

# Distance Approximation in Bounded-Degree and General Sparse Graphs

Sharon Marko\*

Dana Ron†

June 5, 2006

## Abstract

We address the problem of approximating the distance of bounded degree and general sparse graphs from having some predetermined graph property  $\mathcal{P}$ . Namely, we are interested in sublinear algorithms for estimating the fraction of edges that should be added to / removed from a graph so that it obtains  $\mathcal{P}$ . This fraction is taken with respect to a given upper bound  $m$  on the number of edges. In particular, for graphs with degree bound  $d$  over  $n$  vertices,  $m = dn$ . To perform such an approximation the algorithm may ask for the degree of any vertex of its choice, and may ask for the neighbors of any vertex.

The problem of estimating the distance to having a property was first explicitly addressed by Parnas et. al. (*ECCC 2004*). In the context of graphs this problem was studied by Fischer and Newman (*FOCS 2005*) in the dense-graphs model. In this model the fraction of edge modifications is taken with respect to  $n^2$ , and the algorithm may ask for the existence of an edge between any pair of vertices of its choice. Fischer and Newman showed that every graph property that has a testing algorithm in this model with query complexity that is independent of the size of the graph, also has a distance-approximation algorithm with query complexity that is independent of the size of the graph.

In this work we focus on bounded-degree and general sparse graphs, and give algorithms for all properties that were shown to have efficient testing algorithms by Goldreich and Ron (*Algorithmica, 2002*). Specifically, these properties are  $k$ -edge connectivity, subgraph-freeness (for constant size subgraphs), being a Eulerian graph, and cycle-freeness. A variant of our subgraph-freeness algorithm approximates the size of a minimum vertex cover of a graph in sublinear time. This approximation improves on a recent result of Parnas and Ron (*ECCC 2005*).

---

\*This work is part of the author's MSc thesis submitted to the Computer Science Department, Weizmann Institute, Rehovot, Israel

†Dept. of EE-Systems, Tel Aviv University, Tel Aviv, Israel. This research was supported by the Israel Science Foundation, grant number 89/05.

# 1 Introduction

*Distance approximation* is an extension of *Property Testing*. Property testing algorithms are required to distinguish between the case that an object (e.g., graph) has a predetermined property  $\mathcal{P}$  and the case that it has a relatively large distance (i.e., greater than  $\epsilon$  for a given distance parameter  $\epsilon \in [0, 1]$ ) to having  $\mathcal{P}$ . Distance approximation algorithms are required to compute *an estimate of this distance* where the estimate may be up-to an additive error or up-to both an additive and a multiplicative error. Distance approximation and the closely related notion of *tolerant testing* (where the goal is to distinguish between being  $\epsilon_1$ -close and  $\epsilon_2$ -far to having the property) were first studied in [PRR]. Following that work, there have been several results on distance approximation, both positive [ACCL04, GR05, FN05] and negative [FF05]. These works considered properties of functions and strings [PRR, ACCL04, FF05, GR05], ensembles of points [PRR], and (dense) graphs [FN05].

**Distance Approximation in Dense Graphs.** In particular, Fischer and Newman [FN05] proved a general result on the relation between distance approximation and property testing in the *dense-graphs* model (introduced in [GGR98]). In this model, the distance of a graph  $G = (V, E)$  to having a property is defined as the fraction of edges that should be added/removed in order to obtain the property, where the fraction is with respect to  $n^2 = |V|^2$ . This model allows *vertex-pair* queries. That is, the algorithm may query whether there is an edge between any pair of vertices of its choice. Fischer and Newman [FN05] proved that *every* property that has a testing algorithm in the dense-graphs model whose complexity is only a function of the distance parameter  $\epsilon$ , has a distance approximation algorithm with an additive error  $\delta$  in this model, whose complexity is only a function of  $\delta$ . The dependence on  $\delta$  may be quite high (a tower of height polynomial in  $1/\delta$ ), but there is *no* dependence on the size of the graph.

**Bounded Degree and General Sparse Graphs.** The model in which Fischer and Newman obtained their result is clearly appropriate for dense graphs but not for sparse graphs. When studying property testing of sparse graphs, two models were considered (see [GR02] and [PR02]). In both models the testing algorithm may perform *degree queries* and *neighbor queries*.<sup>1</sup> That is, for any vertex  $v$  the algorithm may ask for the degree of  $v$ , and for any index  $i$  it may ask for the  $i$ th neighbor of  $v$ .<sup>2</sup> As in the dense-graphs model, the distance of a graph to having a property  $\mathcal{P}$  is defined as the fraction of edges, normalized with respect to a relevant upper bound, that should be added/removed from a graph so that it obtains  $\mathcal{P}$ .

The difference between the models is the setting of the aforementioned upper bound. In the dense graphs model, the upper bound is  $n^2$ . When dealing with bounded-degree graphs, that is, graphs whose vertices all have degree at most  $d$ , this fraction is defined with respect to  $d \cdot n$ . In general, when the degree of the vertices in the graph is not bounded, then the fraction is taken with respect to the number of edges in the graph, or an upper bound on this number. We denote the (upper bound on the) number of edges by  $m$ . We assume that the algorithm is provided with  $m$  as input. Otherwise, it is possible to obtain an estimate of the number of edges [Fei04, GR06], but this comes at a cost of  $\Theta(\sqrt{n})$  queries (in the case of sparse graphs).

**Our Results.** Focusing on properties that have efficient testing algorithms for bounded-degree and general sparse graphs, we ask which of these also have efficient distance approximation algorithms. We establish that all properties shown to have efficient property testing algorithms in [GR02] also have efficient distance

---

<sup>1</sup>A third model, appropriate for testing properties of graphs that are neither dense nor sparse [KKR04], also allows vertex-pair queries.

<sup>2</sup>If  $v$  has less than  $i$  neighbors then a special symbol is returned.

approximation algorithms. We leave open the interesting question of whether there exists a general transformation from property testing algorithms to distance approximation algorithms as in the case of dense graphs.

To state our results precisely, we introduce some notation. Recall that  $n$  denotes the number of vertices in the graph and  $m$  denotes (an upper bound on) the number of edges. In the case of bounded-degree graphs  $m = dn$  where  $d$  is the maximum degree in the graph. Unless stated otherwise, the graphs we consider are not necessarily simple (i.e., they may have parallel edges), and the multiplicity of each edge is at most  $\beta$ . Let  $\bar{d} \stackrel{\text{def}}{=} \frac{m}{n}$ , so that  $\bar{d}$  is roughly (an upper bound on) the average degree in the graph. For a property  $\mathcal{P}$  and a graph  $G$ , we let  $\Delta_{\mathcal{P}}(G)$  denote the relative *distance* of  $G$  to having the property  $\mathcal{P}$ . That is,  $m \cdot \Delta_{\mathcal{P}}(G)$  is the minimum number of edge modification that should be performed on  $G$  so that it obtains  $\mathcal{P}$ . For  $\alpha \geq 1$ , we say that an algorithm is an  $\alpha$ -*distance approximation algorithm* for a property  $\mathcal{P}$  if, for any given  $\delta \in (0, 1)$ , it outputs an estimate  $\hat{\Delta}$  such that with probability at least  $2/3$ ,  $\Delta_{\mathcal{P}}(G) - \delta \leq \hat{\Delta} \leq \alpha \cdot \Delta_{\mathcal{P}}(G) + \delta$ .<sup>3</sup> We say that it is a *distance approximation algorithm* if it is a 1-distance approximation algorithm. Note that an  $\alpha$ -distance approximation algorithm can be used to perform property testing simply by setting  $\delta = \epsilon/2$  and accepting if and only if  $\hat{\Delta} \leq \epsilon/2$ .

- **$k$ -Edge-Connectivity.** We give a distance approximation algorithm for the  $k$ -edge-connectivity property in general sparse graphs. Its query complexity and running time are  $\text{poly}(k\beta/(\delta\bar{d}))$ .
- **Subgraph-Freeness.** We give a 3-distance approximation algorithm for the triangle-freeness property in bounded-degree graphs. The query and time complexity of the algorithm are  $d^{O(\log(d/\delta))}$ . The algorithm generalizes to subgraph-freeness for any fixed (constant size) subgraph  $H$ .
- **Eulerian.** We give a distance approximation algorithm for the Eulerian property in general sparse graphs. Its query complexity and running time are  $O((\delta\bar{d})^{-4}\beta)$ .
- **Cycle-Freeness.** We give a distance approximation algorithm for the cycle-freeness property in simple bounded-degree graphs. Its query complexity and running time are  $O(\delta^{-3})$ .

By adapting our subgraph-freeness distance-approximation algorithm we can get a sublinear approximation algorithm for the size of a minimum vertex cover. Specifically, an approximation with a multiplicative error of 2 and an additive error of  $\delta n$ , is achieved in time  $d^{O(\log(d/\delta))}$ . This algorithm improves on a recent result presented in [PR05] which achieve the same approximation in time  $d^{O(\delta^{-3} \log d)}$ .

A few notes are in place:

- With the exception of subgraph-freeness, the complexity of our algorithms is polynomially related to the corresponding complexity of the testing algorithms [GR02] (where our error parameter  $\delta$  is replaced by the distance parameter  $\epsilon$ , and  $\bar{d}$  is replaced by  $d$ ).
- Approximating the distance to  $k$ -connectivity for the special case  $k = 1$  was addressed in [CRT05] as a central part of their algorithm for estimating the weight of a minimum spanning tree.
- Subgraph-freeness is the only result in which we have a multiplicative factor in addition to the additive error. In view of the work on dense graphs of [FN05], it is interesting to know whether or not the distance to subgraph-freeness can also be approximated up to any additive factor  $\delta$ , using a number of queries that is independent of  $n$ .

---

<sup>3</sup>We have chosen a non-symmetric definition in terms of the multiplicative factor  $\alpha$ . It is of course possible to use a symmetric definition, in which case a factor  $C$  approximation according to our definition is equivalent to a factor  $\sqrt{C}$  approximation in the symmetric definition. However, we find that it is less natural in our context.

- Testing subgraph-freeness in the *general sparse model* requires  $\Omega(\sqrt{n})$  queries [AKKR06], and the same is true for cycle-freeness.

**Techniques.** Among the aforementioned results, the more interesting techniques are applied in distance approximation of subgraph freeness and  $k$ -connectivity.

In particular, the *testing* algorithm for subgraph freeness is a simple “brute-force” algorithm that uniformly selects vertices and checks, using a local search, whether they belong to a forbidden subgraph. The straightforward adaptation of this testing algorithm to the approximation task would give a multiplicative approximation factor that depends on the maximum degree  $d$  (in addition to the additive error  $\delta$ ). To get a constant factor approximation that does not depend on  $d$ , we take a different approach. Our approach can be viewed as following the paradigm (presented in [PR05]) for transforming local distributed algorithms into sublinear algorithms (e.g., for the minimum vertex cover).

Specifically, we first present an algorithm that inspects the whole graph, but it is essentially a local algorithm in which vertices perform local computations. We later transform this algorithm into a sublinear approximation algorithm. An approximation of the size of the minimum vertex cover can be obtained using modifications of the subgraph-freeness algorithm. The algorithm is similar in spirit to the  $O(\log n)$ -rounds distributed approximation algorithm for the maximal independent set [Lub86].

