removed from A and B to make them intersection-free: the algorithm returns the outcome of a test if a random sample of $\mathcal{O}((d/\varepsilon)\, \log(d/\varepsilon))$ points from A intersects with a random sample of $\mathcal{O}((d/\varepsilon)\, \log(d/\varepsilon))$ points from B.

Sublinear-time algorithms: perspective. The algorithms presented in this section should give a flavor of the area and give us the first impression of what do we mean by sublinear-time and what kind of results one can expect. In the following sections, we will present more elaborate algorithms for various combinatorial problems for graphs and for metric spaces.

3 Sublinear Time Algorithms for Graphs Problems

In the previous section, we introduced the concept of sublinear-time algorithms and we presented two basic sublinear-time algorithms for geometric problems. In this section, we will discuss sublinear-time algorithms for graph problems. Our main focus is on sublinear-time algorithms for graphs, with special emphasizes on sparse graphs represented by adjacency lists where combinatorial algorithms are sought.

3.1 Approximating the Average Degree

Assume we have access to the degree distribution of the vertices of an undirected connected graph $G = (V, E)$, i.e., for any vertex $v \in V$ we can query for its degree. Can we achieve a good approximation of the average degree in G by looking at a sublinear number of vertices? At first sight, this seems to be an impossible task. It seems that approximating the average degree is equivalent to approximating the average of a set of n numbers with values between 1 and $n-1$, which is not possible in sublinear time. However, Feige [25] proved that one can approximate the average degree in $\mathcal{O}(\sqrt{n}/\varepsilon)$ time within a factor of $2+\varepsilon$.

The difficulty with approximating the average of a set of n numbers can be illustrated with the following example. Assume that almost all numbers in the input set are 1 and a few of them are $n-1$. To approximate the average we need to approximate how many occurrences of $n-1$ exist. If there is only a constant number of them, we can do this only by looking at $\Omega(n)$ numbers in the set. So, the problem is that these large numbers can "hide" in the set and we cannot give a good approximation, unless we can "find" at least some of them.

Why is the problem less difficult, if, instead of an arbitrary set of numbers, we have a set of numbers that are the vertex degrees of a graph? For example, we could still have a few vertices of degree $n-1$. The point is that in this case any edge incident to such a vertex can be seen at another vertex. Thus, even if we do not sample a vertex with high degree we will see all incident edges at other vertices in the graph. Hence, vertices with a large degree cannot "hide."

We will sketch a proof of a slightly weaker result than that originally proven by Feige [25]. Let d denote the average degree in $G = (V, E)$ and let d_S denote the random variable for the average degree of a set S of s vertices chosen uniformly at random from V. We will show that if we set $s \geq \beta\sqrt{n}/\varepsilon^{\mathcal{O}(1)}$ for an appropriate constant β, then $d_S \geq (\frac{1}{2} - \varepsilon) \cdot d$ with probability at least $1-\varepsilon/64$. Additionally, we observe that Markov inequality immediately implies that $d_S \leq (1+\varepsilon)\cdot d$ with probability at least $1 - 1/(1+\varepsilon) \geq \varepsilon/2$. Therefore, our algorithm will pick $8/\varepsilon$ sets S_i, each of size s, and output the set with the smallest average degree. Hence, the probability that all of the sets S_i have too high average degree is at most $(1 - \varepsilon/2)^{\varepsilon/8} \leq 1/8$. The probability that one of them has too small average degree is at most $\frac{8}{\varepsilon} \cdot \frac{\varepsilon}{64} = 1/8$. Hence, the output value will satisfy both inequalities with probability at least $3/4$. By replacing ε with $\varepsilon/2$, this will yield a $(2+\varepsilon)$-approximation algorithm.

Now, our goal is to show that with high probability one does not underestimate the average degree too much. Let H be the set of the $\sqrt{\varepsilon\, n}$ vertices with highest degree in G and let $L = V \setminus H$ be the set of the remaining vertices. We first argue that the sum of the degrees of the vertices in L is at least $(\frac{1}{2} - \varepsilon)$ times the sum of the degrees of all vertices. This can be easily seen by distinguishing between edges incident to a vertex from L and edges within H. Edges incident to a vertex from L contribute with at least 1 to the sum of degrees of vertices in L, which is fine as this is at least $1/2$ of their full contribution. So the only edges that may cause problems are edges within H. However, since $|H| = \sqrt{\varepsilon\, n}$, there can be at most $\varepsilon\, n$ such edges, which is small compared to the overall number of edges (which is at least $n-1$, since the graph is connected).

Now, let d_H be the degree of a vertex with the smallest degree in H. Since we aim at giving a lower bound on the average degree of the sampled vertices, we can safely assume that all sampled vertices come from the set L. We know that each vertex in L has a degree between 1 and d_H. Let X_i, $1 \leq i \leq s$, be the random variable for the degree of the ith vertex from S. Then, it follows from Hoeffding bounds that

$$\mathbf{Pr}[\sum_{i=1}^{s} X_i \leq (1-\varepsilon) \cdot \mathbf{E}[\sum_{i=1}^{s} X_i]] \;\; \leq \;\; e^{-\frac{\mathbf{E}[\sum_{i=1}^{r} X_i] \cdot \varepsilon^2}{d_H}} \;\; .$$

We know that the average degree is at least $d_H \cdot |H|/n$, because any vertex in H has at least degree d_H. Hence, the average degree of a vertex in L is at least $(\frac{1}{2} - \varepsilon) \cdot d_H \cdot |H|/n$. This just means $\mathbf{E}[X_i] \geq (\frac{1}{2} - \varepsilon) \cdot d_H \cdot |H|/n$. By linearity of expectation we get $\mathbf{E}[\sum_{i=1}^{s} X_i] \geq s \cdot (\frac{1}{2} - \varepsilon) \cdot d_H \cdot |H|/n$.

This implies that, for our choice of s, with high probability we have $d_S \geq (\frac{1}{2} - \varepsilon) \cdot d$.

Feige showed the following result, which is stronger with respect to the dependence on ε.

Theorem 3 [25] *Using $\mathcal{O}(\varepsilon^{-1} \cdot \sqrt{n/d_0})$ queries, one can estimate the average degree of a graph within a ratio of $(2+\varepsilon)$, provided that $d \geq d_0$.*

Feige also proved that $\Omega(\varepsilon^{-1} \cdot \sqrt{n/d})$ queries are required, where d is the average degree in the input graph. Finally, any algorithm that uses only degree queries and estimates the average degree within a ratio $2 - \delta$ for some constant δ requires $\Omega(n)$ queries.

Interestingly, if one can also use neighborhood queries, then it is possible to approximate the average degree using $\widetilde{\mathcal{O}}(\sqrt{n}/\varepsilon^{\mathcal{O}(1)})$ queries with a ratio of $(1+\varepsilon)$, as shown by Goldreich and Ron [34]. The model for neighborhood queries is as follows. We assume we are given a graph and we can query for the ith neighbor of vertex v. If v has at least i neighbors we get the corresponding neighbor; otherwise we are told that v has less than i neighbors. We remark that one can simulate degree queries in this model with $\mathcal{O}(\log n)$ queries. Therefore, the algorithm from [34] uses only neighbor queries.

