

## Abstract

We address the problem of approximating the distance of sparse graphs from having some predefined property  $\mathcal{P}$ . The distance of  $G$  from having  $\mathcal{P}$  is  $\varepsilon_{\mathcal{P}}(G)$  if the minimum number of edge modifications needed in order to enforce  $\mathcal{P}$  on  $G$  is  $\varepsilon_{\mathcal{P}}(G) \cdot \bar{m}$ , where  $\bar{m}$  is an upper bound on the number of edges in the graph. When dealing with bounded-degree graphs of maximal degree  $d$  we take  $\bar{m} = dn$ , where  $n$  is the number of vertices in the graph. For general sparse graphs of average degree  $\bar{d}$ , we take  $\bar{m} = \bar{d}n$ . We say that  $A$  is an  $(\alpha, \delta)$ -distance approximation algorithm if with high probability its estimate  $\hat{\varepsilon}_{\mathcal{P}}$  satisfies  $\varepsilon_{\mathcal{P}}(G) - \delta \leq \hat{\varepsilon}_{\mathcal{P}} \leq \alpha \cdot \varepsilon_{\mathcal{P}}(G) + \delta$ . We say that it is a  $\delta$ -distance approximation algorithm if  $|\hat{\varepsilon}_{\mathcal{P}} - \varepsilon_{\mathcal{P}}(G)| \leq \delta$ . We present the following distance approximation algorithms:

- **Subgraph-Freeness.** For any fixed subgraph  $H$ , we give an  $(m_H, \delta)$ -distance approximation algorithm for  $H$ -freeness in bounded-degree graphs, where  $m_H$  is the number of edges in  $H$ . Its query and time complexity is  $d^{O(\phi_H \cdot m_H^2 \cdot \log(d_H/\delta))}$ , where  $\phi_H$  is the diameter of  $H$  and  $d_H$  is the number of  $H$ -subgraphs that an edge in  $G$  can belong to. In particular, for triangle-freeness this implies a  $(3, \delta)$ -distance approximation algorithm whose query and time complexity is  $d^{O(\log(d/\delta))}$ .
- **$k$ -Edge-Connectivity.** A  $\delta$ -distance approximation algorithm for  $k$ -edge-connectivity in general sparse graphs whose query and time complexity is  $O\left(\left(\frac{k}{\delta d}\right)^6 \log\left(\frac{k}{\delta d}\right)\right)$ .
- **Eulerian.** A  $\delta$ -distance approximation algorithm for being Eulerian in general sparse graphs whose query and time complexity is  $O(\delta^{-4} \bar{d}^{-4})$ .
- **Cycle-Freeness.** A  $\delta$ -distance approximation algorithm for cycle-freeness in bounded-degree graphs whose query and time complexity is  $O(\delta^{-3} d^{-2})$ .

Our subgraph-freeness distance approximation algorithm can be executed distributively, achieving a multiplicative approximation ratio of  $(m_H + \delta)$  to the minimum subgraph cover of a graph, that is, to the minimum set of edges whose removal results in a subgraph free graph. The number of rounds needed is  $O(m_H^2 \log(d_H/\delta))$ . In [KMW06], a more general algorithm was presented, but for this task it achieves the same approximation ratio using  $O((m_H/\delta)^3 \cdot \log d_H)$  rounds. Further modifications to our algorithm give a  $(2 + \delta)$  distributed approximation algorithm to the minimum vertex cover of a graph in  $O(\log(d/\delta))$  rounds. In the context of distributed vertex cover, our algorithm is similar to the  $O(\log n)$  distributed algorithms for maximal independent set of [Lub86] and maximal matching of [II86].

In addition, a sublinear approximation to the size of the minimum vertex cover,  $VC_{OPT}$ , can be achieved using a modified version our algorithm. The output in this case,  $\hat{C}$ , satisfies  $VC_{OPT} - \delta n \leq \hat{C} \leq 2 \cdot VC_{OPT} + \delta n$ . The query and time complexity is  $d^{O(\log(d/\delta))}$ . In [PR05] it was shown how to use [KMW06] in order to achieve a sublinear approximation for this task of the same quality of approximation using  $d^{O(\delta^{-3} \log d)}$  queries and time.

# Acknowledgments

I would like to thank my advisor Prof. Dana Ron for guiding me in my research, sharing with me a lot of her knowledge and providing insights through many useful discussions. I would also like to thank my advisor Prof. Ran Raz for drawing my attention to this field, providing insights and tips and for his valuable support.

# 1 Introduction

The growing need to cope with large data sets such as the World Wide Web and DNA sequences has made polynomial time algorithms, which are usually considered as efficient, to be impractical in those situations. This has led to the development of algorithms whose running time is sublinear in the input size or even independent of it. Such algorithms are able to read only a small portion of the input and thus it is reasonable and is mostly the case that these algorithms are probabilistic algorithms which output approximate solutions. *Property testing* and its extension *distance approximation* are two of the notions that were proposed in this context. We start by presenting property testing and then turn to distance approximation, which is the main focus of this thesis.

## 1.1 Property Testing

Property testing can be viewed as a relaxation of decision problems. Instead of determining whether a given object (function, graph etc.) has some predefined property (linearity, connectivity etc.) or not, a property testing algorithm is required to distinguish, with high probability, only between objects that have the property and those which are *far* from having it. An object is considered far from having a property if a large fraction of the object needs to be modified so that the new object will have the property. Thus, the behavior of testing algorithms is undefined for objects that do not have the property but are also not far from having it, i.e., for objects which are *close* to having the property. This is inevitable since we cannot expect from a sublinear algorithm to find the few places of the object that distinguish it from having the property. This relaxation allows for the design of very efficient algorithms.

Property Testing was first introduced by Rubinfeld and Sudan [RS96] in the context of testing whether a function is a low-degree polynomial. Combinatorial objects such as graphs were first investigated by Goldreich, Goldwasser and Ron in [GGR98]. Since then, a great progress has been made in testing algorithms for various objects such as functions, graphs, strings and geometrical objects. See [Fis01, Gol98, Ron01] for surveys.

Testing graph properties can be formulated as follows. For a predefined graph property  $\mathcal{P}$ , a testing algorithm is given query access to a graph  $G$  and an approximation parameter  $\varepsilon$ , and is required to distinguish with probability at least  $2/3$  between the case that  $G$  has  $\mathcal{P}$  and the case that  $G$  is  $\varepsilon$ -far from having  $\mathcal{P}$ . A graph is said to be  $\varepsilon$ -far from having a property  $\mathcal{P}$  if the number of edges that must be added or removed so that  $G$  will have  $\mathcal{P}$ , is at least  $\varepsilon\bar{m}$ , where  $\bar{m}$  is some given upper bound on the number of edges in the graph. The distance of  $G$  from having  $\mathcal{P}$ , denoted by  $\varepsilon_{\mathcal{P}}(G)$ , is then the maximal  $\varepsilon$  such that  $G$  is  $\varepsilon$ -far from having  $\mathcal{P}$ . Equivalently, the distance of  $G$  from having  $\mathcal{P}$  is  $\varepsilon_{\mathcal{P}}(G)$  if the minimum number of edge modifications needed in order to enforce  $\mathcal{P}$  on  $G$  is  $\varepsilon_{\mathcal{P}}(G) \cdot \bar{m}$ .

Graph properties were investigated in three models, each defining the type of queries that the algorithm is provided with to access the graph and the exact notion of the term  $\varepsilon$ -far from having a property, which is determined by the upper bound  $\bar{m}$  on the number of edges.

**Dense graphs.** The dense model was presented and first investigated in [GGR98]. The representation used for the graphs in this model is their adjacency-matrix, which is most suitable for dense graphs whose average degree is of order  $\Theta(n)$ , where  $n$  is the number of vertices in the graph. The algorithm is allowed to use *vertex-pair queries* i.e., to query for any pair of vertices  $(u, v)$  whether there is an edge between them or not. An  $n$ -vertex dense graph is said to be  $\varepsilon$ -far from having some property if at least  $\varepsilon n^2$  edges must be added or removed so the graph will have the property. This is in close relation to the graph representation. For some work done in this model see [GGR98, AFKS00, Alo02, AK02, GT03, AS04, AS05a, AS05b].

**Bounded-degree graphs.** In this model, which was introduced in [GR02], graphs are represented by incidence-lists of size  $d$  where  $d = O(1)$  is the maximum degree of the graph. The queries allowed are *neighbor queries* in which the algorithm query for the  $i$ 'th neighbour of a vertex  $v$ , where  $i \leq d$ . Here, again with relation to the graph representation, a graph is  $\varepsilon$ -far from having some property if at least  $\varepsilon dn$  edges must be added or removed so the graph will have the property. See [GR99, GR00, BOT02, GR02] for related work.