In the case of  $k$ -connectivity ( $k > 1$ ), a relatively direct adaptation of the algorithm in [GR02] would give a multiplicative error of  $k$  (in addition to the additive error). To get a purely additive error we need to take a different approach. Specifically, we use different combinatorial representations of the connectivity structure of graphs (based on [NGM97]), rather than those used in [GR02]. On top of this, we adapt and extend some of the ideas in [GR02]. We believe that the analysis we present for distance approximation is actually easier to follow than the analysis of the original testing algorithm.

**Approximating the Size of a Minimum Vertex Cover.** A simple modification of our (non-sublinear time) algorithm for approximating the distance to subgraph freeness, yields a local distributed algorithm for approximating the size of a minimum vertex cover. Specifically, For any  $\delta > 0$  the algorithm gives, with high constant probability, a  $(2 + \delta)$ -factor approximation to the size of the minimum vertex cover in a graph with degree bound  $d$ , using  $O(\log(d/\delta))$  rounds. This improves on the  $(2 + \delta)$ -factor approximation using  $O(\delta^{-3} \cdot \log d)$  rounds given in a recent paper of Kuhn, Moscibroda and Wattenhofer [KMW06]. Their algorithm, which is quite complex, uses Linear Programming and falls into a more general framework of approximation algorithms for covering and packing problems.

As in [PR05] we can use our algorithm to obtain a sublinear-time algorithm for approximating the size of the minimum vertex cover in bounded-degree graphs. The query-complexity and running time of the algorithm are  $d^{O(\log(d/\delta))}$ , and it outputs an estimate with a multiplicative error of at most 2 and an additive error of at most  $\delta n$ . The algorithm in [PR05], which builds on [KMW06] gives an estimate with the same quality in time  $d^{O(\log d/\delta^3)}$ . As shown in [PR05], the dependence on  $d$  can be replaced with a dependence on  $O(\bar{d}/\delta)$ , and this is applicable also in our case.

**Organization.** Our result for  $k$ -connectivity is given in Section 2, and the result for subgraph freeness in Section 3. The results for the Eulerian property and cycle freeness are given in Section 4 and Section 5 respectively.

## 2 Distance Approximation to $k$ -Edge-Connectivity

In this section we consider the graph property of  $k$ -edge-connectivity. Recall that a graph is  $k$ -edge-connected or simply  $k$ -connected if there are  $k$  edge-disjoint paths between any pair of vertices in the graph. Equivalently, a graph is  $k$ -connected if the removal of any  $k - 1$  edges from the graph results in a connected graph.

Goldreich and Ron [GR02] gave a testing algorithm for this property that works for bounded-degree graphs and runs in time<sup>4</sup>  $\tilde{O}(k^3 \cdot \epsilon^{-3+2/k})$ . They improve the running time to  $\tilde{O}(1/\epsilon)$  for  $k = 1, 2$  and to  $\tilde{O}(\epsilon^{-2})$  for  $k = 3$ . Parnas and Ron [PR02] showed that these algorithms can be extended to the general sparse model.

We present a distance approximation algorithm for  $k$ -connectivity whose query complexity and running time are  $O\left(\left(k/(\delta\bar{d})\right)^6 \beta^{3/2} \log\left(k/(\delta\bar{d})\right)\right)$ . For the case  $k = 1$ , this problem was addressed by Chazelle, Rubinfeld and Trevisan [CRT05] as part of their algorithm for approximating the weight of a minimum spanning tree of a graph. Using our terminology, they give a distance approximation algorithm for connectivity of general simple sparse graphs whose query complexity and running time are  $O(1/(\delta^2 \cdot \bar{d}) \cdot \log(1/\delta))$ . For completeness, we describe and analyze a simple (though slightly less efficient) algorithm for distance approximation of 1-connectivity in Subsection 2.1.

The problem of approximating the distance of a graph from being  $k$ -connected for  $k > 1$  is more complicated, but as we shall show, is still solvable in time that is independent of the size of the input graph. As noted earlier, the corresponding testing problem was solved by Goldreich and Ron in [GR02]. By trying to extend their approach to distance approximation, one can get a multiplicative factor of  $k$ , in addition to the additive factor. Here we partly build on their ideas, but use different graph structures. This allows us to obtain an additive approximation, without any multiplicative factor. Specifically, we build on the *extreme-sets tree* and the *extreme-sets partition*, introduced by Naor, Gusfield and Martel in [NGM97] as part of their algorithm for optimally increasing the edge-connectivity of a graph. We first assume that the given graph is connected and handle the case of a non-connected graph in Subsection 2.4.

### 2.1 Connectivity ( $k = 1$ )

The problem of approximating the distance to connectivity is equivalent to approximating the number of connected components in the graph. The idea of the algorithm is to approximate the number of small connected components. Since there are not many large components, the approximation is as required.

#### Algorithm 1 (*Distance approximation to connectivity*)

1. Uniformly and independently sample  $s = \frac{16}{\delta^2 \bar{d}^2}$  vertices from  $G$ . Let  $S$  be the multiset of the sampled vertices.
2. For every  $v \in S$ , perform a BFS starting from  $v$  until  $\frac{4}{\delta \bar{d}}$  vertices have been reached or  $v$ 's connected component has been found. Let  $\hat{n}_v$  be the number of vertices in  $v$ 's connected component in case it was found. Otherwise  $\hat{n}_v = \infty$ .
3. Let  $\hat{C} = \frac{n}{s} \sum_{v \in S} \frac{1}{\hat{n}_v}$  and output  $\frac{1}{m}(\hat{C} - 1)$ .

<sup>4</sup>The  $\tilde{O}(\cdot)$  notation hides logarithmic factors, that is,  $\tilde{O}(f(n))$  means  $O(f(n) \cdot \text{polylog}(f(n)))$ .

**Theorem 1** *Algorithm 1 is a distance approximation algorithm for connectivity. The query and time complexity of the algorithm are  $O((\delta\bar{d})^{-4}\beta)$ .*

**Proof:** For every vertex  $v$  let  $n_v$  be the number of vertices in  $v$ 's connected component. Then, the number of connected components in the graph is  $C = \sum_{v \in S} \frac{1}{n_v}$ . By the definition of  $\hat{n}_v$ ,  $\text{Exp}[\hat{C}]$  is the number of connected components whose size is less than  $\frac{4}{\delta\bar{d}}$ . Since there are at most  $\frac{\delta}{2}m$  connected components of size larger than  $\frac{4}{\delta\bar{d}}$  we get that

$$C - \frac{\delta}{2}m \leq \text{Exp}[\hat{C}] \leq C.$$

Now, by an additive Chernoff bound, for  $s = \frac{16}{\delta^2\bar{d}^2}$ ,

$$\Pr \left[ \left| \hat{C} - \text{Exp}[\hat{C}] \right| > \frac{\delta}{2}m \right] = \Pr \left[ \left| \frac{1}{s} \sum_{v \in S} \frac{1}{\hat{n}_v} - \text{Exp} \left[ \frac{1}{\hat{n}_v} \right] \right| > \frac{\delta}{4}\bar{d} \right] < 2 \cdot e^{-2s(\delta\bar{d}/4)^2} < \frac{1}{3}$$

The claim follows by applying the triangle inequality.

The query and time complexity of every BFS is  $O((\delta\bar{d})^{-2}\beta)$  and so the total query and time complexity of the algorithm is  $O((\delta\bar{d})^{-4}\beta)$ . ■

## 2.2 $k$ -Connectivity ( $k > 1$ ): Preliminaries

Let  $G = (V, E)$  be a connected undirected graph. A *minimum cut* in the graph is a set of edges with minimal size whose removal from the graph disconnects it into two sets of vertices  $A$  and  $\bar{A}$ . We denote the cut by  $(A, \bar{A})$ . The *degree* of the set  $A$ , denoted by  $d(A)$ , is the number of edges with exactly one endpoint in  $A$ , thus it equals to the size of the cut  $(A, \bar{A})$ .

**Definition 1** *We say that a set  $U$  is  $\ell$ -extreme if it has degree  $\ell$  and the degree of every proper subset of  $U$  is strictly larger than  $\ell$ . That is, if  $d(U) = \ell$  and for every  $W \subset U$ ,  $d(W) > \ell$ .*

We note that extreme sets are different but related to the *connectivity classes* of the graph. An  $\ell$ -class is a maximal set of vertices that cannot be disconnected by the removal of less than  $\ell$  edges. By this definition and the definition of extreme sets, every  $(\ell - 1)$ -extreme set is also an  $\ell$ -class but not vice versa since an  $\ell$ -class might have degree greater than  $\ell - 1$ .

Extreme sets have the property that every two of them are either disjoint or one is contained in the other. More precisely, if  $U$  is  $\ell$ -extreme and  $W$  is  $j$ -extreme for  $\ell \geq j$  then either  $U$  and  $W$  are disjoint or  $U \subseteq W$  (see [NGM97]). This property is the key for representing the collection of all the extreme sets of a graph in a tree called *extreme-sets tree*. The leaves of the tree are the vertices of the graph where each vertex of degree  $d$  is a  $d$ -extreme set. The parent of an extreme set  $U$  is the minimal extreme set  $W$  containing  $U$ . If  $U$  is an  $\ell$ -extreme set and  $W$  is a  $j$ -extreme set then necessarily  $\ell > j$ . The root of the tree corresponds to  $V$ , which is a 0-extreme set. We shall use the notation  $U \sqsubset W$  to denote that  $U$  is a child of  $W$  in the tree.

We next make a simple but important observation. Given any partition  $\mathcal{P} = \{V_1, \dots, V_q\}$  of the vertices of  $G$ , the minimum number of edges that should be added to  $G$  in order to make it connected is lower bounded by  $\lceil \phi(\mathcal{P})/2 \rceil$  where  $\phi(\mathcal{P})$  is the edge *demand* of the partition  $\mathcal{P}$ , defined by

$$\phi(\mathcal{P}) = \sum_{i=1}^q \max\{0, k - d(V_i)\}. \quad (1)$$

This is true since for every  $i$ , if  $k - d(V_i) > 0$  then at least  $k - d(V_i)$  endpoints of edges must be attached to vertices of  $V_i$  in order to increase the connectivity to  $k$ . Therefore, the number of edges that must be added to the graph is at least  $\max_{\mathcal{P}} \{\lceil \phi(\mathcal{P})/2 \rceil\}$ .

Naor, Gusfield and Martel [NGM97] defined a partition called the *extreme-sets partition* (ES) and presented an algorithm for increasing the connectivity of  $G$  to  $k$  that adds exactly  $\lceil \phi(ES)/2 \rceil$  edges. Given the aforementioned lower bound, we have that  $\lceil \phi(ES)/2 \rceil$  equals  $m$  times the distance of  $G$  from  $k$ -connectivity, which is just the value that we would like to estimate. In what follows we describe this partition. For an extreme set  $U$  in the extreme-sets tree the *demand* of  $U$  is defined by

$$\phi(U) = \max \left\{ 0, k - d(U), \sum_{W \sqsubset U} \phi(W) \right\}. \quad (2)$$

Thus  $\phi(U)$  is a lower bound on the number of endpoints of edges that must be attached to vertices in  $U$  in order to increase the edge-connectivity of the graph to  $k$ . The demand of the root  $V$  is defined by  $\phi(V) = \sum_{U \sqsubset V} \phi(U)$ . Using these notions, the *extreme-sets partition* is defined as follows.

