Property Testing in Bounded Degree Graphs*

Oded Goldreich
Dept. of Computer Science
and Applied Mathematics
Weizmann Institute of Science
Rehovot, Israel
oded@wisdom.weizmann.ac.il

Dana Ron†
Dept. of EE – Systems
Tel Aviv University
Ramat Aviv, Israel
danar@eng.tau.ac.il

July 2000, with two later comments (May 2008)

Abstract

We further develop the study of testing graph properties as initiated by Goldreich, Goldwasser and Ron. Loosely speaking, given an oracle access to a graph, we wish to distinguish the case the graph has a pre-determined property from the case it is "far" from having this property. Whereas they view graphs as represented by their adjacency matrix and measure distance between graphs as a fraction of all possible vertex pairs, we view graphs as represented by bounded-length incidence lists and measure distance between graphs as a fraction of the maximum possible number of edges. Thus, while the previous model is most appropriate for the study of dense graphs, our model is most appropriate for the study of bounded-degree graphs.

In particular, we present randomized algorithms for testing whether an unknown bounded-degree graph is connected, k-connected (for $k > 1$), cycle-free and Eulerian. Our algorithms work in time polynomial in $1/\epsilon$, always[1] accept the graph when it has the tested property, and reject with high probability if the graph is ϵ-far from having the property. For example, the 2-Connectivity algorithm rejects (w.h.p.) any N-vertex d-degree graph for which more than ϵdN edges need to be added in order to make the graph 2-edge-connected.

In addition we prove lower bounds of $\Omega(\sqrt{N})$ on the query complexity of testing algorithms for the Bipartite and Expander properties.

Keywords: Approximation Algorithms, Randomized Algorithms, Graph Algorithms, Property Testing.

*Work done while visiting LCS, MIT and while visiting ICSI and the CS Dept. at Berkeley. Preliminary version has appeared in the *29th STOC*; [GR97].

†This work was supported by an NSF postdoctoral fellowship.

[1]Errata (2008): With the exception of the cycle-free tester, which may reject with small probability graphs that are cycle-free.

1 Introduction

Approximation is one of the basic paradigms of modern science. One of its facets in computer science is approximation algorithms. Yet, it is not always clear what approximation means. The dominant approach considers a cost function associated with possible solutions of an instance, and seeks algorithms that provide an *approximation of the cost of an optimal solution* (possibly, as well as a solution obtaining such a cost). This approach is most suitable in case there is a natural cost measure for candidate solutions and the optimal solution is preferable due to its low(est) cost. An alternative approach is to consider the *distance* of the given instance *to the closest instance that has a desirable property.* The property may be having a solution of certain cost (w.r.t some cost measure defined as in the first approach), but it can also be of a qualitative nature; for example, being a connected graph (in case the instances are graphs), or being a linear function (in case the instances are functions). The latter approach underlines all work on testing low-degree polynomials [BLR93, RS96, GLR$^+$91, BFL91, BFLS91, FGL$^+$96, ALM$^+$98] and codes [BFLS91, ALM$^+$98, BGS98, Hås96], and its relevance to the construction of probabilistically checkable proofs [BFL91, BFLS91, FGL$^+$96, AS98, ALM$^+$98] is well known. In [GGR98] this approach was applied to testing properties of graphs, and its relation to the more standard approach to approximation was demonstrated.

We stress that approximation is applicable not only when the optimization problems are intractable. Also in case there exists an efficient algorithm for solving the problem optimally, one may wish to have an even faster algorithm and be willing to tolerate its approximative nature. In particular, in a RAM model of computation, an approximation algorithm may even run in sublinear time and still provide valuable information. For example, the testing algorithms of [GGR98] run in constant time and provide "constant error approximations" (e.g., one can approximate the value of the maximum cut in a dense graph to within a constant factor in constant time).

1.1 Testing graph properties

The study of *testing graph properties* was initiated by Goldreich *et. al.* [GGR98], as part of a general study of property testing [RS96, GGR98]. In the general model the algorithm is given oracle access to a function and has to decide whether the function has some specified property or is "far" from having that property. Distance between functions is defined as the fraction of instances on which the functions' values differ. In their study of *testing graph properties*, Goldreich *et. al.* view the graph as a Boolean function defined over the set of all vertex-pairs. Thus, their measure of distance between graphs is the fraction of vertex-pairs that are an edge in one graph and a non-edge in the other graph, taken over the total number of vertex-pairs. This model is most appropriate for the study of dense graphs, and indeed the graph algorithms in [GGR98] refer mainly to dense graphs. For example, their (constant time) Monte Carlo algorithm for testing whether a graph is Bipartite or is 0.1-far from Bipartite is meaningful only for N-vertex graphs which have more than $0.1 \cdot \binom{N}{2}$ edges (since any graph having fewer edges is 0.1-close to being Bipartite). Furthermore, testing connectivity in this model is trivial as long as the distance parameter is bigger than $\frac{2}{N}$ (since every N-vertex graph is $\frac{2}{N}$-close to being connected and so the algorithm may as well accept any graph).

