Sublinear-Time Algorithms for Approximating Graph Parameters

Dana Ron(✉)

School of Electrical Engineering, Tel Aviv University, 69978 Tel Aviv, Israel
danaron@tau.ac.il

1 Introduction

Given a graph $G = (V, E)$, we may be interested in computing various parameters that are associated with the graph. Such parameters include the average degree, the number of connected components, and the size of a minimum vertex cover. These parameters and many others can be computed (exactly or approximately) in an efficient manner. That is, in time that is polynomial in the size of the graph, and possibly even linear in this size. However, for very large graphs, even linear time may by infeasible. Hence, we need to design more efficient algorithms, that is, algorithms that run in *sublinear* time.

Given the constraint on their running time, such algorithms cannot read the entire graph, but can access parts of the graph by performing queries. We mainly consider two types of queries: degree queries and neighbor queries. In a degree query the algorithm specifies a vertex $v \in V$, and the answer to the query is the degree of v in G, denoted $d(v)$. In a neighbor query, the algorithm specifies a vertex v and an index i. The answer to the query is the i^{th} neighbor of v if $i \in \{1, \ldots, d(v)\}$, and is a special symbol, $\perp$, otherwise.[1] A third possible type of query is a vertex-pair query, where the algorithm specifies a pair of vertices $\{u, v\}$ and the answer is 1 if $\{u, v\} \in E$ and 0 otherwise. If the graph is edge weighted, then the answer to a neighbor query (similarly, a vertex-pair query) also includes the weight of the corresponding edge. We assume that the algorithm is given the number of vertices in the graph, denoted n, and, without loss of generality, may assume that $V = \{1, \ldots, n\}$. In all that follows, unless stated explicitly otherwise, the algorithm has access to degree queries and neighbor queries.

The algorithms presented in this survey are randomized algorithms that are allowed a small constant failure probability (e.g., 1/3). This failure probability can be reduced in a standard manner to any desired value $\delta > 0$ at a multiplicative cost of $\log(1/\delta)$. The algorithms compute approximations of various graph parameters. Ideally, we would like to design algorithms that, given any $\epsilon \in (0, 1)$, compute an approximation that is within a multiplicative factor of

D. Ron—Research supported by the Israel Science Foundation grant no. 671/13.

[1] Observe that a degree query to a vertex v can be replaced by $O(\log d(v))$ neighbor queries to v by performing a "doubling" search. For the sake of simplicity we allow both types of queries.

B. Steffen and G. Woeginger (Eds.): Computing and Software Science, LNCS 10000, pp. 105–122, 2019.
https://doi.org/10.1007/978-3-319-91908-9_7

$(1\pm\epsilon)$ from the exact value of the graph parameter in question, and furthermore, their complexity grows polynomially with $1/\epsilon$. While some of the algorithms have this desired behavior, others provide weaker approximations, as we detail when presenting the corresponding results.

In the rest of this section, we present a variety of results for sublinear approximation of graph parameters. In the sections that follow we give more details for a selection of these results.

1.1 Average Degree and Higher Moments of the Degree Distribution

The Average Degree. The problem of estimating the average degree $\overline{d} = \overline{d}(G)$ of a graph G in sublinear time was first studied by Feige [10]. He considered this problem when the algorithm is allowed only degree queries, so that the problem is a special case of estimating the average value of a function given query access to the function. For a general function $d : \{1, \ldots, n\} \to \{0, \ldots, n-1\}$, obtaining a constant-factor estimate of the average value of the function (with constant success probability) requires $\Omega(n)$ queries to the function (and this remains true even if there is a promise that the average value is at least 1). Feige showed that when d is the degree function of a graph, for any $\epsilon \in (0, 1]$ it is possible to obtain an estimate $\widetilde{d}$ such that $\widetilde{d} \in [\overline{d}, (2+\epsilon)\cdot\overline{d}]$ with probability at least $2/3$ by performing $O(\sqrt{n}/\epsilon)$ (uniformly selected) queries. He also showed that in order to go below a factor of 2 in the quality of the estimate, $\Omega(n)$ queries are necessary.

However, given that the object in question is a graph, it is natural to allow the algorithm to query the neighborhood of vertices of its choice and not only their degrees; indeed, the aforementioned problem definition follows this natural convention. Goldreich and Ron [14] showed that by giving the algorithm this extra power, it is possible to break the factor-2 barrier. They provide an algorithm that, given $\epsilon \in (0, 1)$, outputs a $(1 \pm \epsilon)$-factor estimate of the average degree (with probability at least $2/3$) after performing $\widetilde{O}_\epsilon(n^{1/2})$ degree and neighbor queries, assuming $\overline{d} \geq 1$. (We use $\widetilde{O}_\epsilon(\cdot)$ to suppress both $\mathrm{poly}(\log n)$ factors and $\mathrm{poly}(1/\epsilon)$ factors.) More precisely, the number of queries and the running time are $\widetilde{O}_\epsilon((n/\overline{d})^{1/2})$ in expectation. Thus, the complexity decreases as the average degree increases. Furthermore, this result is essentially optimal [14]: a $(1 \pm \epsilon)$-factor estimate requires $\Omega((n/(\epsilon\overline{d}))^{1/2})$ queries.

Higher Moments. For a graph $G = (V, E)$, consider the sum (average) of higher powers of the vertices' degrees: for $s \geq 1$ we let $M_s = M_s(G) \stackrel{\text{def}}{=} \sum_{v\in V} d(v)^s$ and $\mu_s = \mu_s(G) \stackrel{\text{def}}{=} \frac{1}{n} \cdot M_s(G)$. Observe that for $s = 1$ we have that $\mu_1 = \overline{d}$ (and $M_1 = 2m$ where m is the number of edges in the graph), while for $s = 2$, the variance of the degree distribution is $\mu_2 - \mu_1^2$.

Gonen et al. [15] gave a sublinear-time algorithm for approximating μ_s. Technically, their algorithm approximates the number of stars in a graph (with a given

size s), but a simple modification yields an algorithm for moments estimation. A much simpler algorithm (and analysis) was later given by Eden et al. [9] with essentially the same complexity (the dependence on $1/\epsilon$, s and the $\text{poly}(\log n)$ factors are reduced in [9]). Both papers show how to obtain a $(1 \pm \epsilon)$-factor approximation of μ_s by performing $\widetilde{O}_\epsilon\left(\frac{n^{1-\frac{1}{s+1}}}{\mu_s^{\frac{1}{s+1}}} + \min\left\{n^{1-\frac{1}{s}}, \frac{n^{s-1}}{\mu_s^{1-\frac{1}{s}}}\right\}\right)$ queries in expectation, where this bound is essentially optimal [15] (up to a dependence on $1/\epsilon$ and polylogarithmic factors in n). For example, when $s = 2$ this function behaves as follows. For $\mu_2 \leq n^{1/2}$ the bound is $n^{2/3}/\mu_2^{1/3}$, for $n^{1/2} < \mu_2 \leq n$, the bound is $n^{1/2}$, and for $\mu_2 > n$, it is $n/\mu_2^{1/2}$.

Aliakbarpour et al. [1] consider a stronger model that assumes access to uniform random *edges*. They show that in this model $\widetilde{O}_\epsilon\left(\frac{m}{M_s^{1/s}} + n^{1-1/s}\right) = \widetilde{O}_\epsilon\left(n^{1-1/s} \cdot \max\left\{1, \overline{d} \cdot \mu_s^{-\frac{1}{s}}\right\}\right)$ queries suffice for $s > 1$.