**Definition 2** *The extreme-sets partition of a graph  $G$  is the partition  $ES = ES(G) = \{X_1, \dots, X_q\}$  that satisfies the following conditions:*

1. *For every  $i$ ,  $X_i$  is an extreme set with the property that either  $\phi(X_i) = 0$  or  $\phi(X_i) > \sum_{Y \sqsubset X_i} \phi(Y)$ .*
2. *For every  $i$ ,  $X_i$  is not contained in any other extreme set satisfying Condition 1.*

Given the extreme-sets tree, the partition  $ES$  of  $V$  can be constructed by recursively finding the partition of every child of  $V$ . Whenever an extreme set that satisfies Condition 1 from Definition 2 is found, it is added to the partition  $ES$  and the recursion ends. Since the leaves of the tree, i.e., the vertices of the graph, are all extreme sets that satisfy Condition 1, the partition  $ES$  is well defined. Observe that the demand of the partition  $ES$  satisfies  $\phi(ES) = \sum_{i=1}^q \phi(X_i)$ . In addition, this is exactly the demand of the root  $\phi(V) = \sum_{U \sqsubset V} \phi(U)$  since for every  $i \in \{1, \dots, q\}$ , the demand of every ancestor of  $X_i$  exactly equals the sum of the demands of its children. For an illustration of an extreme-sets tree and an extreme-sets partition see Figure 1.

For a graph  $G'$  and for any vertex  $v$  in  $G'$ , we denote by  $X_v(G')$  the set in  $ES(G')$  that contains  $v$ . When  $G' = G$  we use the shorthands  $X_v$  and  $ES$ , respectively.

### 2.3 $k$ -Connectivity ( $k > 1$ ): the Algorithm

When approximating the distance to 1-connectivity the algorithm estimates, for every vertex selected, the size of its connected component. In an analogous way, in order to approximate the distance to  $k$ -connectivity, which equals to  $\frac{1}{m} \lceil \phi(ES)/2 \rceil$  where  $\phi(ES) = \sum_{v \in V} \frac{\phi(X_v)}{|X_v|}$ , we estimate for every vertex  $v$  the demand and the size of  $X_v$ . More precisely, since  $X_v$  may be large for some vertices, we introduce a certain refinement of  $ES$  that consists of subsets of a bounded size.

**Definition 3** *Given a graph  $G$  and a size bound  $t$ , the  $t$ -bounded extreme-sets partition is the partition  $ES^{(t)} = ES^{(t)}(G) = \{X_1^{(t)}, \dots, X_q^{(t)}\}$  that satisfies the following conditions:*

1. *For every  $i$ , the size of  $X_i^{(t)}$  is at most  $t$ .*

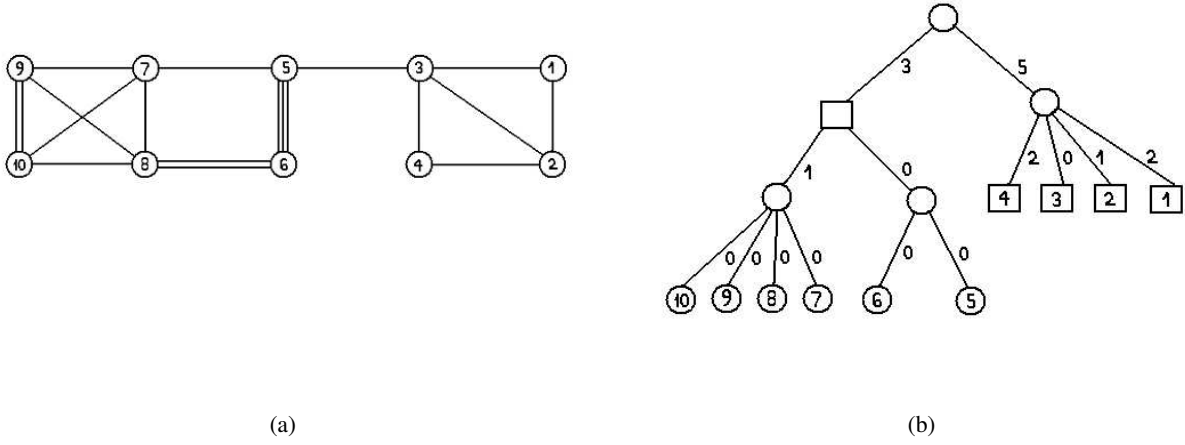


Figure 1: (a) A graph and (b) its extreme-sets tree and extreme-sets partition. Each node represents an extreme set. The values on the edges are the demands of the corresponding extreme sets for  $k = 4$ . The squared nodes represent the sets of the extreme-sets partition.

2. For every  $i$ ,  $X_i^{(t)}$  is an extreme set with the property that either  $\phi(X_i^{(t)}) = 0$  or  $\phi(X_i^{(t)}) > \sum_{Y \sqsubset X_i^{(t)}} \phi(Y)$ .
3. For every  $i$ ,  $X_i^{(t)}$  is not contained in any other extreme set of size at most  $t$  satisfying Conditions 1 and 2.

We claim that  $\phi(ES) \geq \phi(ES^{(t)})$ . To verify this, note that the sets of  $ES$  whose size is at most  $t$  are also sets of  $ES^{(t)}$ . The other sets of  $ES$  are further partitioned in  $ES^{(t)}$  into smaller sets that satisfy Conditions 2 and 3 of Definition 3. That is, every set in  $ES$  whose size is larger than  $t$  is replaced in  $ES^{(t)}$  by smaller extreme sets from its subtree. Now, by the definition of the demand of a set (Equation (2)), the demand of every extreme set in the extreme-sets tree is always greater or equal to the sum of the demands of its children, therefore, the sum of demands of the sets in  $ES^{(t)}$  that replace some set in  $ES$  is at most the demand of that set. For an illustration of a  $t$ -bounded extreme-sets partition see Figure 2.

For any graph  $G'$  and a vertex  $v$  in  $G'$ , we denote the set in  $ES^{(t)}(G')$  that contains  $v$  by  $X_v^{(t)}(G')$ . When  $G' = G$  we use the shorthands  $ES^{(t)}$  and  $X_v^{(t)}$ , respectively.

The following procedure searches for  $X_v^{(t)}$  given a size bound  $t$  and a repetition parameter  $r$ , both of which will be set subsequently. It uses the *contraction* operation of a set  $A$  of vertices in which the vertices of  $A$  are merged into a single vertex  $a$  and for every edge  $(v, u)$  such that  $v \in A$  and  $u \notin A$ , there is an edge between  $a$  and  $u$ .

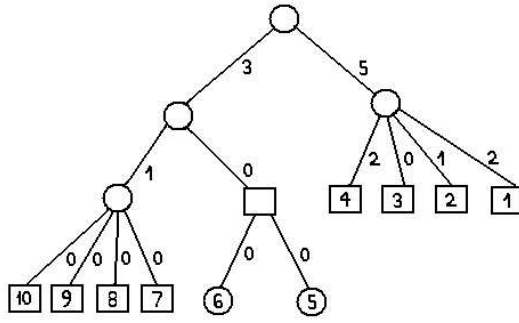


Figure 2: The extreme-sets tree of the graph in Fig 1(a), and the  $t$ -bounded extreme-sets partition for  $t = 3$ . Each node represents an extreme set. The values on the edges are the demands of the corresponding extreme sets for  $k = 4$ . The squared nodes represent the sets of the 3-bounded extreme-sets partition.

**Procedure 1 (Extreme-set search from a given vertex  $v$ )**

1. Repeat the following process for every  $i = 1, \dots, r$ .
  - (a) (Random Search Process) Start with  $S_i = \{v\}$ . As long as  $|S_i| \leq t$  and the size of the cut  $(S_i, \overline{S}_i)$  is less than  $3t^2\beta$ , assign a random cost in the range  $[0, 1]$  to the edges of the cut  $(S_i, \overline{S}_i)$  that were not yet assigned costs. Traverse the edge of lowest cost and add the new vertex reached to  $S_i$ .
  - (b) (Extreme-Set Search) Let  $G_{S_i}$  be the graph obtained from  $G$  by contracting the set  $\overline{S}_i$  to a single vertex  $\overline{s}_i$ . Construct the extreme-sets tree of  $G_{S_i}$  and let  $X_v^{S_i}$  be the set  $X_v(G_{S_i})$ .
2. Let  $X_v^{max}$  be the maximal set among  $\{X_v^{S_i}\}_{i=1}^r$ . Declare  $X_v^{max}$  as the set in ES of  $G$  containing  $v$  i.e., as  $X_v^{(t)}(G)$ .

**Lemma 1** For every  $v$  and size bound  $t$ , Procedure 1 finds  $X_v^{(t)}(G)$  with probability at least  $1 - e^{-2r/t^2}$ . Its query complexity and running time are  $O(t^4 r \beta^{3/2})$ .

**Proof:** The construction of the extreme-sets tree of the graph  $G_{S_i}$  at Step 1.b, can be done by first constructing a hierarchical structure of the connectivity classes of  $G_{S_i}$  (called the *class decomposition tree*) using  $n - 1$  max-flow computations [GH61, Gus90]. This takes  $O(\tilde{n}\tilde{m}^{3/2})$  time where  $\tilde{n} = t + 1$  is an upper bound on the number of vertices in  $G_{S_i}$  and  $\tilde{m} = (t + 1)^2 \cdot \beta$  is an upper bound on the number of edges. Then, the classes that are not extreme sets are removed from the tree to get the extreme-sets tree [NGM97]. This is done using  $O(\tilde{n}\tilde{m})$  steps (see [NGM97] for more details). Thus the total running time for the construction is  $O(t^4 \beta^{3/2})$ . Step 1.a takes  $O(t^3 \beta)$  queries and time and therefore the total query and time complexity of Procedure 1 is  $O(t^4 \beta^{3/2} r)$ .

To analyze the correctness of Procedure 1, assume first that at least one iteration of the random search process (Step 1.a) finds a set  $S$  that contains  $X_v^{(t)}(G)$ . In the following claim we establish that in this case, the procedure declares  $X_v^{(t)}(G)$  as the required set.

**Claim 2** *If at least one iteration of the random search process of Step 1.a finds a set  $S$  that contains  $X_v^{(t)}(G)$ , then Procedure 1 finds  $X_v^{(t)}(G)$ .*

**Proof:** Consider the  $i$ 'th iteration and the set  $S_i$  that is found in Step 1.a. Step 1.b consists of deterministic sub-procedures and therefore always finds  $X_v^{(t)}(G_{S_i})$ . Note that by the transition from the original graph  $G$  to the graph  $G_{S_i}$ , we do not necessarily preserve the connectivity of vertices in  $S_i$ . The connectivity cannot decrease but it might increase. However, we can easily show that the collection of all extreme sets that are contained in  $S_i$  is exactly the same in  $G$  and in  $G_{S_i}$ . This follows immediately by observing that for every set  $U \subseteq S_i$ , the degree of  $U$ ,  $d(U)$ , is exactly the same in  $G$  and in  $G_{S_i}$ . Also, the demand of any extreme set is a local property that depends only on its degree and its sub-extreme sets, thus, the demand of every extreme set that is contained in  $S_i$  is exactly the same in  $G$  and in  $G_{S_i}$ . Therefore, since Step 1.b always finds  $X_v^{(t)}(G_{S_i})$ , if  $S_i$  contains  $X_v^{(t)}(G)$ , then  $X_v^{S_i} = X_v^{(t)}(G)$ . That is, at the  $i$ 'th iteration, the extreme-set search of Step 1.b finds  $X_v^{(t)}(G)$ .