**General graphs.** This model was introduced in [PR02] and further extended in [KKR04]. The graphs studied here lie between the two extremes. Their average degree, denoted by  $\bar{d}$ , is of order  $o(n)$ . Sparse general graphs are general graphs whose average degree  $\bar{d}$  is of order  $O(1)$  but their maximal degree  $d$  is not necessarily bounded. The algorithm can use both vertex-pair queries and neighbor queries. In addition it can use *degree queries* in which the degree of a vertex  $v$  is returned. Independently of graph representation, a graph is considered  $\varepsilon$ -far from having a property if at least  $\varepsilon m$  edges must be added or removed so the graph will have the property, where  $m$  is the number of edges in the graph. A more general definition requires only that  $m$  be an upper bound on the number of edges, as in the dense and bounded-degree models, in which the upper bounds are  $n^2$  and  $dn$  respectively. We note that our results for general graphs works as well in view of this definition. We assume that the algorithm is provided with the average degree  $\bar{d}$  of the graph, otherwise, we can find an estimate of it using  $O(\sqrt{n})$  queries [Fei04, GR06]<sup>1</sup>. The query and time complexity in this model is analyzed with respect to  $\bar{d}$ . For related work see [PR02, AKKR06].

Not all graph properties are interesting in terms of testing in all three models. For example, testing connectivity is trivial in the dense model as long as  $\varepsilon > 1/n$  since every graph is  $1/n$ -close to being connected in this model. Furthermore, there are properties for which the complexity of their testing algorithms differ by much when considered in the different models. For example, testing bipartiteness in bounded-degree graphs requires at least  $\Omega(\sqrt{n})$  queries [GR02] (an almost tight result [GR99] uses  $\sqrt{n} \cdot \text{poly}((\log n)/\varepsilon)$  queries) where in the dense model  $\text{poly}(1/\varepsilon)$  queries suffice [GGR98]. For general graphs,

---

<sup>1</sup>Specifically, Feige [Fei04] presented an algorithm which outputs an estimate  $d^*$  to the average degree of the graph, such that  $d^* \leq \bar{d} \leq (2 + \varepsilon)d^*$ . His algorithm uses  $O(\frac{1}{\varepsilon}\sqrt{n/d_0})$  degree queries and is applicable to all graphs of average degree at least  $d_0$ . By using also neighbor queries, Goldreich and Ron [GR06] improved the estimate so that  $(1 - \varepsilon)d^* \leq \bar{d} \leq (1 + \varepsilon)d^*$  using  $O(\sqrt{n}) \cdot \text{poly}(\frac{1}{\varepsilon})$  queries.

the task is solvable using<sup>2</sup>  $\tilde{O}(\min(\sqrt{n}, n^2/m))$  queries [KKR04], where  $m$  is the number of edges in the graph.

## 1.2 Distance Approximation

In general, property testing algorithms cannot be applied as is to get an estimate of the distance to having the property. This is due to the unexpected behavior of a testing algorithm on objects which are close to having the property. To see this, consider a tester that rejects any object which does not have the property (as a standard decision algorithm). In this case no information regarding the distance is revealed. This has led Parnas, Ron and Rubinfeld [PRR04] to propose the notions of *tolerant testing* and *distance approximation*. A tolerant testing algorithm is provided with two approximation parameters  $\delta_1, \delta_2$  and is required to distinguish between graphs which are  $\delta_1$ -close to having the property and graphs which are  $\delta_2$ -far from having it. In distance approximation, the task is to output an estimate to the distance of an object from having some predefined property. The quality of the estimate depends on a given approximation parameter. Preferably, the algorithm approximate the distance up to any additive factor  $\delta$ , but we allow also other approximations.

Parnas, Ron and Rubinfeld [PRR04] presented tolerant testing algorithms for clustering and a distance approximation algorithm for monotonicity of functions. Ailon et. al. [ACCL04] improve on the latter by reducing the dependence on the size of the domain,  $n$ , from polylogarithmic to logarithmic. A distance approximation algorithm for connectivity of graphs was given by Chazelle, Rubinfeld and Trevisan in [CRT05]. Many distance approximation algorithms for coloring and partitioning problems on dense graph are implied by their testing algorithms presented by Goldreich, Goldwasser and Ron [GGR98]. Guruswami and Rudra [GR05] presented tolerant testing algorithms for several constructions of locally testable codes.

A special interest in property testing and distance approximation algorithms is of algorithms whose query and time complexity depend only on the approximation parameter  $\delta$  and is independent of the input size. One of the goals in the study of property testing is to characterize the properties which are testable in time which is independent of the input size. Another task, introduced by Parnas, Ron and Rubinfeld in [PRR04], is to identify the properties that are also distance approximable to within any additive factor in time which is independent of the size of the input. Fischer and Fortnow [FF05] showed that for boolean properties of functions, not all properties that are testable in time which is independent of the input size, also have distance approximation algorithm whose running time is such.

For graph properties, an important result is that of Fischer and Newman [FN05]. They prove that in the dense model, any graph property that is testable in time which is independent of input size, is also tolerantly testable and therefore distance approximable to within any additive factor, in time independent of  $n$ . An interesting question that motivates this thesis is therefore whether this holds also for the bounded-degree and general sparse graphs.

---

<sup>2</sup>The  $\tilde{O}(\cdot)$  notation hides logarithmic factors, that is,  $\tilde{O}(f(n))$  means  $O(f(n) \cdot \text{polylog}(f(n)))$ .

### 1.3 Our Work

We investigate the approximability of several graph properties which were shown by Goldreich and Ron in [GR02] to be testable in the bounded-degree model in time which is independent of the size of the input. While some of our results hold also for general graphs, the properties studied are interesting in the context of property testing and distance approximation of sparse graphs but not of dense graphs. We start by some definition and then summarize our results and their implications.

A distance approximation algorithm for a predefined graph property  $\mathcal{P}$ , is given a graph  $G$  and an approximation parameter  $\delta$ . It is then required to output an estimate  $\widehat{\varepsilon}_{\mathcal{P}}$  to the distance  $\varepsilon_{\mathcal{P}}(G)$  of  $G$  from having  $\mathcal{P}$ . The *distance* of  $G$  from having  $\mathcal{P}$  is  $\varepsilon_{\mathcal{P}}(G)$  if the minimum number of edges that must be added or removed from  $G$  in order to enforce  $\mathcal{P}$  is  $\varepsilon_{\mathcal{P}}(G) \cdot \bar{m}$ , where  $\bar{m}$  is some upper bound on the number of edges of  $G$ . The result is of course meaningful when the upper bound does not deviate by much from the correct number of edges. In view of the graph models discussed above, in the dense case the upper bound is  $n^2$ , in the bounded degree  $dn$  and in the general case  $\bar{m} = \bar{d}n/2$ .

**Definition 1** For  $\alpha \geq 1$  and  $\delta > 0$ , we say that a distance approximation algorithm is an  $(\alpha, \delta)$ -distance approximation algorithm if with probability at least  $2/3$ ,

$$\varepsilon_{\mathcal{P}}(G) - \delta \leq \widehat{\varepsilon}_{\mathcal{P}} \leq \alpha \cdot \varepsilon_{\mathcal{P}}(G) + \delta .$$

The output of the algorithm,  $\widehat{\varepsilon}_{\mathcal{P}}$ , is said to be an  $(\alpha, \delta)$ -estimate to  $\varepsilon_{\mathcal{P}}(G)$ . A distance approximation algorithm is a  $\delta$ -distance approximation algorithm if  $\alpha = 1$ , i.e., if with probability at least  $2/3$ ,

$$|\widehat{\varepsilon}_{\mathcal{P}} - \varepsilon_{\mathcal{P}}(G)| \leq \delta .$$

Here  $\widehat{\varepsilon}_{\mathcal{P}}$  is said to be a  $\delta$ -estimate to  $\varepsilon_{\mathcal{P}}(G)$ . More generally, if there is some normalized cost measure  $M$ , then an algorithm is an  $(\alpha, \delta)$ -approximation algorithm for  $M$  if for every input  $I$ , with probability at least  $2/3$ , the output  $\widehat{M}$  of the algorithm satisfies:  $M(I) - \delta \leq \widehat{M} \leq \alpha \cdot M(I) + \delta$ .

The following is a summary of our results. Here and in the rest of the thesis, we let  $G$  be a graph on  $n$  vertices,  $m$  edges, maximal degree  $d$  and average degree  $\bar{d}$ .