The Lower Bound and Graphs with Bounded Arboricity. The lower bound constructions showing that the complexity of the aforementioned algorithms for approximating μ_s is essentially optimal [15], are based on "locally dense" graphs. In particular, the first (and simpler) lower bound (corresponding to the first term, $n^{1-\frac{1}{s+1}}/\mu_s^{\frac{1}{s+1}} = n/M_s^{\frac{1}{s+1}}$), is simply based on the difficulty of "hitting" a clique of size $M_s^{\frac{1}{s+1}}$, and the second lower bound (corresponding to the second term), is based on a complete bipartite subgraph. A natural question is whether we can get a better upper bound if we know that there are no dense subgraphs. This question was answered affirmatively by Eden et al. [9]. They showed that a significantly improved complexity can be obtained for graphs with bounded arboricity.[2] For precise details see [9].

Number of Triangles and Larger Cliques. Gonen et al. [15] also considered the problem of approximating the number of triangles in a graph G, denoted $t = t(G)$. They showed a linear lower bound when the algorithm may use degree and neighbor queries and $m = \Theta(n)$. This raises the natural question whether a sublinear bound can be obtained if the algorithm is also allowed pair-queries (which are not helpful in the case of moments estimation). This question was answered affirmatively by Eden et al. [7]. They gave an algorithm whose query complexity and running time are $\widetilde{O}_\epsilon\left(\frac{n}{t^{1/3}} + \frac{m^{3/2}}{t}\right)$ in expectation. To be precise, in the expression for the query complexity, the second term is $\min\{m, m^{3/2}/t\}$ (so that the number of queries is at most linear, and is strictly sublinear as long as $t > m^{1/2}$). This bound on the query complexity is tight (up to factors polynomial in $\log n$ and $1/\epsilon$) [7]. The result was recently extended to approximating the number of k-cliques [8], for any given $k \geq 3$.

[2] The *arboricity* of a graph G, denoted $\text{arb}(G)$, is the minimum number of forests into which its edges can be partitioned. It satisfies [20,21] $\text{arb}(G) = \max_{S \subseteq V}\left\{\left\lceil \frac{|E(S)|}{|S|-1} \right\rceil\right\}$, where $E(S)$ denotes the set of edges in the subgraph induced by S.

1.2 The Number of Connected Components

The problem of approximating the number of connected components in a graph was addressed by Chazelle et al. [4] in the course of designing an algorithm for approximating the minimum weight of a spanning tree. We discuss the latter problem in Subsect. 1.4. Their algorithm for approximating the number of connected components of a graph G, denoted $\mathrm{cc}(G)$, outputs an estimate that with probability at least $2/3$ is within an additive error of ϵn from $\mathrm{cc}(G)$ (for any given $\epsilon \in (0,1)$). The query complexity and running time of the algorithm are $\widetilde{O}(\overline{d}/\epsilon^2)$.

1.3 Minimum Vertex Cover and Related Parameters

Let $\mathrm{vc}(G)$ denote the minimum size of a vertex cover in a graph G. The problem of approximating $\mathrm{vc}(G)$ in sublinear time was first studied by Parnas and Ron [24]. They showed how to obtain an estimate $\widehat{\mathrm{vc}}$ that with probability at least $2/3$ satisfies $\widehat{\mathrm{vc}} \in [\mathrm{vc}(G), 2 \cdot \mathrm{vc}(G) + \epsilon n]$. The query complexity and running time of the algorithm are $d^{O(\log d/\epsilon^3)}$ where d is the maximum degree in the graph. The dependence on d can be replaced by a dependence on $\overline{d}/\epsilon$ (recall that $\overline{d}$ denotes the average degree in the graph) [24]. It is also possible to replace the combination of the multiplicative factor of 2 and the additive term of ϵn by a multiplicative factor of $2+\epsilon$ at a cost that depends on $n/\mathrm{vc}(G)$ (and such a cost is unavoidable).

The upper bound of $d^{O(\log d/\epsilon^3)}$ was significantly improved in a sequence of papers [19,22,23,27]. The best result, appearing in [23] (and building on [22] and [27]), gives an upper bound of $\widetilde{O}(\overline{d}/\epsilon^{O(1)})$.

On the negative side, it was also proved in [24] that at least a linear dependence on the average degree, $\overline{d}$, is necessary. Namely, $\Omega(\overline{d})$ queries are necessary for obtaining an estimate $\widehat{\mathrm{vc}}$ that satisfies (with probability at least $2/3$) $\widehat{\mathrm{vc}} \in [\mathrm{vc}(G), \alpha \cdot \mathrm{vc}(G) + \epsilon n]$ for any $\alpha \geq 1$ and $\epsilon < 1/4$, provided that $\overline{d} = O(n/\alpha)$. In particular this is true for $\alpha = 2$. We also mention that obtaining such an estimate with $\alpha = 2-\gamma$ for any constant γ and sufficiently small constant ϵ requires $\Omega(\sqrt{n})$ queries, as shown by Trevisan (see [24]). For $\alpha < 7/6$, the lower bound [3] is $\Omega(n)$.

Improved Approximation for Restricted Families of Graphs. Hassidim et al. [16] introduced the notion of a *Partition Oracle*, and showed how it can be applied to solve a variety of testing and approximation problems in sublinear time (possibly under a promise that the graph belongs to a certain restricted family of graphs). In particular, for graphs with excluded minors[3] (of constant size, e.g., planar graphs), this implies an algorithm that computes an estimate $\widehat{\mathrm{vc}}$ that satisfies (with probability at least $2/3$) $\widehat{\mathrm{vc}} \in [vc, \mathrm{vc} + \epsilon n]$ (i.e., with no multiplicative factor). The query complexity and running time of the algorithm are $O(d^{\mathrm{poly}(1/\epsilon)})$. An improved partition oracle presented in [17] implies that an

[3] A graph H is a minor of a graph G if H can be obtained from a subgraph of G by a sequence of edge contractions.

estimate with the same quality can be obtained in time $O((d/\epsilon)^{O(\log(1/\epsilon))}) = O(d^{\log^2(1/\epsilon)})$. Similar results hold for the size of a minimum dominating set and maximum independent set.

Maximum Matching. The aforementioned algorithms for approximating $\mathrm{vc}(G)$ work by approximating the size of a maximal matching. Nguyen and Onak [22] showed how such an approximation can be extended and used in a recursive manner (based on *augmenting paths* for matchings) so as to obtain an estimate $\widehat{\mathrm{mm}}$ of the *maximum* size of a matching in a graph G, denoted $\mathrm{mm}(G)$. The estimate satisfies $\widehat{\mathrm{mm}} \in [\mathrm{mm}(G) - \epsilon n, \mathrm{mm}(G)]$ with probability at least $2/3$. The query complexity of the algorithm is $2^{d^{O(1/\epsilon)}}$ in expectation. This result was improved by Yoshida et al. [27] to $d^{O(1/\epsilon^2)}$.