For a sketch of a proof, let us assume that we know the set H. Then we can use the following approach. We only consider vertices from L. If our sample contains a vertex from H we ignore it. By our analysis above, we know that there are only few edges within H and that we make only a small error in estimating the number of edges within L. We loose the factor of two, because we "see" edges from L to H only from one side. The idea behind the algorithm from [34] is to approximate the fraction of edges from L to H and add it to the final estimate. This has the effect that we count any edge between L and H twice, canceling the effect that we see it only from one side. This is done as follows. For each vertex v we sample from L we take a random set of incident edges to estimate the fraction $\lambda(v)$ of its neighbors that is in H. Let $\hat{\lambda}(v)$ denote the estimate we obtain. Then our estimate for the average degree will be $\sum_{v \in S \cap L}(1 + \hat{\lambda}(v)) \cdot d(v)/|S \cap L|$, where $d(v)$ denotes the degree of v. If for all vertices we estimate $\lambda(v)$ within an additive error of ε, the overall error induced by the $\hat{\lambda}$ will be small. This can be achieved with high probability querying $\mathcal{O}(\log n/\varepsilon^2)$ random neighbors. Then the output value will be a $(1+\varepsilon)$-approximation of the average degree. The assumption that we know H can be dropped by taking a set of $\mathcal{O}(\sqrt{n/\varepsilon})$ vertices and setting H to be the set of vertices with larger degree than all vertices in this set (breaking ties by the vertex number).

(We remark that the outline of a proof given above is different from the proof in [34].)

Theorem 4 [34] *Given the ability to make neighbor queries to the input graph G, there exists an algorithm that makes $\mathcal{O}(\sqrt{n/d_0} \cdot \varepsilon^{-\mathcal{O}(1)})$ queries and approximates the average degree in G to within a ratio of $(1+\varepsilon)$.*

3.2 Minimum Spanning Trees

One of the most fundamental graph problems is to compute a minimum spanning tree. Since the minimum spanning tree is of size linear in the number of vertices, no sublinear algorithm for sparse graphs can exists. It is also know that no constant factor approximation algorithm with $o(n^2)$ query

complexity in dense graphs (even in metric spaces) exists [37]. Given these facts, it is somewhat surprising that it is possible to approximate the cost of a minimum spanning tree in sparse graphs [15] as well as in metric spaces [19] to within a factor of $(1+\varepsilon)$.

In the following we will explain the algorithm for sparse graphs by Chazelle et al. [15]. We will prove a slightly weaker result than in [15]. Let $G=(V,E)$ be an undirected connected weighted graph with maximum degree D and integer edge weights from $\{1,\ldots,W\}$. We assume that the graph is given in adjacency list representation, i.e., for every vertex v there is a list of its at most D neighbors, which can be accessed from v. Furthermore, we assume that the vertices are stored in an array such that it is possible to select a vertex uniformly at random. We assume also that the values of D and W are known to the algorithm.

The main idea behind the algorithm is to express the cost of a minimum spanning tree as the number of connected components in certain auxiliary subgraphs of G. Then, one runs a randomized algorithm to estimate the number of connected components in each of these subgraphs.

To start with basic intuitions, let us assume that $W=2$, i.e., the graph has only edges of weight 1 or 2. Let $G^{(1)}=(V,E^{(1)})$ denote the subgraph that contains all edges of weight (at most) 1 and let $c^{(1)}$ be the number of connected components in $G^{(1)}$. It is easy to see that the minimum spanning tree has to link these connected components by edges of weight 2. Since any connected component in $G^{(1)}$ can be spanned by edges of weight 1, any minimum spanning tree of G has $c^{(1)}-1$ edges of weight 2 and $n-1-(c^{(1)}-1)$ edges of weight 1. Thus, the weight of a minimum spanning tree is

$$n-1-(c^{(1)}-1)+2\cdot(c^{(1)}-1) \;=\; n-2+c^{(1)} \;=\; n-W+c^{(1)}\;.$$

Next, let us consider an arbitrary integer value for W. Defining $G^{(i)}=(V,E^{(i)})$, where $E^{(i)}$ is the set of edges in G with weight at most i, one can generalize the formula above to obtain that the cost MST of a minimum spanning tree can be expressed as

$$MST \;=\; n-W+\sum_{i=1}^{W-1} c^{(i)}\;.$$

This gives the following simple algorithm.

APPROXMSTWEIGHT(G,ε)
  for $i=1$ **to** $W-1$
    Compute estimator $\widehat{c}^{(i)}$ for $c^{(i)}$
  output $\widetilde{MST}=n-W+\sum_{i=1}^{W-1}\widehat{c}^{(i)}$

Thus, the key question that remains is how to estimate the number of connected components. This is done by the following algorithm.

APPROXCONNECTEDCOMPS(G, s)
{ *Input: an arbitrary undirected graph* G }
{ *Output:* $\hat{c}$*: an estimation of the number of connected components of* G }
choose s vertices $u_1, \ldots, u_s$ uniformly at random
for $i = 1$ **to** s **do**
choose X according to $\mathbf{Pr}[X \geq k] = 1/k$
run breadth-fist-search (BFS) starting at u_i until either
(1) the whole connected component containing u_i has been explored, or
(2) X vertices have been explored
if BFS stopped in case (1) **then** $b_i = 1$
else $b_i = 0$
output $\hat{c} = \frac{n}{s} \sum_{i=1}^{s} b_i$

To analyze this algorithm let us fix an arbitrary connected component C and let $|C|$ denote the number of vertices in the connected component. Let c denote the number of connected components in G. We can write

$$\mathbf{E}[b_i] \;=\; \sum_{\text{connected component } C} \mathbf{Pr}[u_i \in C] \cdot \mathbf{Pr}[X \geq |C|] = \sum_{\text{connected component } C} \frac{|C|}{n} \cdot \frac{1}{|C|} = \frac{c}{n} \;.$$

And by linearity of expectation we obtain $\mathbf{E}[\hat{c}] = c$.

To show that $\hat{c}$ is concentrated around its expectation, we apply Chebyshev inequality. Since b_i is an indicator random variable, we have

$$\mathbf{Var}[b_i] \;=\; \mathbf{E}[b_i^2] - \mathbf{E}[b_i]^2 \leq \mathbf{E}[b_i^2] = \mathbf{E}[b_i] = c/n \;.$$

The b_i are mutually independent and so we have

$$\mathbf{Var}[\hat{c}] \;=\; \mathbf{Var}\big[\frac{n}{s} \cdot \sum_{i=1}^{s} b_i\big] = \frac{n^2}{s^2} \cdot \sum_{i=1}^{s} \mathbf{Var}[b_i] \leq \frac{n \cdot c}{s} \;.$$

With this bound for $\mathbf{Var}[\hat{c}]$, we can use Chebyshev inequality to obtain

$$\mathbf{Pr}[|\hat{c} - \mathbf{E}[\widehat{c}]| \geq \lambda n] \;\leq\; \frac{n \cdot c}{s \cdot \lambda^2 \cdot n^2} \leq \frac{1}{\lambda^2 \cdot s} \;.$$

From this it follows that one can approximate the number of connected components within additive error of λn in a graph with maximum degree D in $\mathcal{O}(\frac{D \cdot \log n}{\lambda^2 \cdot \varrho})$ time and with probability $1 - \varrho$. The following somewhat stronger result has been obtained in [15]. Notice that the obtained running time is *independent of the input size* n.

Theorem 5 [15] *The number of connected components in a graph with maximum degree* D *can be approximated with additive error at most* $\pm\lambda\, n$ *in* $\mathcal{O}(\frac{D}{\lambda^2} \log(D/\lambda))$ *time and with probability* $3/4$.