Now, consider the collection  $\{X_v^{S_i}\}_{i=1}^r$  of the sets found in iterations  $1, \dots, r$ . Assume w.l.o.g that the sets are ordered by an increasing order of their size. Then for every  $i$ ,  $X_v^{S_i} \subseteq X_v^{S_{i+1}}$ . This follows from the fact that every two extreme sets are either disjoint or one is contained in the other and since  $v \in X_v^{S_i}$  for every  $i$ . In addition, for every  $i$ ,  $X_v^{S_i}$  satisfies conditions 1 and 2 of  $t$ -bounded extreme sets. Thus all the sets except the largest one (or the few largest ones in case the largest set was found more than once) are contained in another extreme set that satisfies conditions 1 and 2 and therefore do not satisfy condition 3. We conclude that if some iteration finds  $X_v^{(t)}$  then necessarily  $X_v^{(t)}$  is the largest set among  $\{X_v^{S_i}\}_{i=1}^r$ . ■

What is left to analyze is the probability that the random search process of Step 1.a finds a set  $S$  containing  $X_v^{(t)}$ . To this end we lower bound the probability that all the vertices of  $X_v^{(t)}$  are added to the growing set  $S$  before any other vertex is. But first, observe that for every  $S \subseteq X_v^{(t)}$ , the size of the cut  $(S, \overline{S})$  is less than  $3t^2\beta$ . Thus, if the algorithm adds to  $S$  only vertices from  $X_v^{(t)}$ , it won't stop before all the vertices of  $X_v^{(t)}$  are in  $S$ . To verify this, for every  $v \in X_v^{(t)}$  let  $d_v^{in}$  denote the degree of  $v$  in the subgraph induced by  $X_v^{(t)}$  ( $d_v^{in}$  is less than  $t\beta$ ) and let  $d_v^{out} = d_v - d_v^{in}$ . Consider any vertex  $u \in X_v^{(t)}$ . Since  $X_v^{(t)}$  is an extreme set,  $\sum_{v \in X_v^{(t)}} d_v^{out} < d_u < d_u^{out} + t\beta$ . In other words,  $\sum_{v \in X_v^{(t)} \setminus \{u\}} d_v^{out} < t\beta$ . Since this is true for every  $u \in X_v^{(t)}$ , the size of the cut  $(X_v^{(t)}, \overline{X_v^{(t)}})$ , which equals to  $\sum_{v \in X_v^{(t)}} d_v^{out}$ , is less than  $2t\beta$  and thus for every  $S \subseteq X_v^{(t)}$ , the size of the cut  $(S, \overline{S})$  is less than  $t^2\beta + 2t\beta \leq 3t^2\beta$ . Note that the algorithm cannot detect the point at which  $S = X_v^{(t)}$  since the value of  $\ell$ , such that  $X_v^{(t)}$  is  $\ell$ -extreme, is unknown.

Now, consider the graph  $G_X$  obtained from  $G$  by contracting the set  $X_v^{(t)}$  into a single vertex  $\bar{x}$ . Assume that the random search process of Step 1.a runs on  $G_X$  for  $t' = |X_v^{(t)}|$  steps. The cut  $(X_v^{(t)}, \bar{x})$  is a minimum cut of  $G_X$  since  $X_v^{(t)}$  is an extreme set. Goldreich and Ron proved in [GR02] that in this case the probability that no cut edge is traversed before  $X_v^{(t)}$  is found is at least  $2t^{-2}$ . Their analysis is based on Karger's analysis of his algorithm for finding minimum cut in a graph [Kar93].

**Lemma 3** [GR02] *For an undirected graph  $G$ , let  $L$  be a set of at most  $t$  vertices such that the cut  $(L, \overline{L})$  is a minimum cut. Then, starting with some vertex  $v \in L$ , the random search process of Step 1.a succeeds in finding the cut  $(L, \overline{L})$  with probability at least  $2t^{-2}$ .*

**Proof Sketch:** Consider the graph  $G_L$  obtained from  $G$  by contracting the set  $\overline{L}$  into a single vertex  $\bar{\ell}$ . Assume that the edges of  $G_L$  are randomly and independently assigned costs in the range  $[0, 1]$  and that the

random search process runs on  $G_L$ . Observe that if the subgraph induced on  $L$  contains a spanning tree that is cheaper than the cut  $(L, \bar{L})$  (i.e., the cost of every edge of the spanning tree is smaller than that of any cut edge) then the random search process finds  $L$ . This is true since in this case, at every step there is some edge whose cost is cheaper than the cost of any cut edge, thus no cut edge is traversed. To analyze the probability that such a spanning tree exists, consider the Contraction Algorithm of Karger for finding a minimum cut in a graph [Kar93]. At every step of his algorithm, the edge with the smallest cost out of the remaining edges in the graph is contracted (as opposed to the smallest cost cut edge in our algorithm). The process continues until one edge remains. Karger showed that for every fixed minimum cut, the probability that no cut edge is contracted is at least  $2t^{-2}$ . The contracted edges form 2 spanning trees, attached to the endpoints of the remaining edge, thus proving that in our case, the subgraph induced on  $L$  contains a spanning tree that is cheaper than the cut  $(L, \bar{L})$ . This completes the proof.  $\square$

**Corollary 4** *If we repeat the random search process  $r$  times, then, with probability at least  $1 - (1 - 2t^{-2})^r > 1 - e^{-2t^{-2} \cdot r}$ , at least one iteration finds a set containing  $X_v^{(t)}$ .*

Combining Corollary 4 with Claim 2, with probability at least  $1 - e^{-2t^{-2}r}$ , Procedure 1 finds  $X_v^{(t)}$ , thus proving Lemma 1.  $\blacksquare$

We now present the distance approximation algorithm that uses Procedure 1 to estimate the distance of a connected graph from being  $k$ -connected.

**Algorithm 2 (Distance approximation to  $k$ -connectivity)**

1. Uniformly and independently sample  $s = 32k^2/(\delta^2\bar{d}^2)$  vertices from  $G$ . Let  $S = \{u_1, \dots, u_s\}$  be the multiset of the sampled vertices.
2. For every sampled vertex  $u_j$ , run Procedure 1 using the size bound  $t = 4k/\delta\bar{d}$  and the repetition constant  $r = t^2 \ln(\frac{32k^2}{\delta^2\bar{d}^2})$ . Let  $X$  be the extreme set found and let  $\hat{n}_j = |X|$ .
3. Calculate the demand of  $X$  and denote it by  $\hat{\phi}_j$ .
4. Let  $\hat{\phi} = \frac{n}{s} \sum_{i=1}^s \frac{\hat{\phi}_i}{\hat{n}_i}$ , let  $\hat{C} = \lceil \frac{\hat{\phi}}{2} \rceil$  and output  $\frac{1}{m} \hat{C}$ .

**Theorem 2** *For every  $k > 1$ , Algorithm 2 is a distance approximation algorithm for  $k$ -connectivity of connected graphs. The query complexity and running time of the algorithm are  $O((k/(\delta\bar{d}))^6 \beta^{3/2} \log(k/(\delta\bar{d})))$ .*

As noted previously, in the full version of this paper [MR06] we show how to deal with the case of unconnected graphs. The only difference is a slight modification in Procedure 1.

**Proof:** Since the query and time complexity of Procedure 1 is  $O(t^4 \cdot r \cdot \beta^{3/2})$ , and since the demand of the set  $X$  found by the procedure is computed in the course of the procedure, the query and time complexity of Algorithm 2 is  $O(s \cdot t^4 \cdot r \beta^{3/2}) = O((k/(\delta\bar{d}))^6 \beta^{3/2} \log(k/(\delta\bar{d})))$ , as claimed.

Let  $ES = \{X_1, \dots, X_q\}$  be the extreme-sets partition of  $G$  and let  $ES^{(t)} = \{X_1^{(t)}, \dots, X_q^{(t)}\}$  be its  $t$ -bounded extreme-sets partition for  $t = 4k/\delta\bar{d}$ . Assume w.l.o.g that the sets are sorted by increasing order of their size. Let  $\ell$  be the maximal index such that  $|X_\ell| \leq t$ . Then  $X_i^{(t)} = X_i$  for every  $i \leq \ell$ . Now, as noted before,  $\phi(ES) \geq \phi(ES^{(t)})$  so

$$\phi(ES) - \phi(ES^{(t)}) = \sum_{i=j+1}^p \phi(X_i) - \sum_{i=j+1}^q \phi(X_i^{(t)}) \quad (3)$$

$$\leq \sum_{i=j+1}^p \phi(X_i) \leq \frac{\delta \bar{d} n}{4k} \cdot k = \frac{\delta}{2} m. \quad (4)$$

where the last inequality is true since there are at most  $\frac{n}{t} = \frac{n}{4k/\delta d}$  sets of size greater than  $t$  and the demand of any set is at most  $k$ .

We next show that with high probability  $\hat{\phi}$  is a good estimate for  $\phi(ES^{(t)})$ . We first calculate the probability that Procedure 1 succeeds in finding  $X_v^{(t)}$  for all the sampled vertices. For every sampled vertex  $u_j$ , as shown by Lemma 1, the probability that Procedure 1 fails to find  $X_{u_j}^{(t)}$  is at most  $e^{-2t^{-2} \cdot r}$ . Thus the probability that it fails for some  $j$  is at most

$$\sum_{i=1}^s e^{-2t^{-2} \cdot r} = s \cdot e^{-2 \ln(\frac{32k^2}{\delta^2 \bar{d}^2})} < 32k^2 / (\delta^2 \bar{d}^2) \cdot ((\delta^2 \bar{d}^2) / (32k^2))^2 < \frac{1}{6}. \quad (5)$$

That is, with probability at least  $5/6$ , the procedure finds  $X_{u_j}^{(t)}$  for every sampled vertex  $u_j$ . Let  $n_v = |X_v^{(t)}|$  and let  $\phi(v)$  be the demand of  $X_v^{(t)}$ . Then, with probability at least  $5/6$ , for every sampled vertex  $u_j$ ,

$$\frac{\hat{\phi}_j}{\hat{n}_j} = \frac{\phi(u_j)}{n_{u_j}}.$$

Assuming this is true, for every  $j$ ,  $\chi_j = \frac{\hat{\phi}_j}{\hat{n}_j}$  is a random variable whose expected value is  $\frac{1}{n} \sum_{v \in V} \frac{\phi(v)}{n_v}$ . Let  $\mu = \text{Exp}[\frac{1}{k} \chi_j]$  for some  $j$ . Then,

$$\phi(ES^{(t)}) = \sum_{i=1}^q \phi(X_i^{(t)}) = \sum_{v \in V} \frac{\phi(v)}{n_v} = k \cdot n \cdot \mu. \quad (6)$$

The random variable  $\frac{1}{k} \chi_j$  gets values in the range  $[0, 1]$ , thus by an additive Chernoff bound, with probability at least  $5/6$ ,

$$\Pr \left[ \left| \hat{\phi} - \phi(ES^{(t)}) \right| > \frac{\delta}{2} m \right] \leq \Pr \left[ \left| \frac{1}{s} \sum_{j=1}^s \frac{\hat{\phi}_j}{k \cdot \hat{n}_j} - \mu \right| > \frac{\delta}{4k} \bar{d} \right] < 2 \cdot e^{-2s\delta^2 \bar{d}^2 / 16k^2} < \frac{1}{6} \quad (7)$$

That is, with probability at least  $2/3$ ,  $|\hat{\phi} - \phi(ES^{(t)})| < \frac{\delta}{2} m$ . Theorem 2 follows by applying the triangle inequality. ■

## 2.4 $k$ -Connectivity for $k \geq 2$ and non-connected graphs

The algorithm of [NGM97] for optimally increasing the connectivity of a graph does not handle the case in which the given graph is not connected. We show, however, that it can be generalized to include this case also and state here the needed modifications.