1.4 Minimum Weight Spanning Tree

Chazelle et al. [4] studied the problem of approximating the minimum weight of a spanning tree in an edge-weighted graph. For a (connected) graph $G = (V, E)$ with an associated weight function w over E, let $\mathrm{st}(G, w)$ denote the minimum weight of a spanning tree in G (according to the weight function w). Assuming $w(e) \in \{1, \ldots, W\}$ for an integer W and every $e \in E$, they show how to obtain an estimate $\widehat{\mathrm{st}}$ that satisfies $\widehat{\mathrm{st}} \in [\mathrm{st}(G, w), (1+\epsilon)\mathrm{st}(G, w)]$ with high constant probability by performing $\widetilde{O}\left(\frac{\overline{d} \cdot W}{\epsilon^2}\right)$ queries. Here a query for a neighbor of a given vertex v also returns the weight of the corresponding edge. They also give an almost-matching lower bound of $\Omega\left(\frac{\overline{d} \cdot W}{\epsilon^2}\right)$. The algorithm can be extended to the case of non-integer weights in the range $[1, W]$ (by discretization of the edge weights).[4]

The problem of approximating the minimum weight of a spanning tree when the distance function is a metric was studied by Czumaj and Sohler [6], and for the special case of the Euclidean metric, by Czumaj et al. [5].

1.5 Distance to Properties

Another type of graph parameter is the distance of a graph to having a particular property. Distance is measured in terms of the fraction of edges that need to be added and/or removed so that the graph obtains the property. Distance approximation was first explicitly introduced by Parnas et al. [25] (together with tolerant property testing).

In what is known as the *dense-graphs model*, a distance-approximation algorithm for a graph property $\mathcal{P}$ may perform vertex-pair queries, and is given an approximation parameter $\epsilon \in (0, 1)$. It should output an estimate of the distance to the property that is within $\pm\epsilon n^2$ from the true value with probability at least

[4] In CRT it was shown how this can be done at a cost of $1/\epsilon$ in the query complexity, and Bansal [2] showed how this cost can be avoided.

2/3. Hence a distance-approximation algorithm is a generalization of a (graph) property-testing algorithm (as defined in [12]). A property-testing algorithm should distinguish between the case that the graph has the property (distance 0), and the case in which it has distance greater than ϵ to the property.

As observed in [25], for some graph properties, known algorithms for property testing in the dense-graphs model presented in [12] immediately imply distance approximation algorithms. In particular this holds for a variety of *Graph Partitioning* properties (such as bipartiteness, and more generally, k-colorability), where the query complexity is polynomial in $1/\epsilon$. (Assuming $\mathcal{P} \neq \mathcal{NP}$, the running time cannot be polynomial in $1/\epsilon$.) Fischer and Newman [11] proved that every property that has a testing algorithm in the dense-graphs model whose query complexity depends only on $1/\epsilon$, has a distance approximation algorithm whose query complexity depends only on $1/\epsilon$ (though the dependence may be quite high (e.g., a tower function)).

Marko and Ron [19] studied distance approximation for bounded-degree and unbounded-degree sparse graphs. In both cases the algorithms can perform neighbor and degree queries. For graphs with a degree bound d, distance is measured with respect to $d \cdot n$, while when there is no bound on the degree, distance is measured with respect to a given upper bound on the number of edges. They present several distance approximation algorithms for properties that have testing algorithms [13], such as k-connectivity and subgraph-freeness.

1.6 Organization

Following a preliminaries section, in Sect. 3 we describe an algorithm for approximating the average degree $\overline{d}$, and more generally, μ_s for $s \geq 1$. In Sect. 4 we give two algorithms for approximating the minimum size of a vertex cover, and in Sect. 5 we describe an algorithm for approximating the minimum weight of a spanning tree. Due to space constraints, some analysis details are omitted.

2 Preliminaries

For an integer s, we let $[s] \stackrel{\text{def}}{=} \{1, \ldots, s\}$. Let $G = (V, E)$ be an undirected graph, which, unless stated otherwise, is simple and unweighted. We denote the number of vertices in G by n and the number of edges by m. Each vertex $v \in V$ is associated with a unique id, denoted $\mathrm{id}(v)$. For a vertex $v \in V$, we let $\Gamma(v)$ denote its set of neighbors, and let $d(v)$ denote its degree. We denote the maximum degree in the graph by $d = d(G)$, and the average degree by $\overline{d} = \overline{d}(G)$. We assume it is possible to uniformly select a vertex in V, and for any vertex $v \in V$ to obtain its degree $d(v)$ (referred to as a *degree query*), as well as its i^{th} neighbor for any $i \in [d(v)]$ (referred to as a *neighbor query*), all at unit cost.

3 Moments of the Degree Distribution

3.1 Average Degree

We start by considering the average degree, $\overline{d} \stackrel{\text{def}}{=} \frac{1}{n}\sum_{v\in V} d(v)$. In what follows we present an algorithm due to [9] (which is a variant of the algorithm appearing in [26]).

This algorithm and its analysis are more elegant than what appears in [14], and also serve as an introduction to higher moments. We have chosen to combine the presentation of the algorithm and its analysis, since we believe it better brings out the ideas behind them. The more general algorithm, for higher moments, is presented in a more conventional and formal manner in Subsect. 3.2.

In what follows we assume we have a "rough" constant factor estimate $\widetilde{m}$ of the number of edges in the graph. That is, $\widetilde{m} = \Theta(m)$. We describe an algorithm that, given such an estimate $\widetilde{m}$, computes a "refined" estimate of $\overline{d} = 2m/n$ that is within $(1 \pm \epsilon)$ of $\overline{d}$ (for any given approximation parameter $\epsilon \in (0, 1)$). In fact, to ensure its correctness, the algorithm only requires that $\widetilde{m} = O(m)$. Furthermore, if $\widetilde{m} = \Omega(m)$, then with high probability it will not overestimate $\overline{d}$ (but may underestimate it). The running time of the algorithm is $\widetilde{O}_\epsilon((n/\widetilde{m})^{1/2})$. Hence, the assumption regarding $\widetilde{m}$ can be removed by a geometric search, as shown in [14, Sect. 3.1.2].

Weight Assignment. Consider assigning each edge $e \in E$ to its endpoint that has *smaller degree*, breaking ties arbitrarily (e.g., by the ids of the vertices). Let the *weight* of vertex v, denoted $w(v)$, be twice[5] the number of edges assigned to v. Observe first that since each edge is assigned to exactly one vertex, $\sum_{v\in V} w(v) = 2m$. Next, observe that $w(u) \leq 2(2m)^{1/2}$ for every vertex u. This is true since $w(u) \leq 2 \cdot d(u)$, and for each of the $w(u)/2$ edges $\{u, v\}$ that are assigned to u, we have that $d(u) \leq d(v)$. Hence, if $w(u) > 2(2m)^{1/2}$ for some u, then $\sum_{v\in V} d(v) > (w(u)/2)\cdot d(u) > 2m$, contradicting the fact that $\sum_{v\in V} d(v) = 2m$.

The algorithm starts by selecting r vertices, uniformly, independently, at random, where $r = \frac{c_r \cdot n}{\widetilde{m}^{1/2}} \cdot \frac{1}{\epsilon^2} = \Theta\left(\frac{n^{1/2}}{\overline{d}^{1/2}} \cdot \frac{1}{\epsilon^2}\right)$ and c_r is a (sufficiently large) constant. Let $R = \{u_1, \ldots, u_r\}$ denote the multiset of vertices selected, and let $w(R) \stackrel{\text{def}}{=} \sum_{i=1}^{r} w(u_i)$. By the definition of the weight of vertices, since each u_i is selected uniformly at random, $\text{Exp}[w(u_i)] = 2m/n = \overline{d}$ for each $i \in [r]$, so that $\text{Exp}_R\left[\frac{1}{r} \cdot w(R)\right] = \overline{d}$.