- **Subgraph-Freeness.** In Section 2 we give a  $(3, \delta)$ -distance approximation algorithm for the triangle-freeness property in bounded-degree graphs. The query and time complexity of the algorithm is  $d^{O(\log(d/\delta))}$ . We generalize the algorithm in Section 2.2 to work for the subgraph-freeness property for any fixed subgraph  $H$ . Let  $m_H$  be the number of edges in  $H$ ,  $\phi_H$  the diameter of  $H$  and  $d_H$  the maximal number of subgraphs a single edge can belong to. Then, the generalization yields an  $(m_H, \delta)$ -distance approximation algorithm whose query and time complexity is  $d^{O(\phi_H \cdot m_H^2 \cdot \log(d_H/\delta))}$ .
- **$k$ -Edge-Connectivity.** In Section 3 we describe a  $\delta$ -distance approximation algorithm for the  $k$ -edge-connectivity property in general graphs. Its query and time complexity is  $O\left(\left(\frac{k}{\delta d}\right)^6 \log\left(\frac{k}{\delta d}\right)\right)$ .

- **Eulerian.** In Section 4 we give a  $\delta$ -distance approximation algorithm for the Eulerian property in general graphs. Its running time is  $O(\delta^{-4}d^{-4})$ .
- **Cycle-Freeness.** In Section 5 we present a  $\delta$ -distance approximation algorithm for the cycle-freeness property in simple bounded-degree graphs. Its query and time complexity is  $O(\delta^{-3}d^{-2})$ .

Note the difference between the result for subgraph-freeness and the rest of the properties in terms of the quality of the estimate. All four properties are testable in their corresponding model (bounded-degree or general) in time which is independent of input size. 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 which is independent of  $n$ . This is still an open question.

### 1.3.1 Relations to Other Settings

Our distance approximation algorithm for subgraph-freeness is related to the approximation of combinatorial problems, such as the minimum vertex cover (VC), in two other settings: distributed algorithms and sublinear algorithms.

*Distributed Approximation Algorithms.* In Section 2.3 we observe that our algorithm can be executed distributively and can approximate the minimum VC and the minimum subgraph cover of a graph (a subgraph cover is a set of edges whose removal results in a subgraph-free graph). We have learnt that in the context of minimum VC, our algorithm is similar to the distributed algorithms of Luby [Lub86] for maximal independent set and of Israeli and Itai [II86] for maximal matching.

When adapted to the distributed setting, our algorithm achieves with high probability, an approximation ratio of  $(2 + \delta)$  to the minimum VC using  $O(\log(d/\delta))$  rounds and also an approximation ratio of  $(m_H + \delta)$  to the subgraph cover using  $O(m_H^2 \log(d_H/\delta))$  rounds. A variant of the distributed algorithms of Luby [Lub86] for maximal independent set and of Israeli and Itai [II86] for maximal matching, can 2-approximate the minimum VC in  $O(\log n)$  expected number of rounds. We note that our adapted algorithm for VC is similar to theirs and that in the context of minimum VC, a modified version of their algorithm can also achieve the same result as ours. An approximation ratio that is slightly above 2 with number of rounds that is independent of  $n$  can also be obtained by the recent result of Kuhn, Moscibroda and Wattenhofer [KMW06]. Their algorithm falls into a more general framework but as for the best approximation ratios they can achieve, their result implies a  $(2 + \delta)$ -approximation for VC using  $O(\delta^{-3} \cdot \log d)$  rounds and an  $(m_H + \delta)$ -approximation for subgraph cover using  $O((m_H/\delta)^3 \cdot \log d_H)$  rounds. Other distributed algorithms for approximating the minimum VC include [PR01, GKPS05].

*Sublinear Approximation Algorithms.* Our algorithm can also be adapted to approximate the minimum VC in sublinear-time. Parnas and Ron [PR05] gave a simple combinatorial  $(2 \log d + 1, \delta)$ -approximation algorithm for this task whose query and time complexity is

$O(\delta^{-2} \cdot d^{\log d})$  (here, the normalization in the approximation ratio is by the number of vertices). They observed that distributed results for VC can be transformed into sublinear approximation algorithms, and in particular, basing on [KMW06] gave a  $(2, \delta)$ -approximation algorithm whose query and time complexity is  $d^{O(\delta^{-3} \log d)}$ .

In Section 2.4 we observe that a variant of our algorithm is a  $(2, \delta)$ -approximation algorithm to the minimum VC whose query and time complexity is  $d^{O(\log(d/\delta))}$ . It can also be viewed as a reduction from a variant of the algorithms of [Lub86, II86].

## 2 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  $H$ . We focus on triangles and then generalize the result to arbitrary subgraphs.

The problem of testing whether a graph is  $\varepsilon$ -far from being  $H$ -free for a constant-size subgraph  $H$ , has been studied quite extensively. In the dense model, Alon et. al. [AFKS00] showed that the problem is solvable using a number of queries which, although has a tower-type dependence in  $1/\varepsilon$ , is independent of the size of the input graph. A distinction between bipartite and non-bipartite graphs was made in [Alo02]. There, it is proved that for any non-bipartite subgraph  $H$ , a super polynomial dependence in  $1/\varepsilon$  is necessary, where for bipartite subgraphs,  $O(1/\varepsilon)$  queries suffice. In any case, for dense graphs, by the work of [FN05] there exists a  $\delta$ -distance approximation algorithm for the  $H$ -freeness property whose query and time complexity is independent of the size of  $n$ . In the general model, Alon et. al. [AKKR06] proved a lower bound of  $\Omega(n^{1/3})$  on the number of queries needed for testing  $H$ -freeness, for every non-bipartite subgraph  $H$ . For triangles, they give an upper bound of  $O(n^{6/7})$ . It is still an open question what is the exact complexity of testing triangle-freeness or  $H$ -freeness in general graphs. For sparse general graphs they prove that testing  $H$ -freeness requires  $\Omega(\sqrt{n})$  queries. In contrast, in the bounded-degree model, Goldreich and Ron [GR02] showed that the problem is solvable using only  $\Theta(1/\varepsilon)$  queries.

We focus here on approximating the distance to triangle-freeness in bounded-degree graphs. We 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  $n$ .

### 2.1 Triangle-Freeness

Let  $G$  be an undirected graph with degree at most  $d$ . 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 then 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  $i \in [\ell - 1]$ , the triangles  $t_i$  and  $t_{i+1}$  are neighbors. The  $k$ -*neighborhood* of a triangle  $t$  is then 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  $TC_{OPT}$  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  $TC_{OPT}$ .

**Algorithm 1 (Minimum triangle cover approximation)**

1. Let  $\mathcal{T}$  be the set of all the triangles in  $G$  and let  $\mathcal{C} = \emptyset$  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) Unselect every two neighboring triangles that were selected.
  - (c) Add all the edges of the selected triangles to  $\mathcal{C}$ .
  - (d) Remove from  $\mathcal{T}$  all the selected triangles and their neighbors and update the degrees of the remaining triangles accordingly.
3. Add to  $\mathcal{C}$  one edge of every remaining triangle in  $\mathcal{T}$ .
4. Output  $\mathcal{C}$ .

**Theorem 1** For every  $\delta$ , Algorithm 1 constructs a triangle cover  $\mathcal{C}$  of size  $C$  such that with probability at least  $5/6$ ,

$$TC_{OPT} \leq C \leq 3 \cdot TC_{OPT} + \frac{\delta m}{2}.$$

**Proof:** First it is clear that  $\mathcal{C}$  is indeed a triangle cover. This is true since the algorithm removes triangles from  $\mathcal{T}$  only after at least one of their edges is added to the cover. The edges of the triangles that remain in  $\mathcal{T}$  at the end of the loop are all added to the cover. Therefore  $C \geq TC_{OPT}$ .

To show that  $C \leq 3 \cdot TC_{OPT} + \frac{1}{2}\delta m$  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 at step 2.a, it unselect them at 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  $\mathcal{C}$  so the number of edges added to  $\mathcal{C}$  during the loop is at most  $3 \cdot TC_{OPT}$ .

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 1** 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  $T^0 = |\mathcal{T}|$ . Then for every  $i > 0$ ,

$$\text{Exp}[T^i | T^{i-1}] \leq \left(1 - \frac{1}{c_1}\right) T^{i-1}$$

where  $c_1 = 3c^2/(c-3)$  and  $c$  is the constant used in step 2.a of Algorithm 1.