Now, we can use this procedure with parameters $\lambda = \varepsilon/(2W)$ and $\varrho = \frac{1}{4W}$ in algorithm APPROXMSTWEIGHT. The probability that at least one call to APPROXCONNECTEDCOMPS is not within an additive error $\pm\lambda n$ is at most $1/4$. The overall additive error is at most $\pm\varepsilon n/2$. Since the cost of the minimum spanning tree is at least $n - 1 \geq n/2$, it follows that the algorithms computes in $\mathcal{O}(D \cdot W^3 \cdot \log n/\varepsilon^2)$ time a $(1 \pm \varepsilon)$-approximation of the weight of the minimum spanning tree with probability at least $3/4$. In [15], Chazelle et al. proved a slightly stronger result which has running time *independent of the input size*.

Theorem 6 [15] *Algorithm* APPROXMSTWEIGHT *computes a value* $\widetilde{MST}$ *that with probability at least* $3/4$ *satisfies*

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

The algorithm runs in $\widetilde{\mathcal{O}}(D \cdot W/\varepsilon^2)$ *time.*

The same result also holds when D is only the average degree of the graph (rather than the maximum degree) and the edge weights are reals from the interval $[1, W]$ (rather than integers) [15]. Observe that, in particular, for sparse graphs for which the ratio between the maximum and the minimum weight is constant, the algorithm from [15] *runs in constant time*!

It was also proved in [15] that any algorithm estimating MST requires $\Omega(D \cdot W/\varepsilon^2)$ time.

3.3 Constant Time Approximation Algorithms for Maximum Matching

The next result we will explain here is an elegant technique to construct constant time approximation algorithms for graphs with bounded degree, as introduced by Nguyen and Onak [48].

Let $G = (V, E)$ be an undirected graph with maximum degree D. Define a randomized (α, β)-*approximation algorithm* to be an algorithm that returns with probability at least $2/3$ a solution with cost at most $\alpha Opt + \beta n$, where n is the size of the input and Opt denotes the cost of an optimal solution. For a graph we will define the input size to be the cardinality of its vertex set. We will consider the problem of computing *the size of maximum matching*, i.e., the size of a maximum size set $M \subseteq E$ such that no two edges are incident to the same vertex of G. It is known that the following simple greedy algorithm (that returns a *maximal matching*) provides a 2-approximation to this problem.

```
GREEDYMATCHING(G)
{ Input: an undirected graph G = (V, E) }
{ Output: a matching M ⊆ E }
    M ← ∅
        for each edge (u, v) ∈ E do
            Let V(M) be the set of vertices of edges in M
            if u, v ∉ V(M) then M ← M ∪ {e}
    return M
```

An important property of GREEDYMATCHING is that in the **for**-loop of the algorithm the edges are considered in an arbitrary ordering. We further observe that at any stage of the algorithm, the set M is a subset of the edges that have already been processed. Furthermore, if we consider an edge e then we know that neighboring edges can only be in M if they appear in the ordering before e. Now assume that the edges are inserted in a random order and let us try to determine for some fixed edge e whether it is contained in the constructed greedy matching. We could, of course, simply run the algorithm to do so by exploring the entire graph. However, our goal is to solve it using local computations that consider only the subgraph of the input graph close to e. In order to determine whether e is in the matching it suffices to determine for all its neighboring edges whether they are in M at the time e is considered by the algorithm. If e appears earlier than all of its neighbors in the random ordering, then we know that e is in the matching. Otherwise, we have to recursively solve the problem for all neighbors of e that appear before e in the random ordering. It may seem in the first place that this reasoning does not help because we now have to determine for a bigger set of edges whether they are in the matching. However, we also gained something: all edges we have to consider recursively are known to appear before e in the random ordering. This makes it less likely that some of their neighbors again appear even earlier in the sequence, which in turn means that we have to recurse for fewer of their neighbors. Thus, typically, this process stops after a constant number of steps.

Let us now try to formalize our findings. We obtain a random ordering of the edges by picking a priority $p(e)$ for each edge uniformly at random from $[0, 1]$. The random order we consider is now defined by increasing priorities. The benefit of this approach is that we do not have to compute a random ordering for the whole vertex set to run the local algorithm. Instead we can draw $p(e)$ at random whenever we consider an edge e for the first time. If we now want to determine whether an edge e is in the matching we only have to recurse with edges having a smaller priority than e. Thus, we have to follow all paths of decreasing priority starting at the endpoints of e.

For a fix path of length k in the graph, the probability that the priorities along the path are decreasing is $1/k!$ (this can be seen by the fact that for any sequence of k distinct priorities just one of them is decreasing; the case that probabilities are equal occurs with probability 0). Since the input graph has maximum degree D, the number of paths of length k starting from a vertex v is at most D^k. Hence, there are at most $2D^k$ paths starting at the endpoints of an edge e. For a large enough constant c this implies that for $k \geq 2^{cD}$, with (large) constant probability there is no path of length k starting from an endpoint of e that has decreasing priorities. This implies that we can determine whether e is in the matching by looking at all vertices with distance at most 2^{cD} from the endpoints of e.

Once we have an oracle to determine whether $e \in M$, we can sample edges to determine whether a given edge e is in M or not. Using a sample of size $\Theta(D/\varepsilon^2)$ we can approximate the number of edges in the matching up to additive error εn. This gives a constant-time $(2, \varepsilon)$-approximation algorithm for estimating the size of maximum matching, assuming D and ε are constant. The algorithm can be further improved to an $(1, \varepsilon)$-approximation using a more complicated approximation algorithm that greedily improves the matching using short augmenting paths. The query complexity of the improved algorithm is $2^{D^{\mathcal{O}(1/\varepsilon)}}$.

A further improvement has been done in a subsequent work by Yoshida et al. [53]. In that

paper, the authors reduce the query complexity to $D^{\mathcal{O}(1/\varepsilon^2)} + \mathcal{O}(1/\varepsilon)^{O(1/\varepsilon)}$ time. The source of improvement is here the idea to consider the edge with lowest priority first. If this edge turns out to be in the matching then we are already done and do not have to perform the remaining recursive calls.

Theorem 7 [48, 53] *For any integer $1 \leq k < \frac{n}{2}$, there is a $(1 + \frac{1}{k}, \varepsilon n)$-approximation algorithm with query complexity $D^{\mathcal{O}(k^2)} k^{\mathcal{O}(k)} \varepsilon^{-2}$ for the size of the maximum matching for graphs with n vertices and degree bound D.*

3.4 Other Sublinear-time Results for Graphs

In this section, our main focus was on combinatorial algorithms for sparse graphs. In particular, we did not discuss a large body of algorithms for dense graphs represented in the adjacency matrix model. Still, we mention the results of approximating the size of the maximum cut in *constant time* for dense graphs [28, 32], and the more general results about approximating all dense problems in Max-SNP in *constant time* [2, 8, 28]. Similarly, we also have to mention about the existence of a large body of property testing algorithms for graphs, which in many situations can lead to sublinear-time algorithms for graph problems. To give representative references, in addition to the excellent survey expositions [26, 30, 31, 40, 50], we want to mention the recent results on testability of graph properties, as described, e.g., in [3, 4, 5, 6, 11, 21, 33, 43].

4 Sublinear Time Approximation Algorithms for Problems in Metric Spaces

One of the most widely considered models in the area of sublinear time approximation algorithms is the *distance oracle model* for metric spaces. In this model, the input of an algorithm is a set P of n points in a metric space (P, d). We assume that it is possible to compute the distance $d(p, q)$ between any pair of points p, q in constant time. Equivalently, one could assume that the algorithm is given access to the $n \times n$ distance matrix of the metric space, i.e., we have oracle access to the matrix of a weighted undirected complete graph. Since the full description size of this matrix is $\Theta(n^2)$, we will call any algorithm with $o(n^2)$ running time a *sublinear algorithm*.