We can now apply the multiplicative Chernoff bound on the sum of the random variables $X_i = w(u_i)$, which satisfy $\text{Exp}[X_i] = \overline{d}$ and $X_i \in [0, 2(2m)^{1/2}]$. By our choice of r and the assumption that $\widetilde{m} = \Theta(m) = \Theta(\overline{d} \cdot n)$, we get that

$$\Pr_R\left[\left|\frac{1}{r} \cdot w(R) - \overline{d}\right| > \epsilon\overline{d}\right] \;<\; 2\exp\left(\frac{-r \cdot \overline{d} \cdot \epsilon^2}{3 \cdot 2(2m)^{1/2}}\right) \;<\; 1/10\,,$$

[5] The factor of 2 is due to the relation between the number of edges and the average degree, as well as for the sake of consistency with the higher moments algorithm.

where the last inequality is for a sufficiently large constant c_r in the setting of the sample size r. The above implies that if we had an oracle for the weight function over vertices, we could compute $w(R)$ and simply output $\frac{1}{r} \cdot w(R)$. Unfortunately, we do not have such an oracle, and furthermore, it is not even clear how to approximate $w(u_i)$ for all $u \in R$ in an efficient manner. Therefore, we approximate $w(R)$ in a different manner, as described next, conditioning on the event that $w(R) = (1 \pm \epsilon) \cdot \overline{d} \cdot r$ (which holds with probability at least 9/10). In what follows we assume without loss of generality that $\epsilon \leq 1/2$ (or else we set $\epsilon = 1/2$).

Approximating $w(R)$. Let $E(R)$ denote the multiset of *ordered* pairs, (u, v) such that $u \in R$ and $\{u, v\} \in E$. Note that if u and v both belong to R, then $E(R)$ contains both (u, v) and (v, u). Consider next "spreading" the weight of the vertices in R onto $E(R)$. Namely, for each $(u, v) \in E(R)$, if $d(u) < d(v)$ or $d(u) = d(v)$ and $\mathrm{id}(u) < \mathrm{id}(v)$, then $w(u, v) = 2$, and otherwise, $w(u, v) = 0$. By this definition,

$$w(R) = \sum_{(u,v) \in E(R)} w(u, v) \ .$$

The benefit of moving the assignment of weight from vertices to (ordered) edges, is that for any edge (u, v), we can determine whether $w(u, v) = 2$ or $w(u, v) = 0$ by simply performing two degree queries. Note that $|E(R)| = \sum_{u \in R} d(u)$, which can be computed by performing degree queries on all vertices in R. Also note that we can select a pair $(u, v) \in E(R)$ uniformly at random as follows: Select a vertex $u \in R$ with probability $d(u)/|E(R)|$, select $i \in [d(u)]$ uniformly at random, and query the i^{th} neighbor of u to obtain (u, v). Finally, observe that $\mathrm{Exp}[|E(R)|] = \overline{d} \cdot r$, and by Markov's inequality, $|E(R)| \leq 10 \cdot \overline{d} \cdot r$ with probability at least 9/10. From this point on, we condition on the last event, in addition to $w(R) = (1 \pm \epsilon) \cdot \overline{d} \cdot r$, which gives us that $w(R)/|E(R)| \geq 1/20$.

Armed with the ability to uniformly sample (ordered) edges from $E(R)$ and obtain their weight, the algorithm selects, uniformly, independently, at random, $q = c_q/\epsilon^2$ edges in $E(R)$ (for an appropriate constant c_q), and sums their weights. Let the sum be denoted by X. By the above discussion, the expected value of X/q is $\frac{w(R)}{|E(R)|}$, which is at least 1/20. By applying the multiplicative Chernoff bound, we get that X/q is within $(1 \pm \epsilon)$ from this expected value with probability at last 9/10. Hence, the algorithm outputs $\frac{|E(R)|}{q \cdot r} \cdot X$ as its estimate for $\overline{d}$.

By summing the probabilities of three "bad" events ((1) $w(R)$ deviates from $\overline{d} \cdot r$ by more than a factor of $(1 \pm \epsilon)$; (2) $|E(R)| > 10 \cdot \overline{d} \cdot r$; (3) X/q deviates from $\frac{w(R)}{|E(R)|}$ by more than a factor of $(1 \pm \epsilon)$), we get that

$$\frac{|E(R)|}{q \cdot r} \cdot X = (1 \pm \epsilon) \cdot \frac{w(R)}{r} = (1 \pm 3\epsilon) \cdot \overline{d} \ ,$$

with probability at least 2/3. By running the algorithm with $\epsilon/3$ instead of ϵ, we obtain the desired accuracy.

Since the geometric search for a "rough" constant factor estimate $\widetilde{m}$ for m increases the complexity of the algorithm by a multiplicative factor of $\mathrm{poly}(\log n)$ (in expectation), we get the following theorem.

Theorem 1. *There exists an algorithm that, given query access to a graph $G = (V, E)$ and an approximation parameter $\epsilon \in (0, 1)$, returns a value that belongs to $[(1-\epsilon)\cdot\overline{d}, (1+\epsilon)\cdot\overline{d}]$ with probability at least $2/3$. The expected query complexity and running time of the algorithm are $\widetilde{O}_\epsilon\left(\frac{n^{1/2}}{\overline{d}^{1/2}}\right)$.*

3.2 Higher Moments

For $s \geq 1$, we consider the sum over all vertices, of their degrees to the power of s, denoted $M_s \stackrel{\text{def}}{=} \sum_v d(v)^s$ and let $\mu_s \stackrel{\text{def}}{=} \frac{1}{n} \cdot M_s$ (so that in particular, $M_1 = 2m$ and $\mu_1 = \overline{d}$). As done implicitly in the case of $s = 1$ (described in Subsect. 3.1), we consider an ordering, denoted $\prec$, over the graph vertices, where $u \prec v$ if $d(u) < d(v)$ or $d(u) = d(v)$ and $id(u) < id(v)$. Our algorithm is given in Fig. 1, and as can be seen, generalizes the algorithm described in Subsect. 3.1. The sample sizes r and q will be determined in the analysis (see the statement of Theorem 2).

Algorithm 1 (An algorithm for approximating μ_s)

1. *Select r vertices, uniformly, independently, at random and denote the resulting multi-set by R. Query the degree of each vertex in R, and let $d(R) = \sum_{v\in R} d(v)$.*
2. *For $i = 1, \ldots, q$ do:*
   (a) *Select a vertex $u_i \in R$ with probability proportional to its degree (i.e., with probability $d(u_i)/d(R)$), and query for a random neighbor v_i of u_i.*
   (b) *If $u_i \prec v_i$, then $X_i = (d^{s-1}(u_i) + d^{s-1}(v_i))$, otherwise, $X_i = 0$.*
3. *Return $X = \frac{1}{r} \cdot \frac{d(R)}{q} \cdot \sum_{i=1}^{q} X_i$.*

Fig. 1. An algorithm for approximating μ_s.