**Proof:** For every iteration  $i$  and degree  $j \in \{0, \dots, 3d\}$ , let  $T_j^i$  be the number of  $j$ -degree triangles left in  $\mathcal{T}$  at the end of the  $i$ 'th iteration so that  $T^i = \sum_{j=0}^{3d} T_j^i$ . At step 2.a, every triangle  $t$  of degree  $j > 0$  is selected with probability  $1/(cj)$  and is unselected at 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

$$\begin{aligned} \sum_{t' \in \Gamma(t)} \frac{1}{c \cdot d(t')} &= \sum_{k=1}^3 \sum_{t' \in \Gamma_k(t)} \frac{1}{c \cdot d(t')} \\ &\leq \sum_{k=1}^3 \sum_{t' \in \Gamma_k(t)} \frac{1}{c \cdot d_k(t)} \\ &= \frac{3}{c} \end{aligned}$$

where the inequality is true since for every  $t' \in \Gamma_k(t)$ ,  $d(t') \geq d_k(t)$ . So, we expect that on each iteration the algorithm unselects only a constant fraction of the selected triangles. Therefore, for every  $j > 0$ , the probability that a  $j$ -degree triangle is added to  $\mathcal{C}$  is at least  $\frac{1}{c_j} \left(1 - \frac{3}{c}\right) = \frac{1}{c'j}$  where  $c' = c^2/(c-3)$ . Thus, the expected number of  $j$ -degree triangles that are added to  $\mathcal{C}$  on the  $i$ 'th iteration is at least  $T_j^{i-1}/(c'j)$ . 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  $j \cdot T_j^{i-1}/(c'j) = T_j^{i-1}/c'$  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  $T_0^{i-1}$  triangles of degree 0 are selected and removed from  $\mathcal{T}$ . Summing up and cancelling the repetitions, the expected total number of triangles removed from  $\mathcal{T}$  at the  $i$ 'th iteration is at least  $(1/3) \sum_{j=0}^{3d} T_j^{i-1}/c' = (1/c_1)T^{i-1}$  where  $c_1$  is as defined in the lemma, which follows. ■

**Corollary 1** 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 1 that for every  $i > 0$ ,

$$\text{Exp}[T^i] = \sum_{a \in \{0 \dots |\mathcal{T}|\}} \text{Exp}[T^i | T^{i-1} = a] \cdot \Pr[T^{i-1} = a]$$

$$\begin{aligned}
&\leq \sum_{a \in \{0 \dots |\mathcal{T}|\}} \left(1 - \frac{1}{c_1}\right) \cdot a \cdot \Pr[T^{i-1} = a] \\
&= \left(1 - \frac{1}{c_1}\right) \text{Exp}[T^{i-1}]
\end{aligned}$$

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,

$$\begin{aligned}
\text{Exp}[T^r] &\leq \left(1 - \frac{1}{c_1}\right)^{c_1 \cdot (\log(\frac{d}{\delta}) + 3)} |\mathcal{T}| \\
&< \left(\frac{1}{e}\right)^{\log(\frac{d}{\delta}) + 3} |\mathcal{T}| \\
&< \frac{\delta}{12d} |\mathcal{T}|
\end{aligned}$$

as required.  $\blacksquare$

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  $\mathcal{C}$  at step 3 is at most  $\frac{1}{2}\delta m$ . We conclude that in this case,  $C \leq 3 \cdot TC_{OPT} + \frac{1}{2}\delta m$ , which proves Theorem 1.  $\blacksquare$

Next we show how to use Algorithm 1 in order to achieve a distance approximation algorithm for triangle-freeness that outputs with high probability a  $(3, \delta)$ -estimate and whose running time is independent of  $n$ . The way we do that is similar in spirit to the way the sublinear approximation algorithm of VC is obtained in [PR05]. That is, by uniformly and independently selecting  $\Theta(1/\delta^2)$  vertices and then for each triangle attached to a sampled vertex, determining whether or not it would have been added to  $\mathcal{C}$  by Algorithm 1. This can be determined by examining only the  $\Theta(\log(d/\delta))$ -neighborhood of every sampled vertex.

**Algorithm 2 (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  $j \in [s]$  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 1.
3. Run Algorithm 1 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  $\widehat{C} = \frac{n}{2s} \sum_{j=1}^s \chi_j$  and output  $\frac{1}{dn} \widehat{C}$ .

**Theorem 2** For every  $\delta > 0$  and every bounded-degree graph  $G$ , Algorithm 2 outputs with probability at least  $2/3$  a  $(3, \delta)$ -estimate to the distance of  $G$  from being triangle-free. The query and time complexity of the algorithm is  $d^{O(\log(d/\delta))}$ .

**Proof:** For every  $j$ , the induced subgraph  $G_r(u_j)$  can be obtained by performing BFS starting at  $u_j$  for  $r + 2$  steps. The number of neighbor queries needed is  $d^{O(\log(d/\delta))}$  so the total query complexity is as claimed. Going over all the triangles in  $\bigcup_{j=1}^s G_r(u_j)$  can be done by sequentially going over the vertices and for every vertex observing the at most  $d^2$  triangles attached to it. Step 3 can be done by  $O(\log(d/\delta))$  such passes thus the total number of steps needed is  $d^{O(\log(d/\delta))}$  and the claimed time complexity follows.

To prove the quality of the estimate, we fix the random bits  $\tau$  that Algorithm 1 uses and assume that Algorithm 2 uses the same  $\tau$  when executing Algorithm 1 at step 3. Let  $\mathcal{C}^\tau$  be the cover found by Algorithm 1 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  $\mathcal{C}^\tau$ . In addition, let  $\widehat{C}^\tau$  be the output of Algorithm 2.  $\widehat{C}^\tau$  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 2, 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 1, every triangle  $t$  can be found in one of the following three states. It might be added to  $\mathcal{C}^\tau$  (and consequently removed from  $T$ ), it might be removed from  $T$  (without being added to  $\mathcal{C}^\tau$ ) and it might remain in  $T$ . 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 its  $(r + 1)$ -neighborhood suffice in order to determine the state of all the triangles incident to it after  $r$  iterations. To see 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 lemma implies, therefore, that for every sampled vertex  $u_j$ ,  $\chi_j^\tau = x_{u_j}^\tau$ .

**Lemma 2** 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 1.

**Proof:** By induction on the iteration number  $i$ . For  $i = 1$ ,  $t$  is added to  $\mathcal{C}$  in case it is selected and none of its neighbors are selected. It is removed from  $T$  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  $T$ . 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  $[0, 1]$  so we can use the additive Chernoff bound to estimate the probability for a fixed deviation of  $\widehat{C}^\tau$  from  $C^\tau$ . That is, for sample size  $s = 2/\delta^2$  and for

every sequence  $\tau$  of random bits,

$$\Pr_S \left[ \left| \widehat{C}^\tau - C^\tau \right| > \frac{\delta}{2} dn \right] < \Pr_S \left[ \left| \frac{1}{s} \sum_{j=1}^s \frac{\chi_j^\tau}{d} - \mu^\tau \right| > \delta \right] < 2 \cdot e^{-2s\delta^2} < 1/6.$$

From Theorem 1 we know that the probability over the possible sequences of random bits  $\tau$ , that  $C^\tau$  is less than  $3 \cdot TC_{OPT} + \frac{\delta m}{2}$  is at least  $5/6$ . Combining this and the fact that  $m \leq dn$ , we get that with probability at least  $2/3$ ,  $\widehat{C}$  is a  $(3, \delta)$ -estimate to the distance of  $G$  from being triangle free. Thus proving Theorem 2 ■

## 2.2 Generalizing the Result to Arbitrary Subgraphs

The result of the previous section can be generalized to arbitrary subgraphs using some subgraph specific parameters. Assume that  $H$  consists of  $m_H$  edges and its diameter is  $\phi_H$ . 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  $d$  subgraph, 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 1 and 2 can be modified to compute the distance of a bounded-degree graph from being  $H$ -free, for every fixed subgraph  $H$ .

**Theorem 3** *For every  $\delta > 0$ , using  $c = 2m_H$  and taking  $G_r(u)$  as the subgraph induced by the  $(\phi_H \cdot (2r+1))$ -neighborhood of  $u$  where  $r = 4m_H^2(\log(\frac{d_H}{\delta}) + 3)$ , Algorithm 2, when using a straightforward adapted version of Algorithm 1, outputs with probability at least  $2/3$  an  $(m_H, \delta)$ -estimate to the distance of  $G$  from being  $H$ -free. The query and time complexity of the algorithm is  $d^{O(\phi_H \cdot m_H^2 \cdot \log(d_H/\delta))}$ .*

**Proof Sketch:** First we analyze the adapted version of Algorithm 1. 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, 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 2 outputs an estimate to  $C$ , the size of the  $H$ -cover  $\mathcal{C}$ . It can be verified that for every fixed subgraph  $H$ , observing the  $2r$ -neighborhood of some isomorphic copy  $h$  of  $H$  is enough in order to determine its state at the end of the

$r$ 'th iteration. Also, for every vertex, its  $(\phi_H \cdot (2r + 1))$ -neighborhood contains the  $2r$ -neighborhood of every subgraph it belongs to. So by the same analysis, with probability at least  $2/3$ , Algorithm 2 outputs an  $(m_H, \delta)$ -estimate to  $HC_{OPT}$ .