Which problems can and cannot be approximated in sublinear time in the distance oracle model? One of the most basic problems is to find (an approximation) of the shortest or the longest pairwise distance in the metric space. It turns out that the shortest distance cannot be approximated. The counterexample is a uniform metric (all distances are 1) with one distance being set to some very small value ε. Obviously, it requires $\Omega(n^2)$ time to find this single short distance. Hence, no sublinear time approximation algorithm for the shortest distance problem exists. What about the longest distance? In this case, there is a very simple $\frac{1}{2}$-approximation algorithm, which was first observed by Indyk [37]. The algorithm chooses an arbitrary point p and returns its furthest neighbor q. Let r, s be the furthest pair in the metric space. We claim that $d(p, q) \geq \frac{1}{2} d(r, s)$. By the triangle inequality, we have $d(r, p) + d(p, s) \geq d(r, s)$. This immediately implies that either $d(p, r) \geq \frac{1}{2} d(r, s)$ or $d(p, s) \geq \frac{1}{2} d(r, s)$. This shows the approximation guarantee.

In the following, we present some recent sublinear-time algorithms for a few optimization problems in metric spaces.

4.1 Minimum Spanning Trees

We can view a metric space as a weighted complete graph G. A natural question is whether we can find out anything about the minimum spanning tree of that graph. As already mentioned in the previous section, it is not possible to find in $o(n^2)$ time a spanning tree in the distance oracle model that approximates the minimum spanning tree within a constant factor [37]. However, it is possible to *approximate the weight* of a minimum spanning tree within a factor of $(1+\varepsilon)$ in $\widetilde{\mathcal{O}}(n/\varepsilon^{\mathcal{O}(1)})$ time [19].

The algorithm builds upon the ideas used to approximate the weight of the minimum spanning tree in graphs described in Section 3.2 [15]. Let us first observe that for the metric space problem we can assume that the maximum distance is $\mathcal{O}(n/\varepsilon)$ and the shortest distance is 1. This can be achieved by first approximating the longest distance in $\mathcal{O}(n)$ time and then scaling the problem appropriately. Since by the triangle inequality the longest distance also provides a lower bound on the minimum spanning tree, we can round up to 1 all edge weights that are smaller than 1. Clearly, this does not significantly change the weight of the minimum spanning tree. Now we could apply the algorithm ApproxMSTWeight from Section 3.2, but this would not give us an $o(n^2)$ algorithm. The reason is that in metric case we have a complete graph, i.e., the average degree is $D = n-1$, and the edge weights are in the interval $[1, W]$, where $W = \mathcal{O}(n/\varepsilon)$. So, we need a different approach. In the following we will outline an idea how to achieve a randomized $o(n^2)$ algorithm. To get a near linear time algorithm as in [19] further ideas have to be applied.

The first difference to the algorithm from Section 3.2 is that when we develop a formula for the minimum spanning tree weight, we use geometric progression instead of arithmetic progression. Assuming that all edge weights are powers of $(1+\varepsilon)$, we define $G^{(i)}$ to be the subgraph of G that contains all edges of length at most $(1+\varepsilon)^i$. We denote by $c^{(i)}$ the number of connected components in $G^{(i)}$. Then we can write

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

where $r = \log_{1+\varepsilon} W - 1$.

Once we have (1), our approach will be to approximate the number of connected components $c^{(i)}$ and use formula (1) as an estimator. Although geometric progression has the advantage that we only need to estimate the connected components in $r = \mathcal{O}(\log n/\varepsilon)$ subgraphs, the problem is that the estimator is multiplied by $(1+\varepsilon)^i$. Hence, if we use the procedure from Section 3.2, we would get an additive error of $\varepsilon\, n \cdot (1+\varepsilon)^i$, which, in general, may be much larger than the weight of the minimum spanning tree.

The basic idea how to deal with this problem is as follows. We will use a different graph traversal than BFS. Our graph traversal runs only on a subset of the vertices, which are called *representative vertices*. Every pair of representative vertices are at distance at least $\varepsilon \cdot (1+\varepsilon)^i$ from each other. Now, assume there are m representative vertices and consider the graph induced

by these vertices (there is a problem with this assumption, which will be discussed later). Running algorithm APPROXCONNECTEDCOMPS on this induced graph makes an error of $\pm\lambda m$, which must be multiplied by $(1+\varepsilon)^i$ resulting in an additive error of $\pm\lambda \cdot (1+\varepsilon)^i \cdot m$. Since the m representative vertices have pairwise distance $\varepsilon \cdot (1+\varepsilon)^i$, we have a lower bound $MST \geq m \cdot \varepsilon \cdot (1+\varepsilon)^i$. Choosing $\lambda = \varepsilon^2/r$ would result in a $(1+\varepsilon)$-approximation algorithm.

Unfortunately, this simple approach does not work. One problem is that we cannot choose a random representative point. This is because we have no a priori knowledge of the set of representative points. In fact, in the algorithm the points are chosen greedily during the graph traversal. As a consequence, the decision whether a vertex is a representative vertex or not, depends on the starting point of the graph traversal. This may also mean that the number of representative vertices in a connected component also depends on the starting point of the graph traversal. However, it is still possible to cope with these problems and use the approach outlined above to get the following result.

Theorem 8 [19] *The weight of a minimum spanning tree of an n-point metric space can be approximated in $\widetilde{\mathcal{O}}(n/\varepsilon^{\mathcal{O}(1)})$ time to within a $(1+\varepsilon)$ factor and with confidence probability at least $\frac{3}{4}$.*

4.1.1 Extensions: Sublinear-time $(2+\varepsilon)$-approximation of metric TSP and Steiner trees

Let us remark here one direct corollary of Theorem 8. By the well known relationship (see, e.g., [52]) between minimum spanning trees, travelling salesman tours, and minimum Steiner trees, the algorithm for estimating the weight of the minimum spanning tree from Theorem 8 immediately yields $\widetilde{\mathcal{O}}(n/\varepsilon^{\mathcal{O}(1)})$ 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*.

4.2 Uniform Facility Location

Similarly to the minimum spanning tree problem, one can estimate the cost of the *metric uniform facility location* problem in $\widetilde{\mathcal{O}}(n/\varepsilon^{\mathcal{O}(1)})$ time [10]. This problem is defined as follows. We are given an n-point metric space (P,d). We want to find a subset $F \subseteq P$ of open facilities such that

$$|F| + \sum_{p \in P} d(p,F)$$

is minimized. Here, $d(p,F)$ denote the distance from p to the nearest point in F. It is known that one cannot find a solution that approximates the optimal solution within a constant factor in $o(n^2)$ time [51]. However, it is possible to approximate the *cost* of an optimal solution within a constant factor.

The main idea is as follows. Let us denote by $B(p,r)$ the set of points from P with distance at most r from p. For each $p \in P$ let r_p be the unique value that satisfies

$$\sum_{q \in B(p,r_p)} (r_p - d(p,q)) = 1 \ .$$

Then one can show that

Lemma 9 [10]

$$\frac{1}{4} \cdot Opt \;\leq\; \sum_{p \in P} r_p \;\leq\; 6 \cdot Opt \;\;,$$

where Opt denotes the cost of an optimal solution to the metric uniform facility location problem.