Here too we assign weights to vertices (and to edges), so that when summing over the weights of all vertices (similarly, all edges) we get M_s. We first introduce some notations. Let $\Gamma_\succ(u) = \{v \in \Gamma(u) : v \succ u\}$, $\Gamma_\prec(u) = \Gamma(u) \setminus \Gamma_\succ(u)$, $d_\succ(u) = |\Gamma_\succ(u)|$ and $d_\prec(u) = |\Gamma_\prec(u)|$. For each vertex u let

$$w_s(u) \stackrel{\text{def}}{=} \sum_{v\in\Gamma_\succ(u)} \left(d(u)^{s-1} + d(v)^{s-1}\right) ,$$

and observe that for $s = 1$ the weight of a vertex u equals $2d_\succ(u)$, which fits the definition in Subsect. 3.1. Taking the sum over all vertices we get

$$\sum_{u \in V} w_s(u) = \sum_{u \in V} d_{\succ}(u) \cdot d(u)^{s-1} + \sum_{v \in V} d_{\prec}(v) \cdot d(v)^{s-1} = M_s. \qquad (1)$$

For a multi-set of vertices R, let $w_s(R) \stackrel{\text{def}}{=} \sum_{u \in R} w_s(u)$, and let $E(R)$ be as defined in Subsect. 3.1 (i.e., $E(R) \stackrel{\text{def}}{=} \{(u,v) : u \in R, \{u,v\} \in E\}$). Observe that if for each $(u,v) \in E(R)$ we define $w_s(u,v) = d(u)^{s-1} + d(v)^{s-1}$ when $u \prec v$ and $w_s(u,v) = 0$ otherwise, then $w_s(R) = \sum_{(u,v) \in E(R)} w_s(u,v)$.

The next lemma provides a bound on the maximum weight of a vertex (recall that for $s = 1$ the bound was $O(m^{1/2})$). It is proved by separately considering "low-degree" vertices and "high-degree" vertices, where the degree threshold is $M_s^{\frac{1}{s+1}}$.

Lemma 1. *For every vertex $v \in V$ we have that $w_s(v) \leq 4M_s^{\frac{s}{s+1}}$.*

Lemma 1 is the main ingredient in the proof of the next theorem.

Theorem 2. *If $r = \frac{c_r \cdot n}{\epsilon^2 \cdot M_s^{\frac{1}{s+1}}}$ for a sufficiently large constant c_r, and $q = \min\left\{n^{1-\frac{1}{s}}, \frac{n^{s-\frac{1}{s}}}{M_s^{1-\frac{1}{s}}}\right\} \cdot \frac{c_q}{\epsilon^2}$ for a sufficiently large constant c_q, then for X as defined in Step 3 of Algorithm 1, $X \in [(1-2\epsilon)\mu_s, (1+3\epsilon)\mu_s]$ with probability at least 2/3.*

Proof Sketch: Lemma 1 implies that for r as stated in the theorem, with high constant probability, the sample R is such that $w(R)$ is close to its expected value, $r \cdot \mu_s$. The size of r also ensures that with high constant probability $d(R)$ (as defined in Algorithm 1) is not much larger than its expected value, $r \cdot \overline{d}$. Conditioned on these two events we get that $\mathrm{Exp}[X_i] = \frac{w_s(R)}{d(R)} = \Omega\left(\frac{M_s}{m}\right)$, for the random variables X_i defined in Step 2b of Algorithm 1. Since it can be shown that $m \leq M_s^{\frac{1}{s}} \cdot n^{1-\frac{1}{s}}$, we get that $\mathrm{Exp}[X_i] = \Omega\left(\frac{M_s^{1-\frac{1}{s}}}{n^{1-\frac{1}{s}}}\right)$. We also use the fact that each X_i is upper bounded by $2\max_v\{d(v)^{s-1}\} \leq 2\min\{M_s^{1-\frac{1}{s}}, n^{s-1}\}$ (since $d(v) \leq M_s^{1/s}$ and $d(v) < n$). By the multiplicative Chernoff bound we get that for a sufficiently large constant c_q in the setting of q, $\Pr\left[\left|\frac{1}{q}\sum_{i=1}^{q} X_i - \frac{w(R)}{d(R)}\right| > \epsilon \cdot \frac{w(R)}{d(R)}\right] < \frac{1}{10}$, and the theorem follows. □

As stated in Theorem 2, the sample sizes r and q used in the algorithm depend on M_s. Similarly to the case of the average degree ($s = 1$), a constant factor estimate suffices, and such an estimate can be found by performing a geometric search (at a multiplicative cost of $\mathrm{poly}(s \log n, 1/\epsilon)$ [9, Sect. 6]), obtaining the following theorem:

Theorem 3. *There exists an algorithm that, given query access to a graph $G = (V,E)$ and an approximation parameter $\epsilon \in (0,1)$, returns a value that belongs to $[(1-\epsilon) \cdot \mu_s, (1+\epsilon) \cdot \mu_s]$ with probability at least*

2/3. *The expected query complexity and running time of the algorithm are* $O\left(\frac{n^{1-\frac{1}{s+1}}}{\mu_s^{\frac{1}{s+1}}} + \min\left\{n^{1-\frac{1}{s}}, \frac{n^{s-1}}{\mu_s^{1-\frac{1}{s}}}\right\} \cdot \mathrm{poly}(s \log n, 1/\epsilon)\right)$.

4 Minimum Vertex Cover and Maximum Matching

There are several approaches to the problem of approximating the size of a minimum vertex cover in sublinear time. Here we present two. Both are based on the relation between vertex covers and matchings. Namely, for any graph $G = (V, E)$ and any matching $M \subseteq E$, the size of a minimum vertex cover of G, denoted $\mathrm{vc}(G)$, satisfies $\mathrm{vc}(G) \geq |M|$ (because any vertex cover must include at least one endpoint of every edge in M). Furthermore, if M is a maximal matching, then $\mathrm{vc}(G) \leq 2|M|$ (because taking both endpoints of each edge in M gives us a vertex cover). Both algorithms provide an estimate $\widehat{\mathrm{vc}}$, that with high constant probability satisfies $\widehat{\mathrm{vc}} \in [\mathrm{vc}(G), 2\mathrm{vc}(G) + \epsilon n]$. The algorithms are described for bounded degree graphs, where their complexity depends on the degree bound, d. They can be adapted to work with bounded average degree, $\overline{d}$, as we discuss shortly following Theorem 5.

4.1 Building on a Distributed Algorithm

In this subsection we describe an algorithm that is due to [19]. The basic underlying idea (first applied in [24]) is to transform a local distributed algorithm into a sublinear algorithm. Recall that in the local distributed model, there is a processor residing on each vertex, and the computation proceeds in rounds. In each round, each vertex can send messages to all its neighbors. When the computation ends, each vertex knows "its part" of the output, where in the case of the computation of a vertex cover, it knows whether or not it belongs to the cover.

The distributed algorithm described in [19] is similar to the $O(\log n)$-rounds distributed algorithm for the maximal independent set of Luby [18]. The algorithm, presented in Fig. 2, is described as if there is a processor assigned to every edge, but clearly this can be emulated by processors that are assigned to the vertices. For an edge $e = \{u, v\}$, we let $d(e) \stackrel{\mathrm{def}}{=} d(u)+d(v)$ denote the number of edges that have a common endpoint with e, which are considered to be its neighbors.