The induced subgraph  $G_r(u)$  can be constructed using  $d^{O(\phi_H \cdot m_H^2 \cdot \log(d_H/\delta))}$  queries and so the total query complexity is as claimed. As for the running time, there are  $d^{O(\phi_H \cdot m_H^2 \cdot \log(d_H/\delta))}$  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  $\phi_H$  vicinity, i.e., using  $O(d^{\phi_H \cdot m_H})$  steps. So, going over all the subgraphs  $O(r)$  times takes  $d^{O(\phi_H \cdot m_H^2 \cdot \log(d_H/\delta))}$  steps.  $\square$

## 2.3 Relation to Distributed Approximation Algorithms

We observe that Algorithm 1 is readily seen to be a randomized distributed approximation algorithm. By applying some modifications, it can approximate the minimum subgraph-cover and the minimum VC of a graph, which is one of the fundamental problems extensively studied in various settings in Computer Science.

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  $k$ , such that, 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.

Consider the distributed setting that is used in [KMW06]. There, when the problem in question is vertex cover, there is a processor on every vertex as well as on every edge while when dealing with the  $H$ -cover problem, there is a processor on every edge and every copy of  $H$ . The connection channels connect edges with their vertices or subgraphs with their edges. In practice, the processes that correspond to the edges and subgraphs can be simulated by one of their vertices.

The distributed nature of Algorithm 1 in the above setting is obvious. An adapted version of it achieves, with probability at least  $2/3$ , an approximation ratio of  $(2 + \delta)$  to the minimum VC using  $O(\log(d/\delta))$  rounds. For the subgraph cover it gives an approximation ratio of  $(m_H + \delta)$  using  $O(m_H^2 \log(d_H/\delta))$  rounds. The distributed algorithm of Israeli and Itai [II86] finds maximal matching in  $O(\log n)$  expected number of rounds. The same time complexity holds for the distributed algorithm of Luby [Lub86] which finds maximal independent set but can be transformed to yield maximal matching. By taking the vertices of the matching one can 2-approximate the minimum VC in  $O(\log n)$  expected number of rounds. As mentioned in the introduction, in this context, our algorithm is similar to theirs. By applying small modifications to their algorithms, one can achieve the same result as our result for minimum VC.

A 2-approximation algorithm to the minimum VC was presented also by Panconesi and Rizzi [PR01]. Their algorithm uses  $O(d + \log^* n)$  rounds. Grandoni et. al. [GKPS05], gave

a more general algorithm which uses Linear Programming and gives  $(2 + \delta)$ -approximation to minimum VC using  $O(\delta^{-1} \log n)$  rounds.

An approximation ratio that is slightly above 2 with number of rounds which is independent of  $n$  can also be obtained by the recent result of Kuhn, Moscibroda and Wattenhofer [KMW06]. Their algorithm which is quite complex, uses Linear Programming and falls into a more general framework. Their result gives a tradeoff between quality and number of rounds but focusing on the best quality they can achieve, their result implies an approximation ratio of  $(2 + \delta)$  for VC using  $O(\delta^{-3} \cdot \log d)$  rounds and an approximation ratio of  $(m_H + \delta)$  for subgraph cover using  $O((m_H/\delta)^3 \cdot \log d_H)$  rounds.

For completeness we present the distributed version of Algorithm 1 for the minimum VC problem. A similar modification gives a distributed algorithm for the subgraph cover. 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 the minimum VC. The following is a randomized distributed algorithm that approximates the minimum VC of a graph.

**Algorithm 3 (Distributed approximation for minimum VC)**

1. Every edge activates itself.
2. From  $i = 1$  to  $r = \Theta(\log(d/\delta))$ 
  - (a) Every active edge  $e$  selects itself with probability  $\frac{1}{4d(e)}$ . 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 4** For every graph with degree at most  $d$  and every  $\delta > 0$ , Algorithm 3 constructs a vertex cover that, with probability at least  $2/3$ , approximates  $VC_{OPT}$  up to a multiplicative factor of  $2 + \delta$ . The number of rounds needed is  $O(\log(d/\delta))$ .

Also, by applying the straightforward modifications, it computes a triangle-cover ( $H$ -cover) that, with probability at least  $2/3$ , approximates  $TC_{OPT}$  ( $HC_{OPT}$ ) up to a multiplicative factor of  $3 + \delta$  ( $m_H + \delta$ ). The number of rounds needed is  $O(\log(d/\delta))$  ( $O(m_H^2 \log(d_H/\delta))$ ).

**Proof Sketch:** Let  $C$  be the size of the cover found by the algorithm. As shown in the proof of Theorem 1 (translating it to VC), with probability at least  $2/3$ , at the end of the loop, the number of inactive edges is at most  $\frac{\delta}{2d}|E|$ . Now, 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  $|E| \leq d \cdot VC_{OPT}$ . Therefore, combining the fact that the number of vertices selected during the loop is at most  $2 \cdot VC_{OPT}$  we have,

$$C \leq (2 + \delta)VC_{OPT}.$$

The claim regarding the  $H$ -cover problem is proved in the same way.  $\square$

## 2.4 Relation to Sublinear Approximation of Minimum VC

The problem of approximating the minimum VC has been studied also in the context of sublinear algorithms. Parnas and Ron [PR05] presented a  $(2 \log d + 1, \delta)$ -approximation algorithm whose query and time complexity is  $O(\delta^{-2} \cdot d^{\log d})$ . In addition, they show a reduction from local distributed approximation algorithms to sublinear algorithms. Using the distributed algorithm of Kuhn, Moscibroda and Wattenhofer [KMW06], discussed in the previous section, their reduction gives a  $(2, \delta)$ -approximation algorithm whose query complexity is  $d^{O(\delta^{-3} \cdot \log(d))}$ . Both of these algorithms can be modified to achieve a dependence in  $\Theta(\bar{d}/\delta)$  instead of  $d$ .

Algorithm 2, when modified to solve the minimum VC, achieves a  $(2, \delta)$ -approximation algorithm whose query and time complexity is  $d^{O(\log(d/\delta))}$ . It can also be viewed as a reduction from Algorithm 3. For completeness we state the modified algorithm.

**Algorithm 4 (Sublinear approximation for minimum VC)**

1. Uniformly and independently sample  $s = 2/\delta^2$  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 3.
3. Run Algorithm 3 on  $\bigcup_{v \in S} G_r(v)$  (in a sequential way). For every  $v \in S$ , let  $\chi_v = 1$  if the algorithm adds  $v$  to the cover, otherwise  $\chi_v = 0$ .
4. Output  $\hat{C} = \frac{n}{s} \sum_{v \in S} \chi_v$ .

**Theorem 5** For every  $\delta > 0$ , using  $c = 4$  and  $r = 16(\log(\frac{d}{\delta}) + 3)$  Algorithm 4 outputs with probability at least  $2/3$  a  $(2, \delta)$ -estimate to  $VC_{OPT}$ , that is,

$$VC_{OPT} - \delta n \leq \hat{C} \leq 2 \cdot VC_{OPT} + \delta n.$$

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

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

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

In this section we focus on the graph property of  $k$ -edge-connectivity. A graph is said to be  $k$ -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  $\tilde{O}(k^3 \cdot \varepsilon^{-3+2/k})$ . They improve the running time to  $\tilde{O}(1/\varepsilon)$  for  $k = 1, 2$  and to  $\tilde{O}(\varepsilon^{-2})$  for  $k = 3$ . Parnas and Ron [PR02] observed that the algorithm can be extended to the general model. In dense graphs, the problem of testing connectivity becomes trivial for  $\varepsilon \geq 1/n$  since, in this model, any graph on  $n$  vertices is  $1/n$ -close to being connected.

We present an algorithm that approximate, up to an additive factor of  $\delta m$ , the distance of a sparse connected graph from being  $k$ -connected for any  $k \geq 1$ . The query and time complexity of our algorithm is  $O\left(\left(\frac{k}{\delta d}\right)^6 \log\left(\frac{k}{\delta d}\right)\right)$ . For  $k = 1$ , this was solved by Chazelle, Rubinfeld and Trevisan [CRT05] as part of their algorithm for approximating the minimum spanning tree of a graph. They approximate the distance to 1-connectivity up to an additive factor of  $\delta n$  in  $O(\bar{d}\delta^{-2} \log(\bar{d}/\delta))$  time. For completeness, we describe here a similar algorithm.

#### 3.1 Connectivity

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 5 (Distance approximation to connectivity)***

1. Uniformly and independently sample  $s = \frac{16}{\delta^2 d^2}$  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  $\frac{4}{\delta d}$  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  $\frac{1}{m}(\hat{C} - 1)$ .