Now, the algorithm is based on a randomized algorithm that for a given point p, estimates r_p to within a constant factor in time $\mathcal{O}(r_p \cdot n \cdot \log n)$ (recall that $r_p \leq 1$). Thus, the smaller r_p, the faster the algorithm. Now, let p be chosen uniformly at random from P. Then the expected running time to estimate r_p is $\mathcal{O}(n \log n \cdot \sum_{p \in P} r_p/n) = \mathcal{O}(n \log n \cdot \mathbf{E}[r_p])$. We pick a random sample set S of $s = 100 \log n / \mathbf{E}[r_p]$ points uniformly at random from P. (The fact that we do not know $\mathbf{E}[r_p]$ can be dealt with by using a logarithmic number of guesses.) Then we use our algorithm to compute for each $p \in S$ a value $\widehat{r}_p$ that approximates r_p within a constant factor. Our algorithm outputs $\frac{n}{s} \cdot \sum_{p \in S} \widehat{r}_p$ as an estimate for the cost of the facility location problem. Using Hoeffding bounds it is easy to prove that $\frac{n}{s} \cdot \sum_{p \in S} r_p$ approximates $\sum_{p \in P} r_p = Opt$ within a constant factor and with high probability. Clearly, the same statement is true, when we replace the r_p values by their constant approximations $\widehat{r}_p$. Finally, we observe that expected running time of our algorithm will be $\widetilde{\mathcal{O}}(n/\varepsilon^{\mathcal{O}(1)})$. This allows us to conclude with the following.

Theorem 10 [10] *There exists an algorithm that computes a constant factor approximation to the cost of the metric uniform facility location problem in $\mathcal{O}(n \log^2 n)$ time and with high probability.*

4.3 Clustering via Random Sampling

The problems of clustering large data sets into subsets (clusters) of similar characteristics are one of the most fundamental problems in computer science, operations research, and related fields. Clustering problems arise naturally in various massive datasets applications, including data mining, bioinformatics, pattern classification, etc. In this section, we will discuss the *uniformly random sampling* for clustering problems in metric spaces, as analyzed in two recent papers [20, 46].

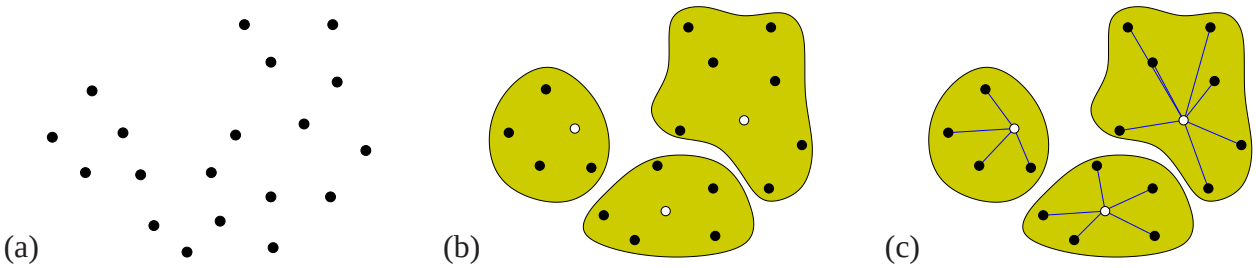

Figure 2: *(a) A set of points in a metric space, (b) its 3-clustering (white points correspond to the center points), and (c) the distances used in the cost for the 3-median.*

Let us consider a classical clustering problem known as the *k-median problem*. Given a finite metric space (P, d), the goal is to find a set $C \subseteq P$ of k centers (points in P) that minimizes $\sum_{p \in P} d(p, C)$, where $d(p, C)$ denotes the distance from p to the nearest point in C. The k-median

problem has been studied in numerous research papers. It is known to be $\mathcal{NP}$-hard and there exist constant-factor approximation algorithms running in $\widetilde{\mathcal{O}}(n\,k)$ time. In two recent papers [20, 46], the authors asked the question about the quality of the uniformly random sampling approach to k-median, that is, is the quality of the following generic scheme:

> (1) choose a multiset $S \subseteq P$ of size s i.u.r. (with repetitions),
> (2) run an α-approximation algorithm $\mathbb{A}_\alpha$ on input S to compute a solution C^*, and
> (3) **return** set C^* (the clustering induced by the solution for the sample).

The goal is to show that already a sublinear-size sample set S will suffice to obtain a good approximation guarantee. Furthermore, as observed in [46] (see also [45]), in order to have any guarantee of the approximation, one has to consider the quality of the approximation as a function of the diameter of the metric space. Therefore, we consider a model with the diameter of the metric space Δ given, that is, with $d : P \times P \to [0, \Delta]$.

Using techniques from statistics and computational learning theory, Mishra et al. [46] proved that if we sample a set S of $s = \widetilde{O}\left(\left(\frac{\alpha\Delta}{\varepsilon}\right)^2 (k \ln n + \ln(1/\delta))\right)$ points from P i.u.r. (*independently and uniformly at random*) and run α-approximation algorithm $\mathbb{A}_\alpha$ to find an approximation of the k-median for S, then with probability at least $1-\delta$, the output set of k centers has *average distance* to the nearest center of at most $2 \cdot \alpha \cdot \overline{\mathsf{med}}(P,k) + \varepsilon$, where $\overline{\mathsf{med}}(P,k)$ denotes the *average distance* to the k-median C, that is, $\overline{\mathsf{med}}(P,k) = \frac{\sum_{v \in P} d(v,C)}{n}$. We will now briefly sketch the analysis due to Czumaj and Sohler [20] of a similar approximation guarantee but with a smaller bound for s.

Let C_{opt} denote an optimal set of centers for P and let $\overline{\mathsf{cost}}(X,C)$ be the average cost of the clustering of set X with center set C, that is, $\overline{\mathsf{cost}}(X,C) = \frac{\sum_{x \in X} d(x,C)}{|X|}$. Notice that $\overline{\mathsf{cost}}(P, C_{opt}) = \overline{\mathsf{med}}(P,k)$. The analysis of Czumaj and Sohler [20] is performed in two steps.

(i) We first show that there is a set of k centers $C \subseteq S$ such that $\overline{\mathsf{cost}}(S,C)$ is a good approximation of $\overline{\mathsf{med}}(P,k)$ with high probability.

(ii) Next we show that with high probability, every solution C for P with cost much bigger than $\overline{\mathsf{med}}(P,k)$ is either not a feasible solution for S (i.e., $C \not\subseteq S$) or $\overline{\mathsf{cost}}(S,C) \gg \alpha \cdot \overline{\mathsf{med}}(P,k)$ (that is, the cost of C for the sample set S is large with high probability).

Since S contains a solution with cost at most $c \cdot \overline{\mathsf{med}}(P,k)$ for some small c, $\mathbb{A}_\alpha$ will compute a solution C^* with cost at most $\alpha \cdot c \cdot \overline{\mathsf{med}}(P,k)$. Now we have to prove that no solution C for P with cost much bigger than $\overline{\mathsf{med}}(P,k)$ will be returned, or in other words, that if C is feasible for S then its cost is larger than $\alpha \cdot c \cdot \overline{\mathsf{med}}(P,k)$. But this is implied by (ii). Therefore, the algorithm will not return a solution with too large cost, and the sampling is a $(c \cdot \alpha)$-approximation algorithm.