In the course of the algorithm (described in Fig. 2), the edges (processors assigned to them) make various decisions (to activate/inactivate themselves, to select/un-select themselves, and to add their endpoints to the cover). Following each such decision, a corresponding message is sent to all neighboring edges (this notification is not stated explicitly in the algorithm). On a high level, the algorithm works in iterations, where in each iteration a new subset of vertices is added to the cover C (based on a certain (distributed) random process). In each iteration, the vertices added to C constitute endpoints of a matching. After the last iteration, for each edge that has remained uncovered by C, one of its endpoints is added to C. In the analysis of the algorithm, we show that with high probability (over the random selection process), the number of edges remaining in the final stage is small.

Theorem 4. *For every graph $G = (V, E)$ with degree-bound d and every $\delta > 0$, Algorithm 2 constructs a vertex cover $C \subseteq V$ such that with probability at least $5/6$, $|C| \in [\mathrm{vc}(G), 2 \cdot \mathrm{vc}(G) + \delta n]$.*

Algorithm 2 (Distributed approximation for minimum vertex cover)

1. *Each edge initially activates itself.*
2. *From $i = 1$ to $r = 16 \cdot \log(6d/\delta)$:*
   (a) *Each active edge e selects itself with probability $\frac{1}{4 \cdot d(e)}$. If $d(e) = 0$ then e is selected with probability 1.*
   (b) *Every two neighboring edges that were both selected, un-select themselves.*
   (c) *Each vertex that is incident to a selected edge (that was not un-selected), adds itself to the vertex cover C.*
   (d) *All selected edges and neighbors of selected edges, inactivate themselves.*
   (e) *Active edges update their degrees to be the number of their active neighbors.*
3. *For every edge that remained active, its endpoint with the smaller id adds itself to the vertex cover C.*

Fig. 2. A distributed algorithm for an approximate minimum vertex cover.

Proof Sketch: Since an edge inactivates itself only when one of its endpoints is added to C, and in Step 3 one endpoint from each edge that is still active is added to C, all edges are covered by the end of the algorithm. Hence C is a vertex cover, and this implies the lower bound on its size.

By the definition of the algorithm, the vertices that are added to C in the r iterations of Step 2 are endpoints of a matching. Hence, their number is at most $2 \cdot \mathrm{vc}(G)$. It remains to show that with probability at least $5/6$, the number of edges that remain active at the start of Step 3 is at most δn. To this end we introduce the following notation: for each $i \in [r]$, let m_i be the number of active edges remaining at the end of the i^{th} iteration of Step 2. For $i = 0$ let $m_0 = m$. It can be shown, that given the process by which the algorithm selects and de-activates edges,

$$\mathrm{Exp}\left[m_i \mid m_{i-1}\right] \leq \left(1 - \frac{1}{16}\right) m_{i-1} \,. \tag{2}$$

The heart of the argument for establishing Eq. (2) is that the following holds for each iteration i and integer $j > 0$. If we consider at the start of iteration i an active edge e that has j active neighbors, then the probability that e is selected in iteration i and remains selected (since none of its active neighbors is selected), is $\Omega(1/j)$. But if e remains selected, then it, as well as its j active neighbors, are inactivated. The inactivation of an edge can be caused by more than one selected neighbor, but since the selected edges do not neighbor each other, an edge can be inactivated due to at most two of its neighbors. Equation (2) follows by summing the "inactivation contribution" of edges with varying numbers of (active) neighbors.

Equation (2) in turn implies that for $r = 16\log(6d/\delta)$, $\mathrm{Exp}[m_r] \leq (1-1/16)^r \cdot m < (\delta/6)n$. By Markov's inequality, $m_r \leq \delta n$ with probability at least 5/6, as desired. □

In order to estimate the size of a minimum vertex cover we apply the observation that it is possible to emulate the outcome of the distributed algorithm (Algorithm 2) at any vertex v of our choice by considering the subgraph induced by all vertices at distance at most $r+1$ from v. Since the distributed algorithm is randomized, we only need to take care to use the same coin-flips if we encounter the same vertex u in the neighborhoods of two different vertices v_1 and v_2. The sublinear approximation algorithm is given in Fig. 3.

Algorithm 3 (Sublinear Approximation for $\mathrm{vc}(G)$**, Version I)**

1. *Uniformly and independently sample* $s = 2/\epsilon^2$ *vertices from* G*. Let* $S = \{v_1, \ldots, v_s\}$ *be the multiset of the sampled vertices.*
2. *For each* $v_i \in S$*, query* G *in order to obtain the subgraph* $G_r(v_i)$ *induced by the* $(r+1)$*-neighborhood of* v_i*, where* $r = 16\log(6d/\delta)$ *is as in Algorithm 2, and* $\delta = \epsilon/2$*.*
3. *Run Algorithm 2 on the graph that is the union of all subgraphs* $G_r(v_i)$ *for* $v_i \in S$ *(in a sequential manner). For each* $i \in [s]$*, let* $\chi_i = 1$ *if the algorithm adds* v_i *to the cover, otherwise* $\chi_i = 0$*.*
4. *Output* $\widehat{\mathrm{vc}} = \frac{n}{s}\sum_{i=1}^{s}\chi_i + (\epsilon/2)n$ *.*

Fig. 3. A sublinear algorithm for approximating the minimum size of a vertex cover.

The next theorem follows by applying Theorem 4 together with the multiplicative Chernoff bound.

Theorem 5. *For every graph* G *with degree bound* d*, and every* $\epsilon \in (0,1]$*, Algorithm 3 outputs an estimate* $\widehat{\mathrm{vc}}$*, that with probability at least* $2/3$ *satisfies* $\widehat{\mathrm{vc}} \in [\mathrm{vc}(G), 2\cdot\mathrm{vc}(G) + \epsilon n]$*. The query and time complexity of the algorithm are* $d^{O(\log(d/\epsilon))}$*.*

We remark that the same modifications of the algorithm in [24] can be applied here to achieve a dependence on $\Theta(\overline{d}/\epsilon)$ instead of d in the query complexity. The idea is to slightly modify the distributed algorithm so that initially, each vertex with degree greater than $2\overline{d}/\epsilon$ is added to the cover, and all edges incident to these vertices are inactivated. This increases the size of the cover by an additive term of at most $\epsilon n/2$, and reduces the maximum degree in the graph induced by active edges to $2\overline{d}/\epsilon$.

4.2 Building on a Random Ordering

The local emulation of the distributed algorithm described in the previous subsection can be viewed as an *oracle* (which is randomized) for a vertex cover C.

Namely, the cover C is defined by the protocol of the distributed algorithm, and the coin flips used in the course of the execution of the distributed algorithm (that determine which edges are selected in each iteration). The oracle is given a vertex v and should answer whether $v \in C$. To this end it emulates the execution of the distributed algorithm in the neighborhood of v (flipping coins "on the fly", while keeping track of previous coin flips if needed). The sublinear algorithm for approximating the size of a minimum vertex cover can now be viewed as simply querying the oracle on $\Theta(1/\epsilon^2)$ uniformly selected vertices, and using the fraction of sampled vertices that belong to the cover to determine its estimate.

Nguyen and Onak [22] also design such a randomized oracle for a vertex cover, but their oracle is not based on a distributed algorithm but rather on the greedy *sequential* algorithm for constructing a maximal matching (and adding to the cover both endpoints of each edge in the matching). This algorithm considers an arbitrary ranking $\pi : E \to [m]$ of the edges of the graph (where each edge is given a unique rank). In each step the algorithm checks whether the next edge according to this ranking neighbors any edge that was already added to the matching M_π (initially, M_π is empty). If not, then the new edge is added to M_π. While for different rankings π we may get a different matching M_π, we always obtain a maximal matching (and hence $|M_\pi| \in [\mathrm{vc}(G), 2\mathrm{vc}(G)]$).