Naor, Gusfield and Martel [NGM97] presented an algorithm that increases the connectivity of a given  $\lambda$ -connected graph to  $k$ , by adding exactly  $\lceil \phi(ES)/2 \rceil$  edges. This number of edges matches the lower bound discussed before, thus proving optimality. Their algorithm works in  $k - \lambda$  phases, each increases the connectivity of the graph by one. They prove that for every phase  $i \in \{1, \dots, k - \lambda - 1\}$ , if  $\ell_i$  is the number of edges added, then the demand is decreased by  $2\ell_i$ . In the last phase, since at most one endpoint is added

without satisfying a demand, the demand is decreased by either  $2\ell_{k-\lambda}$  or  $2\ell_{k-\lambda} - 1$ . The way the edges are selected is by first selecting pairs of extreme sets to connect using a graph structure called *cactus* and then selecting the vertices inside the extreme sets, connecting them by an edge.

We focus on the first phase in which the connectivity of a graph  $G$  with  $p$  connected components is increased to 1. We describe how to choose  $p - 1$  edges so that  $G$  will become 1-connected and the demand  $\phi(ES)$  will be decreased by  $2(p - 1)$ . Consider the graph  $G'$  obtained by contracting every 2-class of  $G$  into a single node.  $G'$  is a forest of  $p$  trees where each tree consists of one or more nodes. Let  $T_1, \dots, T_p$  be the trees of  $G'$ . If  $T_i$  consists of one node then it corresponds to a 2-class of  $G$ , otherwise, its leaves correspond to 1-extreme sets of  $G$ . For every  $i$ , choose two arbitrary leaves  $u_{i1}$  and  $u_{i2}$  in the tree  $T_i$ . For trees of only one node, choose this node twice. For every  $i \in \{1, \dots, p - 1\}$  match the pair  $(u_{i2}, u_{(i+1)1})$ . Then, if  $u_{i1} = u_{i2}$  (the tree  $T_i$  consists of one node) choose two vertices in the corresponding 2-class of  $T_i$ , otherwise, choose one vertex in the corresponding extreme set of  $u_{i1}$  and one vertex in the corresponding extreme set of  $u_{i2}$ . Now, for every pair  $(u_{i2}, u_{(i+1)1})$  add an edge connecting the selected vertices in the corresponding extreme sets. The selection of those vertices is done using a similar rule to that defined in [NGM97]:

For every leaf  $u_{ij}$ , let  $U_{ij}$  be its corresponding extreme set in the extreme-sets tree of  $G$ . If only one endpoint should be added to a vertex in  $U_{ij}$  then find an extreme set  $W_{ij}$  in the subtree of  $U_{ij}$  such that  $\phi(W_{ij}) > 0$  and  $\phi(Z) = 0$  for every  $Z$  child of  $W_{ij}$ . Then choose an arbitrary vertex in  $W_{ij}$ . If on the other hand two endpoints should be added to vertices in  $U_{ij}$  then there are two cases: If  $U_{ij}$  is a  $k$ -class, select two arbitrary vertices in  $U_{ij}$  (unless it contains only one vertex, in which case this vertex is selected twice). Otherwise, find two extreme sets  $W_{i1}$  and  $W_{i2}$  in the subtrees of two separated children of  $U_{ij}$  satisfying the same rule as in the case where only one endpoint should be selected. Then choose two arbitrary vertices, one in  $W_{i1}$  and the other in  $W_{i2}$ .

We claim that the addition of such  $p - 1$  edges to  $G$ , which clearly increase the connectivity of  $G$  by 1, decrease the demand  $\phi(ES)$  by  $2(p - 1)$ . The proof is similar to the proof for the other phases in [NGM97]. The only difference is the proof for Lemma 4.5 of [NGM97] that in our case follows immediately from the construction. We refer the reader to [NGM97] for more details. This ensures that when combining the rest of the phases as described in [NGM97], the total number of edges added is  $\lceil \phi(ES)/2 \rceil$  and the resulting graph is  $k$ -connected. Thus, if the given graph is not connected, its distance from  $k$ -connectivity, for  $k \geq 2$  is exactly  $\frac{1}{dn} \lceil \phi(ES)/2 \rceil$  as for a connected graph.

As for the implications on our algorithm, Procedure 1 has to be modified in the following way. Whenever the Random Search Process of Step 1.a finds the connected component  $C_v$  of  $v$ , the extreme-sets tree of  $C_v$  is built and the procedure declares the set  $X_v(C_v)$  as  $X_v(G)$ . Note that  $X_v(C_v)$  is exactly  $X_v(G)$  so that if  $v$  belongs to a small connected component, the procedure always finds  $X_v(G)$ .

### 3 Distance Approximation to Subgraph-Freeness

For a fixed graph  $H$ , we say that  $G$  is  $H$ -free if it contains no subgraph isomorphic to  $H$ . In this section we consider the problem of approximating the distance of a graph from being  $H$ -free for some fixed subgraph  $H$  in the bounded-degree model. We note that testing subgraph-freeness in the general sparse model requires  $\Omega(\sqrt{n})$  queries [AKKR06].

In what follows we focus on triangles and then generalize the result to arbitrary subgraphs (in Subsection 3.2). We first present a non-sublinear algorithm for approximating the minimum number of edges that should be removed in order to obtain a triangle-free graph. Later we show how to transform it into a distance approximation algorithm whose running time is independent of  $n$ .

### 3.1 Triangle-Freeness

Let  $G$  be an undirected graph with degree at most  $d$  and let  $m = dn$ . We say that two triangles in  $G$  are *neighbors* if they share a common edge. For a triangle  $t$ , the set of its neighboring triangles is denoted by  $\Gamma(t)$ . The *degree* of a triangle, denoted by  $d(t)$ , is defined as the size of  $\Gamma(t)$ . The *distance* between two triangles  $t$  and  $t'$  is the minimum number of triangles minus 1 in a sequence  $t_1, \dots, t_\ell$  of triangles for which  $t_1 = t$  and  $t_\ell = t'$  and for every  $i \in \{1, \dots, \ell - 1\}$ , the triangles  $t_i$  and  $t_{i+1}$  are neighbors. The *k-neighborhood* of a triangle  $t$  is defined as the set of triangles whose distance from  $t$  is at most  $k$ . In an analogous way, the *k-neighborhood* of a vertex  $v$  is the set of vertices whose distance from  $v$  is at most  $k$ . For a set  $S$  of edges, we say that  $S$  is a *triangle cover* if its removal from the graph results in a triangle-free graph and denote by  $C_{OPT}$  the minimum size of a triangle cover of  $G$ .

The following algorithm gets as input a graph  $G$  with degree at most  $d$  and a parameter  $\delta$ , and approximates  $C_{OPT}$ .

**Algorithm 3 (Minimum triangle cover approximation)**

1. Let  $\mathcal{T}$  be the set of all the triangles in  $G$  and let  $\mathcal{TC} = \emptyset$  be the initial triangle cover.
2. From  $i = 1$  to  $r = \Theta(\log(d/\delta))$ 
  - (a) Select each triangle  $t \in \mathcal{T}$  with probability  $\frac{1}{c \cdot d(t)}$ , where  $c$  is some constant that will be defined later. If  $d(t) = 0$  then  $t$  is selected with probability 1.
  - (b) Un-select every two neighboring triangles that were selected.
  - (c) Add all the edges of the selected triangles to  $\mathcal{TC}$ .
  - (d) Remove from  $\mathcal{T}$  all the selected triangles and their neighbors and update the degrees of the remaining triangles accordingly.
3. Add to  $\mathcal{TC}$  one edge of every remaining triangle in  $\mathcal{T}$ .
4. Output  $\mathcal{TC}$ .

**Theorem 3** For every  $\delta$ , Algorithm 3 constructs a triangle cover  $\mathcal{TC}$  of size  $C$  such that with probability at least  $5/6$ ,  $C_{OPT} \leq C \leq 3 \cdot C_{OPT} + \frac{\delta m}{2}$ .

**Proof:** First it is clear that  $\mathcal{TC}$  is indeed a triangle cover and therefore  $C \geq C_{OPT}$ .

To show that  $C \leq 3 \cdot C_{OPT} + \frac{1}{2}\delta m$  consider first the triangles that the algorithm adds to the cover during the loop of Step 2. Observe that these triangles are all edge-disjoint since whenever the algorithm selects neighboring triangles in Step 2.a, it un-selects them in Step 2.b. Also, any neighbor of a selected triangle is removed from  $\mathcal{T}$  and cannot be selected on the following iterations. Therefore, any other triangle cover must contain at least one edge of every triangle from  $\mathcal{TC}$  so the number of edges added to  $\mathcal{TC}$  during the loop is at most  $3 \cdot C_{OPT}$ .

In order to upper bound the number of triangles left in  $\mathcal{T}$  at the end of the loop of Step 2 we apply the following lemma.

**Lemma 5** For every  $i \in \{1, \dots, r\}$  let  $T_i$  be the number of triangles left in  $\mathcal{T}$  at the end of the  $i$ 'th iteration of Step 2. For  $i = 0$  let  $T_i = |\mathcal{T}|$ . Then for every  $i > 0$ ,

$$\text{Exp}[T_i \mid T_{i-1}] \leq \left(1 - \frac{1}{c_1}\right) T_{i-1}$$

where  $c_1 = 3c^2/(c-3)$  and  $c$  is the constant used in Step 2.a of Algorithm 3.