**Theorem 6** For every  $\delta > 0$  and every graph  $G$  on  $n$  vertices, with probability at least  $2/3$ , Algorithm 5 outputs a  $\delta$ -estimate to the distance of  $G$  from being 1-connected. The query and time complexity of the algorithm is  $O(\delta^{-4}\bar{d}^{-4})$ .

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

## 3.2 $k$ -Connectivity for $k \geq 2$

The problem of approximating the distance of a graph from being  $k$ -connected is more complicated, but as we show here, still solvable in time which is independent of the size of the input graph. 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. Furthermore, our analysis is somewhat simpler and “cleaner”. Specifically, we build on the *extreme-sets tree* and the *extreme-sets partition*. The latter was 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 3.2.3. We start by some definitions and structures and then describe the algorithm.

### 3.2.1 Preliminaries

Let  $G = (V, E)$  be a connected undirected graph. A *cut* in the graph is a minimal set of edges whose removal from the graph disconnects it into two sets of vertices  $A$  and  $\bar{A}$ . It corresponds to the set of edges between  $A$  and  $\bar{A}$  and is denoted 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  $(A, \bar{A})$ . A pair of edges  $\{v, u\}$  is called  *$j$ -edge-connected* if there are  $j$  edge-disjoint paths between them, or equivalently, if any cut that separates them is of size at least  $j$ . The  $j$ -edge connected relation defines a partition of the vertices into equivalence classes called  *$j$ -classes*. Thus a  *$j$ -class* is a maximal set of vertices that

cannot be disconnected by the removal of less than  $j$  edges. A graph  $G$  is  $j$ -edge-connected if the set  $V$  of all the vertices in  $G$  form a  $j$ -class. The edge-connectivity  $\lambda$  of  $G$  is the maximal  $j$  such that  $G$  is  $j$ -edge-connected. The size of the minimum cut in  $G$  is  $\lambda$ .

It is easy to see that the partition of  $V$  into its  $(j + 1)$ -classes is a refinement of the partition of  $V$  into its  $j$ -classes. Thus the connectivity classes have a hierarchical structure that can be represented by a *class decomposition tree*. The root of the tree corresponds to  $V$  which is of connectivity  $\lambda$ . Any intermediate node corresponds to some  $j$ -class  $U \subset V$  for some  $j > \lambda$  and its children correspond to the partition of  $U$  into its  $(j + 1)$ -classes. The leaves in the tree correspond to the vertices of the graph. See Figure 1 for an example of a graph and its class decomposition tree. We now turn to describe the extreme-sets structures of a graph.

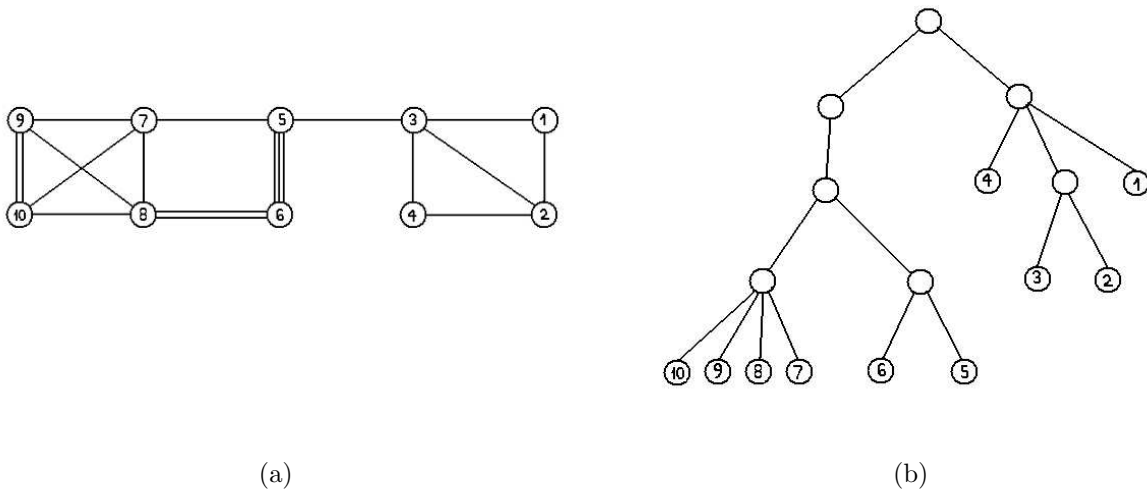


Figure 1: (a) A graph  $G_1$  and (b) its class decomposition tree. A node of depth  $j$  represents a  $j$ -class.

**Definition 2** 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  $W \subset U$ ,  $d(W) > \ell$ .

Observe that 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  $j \geq \ell$  then either  $U$  and  $W$  are disjoint or  $W \subseteq U$  (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.

Before we describe the extreme-sets partition we need to observe the following fact. Given any partition  $\mathcal{P} = \{P_1, \dots, P_t\}$  of the vertices of  $G$ , the distance of  $G$  from  $k$ -connectivity 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

$$\phi(\mathcal{P}) = \sum_{i=1}^t \max\{0, k - d(P_i)\}$$

This is true since for every  $i$ , if  $k - d(P_i) > 0$  then at least  $k - d(P_i)$  ends of edges must be attached to vertices of  $P_i$  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. Considering the lower bound, this is  $m$  times the distance of  $G$  from  $k$ -connectivity, the value which we try to estimate. In what follows we describe this partition. For an extreme-set  $U$  in the extreme-set tree they define the *demand* of  $U$  by

$$\phi(U) = \max \left\{ 0, k - d(U), \sum_{w \text{ child of } U} \phi(W) \right\}.$$

$\phi(U)$  is a lower bound on the number of ends 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

$$\phi(V) = \sum_{U \text{ child of } V} \phi(U).$$

Using these notions, the *extreme-sets partition* is defined as follows.

**Definition 3** *The extreme-sets partition of a graph  $G$  is the partition  $ES = \{X_1, \dots, X_p\}$  such that*

1. *For every  $i$ ,  $X_i$  is an extreme set with the property that either  $\phi(X_i) = 0$  or  $\phi(X_i) > \sum_{Y \text{ child of } X_i} \phi(Y)$*
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 found by recursively finding the partition of every child of  $V$ . Whenever an extreme-set that satisfies condition 1 is found, it is added to the partition  $ES$  and its subtree is ignored. 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  $\phi(ES) = \sum_{i=1}^p \phi(X_i)$ . In addition, this is exactly the demand of the root  $\phi(V) = \sum_{U \text{ child of } V} \phi(U)$  since for every  $i \in [p]$ , 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 2.

For every vertex  $v$ , we denote by  $X_v$  the set in  $ES$  that contains  $v$ . Whenever it is not clear which graph is in question we denote the extreme-set partition by  $ES(G)$  and the set containing  $v$  by  $X_v(G)$ .

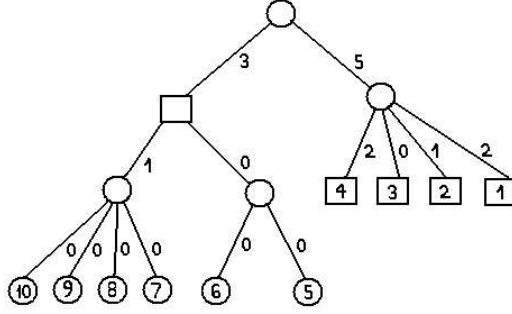


Figure 2: The extreme-sets tree of the graph  $G_1$  of Fig 1(a). Each node represents an extreme-set. The values of the edges are the demands of the corresponding extreme-sets for  $k = 4$ . The squared nodes represent the sets of the extreme-sets partition.

### 3.2.2 The algorithm

When approximating the distance to 1-connectivity we estimated for every vertex 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  $X_v$ .

The following procedure searches for  $X_v$  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  $u$ .

**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  $(t^2 + 3t)/2$ , assign a random cost in the range  $[0, 1]$  to the edges of the cut  $(S_i, \overline{S}_i)$  which 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  $X_v(G)$ .

To analyze Procedure 1 we define a variant of the extreme-sets partition:

**Definition 4** Given a graph  $G$  and a size bound  $t$ , the  $t$ -bounded extreme-sets partition is the partition  $ES^{(t)} = \{X_1^{(t)}, \dots, X_\ell^{(t)}\}$  satisfying the conditions of the extreme-sets partition with the additional constraint that for every  $i$ , the size of  $X_i^{(t)}$  is bounded by  $t$ . That is,

1. For every  $i$ ,  $X_i^{(t)}$  is an extreme-set with the property that either  $\phi(X_i^{(t)}) = 0$  or  $\phi(X_i^{(t)}) > \sum_{Y \text{ child of } X_i^{(t)}} \phi(Y)$
2. For every  $i$ , the size of  $X_i^{(t)}$  is at most  $t$ .
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 see 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 condition 1 of Definition 4. 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, 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 3.