Theorem 11 [20] *Let $0 < \delta < 1$, $\alpha \geq 1$, $0 < \beta \leq 1$ and $\varepsilon > 0$ be approximation parameters. If $s \geq \frac{c \cdot \alpha}{\beta} \cdot \left(k + \frac{\Delta}{\varepsilon \cdot \beta} \cdot \left(\alpha \cdot \ln(1/\delta) + k \cdot \ln\left(\frac{k\,\Delta\,\alpha}{\varepsilon\,\beta^2}\right)\right)\right)$ for an appropriate constant c, then for the solution set of centers C^*, with probability at least $1-\delta$ it holds the following*

$$\overline{cost}(V, C^*) \;\leq\; 2\,(\alpha+\beta) \cdot \overline{\mathsf{med}}(P,k) + \varepsilon \;\;.$$

To give the flavor of the analysis, we will sketch (a simpler) part (i) of the analysis:

Lemma 12 *If* $s \geq \frac{3\Delta\alpha(1+\alpha/\beta)\ln(1/\delta)}{\beta\cdot\overline{\mathsf{med}}(P,k)}$ *then* $\mathbf{Pr}\big[\overline{\mathit{cost}}(S,C^*) \leq 2\,(\alpha+\beta)\cdot\overline{\mathsf{med}}(P,k)\big] \geq 1-\delta$.

Proof. We first show that if we consider the clustering of S with the optimal set of centers C_{opt} for P, then $\overline{\mathsf{cost}}(S,C_{opt})$ is a good approximation of $\overline{\mathsf{med}}(P,k)$. The problem with this bound is that in general, we cannot expect C_{opt} to be contained in the sample set S. Therefore, we have to show also that the optimal set of centers for S cannot have cost much worse than $\overline{\mathsf{cost}}(S,C_{opt})$.

Let X_i be the random variable for the distance of the ith point in S to the nearest center of C_{opt}. Then, $\overline{\mathsf{cost}}(S,C_{opt}) = \frac{1}{s}\sum_{1\leq i\leq s} X_i$, and, since $\mathbf{E}[X_i] = \overline{\mathsf{med}}(P,k)$, we also have $\overline{\mathsf{med}}(P,k) = \frac{1}{s}\cdot\mathbf{E}\big[\sum X_i\big]$. Hence,

$$\mathbf{Pr}\big[\overline{\mathsf{cost}}(S,C_{opt}) > (1+\tfrac{\beta}{\alpha})\cdot\overline{\mathsf{med}}(P,k)\big] = \mathbf{Pr}\big[\sum_{1\leq i\leq s} X_i > (1+\tfrac{\beta}{\alpha})\cdot\mathbf{E}\big[\sum_{1\leq i\leq s} X_i\big]\big] \ .$$

Observe that each X_i satisfies $0 \leq X_i \leq \Delta$. Therefore, by Chernoff-Hoeffding bound we obtain:

$$\mathbf{Pr}\big[\sum_{1\leq i\leq s} X_i > (1+\beta/\alpha)\cdot\mathbf{E}\big[\sum_{1\leq i\leq s} X_i\big]\big] \ \leq \ e^{-\frac{s\cdot\overline{\mathsf{med}}(P,k)\cdot\min\{(\beta/\alpha),(\beta/\alpha)^2\}}{3\,\Delta}} \ \leq \ \delta \ . \qquad (2)$$

This gives us a good bound for the cost of $\overline{\mathsf{cost}}(S,C_{opt})$ and now our goal is to get a similar bound for the cost of the optimal set of centers for S. Let C be the set of k centers in S obtained by replacing each $c \in C_{opt}$ by its nearest neighbor in S. By the triangle inequality, $\overline{\mathsf{cost}}(S,C) \leq 2\cdot\overline{\mathsf{cost}}(S,C_{opt})$. Hence, multiset S contains a set of k centers whose cost is at most $2\cdot(1+\beta/\alpha)\cdot\overline{\mathsf{med}}(P,k)$ with probability at least $1-\delta$. Therefore, the lemma follows because $\mathbb{A}_\alpha$ returns an α-approximation C^* of the k-median for S. □

Next, we only state the other lemma that describe part (ii) of the analysis of Theorem 11.

Lemma 13 *Let* $s \geq \frac{c\cdot\alpha}{\beta}\cdot\Big(k+\frac{\Delta}{\varepsilon\cdot\beta}\cdot\Big(\alpha\cdot\ln(1/\delta)+k\cdot\ln\Big(\frac{k\,\Delta\,\alpha}{\varepsilon\,\beta^2}\Big)\Big)\Big)$ *for an appropriate constant* c. *Let* $\mathbb{C}$ *be the set of all sets of* k *centers* C *of* P *with* $\overline{\mathit{cost}}(P,C) > (2\,\alpha+6\,\beta)\cdot\overline{\mathsf{med}}(P,k)$. *Then,*

$$\mathbf{Pr}\big[\exists C_b\in\mathbb{C} : C_b\subseteq S \ \textit{and} \ \overline{\mathit{cost}}(S,C_b) \ \leq \ 2\,(\alpha+\beta)\,\overline{\mathsf{med}}(P,k)\big] \ \leq \ \delta \ . \qquad \square$$

Observe that comparing the result from [46] to the result in Theorem 11, Theorem 11 improves the sample complexity by a factor of $\Delta\cdot\log n/\varepsilon$ while obtaining a slightly worse approximation ratio of $2\,(\alpha+\beta)\,\overline{\mathsf{med}}(P,k)+\varepsilon$, instead of $2\,\alpha\,\overline{\mathsf{med}}(P,k)+\varepsilon$ as in [46]. However, since the polynomial-time algorithm with the best known approximation guarantee has $\alpha = 3+\frac{1}{c}$ for the running time of $\mathcal{O}(n^c)$ time [9], this significantly improves the running time of [46] for all realistic choices of the input parameters while achieving the same approximation guarantee. As a highlight, Theorem 11 yields a sublinear-time algorithm that in time $\tilde{\mathcal{O}}((\frac{\Delta}{\varepsilon}\cdot(k+\log(1/\delta)))^2)$ — *fully independent of* n — returns a set of k centers for which the average distance to the nearest median is at most $\mathcal{O}(\overline{\mathsf{med}}(P,k))+\varepsilon$ with probability at least $1-\delta$.

Extensions. The result in Theorem 11 can be significantly improved if we assume the input points are in *Euclidean space* $\mathbb{R}^d$. In this case the approximation guarantee can be improved to $(\alpha+\beta)\,\overline{\mathsf{med}}(P,k)+\varepsilon$ at the cost of increasing the sample size to $\widetilde{O}(\frac{\Delta\cdot\alpha}{\varepsilon\cdot\beta^2}\cdot(k\,d+\log(1/\delta)))$.

Furthermore, a similar approach as that sketched above can be applied to study similar generic sample schemes for other clustering problems. As it is shown in [20], almost identical analysis lead to sublinear (independent on n) sample complexity for the classical k-*means problem*. Also, a more complex analysis can be applied to study the sample complexity for the *min-sum* k-*clustering problem* [20].

4.4 Other Results

Indyk [37] was the first who observed that some optimization problems in metric spaces can be solved in sublinear-time, that is, in $o(n^2)$ time. He presented $(\frac{1}{2}-\varepsilon)$-approximation algorithms for MaxTSP and the maximum spanning tree problems that run in $O(n/\varepsilon)$ time [37]. He also gave a $(2+\varepsilon)$-approximation algorithm for the minimum routing cost spanning tree problem and a $(1+\varepsilon)$ approximation algorithm for the average distance problem; both algorithms run in $O(n/\varepsilon^{O(1)})$ time.