**Proof:** For every iteration  $i$  and degree  $j \in \{0, \dots, 3d\}$ , let  $T_{i,j}$  be the number of  $j$ -degree triangles left in  $\mathcal{T}$  at the end of the  $i$ 'th iteration so that  $T_i = \sum_{j=0}^{3d} T_{i,j}$ . At Step 2.a, every triangle  $t$  of degree  $j > 0$  is selected with probability  $1/(c^j)$  and is un-selected in Step 2.b only in case at least one of its neighbors is also selected. To calculate the probability that at least one neighbor of  $t$  is selected, let  $\Gamma_k(t)$  be the set of  $t$ 's neighbors at its  $k$ 'th edge for  $k \in \{1, 2, 3\}$ , and let  $d_k(t)$  be the size of  $\Gamma_k(t)$ . Then, the probability that at least one of  $t$ 's neighbors is selected is at most

$$\sum_{t' \in \Gamma(t)} \frac{1}{c \cdot d(t')} = \sum_{k=1}^3 \sum_{t' \in \Gamma_k(t)} \frac{1}{c \cdot d(t')} \leq \sum_{k=1}^3 \sum_{t' \in \Gamma_k(t)} \frac{1}{c \cdot d_k(t)} = \frac{3}{c} \quad (8)$$

where the inequality is true since for every  $t' \in \Gamma_k(t)$ ,  $d(t') \geq d_k(t)$ . Hence, we expect that on each iteration the algorithm un-selects only a constant fraction of the selected triangles. Therefore, for every  $j > 0$ , the probability that a  $j$ -degree triangle is added to  $\mathcal{TC}$  is at least  $\frac{1}{c^j} (1 - \frac{3}{c}) = \frac{1}{c'j}$  where  $c' = c^2/(c-3)$ . It follows that the expected number of  $j$ -degree triangles that are added to  $\mathcal{TC}$  on the  $i$ 'th iteration is at least  $T_{i-1,j}/(c'j)$ . Such triangles are removed from  $\mathcal{T}$  together with their  $j$  neighbors. So for every  $j > 0$ , we expect that the chosen  $j$ -degree triangles cause the removal of at least  $j \cdot T_{i-1,j}/(c'j) = T_{i-1,j}/c'$  additional triangles of some degree. The removal of a triangle can be caused by more than one selected neighbor, but since the selected triangles are all edge-disjoint, we counted every removed triangle at most three times. For  $j = 0$ , all the  $T_{i-1,0}$  triangles of degree 0 are selected and removed from  $\mathcal{T}$ . Summing up and canceling the repetitions, the expected total number of triangles removed from  $\mathcal{T}$  at the  $i$ 'th iteration is at least  $(1/3) \sum_{j=0}^{3d} T_{i-1,j}/c' = (1/c_1)T_{i-1}$  where  $c_1$  is as defined in the lemma, and the proof of the lemma is completed. ■

The next corollary follows from Lemma 5.

**Corollary 6** *By taking  $c = 6$ , after  $r = 36(\log(\frac{d}{\delta}) + 3)$  iterations,  $\text{Exp}[T_r] \leq \frac{\delta}{12d}|\mathcal{T}|$ .*

**Proof:** It follows from Lemma 5 that for every  $i > 0$ ,

$$\text{Exp}[T_i] = \sum_{a \in \{0 \dots |\mathcal{T}|\}} \text{Exp}[T_i | T_{i-1} = a] \cdot \Pr[T_{i-1} = a] \quad (9)$$

$$\leq \sum_{a \in \{0 \dots |\mathcal{T}|\}} \left(1 - \frac{1}{c_1}\right) \cdot a \cdot \Pr[T_{i-1} = a] \quad (10)$$

$$= \left(1 - \frac{1}{c_1}\right) \text{Exp}[T_{i-1}] \quad (11)$$

and therefore by simple induction  $\text{Exp}[T_r] \leq \left(1 - \frac{1}{c_1}\right)^r |\mathcal{T}|$ . Now  $c_1$  is minimal when  $c = 6$ , in which case it equals 36 and so,

$$\text{Exp}[T_r] \leq \left(1 - \frac{1}{c_1}\right)^{c_1 \cdot (\log(\frac{d}{\delta}) + 3)} |\mathcal{T}| < \exp(-\log(\frac{d}{\delta}) + 3) \cdot |\mathcal{T}| < \frac{\delta}{12d} |\mathcal{T}| \quad (12)$$

as required. ■

Using Markov's inequality, the probability that  $T_r > \frac{\delta}{2d}|\mathcal{T}|$  is less than 1/6. Now, since every edge belongs to at most  $d$  triangles, we have  $|\mathcal{T}| \leq dm$  and so with probability at least 5/6 the number of edges

added to the cover  $\mathcal{TC}$  in Step 3 is at most  $\frac{1}{2}\delta m$ . We conclude that in this case, the size of the cover is upper bounded by  $3 \cdot C_{OPT} + \frac{1}{2}\delta m$ , which completes the proof of Theorem 3. ■

Next we show how to modify Algorithm 3 in order to achieve a 3-distance approximation algorithm for triangle-freeness whose running time is independent of  $n$ . Specifically, the algorithm uniformly and independently selects  $\Theta(1/\delta^2)$  vertices and then for each triangle attached to a sampled vertex, determines whether or not it would have been added to  $\mathcal{TC}$  by Algorithm 3. This can be determined by examining only the  $\Theta(\log(d/\delta))$ -neighborhood of every sampled vertex.

**Algorithm 4 (Distance approximation to triangle-freeness)**

1. Uniformly and independently sample  $s = 2/\delta^2$  vertices from  $G$ . Let  $S = \{u_1, \dots, u_s\}$  be the multiset of the sampled vertices.
2. For every  $j \in \{1, \dots, s\}$  observe the subgraph  $G_r(u_j)$  induced by the  $(r+1)$ -neighborhood of  $u_j$ , where  $r = \Theta(\log \frac{d}{\delta})$  is as in Algorithm 3.
3. Run Algorithm 3 on  $\bigcup_{j=1}^s G_r(u_j)$ . For every  $u_j \in S$ , let  $\chi_j$  be the number of edges incident to  $u_j$  that the algorithm adds to the cover.
4. Let  $\hat{C} = \frac{n}{2s} \sum_{j=1}^s \chi_j$  and output  $\frac{1}{dn} \hat{C}$ .

**Theorem 4** Algorithm 4 is a 3-distance approximation algorithm for triangle-freeness. The query complexity and running time of the algorithm are  $d^{O(\log(d/\delta))}$ .

**Proof:** The claimed complexity of the algorithm can be easily verified. To prove the quality of the estimate, we fix the random bits  $\tau$  that Algorithm 3 uses and assume that Algorithm 4 uses the same  $\tau$  when executing Algorithm 3 in Step 3. Let  $\mathcal{TC}^\tau$  be the cover found by Algorithm 3 when using  $\tau$  and let  $C^\tau$  be its size. For every vertex  $v$  let  $x_v^\tau$  be the number of edges incident to  $v$  in  $\mathcal{TC}^\tau$ . In addition, let  $\hat{C}^\tau$  be the output of Algorithm 4.  $\hat{C}^\tau$  is an estimate to  $C^\tau$ . For every  $j$ , let  $\chi_j^\tau$  be the value of  $\chi_j$  when using  $\tau$ . We will show that for every sampled vertex  $u_j$ , the number  $\chi_j^\tau$  calculated by Algorithm 4, exactly equals  $x_{u_j}^\tau$ . The only error, therefore, is due to sampling which, as we will show, is not too large.

We first observe that at the end of every iteration of Algorithm 3, every triangle  $t$  can be found in one of the following three states. It might be added to  $\mathcal{TC}^\tau$  (and consequently removed from  $\mathcal{T}$ ), it might be removed from  $\mathcal{T}$  (without being added to  $\mathcal{TC}^\tau$ ) and it might remain in  $\mathcal{T}$ . In the following lemma we'll show that it is enough to observe the  $2r$ -neighborhood of a triangle in order to determine its state after  $r$  iterations. From a vertex point of view, this implies that inspecting its  $(r+1)$ -neighborhood suffices in order to determine the state of all the triangles incident to it after  $r$  iterations. To verify this, one can easily show by induction on  $i$  that the  $(i+1)$ -neighborhood of a vertex  $v$  contains the  $2i$ -neighborhood of the triangles incident to  $v$ . The following claim implies, therefore, that for every sampled vertex  $u_j$ ,  $\chi_j^\tau = x_{u_j}^\tau$ .

**Claim 7** For every triangle  $t$ , it is enough to observe its  $2r$ -neighborhood in order to determine its state at the end of the  $r$ 'th iteration of Algorithm 3.

**Proof:** We prove the lemma by induction on the iteration number  $i$ . For  $i = 1$ ,  $t$  is added to  $\mathcal{TC}$  in case it is selected and none of its neighbors are selected. It is removed from  $\mathcal{T}$  in case it is not selected and at least one of its neighbors,  $t'$ , is selected while none of  $t'$ 's neighbors is. Otherwise it remains in  $\mathcal{T}$ . So, in order to determine  $t$ 's state after the first iteration, it is enough to observe its 2-neighborhood. Assume that the

claim is true for  $i < r$ . Then for the  $r$ 'th iteration, by the same argument, one needs to know the state of  $t$ 's 2-neighborhood at the end of the  $r - 1$  iteration. That is, observing the  $2(r - 1) + 2 = 2r$ -neighborhood of  $t$  suffice, as required. ■

Now, for every  $j$ ,  $\chi_j^\tau$  is a random variable whose expected value is  $\frac{1}{n} \sum_{v \in V} x_v^\tau$ . Let  $\mu^\tau = \text{Exp}[\frac{1}{d} \chi_j^\tau]$  for some  $j$ . Thus,  $C^\tau = \frac{1}{2} \sum_{v \in V} x_v^\tau = \frac{dn}{2} \cdot \mu^\tau$ . The random variable  $\frac{1}{d} \chi_j^\tau$  gets values in the range  $[0, 1]$ . By an additive Chernoff bound,  $\Pr_S \left[ \left| \widehat{C}^\tau - C^\tau \right| > \frac{\delta}{2} dn \right] < 1/6$ . From Theorem 3 we know that the probability over the possible sequences of random bits  $\tau$ , that  $C^\tau$  is less than  $3 \cdot C_{OPT} + \frac{\delta m}{2}$  is at least  $5/6$ . Combining this and the fact that  $m \leq dn$ , we get that with probability at least  $2/3$ ,  $C_{OPT} \leq \widehat{C} \leq 3C_{OPT} + \delta dn$ . ■ (Theorem 4)

### 3.2 Generalizing the Result to Arbitrary Subgraphs

The result for triangle-freeness can be generalized to arbitrary subgraphs using some subgraph specific parameters. Assume that  $H$  consists of  $m_H$  edges and its diameter is  $\rho_H$ . Also, let  $d_H$  be the maximal number of subgraphs a single edge can belong to. For triangles  $d_H = O(d)$  but for other subgraphs it might be much larger. For example, for the cycle of length  $d$ , an edge in a complete subgraph of  $d$  vertices, belongs to  $(d - 2)!$  cycles of length  $d$ . For a fixed graph  $H$ ,  $G$  is called  $H$ -free if no subgraph of  $G$  is isomorphic to  $H$ . We say that a set of edges is an  $H$ -cover if its removal from the graph results in an  $H$ -free graph and denote by  $HC_{OPT}$  the minimum size of an  $H$ -cover of  $G$ . Two isomorphic copies of  $H$  in  $G$  are called *neighbors* if they share at least one edge. For an isomorphic copy  $h$  of  $H$ , the set of its neighbors is denoted by  $\Gamma(h)$  and its *degree*,  $d(h)$ , is defined as the size of  $\Gamma(h)$ . Using these definitions, Algorithms 3 and 4 can be modified to compute the distance of a bounded-degree graph from being  $H$ -free, for every fixed subgraph  $H$ .