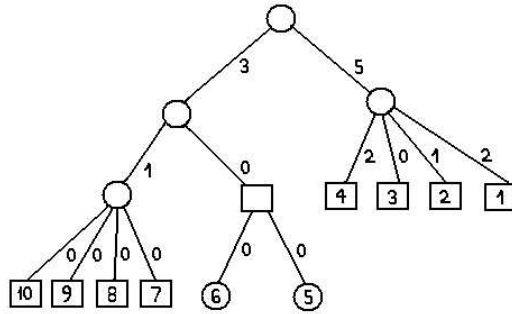


Figure 3: The extreme-sets tree of the graph  $G_1$  of Fig 1(a). Each node represents an extreme-set. The values of the edges are the demands of the corresponding extreme-sets for  $k = 4$ . The squared nodes represent the sets of the 3-bounded extreme-sets partition.

As before, for every  $v$ , we denote the set in  $ES^{(t)}$  which contains  $v$  by  $X_v^{(t)}$  and whenever it is not clear which graph is in question we denote the  $t$ -bounded extreme-sets partition by  $ES^{(t)}(G)$  and the set containing  $v$  by  $X_v^{(t)}(G)$ . In the following theorem we show that Procedure 1 actually finds  $X_v^{(t)}(G)$ .

**Theorem 7** For every  $v$  and size bound  $t$ , Procedure 1 finds  $X_v^{(t)}(G)$  with probability at least  $1 - e^{-2t^{-2} \cdot r}$ . Its query and time complexity is  $O(t^4 r)$ .

**Proof:** The construction of the extreme-sets tree of the graph  $G_S$  at step 1.b, can be done by first constructing the class decomposition tree using  $n-1$  max-flow computations [GH61,

Gus90]. This takes  $O(n'm^{3/2})$  time where  $n' = t + 1$  is the maximum number of vertices in  $G_S$  and  $m' = \binom{t+1}{2}$  is its maximum number of edges. Then, the classes which are not extreme-sets are removed from the tree to get the extreme-sets tree [NGM97]. This is done using  $O(n'm')$  steps, see [NGM97] for more details. Thus the total running time for the construction is  $O(t^4)$ . Step 1.a takes  $O(t^3)$  queries and time thus the total query and time complexity of Procedure 1 is therefore  $O(t^4r)$ .

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 lemma we show that in this case, the procedure declares  $X_v^{(t)}(G)$  as the required set.

**Lemma 3** *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 any iteration and let  $S$  be the set found in step 1.a. Step 1.b consists of deterministic sub-procedures and therefore always finds  $X_v^{(t)}(G_S)$ . Note that by the transition from the original graph  $G$  to the graph  $G_S$  we do not necessarily preserve the connectivity of vertices in  $S$ . The connectivity cannot decrease but it might increase so that the collection of all connected components that are contained in  $S$  might not be the same in  $G$  and in  $G_S$ . However, we can easily show that the collection of all extreme-sets that are contained in  $S$  is exactly the same in  $G$  and in  $G_S$ . This follows immediately by observing that for every set  $U \subseteq S$ , the degree of  $U$ ,  $d(U)$ , is exactly the same in  $G$  and in  $G_S$ . 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 which is contained in  $S$  is exactly the same in  $G$  and in  $G_S$ . Therefore, since step 1.b always finds  $X_v^{(t)}(G_S)$ , if on the  $i$ 'th iteration step 1.a finds a set  $S_i$  that contains  $X_v^{(t)}(G)$ , then  $X_v^{S_i} = X_v^{(t)}(G)$ . That is, at the same 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 analyzed is the probability that the random search process of step 1.a finds a set  $S$  containing  $X_v^{(t)}$ . Instead, we will 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  $(t^2 + 3t)/2$ . 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 see 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  $t$ ) and let  $d_v^{out} = d_v - d_v^{in}$ . Let  $u \in X_v^{(t)}$ , then since  $X_v^{(t)}$  is an extreme-set,  $\sum_{v \in X_v^{(t)}} d_v^{out} < d_u < d_u^{out} + t$ . In other words,  $\sum_{v \in X_v^{(t)} \setminus \{u\}} d_v^{out} < t$ .

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  $2t$  and thus for every  $S \subseteq X_v^{(t)}$ , the size of the cut  $(S, \overline{S})$  is less than  $\binom{t}{2} + 2t = (t^2 + 3t)/2$ . 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  $\overline{X_v^{(t)}}$  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]. We sketch their proof for completeness.

**Lemma 4** ([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  $\bar{l}$ . 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 which 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. 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 which is cheaper than the cut  $(L, \overline{L})$ . This completes the proof.  $\square$

**Corollary 2** *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 Lemma 3, with probability at least  $1 - e^{-2t^{-2}}$ , Procedure 1 finds  $X_v^{(t)}$ , thus proving Theorem 7.  $\blacksquare$

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

**Algorithm 6 (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  $r = t^2 \log(\frac{14k^2}{\delta^2\bar{d}^2})$ . 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  $\hat{\phi} = \frac{n}{s} \sum_{i=1}^s \frac{\hat{\phi}_j}{\hat{n}_j}$ , let  $\hat{C} = \left\lceil \frac{\hat{\phi}}{2} \right\rceil$  and output  $\frac{1}{m} \hat{C}$ .

**Theorem 8** For every  $\delta > 0$  and every graph  $G$  on  $n$  vertices, with probability at least  $2/3$ , Algorithm 6 outputs a  $\delta$ -estimate to the distance of  $G$  from being  $k$ -connected. The query and time complexity of the algorithm is  $O\left(\left(\frac{k}{\delta\bar{d}}\right)^6 \log\left(\frac{k}{\delta\bar{d}}\right)\right)$ .

**Proof:** Since the query and time complexity of Procedure 1 is  $O(t^4 \cdot r)$ , and since the result for step 3 can be extracted from Procedure 1, the query and time complexity of Algorithm 6 is as claimed.

Let  $ES = \{X_1, \dots, X_p\}$  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

$$\begin{aligned}
 \phi(ES) - \phi(ES^{(t)}) &= \sum_{i=j+1}^p \phi(X_i) - \sum_{i=j+1}^q \phi(X_i^{(t)}) \\
 &\leq \sum_{i=j+1}^p \phi(X_i) \\
 &\leq \frac{\delta\bar{d}n}{4k} \cdot k \\
 &= \frac{\delta}{2}m.
 \end{aligned}$$

where the last inequality is true since there are at most  $\frac{n}{t} = \frac{n}{4k/\delta\bar{d}}$  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 Theorem 7, 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

$$\sum_{i=1}^s e^{-2t^{-2} \cdot r} = s \cdot e^{-2 \log\left(\frac{14k^2}{\delta^2\bar{d}^2}\right)}$$

$$\begin{aligned}
&< s \cdot \left( \frac{\delta^2 \bar{d}^2}{14k^2} \right)^2 \\
&< \frac{1}{6}.
\end{aligned}$$

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,

$$\begin{aligned}
\phi(ES^{(t)}) &= \sum_{i=1}^q \phi(X_i^{(t)}) \\
&= \sum_{v \in V} \frac{\phi(v)}{n_v} \\
&= k \cdot n \cdot \mu
\end{aligned}$$

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  $5/6$ ,

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

That is, with probability at least  $2/3$ ,  $|\hat{\phi} - \phi(ES^{(t)})| < \frac{\delta}{2} m$ . Theorem 8 follows by applying the triangle inequality. ■

### 3.2.3 $k$ -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 increase 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 vertex 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 assures 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 resulted 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  $X_v(G)$ .

## 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. In dense graphs,

approximating the distance to this property is trivial since any graph is  $\delta$ -close to being Eulerian for every  $\delta \geq \frac{1}{n}$ . In bounded-degree graphs, Goldreich and Ron [GR02] gave a testing algorithm for this property, which can be easily adapted to general graphs, whose query and time complexity is  $\tilde{O}(\varepsilon^{-1})$ . Here we give a  $\delta$ -distance approximation algorithm for general graphs.

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

1. Run the following variant of Algorithm 5: 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 estimation 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  $j \in [s]$ , if  $d_j$  is odd then let  $\chi_j = 1$  otherwise let  $\chi_j = 0$ .
5. Let  $\hat{R} = \frac{n}{2s} \sum_{j=1}^s \chi_j - 1 + \hat{C}_e$  and output  $\hat{R}/(\bar{d}n)$

**Theorem 9** For every  $\delta > 0$  and every graph  $G$  on  $n$  vertices and average degree  $\bar{d}$ , with probability at least  $2/3$ , Algorithm 7 outputs a  $\delta$ -estimate to the distance of  $G$  from being Eulerian. The query and time complexity of the algorithm is  $O(\delta^{-4} \bar{d}^{-4})$ .