Suppose we are given an edge e, and would like to determine whether $e \in M_\pi$ (without necessarily constructing the entire matching M_π). Consider the edges that neighbor e. Observe that in order to decide whether $e \in M_\pi$, it suffices to know whether any of its neighbors with *lower rank* (according to π) is in M_π. If at least one of them is, then $e \notin M_\pi$, and if none of them belong to M_π, then $e \in M_\pi$. This gives rise to the (recursively defined) oracle in Fig. 4.

Algorithm 4 (Oracle for M_π, given an edge e as input)

1. *For each edge e' neighboring e such that $\pi(e') < \pi(e)$, recursively call the oracle (Algorithm 4) on e'.*
2. *If the oracle returns TRUE for one of these neighbors, then return FALSE, else return TRUE.*

Fig. 4. An oracle of a maximal matching M_π

The first question that arises is what is the number of recursive calls that the oracle needs to perform in order to decide whether e belongs to M_π. This of course depends on $\pi(e)$ (e.g., if $\pi(e) = 1$ then there are no recursive calls) and more generally on the ranking of edges in the neighborhood of e. To be precise, if we consider the tree of recursive calls, then the paths in the tree correspond to edges with decreasing ranks. Nguyen and Onak consider a random choice of π, and analyze the expected number of recursive calls, where the expectation is taken both over the choice of π and the choice of a random edge e. Observe that if we increase the range of π from $[m]$ to $[\mathrm{poly}(m)]$, then $\pi(e)$ can be selected

on-the-fly (that is, independently for each encountered edge), with only a small probability of a collision (i.e., $\pi(e) = \pi(e')$ for $e \neq e'$).

The next lemma directly follows from Lemma 12 in [22] (a more general *Locality Lemma* regarding random rankings of edges is Lemma 4 in [22]).

Lemma 2. *Let $G = (V, E)$ be any graph with maximum degree bounded by d. For a uniformly selected ranking π over E and a uniformly selected edge $e \in E$, the expected number of recursive calls made by Algorithm 4 when called on e is $2^{O(d)}$.*

The resulting sublinear approximation algorithm for the size of a minimum vertex cover is similar to Algorithm 3, and is provided in Fig. 5.

Algorithm 5 (Sublinear Approximation for $\mathrm{vc}(G)$, Version II)

1. *Uniformly and independently sample $s = 2/\epsilon^2$ vertices from G. Let S be the multiset of the sampled vertices.*
2. *For each $v \in S$, query the maximal matching oracle (Algorithm 4) on all edges incident to v (where π is a random ranking selected on the fly by Algorithm 4). If the oracle returns TRUE on one of these edges, then set $\chi_v = 1$, otherwise $\chi_v = 0$.*
3. *Output $\widehat{\mathrm{vc}} = \frac{n}{s} \sum_{v \in S} \chi_v + (\epsilon/2)n$.*

Fig. 5. A sublinear algorithm for approximating the minimum size of a vertex cover.

The proof of the correctness of Algorithm 5, stated next, is essentially the same as the proof of Theorem 5, and the bound on the query complexity follows from Lemma 2.

Theorem 6. *For every $\epsilon > 0$, and every graph G, Algorithm 5 outputs an estimate $\widehat{\mathrm{vc}}$, that with probability at least $2/3$ satisfies $\widehat{\mathrm{vc}} \in [\mathrm{vc}(G), 2 \cdot \mathrm{vc}(G) + \epsilon n]$. The query complexity of the algorithm is $2^{O(d)}/\epsilon^2$.*

Comparing the bound in Theorem 6 to the bound in Theorem 5 we see that while the dependence on d is larger, the dependence on $1/\epsilon$ is improved. More importantly, the approach suggested in [22] led to a significant improvement in the complexity, as we discuss shortly next. Here too we remark that it is possible to achieve a dependence on $\Theta(\bar{d}/\epsilon)$ instead of d in the complexity of the algorithm.

Reducing the Query Complexity. Nguyen and Onak [22] also suggested the following variant of their algorithm. When making recursive calls on edges neighboring an edge e, perform the calls from the smallest to the largest rank. Since once some neighboring edge of e returns TRUE, we know that e should return FALSE (so that there is no need to make calls on the other neighboring

edges), they asked whether it can be proved that this variant has smaller query complexity (in expectation). A very clever analysis of Yoshida et al. [27] showed that indeed the expected number of recursive calls can be upper bounded by a polynomial in d. This yields an algorithm for approximating the size of a minimum vertex cover whose query complexity is $O(d^4/\epsilon^2)$, or $O(\overline{d}^4/\epsilon^4)$ in terms of the average degree $\overline{d}$. Onak et al. [23] showed how to further modify the algorithm so as to obtain an algorithm whose query complexity is $\widetilde{O}(\overline{d}) \cdot \mathrm{poly}(1/\epsilon))$, which almost matches the lower bound of $\Omega(\overline{d})$ for constant ϵ [24].

5 Minimum Weight Spanning Tree

In this section we present a slight variant of Chazelle et al. [4] algorithm for approximating the minimum weight of a spanning tree in an edge weighted graph with weights in $[W]$ for an integer W. We denote the minimum weight of a spanning tree by $\mathrm{st}(G, w)$ where $G = (V, E)$ is the underlying graph and $w : E \to [W]$ is the weight function.

The first idea underlying the algorithm is to reduce the problem of approximating $\mathrm{st}(G, w)$ to that of approximating the number of connected components in a graph. Specifically, for each $j \in [W]$, let $G_j = (V, E_j)$ for $E_j \stackrel{\mathrm{def}}{=} \{e \in E : w(e) \leq j\}$, and let cc_j denote the number of connected components in G_j. The next lemma relates between $\mathrm{st}(G, w)$ and $\mathrm{cc}_1, \ldots, \mathrm{cc}_{W-1}$. It can be established by recalling Kruskal's algorithm for finding a minimum-weight spanning tree.

Lemma 3. $\mathrm{st}(G, w) = n - W + \sum_{j=1}^{W-1} \mathrm{cc}_j$.

Armed with Lemma 3 it remains to show how to obtain an approximation of the number of connected components $\mathrm{cc}(H)$ of a graph H (and to apply this to the graphs $G_1, \ldots, G_{W-1}$). For the sake of simplicity, in what follows we describe an algorithm whose complexity depends on the maximum degree d (rather than the average degree $\overline{d}$, as done in [4]). The algorithm, which is due to Czumaj

Algorithm 6 (Sublinear Approximation for $\mathrm{cc}(H)$)

1. *Repeat the following $s = 1/\gamma^2$ times:*
   (a) *Select a vertex $v_i \in V$ uniformly at random.*
   (b) *Pick a random integer X_i according to the probability distribution $\Pr[X_i \geq k] = 1/k$.*
   (c) *If $X_i > B$ then set $\chi_i = 0$.*
   (d) *Else, perform a Breadth First Search (BFS) from v_i until $X_i + 1$ vertices are reached, or the BFS can reach at most X_i vertices (since v_i belongs to a connected component with at most X_i vertices). In the former case set $\chi_i = 0$ and in the latter case set $\chi_i = 1$.*
2. *Output $\widehat{\mathrm{cc}} = \frac{n}{s} \sum_{i=1}^{s} \chi_i$.*

Fig. 6. A sublinear algorithm for approximating the number of connected components in a graph H.

and Sohler [6], receives both an approximation parameter γ and a size bound B (and its performance is analyzed as a function of these two parameters).