**Theorem 5** *For every constant size subgraph  $H$ , there exists a variant of Algorithm 4 that is an  $m_H$ -distance approximation algorithm for  $H$ -freeness. The query and time complexity of the algorithm is  $d^{O(\rho_H \cdot m_H^2 \cdot \log(d_H/\delta))}$ .*

**Proof Sketch:** First we describe and analyze a variant of Algorithm 3 that constructs an  $H$ -cover of  $G$ . Let  $\mathcal{H}$  be the set of all the subgraphs isomorphic to  $H$  at the beginning of the algorithm and for every iteration  $i$ , let  $\mathcal{H}_i$  be the set of subgraphs left in  $\mathcal{H}$  at the end of the  $i$ 'th iteration. At Step 2.a every subgraph  $h$  selects itself with probability  $1/(c \cdot d(h))$ . By the same analysis as that of Algorithm 3, it is unselected at Step 2.b with probability at most  $m_H/c$  and therefore, for every  $i$ , at the  $i$ 'th iteration, at least  $(1/c_1)|\mathcal{H}_{i-1}|$  subgraphs are removed from  $\mathcal{H}$ . The constant  $c_1$  in this case equals  $m_H c^2 / (c - m_H)$ . By taking  $c = 2m_H$ , after  $r = 4m_H^2 (\log(d_H/\delta) + 3)$  iterations, with probability at least  $5/6$  the number of the remaining subgraphs is at most  $\frac{\delta}{2d_H} |\mathcal{H}|$ . Since every edge belongs to at most  $d_H$  subgraphs,  $|\mathcal{H}| \leq d_H m$ . Therefore, combining the fact that the number of subgraphs added to the cover during the loop is at most  $m_H \cdot HC_{OPT}$ , with probability at least  $5/6$ ,  $C \leq m_H \cdot HC_{OPT} + \frac{1}{2} \delta m$ .

The adapted version of Algorithm 4 outputs an estimate to  $HC$ , the size of the  $H$ -cover  $\mathcal{TC}$ . It can be verified that for every fixed subgraph  $H$ , observing the  $2r$ -neighborhood of some isomorphic copy  $h$  of  $H$  suffices in order to determine its state at the end of the  $r$ 'th iteration. Also, for every vertex, its  $(\rho_H \cdot (2r + 1))$ -neighborhood contains the  $2r$ -neighborhood of every subgraph it belongs to. So by the same analysis, we get an  $m_H$ -distance approximation for  $H$ -freeness.

The induced subgraph  $G_\tau(u)$  can be constructed using  $d^{O(\rho_H \cdot m_H^2 \cdot \log(d_H/\delta))}$  queries and so the total query complexity is as claimed. As for the running time, there are  $d^{O(\rho_H \cdot m_H^2 \cdot \log(d_H/\delta))}$  vertices. For every vertex, checking all the possible subgraphs it might belong to can be done by checking all the possible  $m_H$

combinations of edges in its  $\rho_H$  vicinity, i.e., using  $O(d^{\rho_H \cdot m_H})$  steps. So, going over all the subgraphs  $O(r)$  times takes  $d^{O(\rho_H \cdot m_H^2 \cdot \log(d_H/\delta))}$  steps.  $\square$

### 3.3 Approximating the Minimum Vertex Cover

We observe that Algorithm 3 is readily seen to be a randomized distributed approximation algorithm. By applying some modifications, it can yield a distributed algorithm for approximation the minimum vertex cover of a graph, which is one of the fundamental problems extensively studied in various settings in Computer Science. Details follow.

The distributed computation model consists of an underlying (synchronous) network  $G$  in which the vertices represent processors and the edges represent the communication channels. A distributed algorithm runs in  $k$  rounds for some number  $k$ , where in each round every vertex is allowed to send messages to its neighbors. After  $k$  rounds, each vertex completes its computation and the entire network achieves this way some global goal. For example, if the goal is to compute a vertex cover, then each vertex should decide whether or not it belongs to the vertex cover. In the local computation model, each vertex performs its task based on local information only, that is,  $k$  is smaller than the diameter of the graph. Usually there is a trade-off between the locality of the algorithm and the quality of the solution.

For completeness we present the distributed version of Algorithm 3 for the minimum vertex-cover problem. We start with some definitions. Two edges are considered *neighbors* if they share a common vertex. For an edge  $e$  we denote by  $\Gamma(e)$  the set of its neighbors and by  $d(e)$  the size of  $\Gamma(e)$ . We denote by  $VC_{OPT}$  the size of a minimum vertex cover. The following is a randomized distributed algorithm the finds a vertex cover whose size is at most  $(2 + \delta)$  times the minimum size with high constant probability.

**Algorithm 5 (Distributed approximation for minimum vertex cover)**

1. Every edge activates itself.
2. From  $i = 1$  to  $r = \Theta(\log(d/\delta))$ 
  - (a) Every 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 selected, unselect themselves.
  - (c) Every vertex that is attached to a selected edge, adds itself to the vertex cover.
  - (d) Selected edges and neighbors of selected edges, inactivate themselves.
  - (e) Active edges update their degrees to be the number of their active neighbors.
3. One vertex of every still active edge, adds itself to the vertex cover.

**Theorem 6** For every  $\delta > 0$  and every graph  $G = (V, E)$  with degree-bound  $d$ , Algorithm 5 constructs a vertex cover  $C \subseteq V$  such that with probability at least  $5/6$ ,  $VC_{OPT} \leq |C| \leq (2 + \delta) \cdot VC_{OPT}$ . The query complexity and running time of the algorithm are  $d^{O(\log(d/\delta))}$ .

The proof is very similar to the proof of Theorem 3, and hence we only give a sketch. **Proof Sketch:** Since an edge inactivates itself only when one of its endpoints is added to the vertex cover, and in Step 3 one end-point from each edge that is still active is added to the cover, all edges are covered by the end of the algorithm, as required. This implies also the lower bound on  $|C|$ .

Analogously to the proof of Theorem 3, with probability at least  $5/6$ , at the end of the Step 2, the number of inactive edges is at most  $\frac{\delta}{2d}|E|$ . Since the degree of every vertex is at most  $d$  and at least  $VC_{OPT}$  vertices must be selected in order to cover all the edges, we have  $|E| \leq d \cdot VC_{OPT}$ . Combining this with the fact that the number of vertices selected during Step 2 is at most  $2 \cdot VC_{OPT}$  we have,  $|C| \leq (2 + \delta)VC_{OPT}$ .  $\square$

### 3.3.1 A Sublinear Approximation of the Size of a Minimum Vertex Cover

The problem of approximating the size of a minimum vertex cover has been studied also in the context of sublinear algorithms. Parnas and Ron [PR05] show a reduction from local distributed approximation algorithms to sublinear algorithms. Using the distributed algorithm of Kuhn, Moscibroda and Wattenhofer [KMW06] mentioned in the introduction, their reduction gives an approximation with a multiplicative factor of 2 and an additive factor of  $\delta n$ , in time  $d^{O(\delta^{-3} \cdot \log(d))}$ . The dependence on the maximum degree  $d$  can be replaced by a dependence on  $\Theta(\bar{d}/\delta)$ . Algorithm 4, when modified to approximate the size of the minimum vertex cover, obtains an estimate of the same quality in time  $d^{O(\log(d/\delta))}$ . It can also be viewed as a reduction from Algorithm 5. For completeness we describe the modified algorithm.

#### Algorithm 6 (Sublinear Approximation for $VC_{OPT}$ )

1. Uniformly and independently sample  $s = 4/\delta^2$  vertices from  $G$ . Let  $S$  be the multiset of the sampled vertices.
2. For every  $v \in S$ , observe the subgraph  $G_r(v)$  induced by the  $(r + 1)$ -neighborhood of  $v$ , where  $r = \Theta(\log(\frac{d}{\delta}))$  is as in Algorithm 5.
3. Run Algorithm 5 on  $\bigcup_{v \in S} G_r(v)$  (in a sequential manner). For every  $v \in S$ , let  $\chi_v = 1$  if the algorithm adds  $v$  to the cover, otherwise  $\chi_v = 0$ .
4. Output  $\widehat{VC} = \frac{n}{s} \sum_{v \in S} \chi_v$ .

The proof is analogous to the proof of Theorem 4 and is omitted.

**Theorem 7** For every  $\delta > 0$ , and every graph  $G$ , Algorithm 6 outputs with probability at least  $2/3$  an estimate  $\widehat{C}$  that satisfies

$$VC_{OPT} - \delta n \leq \widehat{VC} \leq 2 \cdot VC_{OPT} + \delta n .$$

The query and time complexity of the algorithm are  $d^{O(\log(d/\delta))}$ .

We remark that the same modifications of the algorithm in [PR05] can be applied here to achieve a dependence on  $\Theta(\bar{d}/\delta)$  instead of  $d$  in the running time and query complexity.

## 4 Distance Approximation to Being Eulerian

A graph  $G$  is Eulerian if there exist a path in the graph that traverses every edge of  $G$  exactly once. It is a well known fact that a graph is Eulerian if and only if it is connected and all vertices have even degree or exactly two vertices have odd degree. Goldreich and Ron [GR02] gave a testing algorithm for this property in bounded-degree graphs, where the complexity of the algorithm is roughly linear in  $1/\epsilon$ . The algorithm can be easily adapted to general sparse graphs at a slight increase in the complexity. Here we give a distance approximation algorithm for general sparse graphs.

### Algorithm 7 (Distance approximation to being Eulerian)

1. Run the following variant of Algorithm 1: The sample size is  $\frac{128}{\delta^2 d^2}$ , the size bound is  $\frac{8}{\delta d}$  and for every sampled vertex  $v$ , if its connected component was found and all the vertices in this component are of even degree then  $\hat{n}_v$  is the number of vertices in this component, otherwise,  $\hat{n}_v = \infty$ . Let  $\hat{C}_e$  be the estimate of the algorithm for the number of connected components.
2. Uniformly and independently sample  $s = \frac{16}{\delta^2 d^2}$  vertices from  $G$ . Let  $S$  be the multiset of the sampled vertices.
3. Query the degree  $d_j$  of every sampled vertex  $u_j \in S$ .
4. For every  $j \in \{1, \dots, s\}$ , if  $d_j$  is odd then let  $\chi_j = 1$  otherwise let  $\chi_j = 0$ .
5. Let  $\hat{R} = \frac{n}{2s} \sum_{j=1}^s \chi_j + \hat{C}_e - 1$  and output  $\hat{R}/(\bar{d}n)$

**Theorem 8** For every  $\delta > 0$  and every graph  $G$  over  $n$  vertices and average degree  $\bar{d}$ , with probability at least  $2/3$ , Algorithm 7 outputs a estimate to the distance of  $G$  from being Eulerian. The query and time complexity of the algorithm is  $O((\delta \cdot \bar{d})^{-4} \beta)$ .

**Proof:** Consider the connected components of  $G$ . In each connected component, either all vertices have an even degree or there is a positive even number of vertices with an odd degree. In the former case we refer to the component as an *even-degree* component, and in the latter case we refer to it as an *odd-degree* component. Let  $C_e$  denote the number of even-degree components, and for each vertex  $v$  let  $x_v = 1$  if  $v$  is an odd-degree vertex, and let  $x_v = 0$  otherwise. We claim that the minimum number of edges that must be added to  $G$  in order to make it Eulerian is  $\frac{1}{2} \cdot \sum_{v \in V} x_v + C_e - 1$ .