**Proof:** One way to optimally make  $G$  Eulerian using edge modifications is the following. First,  $G$  should be connected so that one end of an edge is attached to every connected component. This edge should be attached to an odd-degree vertex, if exists in the component. Note that one edge must be attached to every such vertex so this selection is optimal. Then, all (except for two) of the odd-degree vertices in the graph (whose number is even) are matched into pairs and one edge is attached to every pair of vertices. This way, the number of ends of edges attached to vertices of some connected component exactly equals the number of its odd-degree vertices if there are such, or 2 if all of its vertices are of even degree. For every vertex  $v \in V$ , let  $x_v = 1$  if the degree of  $v$ ,  $d_v$ , is odd, otherwise  $x_v = 0$ . Also, let  $C_e$  be the number of connected components in which all vertices are of even degree. Then, the minimum number of edges that must be added to  $G$  in order to make it Eulerian is  $R = \frac{1}{2} (\sum_{v \in V} x_v - 2) + C_e$ .

Algorithm 5 with the mentioned modifications, estimates the number of small connected components in which all the vertices are of even degree. As proved in Theorem 6, 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  $5/6$ ,  $|\frac{n}{s} \sum_{j=1}^s \chi_j - \sum_{v \in V} x_v| < \frac{\delta}{2} m$ . By applying the triangle inequality, the claimed approximation is achieved.

The query and time complexity of Algorithm 5 (step 1 of Algorithm 7) is  $O(\delta^{-4}\bar{d}^{-4})$ . The rest of Algorithm 7 takes  $O(\delta^{-2}\bar{d}^{-2})$  queries and time. Thus the total query and time complexity is  $O(\delta^{-4}\bar{d}^{-4})$ . ■

## 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 which is independent of  $n$ . In the general model, testing this property requires at least  $\Omega(\sqrt{n})$  queries. To see 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  $\delta$ -distance approximation algorithm for this property for bounded-degree graphs.

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  $s = \frac{32}{\delta^2}$  vertices from  $G$ . Let  $S = \{u_1, \dots, u_s\}$  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  $j \in [s]$ , 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  $\hat{X}_s = \frac{n}{s} \sum_{j=1}^s \frac{\hat{x}_j}{\hat{n}_j}$ ,  $\hat{m}_\ell = \frac{n}{2s} \sum_{j=1}^s d_j \cdot y_j$  and  $\hat{n}_\ell = \frac{n}{s} \sum_{j=1}^s y_j$ .  
Let  $\hat{X} = \hat{X}_s + \hat{m}_\ell - \hat{n}_\ell$  and output  $\frac{1}{dn} \hat{X}$ .

**Theorem 10** 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  $\delta$ -estimate to the distance of  $G$  from being cycle free. The query and time complexity of the algorithm is  $O(\delta^{-3}d^{-2})$ .

**Proof:** We say that a connected component is *small* if its size is bounded by  $b$ , otherwise it is a *large* component. Let  $C_s$  and  $C_\ell$  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.

Now, for every  $j \in [s]$ , the random variable  $\chi_j = \frac{\hat{x}_j}{\hat{n}_j d}$  gets values in the range  $[0, 1]$  and its expected value is  $\mu = \frac{1}{n} \sum_{v \in C_s} \frac{x_v}{d \cdot n_v} = \frac{1}{dn} X_s$ . 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  $j \in [s]$ ,  $\text{Exp}[y_j] = \frac{1}{n} n_\ell$  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  $O(\delta^{-1} d^{-2})$  and so the total query and time complexity is  $O(\delta^{-3} d^{-2})$ . ■

## References

- [ACCL04] N. Ailon, B. Chazelle, S. Comandur, and D. Liue. Estimating the distance to a monotone function. In *Proceedings of the Eight International Workshop on Randomization and Computation (RANDOM)*, pages 229–236, 2004. To appear in *Random Structures and Algorithms*.
- [AFKS00] N. Alon, E. Fischer, M. Krivelevich, and M. Szegedy. Efficient testing of large graphs. *Combinatorica*, 20:451–476, 2000.
- [AK02] N. Alon and M. Krivelevich. Testing  $k$ -colorability. *SIAM Journal on Discrete Math*, 15(2):211–227, 2002.
- [AKKR06] N. Alon, T. Kaufman, M. Krivilevich, and D. Ron. Testing triangle-freeness in general graphs. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 279–288, 2006.
- [Alo02] N. Alon. Testing subgraphs in large graphs. *Random Structures and Algorithms*, 21(3–4):359–370, 2002.
- [AS04] N. Alon and A. Shapira. A characterization of easily testable induced subgraphs. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2004.
- [AS05a] N. Alon and A. Shapira. A characterization of the (natural) graph properties testable with one-sided error. In *Proceedings of the Forty-Sixth Annual Symposium on Foundations of Computer Science (FOCS)*, 2005.
- [AS05b] N. Alon and A. Shapira. Every monotone graph property is testable. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on the Theory of Computing (STOC)*, pages 128–137, 2005.
- [BOT02] A. Bogdanov, Kenji Obata, and L. Trevisan. A lower bound for testing 3-colorability in bounded-degree graphs. In *Proceedings of the Forty-Third Annual Symposium on Foundations of Computer Science (FOCS)*, pages 93–102, 2002.

- [CRT05] B. Chazelle, R. Rubinfeld, and L. Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SIAM Journal on Computing*, 34(6):1370–1379, 2005.
- [Fei04] U. Feige. On sums of independent random variables with unbounded variance and estimating the average degree in a graph. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on the Theory of Computing (STOC)*, pages 594–603, 2004. To appear in SICOMP.
- [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.
- [Fis01] E. Fischer. The art of uninformed decisions: A primer to property testing. *Bulletin of the European Association for Theoretical Computer Science*, 75:97–126, 2001.
- [FN05] E. Fischer and I. Newman. Testing versus estimation of graph properties. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on the Theory of Computing (STOC)*, pages 138–146, 2005.
- [GGR98] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4):653–750, 1998.
- [GH61] R. E. Gomory and T. C. Hu. Multi-terminal network flows. *SIAM Journal on Applied Math*, 9(4):551–560, 1961.
- [GKPS05] F. Grandoni, J. Konemann, A. Panconesi, and M. Sozio. Primal-dual based distributed algorithms for vertex cover with semi-hard capacities. In *Proceedings of the 24th ACM Symposium on Principles of Distributed Computing*, pages 118–125, 2005.
- [Gol98] O. Goldreich. Combinatorial property testing - a survey. In *Randomization Methods in Algorithm Design*, pages 45–60, 1998.
- [GR99] O. Goldreich and D. Ron. A sublinear bipartite tester for bounded degree graphs. *Combinatorica*, 19(3):335–373, 1999.
- [GR00] O. Goldreich and D. Ron. On testing expansion in bounded-degree graphs. ECCC Report TR00-020, 2000.
- [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 Ninth International Workshop on Randomization and Computation (RANDOM)*, pages 306–317, 2005.

- [GR06] O. Goldreich and D. Ron. Approximating average parameters of graphs. In *Proceedings of the Tenth International Workshop on Randomization and Computation (RANDOM)*, pages 363–374, 2006.
- [GT03] O. Goldreich and L. Trevisan. Three theorems regarding testing graph properties. *Random Structures and Algorithms*, 23(1):23–57, 2003.
- [Gus90] D. Gusfield. Very simple methods for all pairs network flow analysis. *SIAM Journal on Computing*, 19(1):143–155, 1990.
- [II86] A. Israeli and A. Itai. A fast and simple randomized parallel algorithm for maximal matching. *Information Processing Letters*, 22:77–80, 1986.
- [Kar93] D. Karger. Global min-cuts in RNC and other ramifications of a simple mincut algorithm. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 21–30, 1993.
- [KKR04] T. Kaufman, M. Krivelevich, and D. Ron. Tight bounds for testing bipartiteness in general graphs. *SIAM Journal on Computing*, 33(6):1441–1483, 2004.
- [KMW06] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. The price of being near-sighted. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 980–989, 2006.
- [Lub86] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15(2):1036–1055, 1986.
- [NGM97] D. Naor, D. Gusfield, and C. Martel. A fast algorithm for optimally increasing the edge connectivity. *SIAM Journal on Computing*, 26(4):1139–1165, 1997.
- [PR01] A. Panconesi and R. Rizzi. Some simple distributed algorithms for sparse networks. *Distributed Computing*, 14(2):97–100, 2001.
- [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.
- [PRR04] M. Parnas, D. Ron, and R. Rubinfeld. Tolerant property testing and distance approximation. ECCC Report TR04-010, to appear in JCSS, 2004.
- [Ron01] D. Ron. Property testing. In *Handbook on Randomization, Volume II*, pages 597–649, 2001.
- [RS96] R. Rubinfeld and M. Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, 1996.