There is also a number of sublinear-time algorithms for various clustering problems in either Euclidean spaces or metric spaces, when the number of clusters is small. For radius (k-*center*) and *diameter clustering* in Euclidean spaces, sublinear-time property testing algorithms [1, 21] and tolerant testing algorithms [49] have been developed. The first sublinear algorithm for the k-*median* problem was a bicriteria approximation algorithm [37]. This algorithm computes in $\widetilde{\mathcal{O}}(n\,k)$ time a set of $\mathcal{O}(k)$ centers that are a constant factor approximation to the k-median objective function. Later, standard constant factor approximation algorithms were given that run in time $\widetilde{O}(n\,k)$ (see, e.g., [44, 51]). These sublinear-time results have been extended in many different ways, e.g., to efficient data streaming algorithms and very fast algorithms for Euclidean k-median and also to k-*means*, see, e.g., [9, 12, 16, 27, 35, 36, 41, 42, 45]. For another clustering problem, the *min-sum* k-*clustering problem* (which is complement to the Max-k-Cut), for the basic case of $k=2$, Indyk [39] (see also [38]) gave a $(1+\varepsilon)$-approximation algorithm that runs in time $O(2^{1/\varepsilon^{O(1)}}\,n\,(\log n)^{O(1)})$, which is sublinear in the full input description size. No such results are known for $k\geq 3$, but recently, [22] gave a constant-factor approximation algorithm for min-sum k-clustering that runs in time $\mathcal{O}(n\,k\,(k\,\log n)^{\mathcal{O}(k)})$ and a polylogarithmic approximation algorithm running in time $\widetilde{\mathcal{O}}(n\,k^{\mathcal{O}(1)})$.

4.5 Limitations: What Cannot be done in Sublinear-Time

The algorithms discussed in the previous sections may suggest that many optimization problems in metric spaces have sublinear-time algorithms. However, it turns out that the problems listed in the previous sections are more like exceptions than a norm. Indeed, most of the problems have a trivial lower bound that exclude sublinear-time algorithms. We have already mentioned in Section 4 that the problem of approximating the cost of the lightest edge in a finite metric space (P,d) requires $\Omega(n^2)$, even if randomization is allowed. The other problems for which no sublinear-time algorithms are possible include estimation of the cost of minimum-cost matching, the cost of

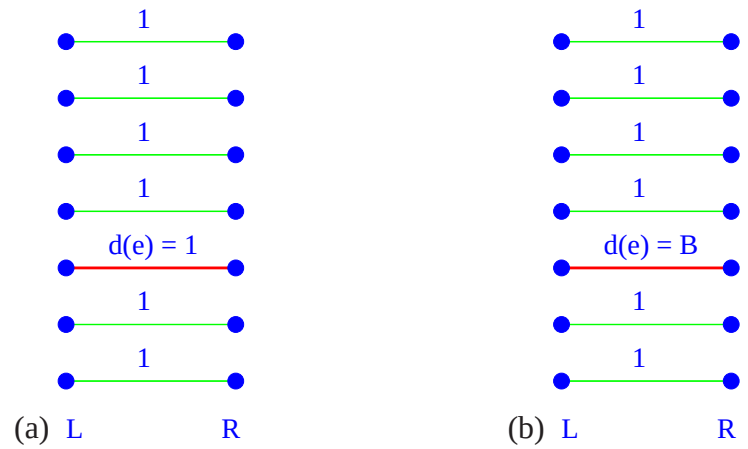

Figure 3: *Two instance of the metric matching which are indistinguishable in $o(n^2)$ time and whose cost differ by a factor greater than λ. The perfect matching connecting L with R is selected at random and the edge e is selected as a random edge from the matching. We set $B = n\,(\lambda - 1) + 2$. The distances not shown are all equal to $n^3\,\lambda$.*

minimum-cost bi-chromatic matching, the cost of minimum *non-uniform* facility location, the cost of k-median for $k = n/2$; all these problems require $\Omega(n^2)$ (randomized) time to estimate the cost of their optimal solution to within any constant factor [10].

To illustrate the lower bounds, we give two instances of the metric spaces which are indistinguishable by any $o(n^2)$-time algorithm for which the cost of the minimum-cost matching in one instance is greater than λ times the one in the other instance (see Figure 3). Consider a metric space (P, d) with $2n$ points, n points in L and n points in R. Take a random perfect matching $\mathbb{M}$ between the points in L and R, and then choose an edge $e \in \mathbb{M}$ at random. Next, define the distance in (P, d) as follows:

- $d(e)$ is either 1 or B, where we set $B = n\,(\lambda - 1) + 2$,
- for any $e^* \mathbb{M} \setminus \{e\}$ set $d(e^*) = 1$, and
- for any other pair of points $p, q \in P$ not connected by an edge from $\mathbb{M}$, $d(p, q) = n^3\,\lambda$.

It is easy to see that both instances define properly a metric space (P, d). For such problem instances, the cost of the minimum-cost matching problem will depend on the choice of $d(e)$: if $d(e) = B$ then the cost will be $n - 1 + B > n\,\lambda$, and if $d(e) = 1$, then the cost will be n. Hence, any λ-factor approximation algorithm for the matching problem must distinguish between these two problem instances. However, this requires to find if there is an edge of length B, and this is known to require time $\Omega(n^2)$, even if a randomized algorithm is used.

5 Conclusions

It would be impossible to present a complete picture of the large body of research known in the area of sublinear-time algorithms in such a short paper. In this survey, our main goal was to give

some flavor of the area and of the types of the results achieved and the techniques used. For more details, we refer to the original works listed in the references.

We did not discuss two important areas that are closely related to sublinear-time algorithms: property testing and data streaming algorithms. For interested readers, we recommend the surveys in [7, 26, 30, 31, 40, 50] and [47], respectively.

The current paper is a slightly updated version of the paper A. Czumaj and C. Sohler. Sublinear-time algorithms. *Bulletin of the EATCS*, 89: 23–47, June 2006.

References

[1] N. Alon, S. Dar, M. Parnas, and D. Ron. Testing of clustering. *SIAM Journal on Discrete Mathematics*, 16(3): 393–417, 2003.

[2] N. Alon, W. Fernandez de la Vega, R. Kannan, and M. Karpinski. Random sampling and approximation of MAX-CSPs. *Journal of Computer and System Sciences*, 67(2): 212–243, 2003.

[3] N. Alon, E. Fischer, M. Krivelevich, and M. Szegedy. Efficient testing of large graphs. *Combinatorica*, 20(4): 451–476, 2000.

[4] N. Alon, E. Fischer, I. Newman, and A. Shapira. A combinatorial characterization of the testable graph properties: it's all about regularity. *SIAM Journal on Computing*, 39(1): 143–167, 2009.

[5] N. Alon and A. Shapira. Every monotone graph property is testable. *SIAM Journal on Computing*, 38(2): 505–522, 2008.

[6] N. Alon and A. Shapira. A characterization of the (natural) graph properties testable with one-sided error. *SIAM Journal on Computing*, 37(6): 1703–1727, 2008.

[7] N. Alon and A. Shapira. Homomorphisms in graph property testing - A survey. In *Topics in Discrete Mathematics*, dedicated to Jarik Nesetril on the occasion of his 60th Birthday, M. Klazar, J. Kratochvil, M. Loebl, J. Matousek, R. Thomas and P. Valtr, eds., pp. 281–313.

[8] S. Arora, D. R. Karger, and M. Karpinski. Polynomial time approximation schemes for dense instances of $\mathcal{NP}$-hard problems. *Journal of Computer and System Sciences*, 58(1): 193–210, 1999.