It is easy to verify that these many edges suffice to make the graph Eulerian. To show that this number is necessary consider first the process of adding any  $C_e + C_o - 1$  edges that make the graph connected. Such a process necessarily leaves us with at least  $\sum_{v \in V} x_v - 2C_o + 2$  odd-degree vertices. To verify this observe that the addition of the edges defines a tree, whose nodes are the connected components. Let  $k$  be the number of leaves in this tree that correspond to even-degree components (where  $k \geq 0$ ). Each such leaf contributes an odd-degree vertex, and “uses”  $k$  end-points among the  $C_e + C_o - 1$  added edges. Every even-degree component that does not correspond to a leaf “uses” at least 2 additional edges. Hence, the remaining number of end points, which are attached to vertices in odd-degree components, is at most  $2(C_e + C_o - 1) - (k + 2(C_e - k)) = 2C_o + k - 2$ . Therefore, the number of odd-degree vertices that remain in odd-degree components is at least  $\sum_v x_v - (2C_o + k - 2)$ . Adding to this the (at least)  $k$  odd-degree vertices that belong to the even-degree components, we get at least  $\sum_v x_v - 2C_o + 2$  remaining odd-degree vertices.

So the total number of edges required is at least  $C_e + C_o - 1 + \frac{1}{2} \cdot (\sum_{v \in V} x_v - 2C_o) = \frac{1}{2} \cdot \sum_{v \in V} x_v + C_e - 1$  as claimed.

Now, Algorithm 1 with the modifications described in Step 1 of Algorithm 7, estimates the number of small connected components in which all the vertices are of even degree. As proved in Theorem 1, using the modified parameters, with probability at least  $5/6$ ,  $|\hat{C}_e - C_e| < \frac{\delta}{2}m$ . In addition, using an additive Chernoff bound, with probability at least  $5/6$ ,  $|\frac{n}{s} \sum_{j=1}^s \chi_j - \sum_{v \in V} x_v| < \frac{\delta}{2}m$ , and the claimed approximation is obtained.

The query and time complexity of Algorithm 1 (Step 1 of Algorithm 7) is  $O((\delta \cdot \bar{d})^{-4}\beta)$ . The rest of Algorithm 7 takes  $O((\delta \cdot \bar{d})^{-2})$  queries and time. Thus the total query and time complexity is  $O((\delta \cdot \bar{d})^{-4}\beta)$ . ■

## 5 Distance Approximation to Cycle-Freeness

The cycle-freeness property has been shown by Goldreich and Ron [GR02] to be testable in bounded-degree graphs in running time that is independent of  $n$ . In the general sparse model, testing this property requires at least  $\Omega(\sqrt{n})$  queries. To verify this, consider 2 families of graphs, each consisting of all the  $n!$  labelings of the following two  $n$ -vertex graphs. The first is the empty graph and the second consists only of a clique of size  $\sqrt{n}$  so that its distance to cycle-freeness is  $\Theta(n)$ . In order to distinguish between graphs that are selected uniformly from each of the two families, a testing algorithm must perform at least  $\Omega(\sqrt{n})$  queries. In this section we give a distance approximation algorithm for this property for bounded-degree graphs.

Let  $G$  be a simple graph with degree-bound  $d$ , and let  $C_1, \dots, C_k$  be the connected components of  $G$ . For every  $i$  we define the number of *extra* edges in  $C_i$  by  $x_i = m_i - (|C_i| - 1)$  where  $m_i$  is the number of edges in  $C_i$ . The distance of  $G$  to cycle-freeness equals therefore to  $X = \sum_{i=1}^k x_i$ . The following algorithm approximates this number up to an additive factor of  $\delta dn$  for every given  $\delta > 0$  and every bounded-degree graph.

### Algorithm 8 (Distance approximation to cycle-freeness)

1. Uniformly and independently sample  $t = \frac{32}{\delta^2}$  vertices from  $G$ . Let  $S = \{u_1, \dots, u_t\}$  be the multiset of the sampled vertices.
2. For every  $u_j \in S$ , perform a BFS starting from  $u_j$  until  $b = \frac{2}{\delta d}$  vertices have been reached or  $u_j$ 's connected component has been found.
3. For every  $j \in \{1, \dots, t\}$ , let  $\hat{n}_j$  be the number of vertices visited and let  $\hat{x}_j$  be the number of extra edges in  $u_j$ 's connected component in case it was found, otherwise  $\hat{x}_j = 0$ . Let  $d_j$  be the degree of  $u_j$  and also, let  $y_j = 1$  if  $u_j$  belongs to a connected component of size larger than  $b$ , otherwise  $y_j = 0$ .
4. Let  $\hat{X}_s = \frac{n}{t} \sum_{j=1}^t \frac{\hat{x}_j}{\hat{n}_j}$ ,  $\hat{m}_\ell = \frac{n}{2t} \sum_{j=1}^t d_j \cdot y_j$  and  $\hat{n}_\ell = \frac{n}{t} \sum_{j=1}^t y_j$ . ('s' stands for 'small' and 'l' for 'large'.) Let  $\hat{X} = \hat{X}_s + \hat{m}_\ell - \hat{n}_\ell$  and output  $\frac{1}{dn} \hat{X}$ .

**Theorem 9** For every  $\delta > 0$  and every graph  $G$  on  $n$  vertices and maximum degree  $d$ , with probability at least  $2/3$ , Algorithm 8 outputs a estimate to the distance of  $G$  from being cycle free. The query complexity

and running time complexity of the algorithm are  $O(\delta^{-3})$ .

**Proof:** We say that a connected component is *small* if its size is bounded by  $b$ , otherwise it is a *large* component. Let  $\mathcal{C}_s$  and  $\mathcal{C}_\ell$  be the sets of small and large connected components respectively and let  $X_s$  and  $X_\ell$  be the number of extra edges in these sets. In addition, for every vertex  $v$ , let  $n_v$  be the size of its connected component and  $x_v$  the number of extra edges in this component. In what follows, with slightly abuse of notation, we shall use  $v \in \mathcal{C}_s$  (similarly,  $v \in \mathcal{C}_\ell$ ) to mean that  $v$  belongs to a connected component in  $\mathcal{C}_s$  (respectively,  $\mathcal{C}_\ell$ ). Note that for every vertex  $u_j$  in the sample, if  $u_j \in \mathcal{C}_s$  then  $\hat{x}_j = x_j$ , while if  $u_j \in \mathcal{C}_\ell$  then  $\hat{x}_j = 0$ .

Now, for every  $j \in \{1, \dots, t\}$ , the random variable  $\chi_j = \frac{\hat{x}_j}{\hat{n}_j \cdot d}$  gets values in the range  $[0, 1]$  and (since  $\hat{x}_j = 0$  if  $u_j \in \mathcal{C}_\ell$ ), its expected value is  $\mu = \frac{1}{n} \sum_{v \in \mathcal{C}_f} \frac{x_v}{d \cdot n_v} = \frac{1}{dn} X_s$ . Thus by an additive Chernoff bound, with probability at least  $5/6$ ,  $|\hat{X}_s - X_s| < \frac{\delta}{4} dn$ .

As for the large connected components, since there are at most  $\frac{\delta}{2} dn$  such components,  $X_\ell \leq m_\ell - n_\ell + \frac{\delta}{2} dn$ . Now, for every  $j \in \{1, \dots, s\}$ ,  $\text{Exp}[y_j] = \frac{1}{n} n_\ell$  and  $\text{Exp}[d_j] = \frac{2}{n} m_\ell$ . Thus, with probability at least  $5/6$ ,  $|(m_\ell - n_\ell) - (\hat{m}_\ell - \hat{n}_\ell)| < \frac{\delta}{4} dn$  and the theorem follows.

The query and time complexity of every BFS is  $(b \cdot d) = O(1/\delta)$ , and so the total query and time complexity is  $O(\delta^{-3})$ . ■

## References

- [ACCL04] N. Ailon, B. Chazelle, S. Comandur, and D. Liue. Estimating the distance to a monotone function. In *Proceedings of the 8th RANDOM*, pages 229–236, 2004. To appear in *Random Structures and Algorithms*.
- [AKKR06] N. Alon, T. Kaufman, M. Krivilevich, and D. Ron. Testing triangle-freeness in general graphs. In *Proceedings of the 17th SODA*, pages 279–288, 2006.
- [CRT05] B. Chazelle, R. Rubinfeld, and L. Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SICOMP*, 34(6):1370–1379, 2005.
- [Fei04] U. Feige. On sums of independent random variables with unbounded variance and estimating the average degree in a graph. In *Proceedings of the 36th STOC*, pages 594–603, 2004.
- [FF05] E. Fischer and L. Fortnow. Tolerant versus intolerant testing for boolean properties. In *Proceedings of the 20th IEEE Conference on Computational Complexity*, pages 135–140, 2005.
- [FN05] E. Fischer and I. Newman. Testing versus estimation of graph properties. In *Proceedings of the 37th STOC*, pages 138–146, 2005.
- [GGR98] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *JACM*, 45(4):653–750, 1998.
- [GH61] R. E. Gomory and T. C. Hu. Multi-terminal network flows. *SIAM Journal on Applied Math*, 9(4):551–560, 1961.
- [GR02] O. Goldreich and D. Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002.

- [GR05] V. Guruswami and A. Rudra. Tolerant locally testable codes. In *Proceedings of the 9th RANDOM*, pages 306–317, 2005.
- [GR06] O. Goldreich and D. Ron. Approximating average parameters of graphs. In these proceedings., 2006.
- [Gus90] D. Gusfield. Very simple methods for all pairs network flow analysis. *SICOMP*, 19(1):143–155, 1990.
- [Kar93] D. Karger. Global min-cuts in RNC and other ramifications of a simple mincut algorithm. In *Proceedings of the 4th SODA*, pages 21–30, 1993.
- [KKR04] T. Kaufman, M. Krivelevich, and D. Ron. Tight bounds for testing bipartiteness in general graphs. *SICOMP*, 33(6):1441–1483, 2004.
- [KMW06] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. The price of being near-sighted. In *Proceedings of the 17th SODA*, pages 980–989, 2006.
- [Lub86] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SICOMP*, 15(2):1036–1055, 1986.
- [MR06] S. Marko and D. Ron. Distance approximation in bounded-degree and general sparse graphs. Manuscript. Available from: [www.eng.tau.ac.il/~danar/](http://www.eng.tau.ac.il/~danar/), 2006.
- [NGM97] D. Naor, D. Gusfield, and C. Martel. A fast algorithm for optimally increasing the edge connectivity. *SICOMP*, 26(4):1139–1165, 1997.
- [PR02] M. Parnas and D. Ron. Testing the diameter of graphs. *Random Structures and Algorithms*, 20(2):165–183, 2002.
- [PR05] M. Parnas and D. Ron. On approximating the minimum vertex cover in sublinear time and the connection to distributed algorithms. ECCC Report TR05-094., 2005.
- [PRR] M. Parnas, D. Ron, and R. Rubinfeld. Tolerant property testing and distance approximation. ECCC Report TR04-010, 2004, to appear in JCSS.