Lemma 4. *For every graph G with degree bounded by d and for every $\gamma \in (0,1]$ and integer B, $\mathrm{Exp}[\widehat{\mathrm{cc}}] \in [\mathrm{cc}(H) - n/B, \mathrm{cc}(H)]$ and $\mathrm{Var}[\widehat{\mathrm{cc}}] \leq \gamma^2 \cdot n \cdot \mathrm{cc}(H)$. The expected number of queries performed by Algorithm 6 is $O\left(\frac{d}{\gamma^2} \log B\right)$.*

Lemma 4 can be established by a fairly standard probabilistic analysis.

In Fig. 7 we give an algorithm for approximating the minimum weight of a spanning tree by using Algorithm 6 as a subroutine.

Algorithm 7 (Sublinear Approximation for $\mathrm{st}(G,w)$)

1. *For $j = 1$ to $W-1$:*
   (a) *Run Algorithm 6 on G_j with parameters $\gamma = \epsilon/8$ and $B = 4W/\epsilon$ (the degree bound d is the maximum degree in G).*
   (b) *Let $\widehat{\mathrm{cc}}_j$ be the estimate it returns.*
2. *Output $\widehat{\mathrm{st}} = n - W + \sum_{j=1}^{W-1} \widehat{\mathrm{cc}}_j$.*

Fig. 7. A sublinear algorithm for approximating the minimum weight of a spanning tree.

The next theorem follows by applying Lemmas 3 and 4 and Chebishev's inequality.

Theorem 7. *For every edge-weighted graph G with degree bounded by d and weights in $[W]$, and for every $\epsilon \in (0,1]$ Algorithm 7 returns an estimate $\widehat{\mathrm{st}}$ that satisfies $\widehat{\mathrm{st}} \in [(1-\epsilon)\cdot \mathrm{st}(G,w), (1+\epsilon)\cdot \mathrm{st}(G,w)]$ with probability at least $2/3$. Its expected query complexity is $O\left(\frac{d \cdot W}{\epsilon^2} \log \frac{W}{\epsilon}\right)$.*

References

1. Aliakbarpour, M., Biswas, A.S., Gouleakis, T., Peebles, J., Rubinfeld, R., Yodpinyanee, A.: Sublinear-time algorithms for counting star subgraphs with applications to join selectivity estimation. Technical report 1601.04233, Arxiv (2016). To appear in Algorithmica. 107
2. Bansal, V.: Sublinear-time algorithms for estimating the weight of minimum spanning trees. Unpublished manuscript (2003). 109
3. Bogdanov, A., Obata, K., Trevisan, L.: A lower bound for testing 3-colorability in bounded-degree graphs. In: Proceedings of FOCS, Los Alamitos, CA, pp. 93–102 (2002). 108
4. Chazelle, B., Rubinfeld, R., Trevisan, L.: Approximating the minimum spanning tree weight in sublinear time. SIAM J. Comput. **34**(6), 1370–1379 (2005). 108, 109, 120
5. Czumaj, A., Ergun, F., Fortnow, L., Magen, A., Newman, I., Rubinfeld, R., Sohler, C.: Approximating the weight of the euclidean minimum spanning tree in sublinear time. SIAM J. Comput. **35**(1), 91–109 (2005). 109

6. Czumaj, A., Sohler, C.: Estimating the weight of metric minimum spanning trees in sublinear time. SIAM J. Comput. **39**(3), 904–922 (2009). 109, 120
7. Eden, T., Levi, A., Ron, D., Seshadhri, C.: Approximately counting triangles in sublinear time. SIAM J. Comput. **46**(5), 1603–1646 (2017). 107
8. Eden, T., Ron, D., Seshadhri, C.: On approximating the number of k-cliques in sublinear time. CoRR, abs/1707.04858 (2017). 107
9. Eden, T., Ron, D., Seshadhri, C.: Sublinear time estimation of degree distribution moments: the arboricity connection. In: Proceedings of ICALP, pp. 7:1–7:13 (2017). 106, 107, 110, 114
10. Feige, U.: On sums of independent random variables with unbounded variance, and estimating the average degree in a graph. SIAM J. Comput. **35**(4), 964–984 (2006). 106
11. Fischer, E., Newman, I.: Testing versus estimation of graph properties. SIAM J. Comput. **37**(2), 482–501 (2007). 110
12. Goldreich, O., Goldwasser, S., Ron, D.: Property testing and its connections to learning and approximation. J. ACM **45**, 653–750 (1998). 110
13. Goldreich, O., Ron, D.: Property testing in bounded degree graphs. Algorithmica **32**, 302–343 (2002). 110
14. Goldreich, O., Ron, D.: Approximating average parameters of graphs. Random Struct. Algorithms **32**(4), 473–493 (2008). 106, 111
15. Gonen, M., Ron, D., Shavitt, Y.: Counting stars and other small subgraphs in sublinear time. SIAM J. Discret. Math. **25**(3), 1365–1411 (2011). 106, 107
16. Hassidim, A., Kelner, J.A., Nguyen, H.N., Onak, K.: Local graph partitions for approximation and testing. In: Proceedings of FOCS, pp. 22–31 (2009). 108
17. Levi, R., Ron, D.: A quasi-polynomial time partition oracle for graphs with an excluded minor. ACM Trans. Algorithms **11**(3), 24 (2015). 109
18. Luby, M.: A simple parallel algorithm for the maximal independent set problem. SIAM J. Comput. **15**(2), 1036–1055 (1986). 115
19. Marko, S., Ron, D.: Distance approximation in bounded-degree and general sparse graphs. ACM Trans. Algorithms **5**(2), 22 (2009). 108, 110, 115
20. Nash-Williams, C.S.J.A.: Edge-disjoint spanning trees of finite graphs. J. Lond. Math. Soc. **1**(1), 445–450 (1961). 107
21. Nash-Williams, C.S.J.A.: Decomposition of finite graphs into forests. J. Lond. Math. Soc. **1**(1), 12 (1964). 107
22. Nguyen, H.N., Onak, K.: Constant-time approximation algorithms via local improvements. In: Proceedings of FOCS, pp. 327–336 (2008). 108, 109, 118, 119
23. Onak, K., Ron, D., Rosen, M., Rubinfeld, R.: A near-optimal sublinear-time algorithm for approximating the minimum vertex cover size. In: Proceedings of SODA, pp. 1123–1131 (2012). 108, 119
24. Parnas, M., Ron, D.: Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. Theoret. Comput. Sci. **381**(1–3), 183–196 (2007). 108, 115, 117, 120
25. Parnas, M., Ron, D., Rubinfeld, R.: Tolerant property testing and distance approximation. J. Comput. Syst. Sci. **72**(6), 1012–1042 (2006). 109, 110
26. Seshadhri. C.: A simpler sublinear algorithm for approximating the triangle count. CoRR, abs/1505.01927 (2015). 111
27. Yoshida, Y., Yamamoto, M., Ito, H.: An improved constant-time approximation algorithm for maximum matchings and other optimization problems. SIAM J. Comput. **41**(4), 1074–1093 (2012). 108, 109, 119