[9] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristics for k-median and facility location problems. *SIAM Journal on Computing*, 33(3): 544–562, 2004.

[10] M. Bădoiu, A. Czumaj, P. Indyk, and C. Sohler. Facility location in sublinear time. *Proceedings of the 32nd Annual International Colloquium on Automata, Languages and Programming (ICALP)*, pp. 866-877, 2005.

[11] C. Borgs, J. Chayes, L. Lovász, V. T. Sos, B. Szegedy, and K. Vesztergombi. Graph limits and parameter testing. *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC)*, 2006.

[12] M. Charikar, L. O'Callaghan, and R. Panigrahy. Better streaming algorithms for clustering problems. *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 30–39, 2003.

[13] B. Chazelle and D. P. Dobkin. Intersection of convex objects in two and three dimensions. *Journal of the ACM*, 34(1): 1–27, 1987.

[14] B. Chazelle, D. Liu, and A. Magen. Sublinear geometric algorithms. *SIAM Journal on Computing*, 35(3): 627–646, 2006.

[15] B. Chazelle, R. Rubinfeld, and L. Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SIAM Journal on Computing*, 34(6): 1370–1379, 2005.

[16] K. Chen. On k-median clustering in high dimensions. *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 1177–1185, 2006.

[17] A. Czumaj, F. Ergün, L. Fortnow, A. Magen, I. Newman, R. Rubinfeld, and C. Sohler. Sublinear-time approximation of Euclidean minimum spanning tree. *SIAM Journal on Computing*, 35(1): 91–109, 2005.

[18] A. Czumaj and C. Sohler. Property testing with geometric queries. *Proceedings of the 9th Annual European Symposium on Algorithms (ESA)*, pp. 266–277, 2001.

[19] A. Czumaj and C. Sohler. Estimating the weight of metric minimum spanning trees in sublinear-time. *SIAM Journal on Computing*, 39(3): 904–922, 2009.

[20] A. Czumaj and C. Sohler. Sublinear-time approximation for clustering via random sampling. *Random Structures and Algorithms*, 30(1-2): 226–256, 2007.

[21] A. Czumaj and C. Sohler. Abstract combinatorial programs and efficient property testers. *SIAM Journal on Computing*, 34(3): 580–615, 2005.

[22] A. Czumaj and C. Sohler. Small space representations for metric min-sum k-clustering and their applications. *Theory of Computing Systems*, 46(3): 416–442, 2010.

[23] A. Czumaj, C. Sohler, and M. Ziegler. Property testing in computational geometry. *Proceedings of the 8th Annual European Symposium on Algorithms (ESA)*, pp. 155–166, 2000.

[24] M. Dyer, N. Megiddo, and E. Welzl. Linear programming. In *Handbook of Discrete and Computational Geometry*, 2nd edition, edited by J. E. Goodman and J. O'Rourke, CRC Press, 2004, pp. 999–1014.

[25] U. Feige. On sums of independent random variables with unbounded variance and estimating the average degree in a graph. *SIAM Journal on Computing*, 35(4): 964–984, 2006.

[26] E. Fischer. The art of uninformed decisions: A primer to property testing. *Bulletin of the EATCS*, 75: 97–126, October 2001.

[27] G. Frahling and C. Sohler. Coresets in dynamic geometric data streams. *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 209–217, 2005.

[28] A. Frieze and R. Kannan. Quick approximation to matrices and applications. *Combinatorica*, 19(2): 175–220, 1999.

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

[30] O. Goldreich. Combinatorial property testing (a survey). In P. Pardalos, S. Rajasekaran, and J. Rolim, editors, *Proceedings of the DIMACS Workshop on Randomization Methods in Algorithm Design*, volume 43 of *DIMACS, Series in Discrete Mathetaics and Theoretical Computer Science*, pp. 45–59, 1997. American Mathematical Society, Providence, RI, 1999.

[31] O. Goldreich. Property testing in massive graphs. In J. Abello, P. M. Pardalos, and M. G. C. Resende, editors, *Handbook of massive data sets*, pp. 123–147. Kluwer Academic Publishers, 2002.

[32] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4): 653–750, 1998.

[33] O. Goldreich and D. Ron. A sublinear bipartiteness tester for bounded degree graphs. *Combinatorica*, 19(3):335–373, 1999.

[34] O. Goldreich and D. Ron. Approximating average parameters of graphs. *Random Structures and Algorithms*, 32(4): 473–493, 2008.

[35] S. Har-Peled and S. Mazumdar. Coresets for k-means and k-medians and their applications. *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 291–300, 2004.

[36] S. Har-Peled and A. Kushal. Smaller coresets for k-median and k-means clustering. *Discrete & Computational Geometry*, 37(1): 3–19, 2007.

[37] P. Indyk. Sublinear time algorithms for metric space problems. *Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC)*, pp. 428–434, 1999.

[38] P. Indyk. A sublinear time approximation scheme for clustering in metric spaces. *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 154–159, 1999.

[39] P. Indyk. *High-Dimensional Computational Geometry*. PhD thesis, Stanford University, 2000.

[40] R. Kumar and R. Rubinfeld. Sublinear time algorithms. *SIGACT News*, 34: 57–67, 2003.

[41] A. Kumar, Y. Sabharwal, and S. Sen. A simple linear time $(1 + \varepsilon)$-approximation algorithm for k-means clustering in any dimensions. *Proceedings of the 45th IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 454–462, 2004.

[42] A. Kumar, Y. Sabharwal, and S. Sen. Linear time algorithms for clustering problems in any dimensions. *Proceedings of the 32nd Annual International Colloquium on Automata, Languages and Programming (ICALP)*, pp. 1374–1385, 2005.

[43] L. Lovász and B. Szegedy. Graph limits and testing hereditary graph properties. Technical Report, MSR-TR-2005-110, Microsoft Research, August 2005.

[44] R. Mettu and G. Plaxton. Optimal time bounds for approximate clustering. *Machine Learning*, 56(1-3):35–60, 2004.

[45] A. Meyerson, L. O'Callaghan, and S. Plotkin. A k-median algorithm with running time independent of data size. *Machine Learning*, 56(1–3): 61–87, July 2004.

[46] N. Mishra, D. Oblinger, and L. Pitt. Sublinear time approximate clustering. *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 439–447, 2001.

[47] S. Muthukrishnan. Data streams: Algorithms and applications. In *Foundations and Trends in Theoretical Computer Science*, volume 1, issue 2, August 2005.

[48] H. Nguyen and K. Onak. Constant-time approximation algorithms via local improvements. *Proceedings of the 49th IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 489–498, 2008.

[49] M. Parnas, D. Ron, and R. Rubinfeld. Tolerant property testing and distance approximation. *Journal of Computer and System Sciences*, 72(6): 1012–1042, 2006.

[50] D. Ron. Property testing. In P. M. Pardalos, S. Rajasekaran, J. Reif, and J. D. P. Rolim, editors, *Handobook of Randomized Algorithms*, volume II, pp. 597–649. Kluwer Academic Publishers, 2001.

[51] M. Thorup. Quick k-median, k-center, and facility location for sparse graphs. *SIAM Journal on Computing*, 34(2):405–432, 2005.

[52] V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, New York, 2004.

[53] Y. Yoshida, M. Yamamoto, and H. Ito. Improved constant-time approximation algorithms for maximum independent sets and maximum matchings. *Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC)*, pp. 225–234, 2009.