In this paper we present an alternative model. We view bounded-degree graphs as functions defined over pairs (v,i), where v is a vertex and i is a positive integer within a predetermined (degree) bound, denoted d. The range of the function is the vertex set augmented by a special symbol. Thus the value on argument (v,i) specifies the $i^{\rm th}$ neighbor of v (with the special symbol indicating non-existence of such a neighbor). Our measure of distance between (N-vertex) graphs is the fraction of vertex-pairs which are an edge in one graph and a non-edge in the other, taken over the size of the domain (i.e., over dN). Unless $d = \Theta(N)$, this model does not allow to consider dense graphs, yet it is most appropriate for the study of bounded-degree graphs. In particular, in contrast to the model studied in [GGR98], testing connectivity is no longer trivial in our model.[2] The two models differ not only in the type of properties that are non-trivial, but also in the applicable techniques and the results that can be obtained for specific properties. For example, we show that no (Monte Carlo) algorithm running in $o(\sqrt{N})$ time can test whether a bounded-degree graph is Bipartite or is 0.1-far from Bipartite, where distance is as defined in our model. This stands in contrast to the constant-time algorithm for testing bipartiteness in the [GGR98] model.

To demonstrate the viability of our model, we present randomized algorithms for testing several natural properties of bounded-degree graphs. All algorithms get as input a degree bound d and an approximation parameter ϵ. The algorithms make queries of the form (v, i) that are answered with the name of the i^{th} neighbor of v (or with a special symbol in case v has less than i neighbors). With probability at least $2/3$, each algorithm accepts any graph having the tested property and rejects any graph which is at distance greater than ϵ from any graph having the property. Actually, except for the cycle-freeness tester, all algorithms have one-sided error (i.e., always accept graphs which have the property), and furthermore when rejecting they present a short certificate vouching that the property does not hold in the tested graph. Assuming that vertex names are manipulated at constant time, all algorithms have $\text{poly}(d/\epsilon)$ running-time (i.e., independent of the size of the graph). Actually, most algorithms have $\text{poly}(1/\epsilon)$ running-time and some have $\tilde{O}(1/\epsilon)$ running-time, where $\tilde{O}(\ell) = \text{poly}(\log(\ell)) \cdot \ell$. In particular, we present testing algorithms for the following properties:

connectivity: Our algorithm runs in time $\tilde{O}(1/\epsilon)$. Recall that by the above this means that in case the graph is connected the algorithm always accepts, whereas in case the graph is ϵ-far from being connected the algorithm rejects with probability at least $\frac{2}{3}$. Furthermore, the algorithm supplies a small counter-example to connectivity (in the form of an induced subgraph which is disconnected from the rest of the graph).

***k*-edge-connectivity:** Our algorithms run in time $\tilde{O}(k^3 \cdot \epsilon^{-3+\frac{2}{k}})$. For $k = 2, 3$ we have improved algorithms whose running-times are $\tilde{O}(\epsilon^{-1})$ and $\tilde{O}(\epsilon^{-2})$, respectively. Our techniques extend to testing k-vertex-connectivity, for $k = 2, 3$, see [GR99a, Sec. 4].

Eulerian: Our algorithm runs in time $\tilde{O}(\epsilon^{-1})$.

cycle-freeness: Our algorithm runs in time $O(\epsilon^{-3})$. Unlike all other algorithms, this algorithm has two-sided error probability, which is shown to be unavoidable for testing this property (within $o(\sqrt{N})$ queries, where N is the size of the graph).

In addition, we establish $\Omega(\sqrt{N})$ lower bounds on the query complexity of testing algorithms for the **Bipartite** and **Expander** properties. The first lower bound stands in sharp contrast to a result on testing bipartiteness which is described in [GGR98]. Recall that in [GGR98] graphs are represented by their $N \times N$ adjacency matrices, and the distance between two graphs is defined to be the fraction of entries on which their respective adjacency matrices differ. The Bipartite tester of [GGR98] works in time $\text{poly}(1/\epsilon)$ and distinguishes Bipartite graphs from graphs in which

[2] Recall that in the former model, for every $\epsilon \geq 2/N$, every graph is ϵ-close to being connected, and that typically we focus on constant ϵ (or $\epsilon > N^{-1/O(1)}$). Thus typically, testing connectivity is trivial in that model. Indeed, in our model every graph is $(2/d)$-close to being connected, but this leaves a wide range of ϵ's (e.g., constant $\epsilon < 2/d$) for which the problem of testing connectivity is non-trivial.

at least $\frac{1}{2}\epsilon N^2$ edges must be omitted in order to be bipartite. Recall that in the current paper, graphs are represented by incidence lists of length d and distance is measured as the number of edge modifications divided by dN (rather than by N^2).

Finally, we observe that the known results on inapproximability of Minimum Vertex Cover (and Dominating Set) for bounded-degree graphs [ALM+98, PY91], rule out the possibility of efficient testing algorithms for these properties in our model.

1.2 What does this type of approximation mean?

To make the discussion less abstract, let us consider the k-(edge)-connectivity tester. As evident from above, this algorithm is very fast; its running-time is polynomial in the error parameter, which one may think of as being a constant. Yet, what does one gain by using it?

One possible answer is that since the tester is so fast, it may make sense to run it before running an algorithm for k-connectivity. In case the graph is very far from being k-connected, we will obtain (w.h.p.) a proof towards this fact and save the time we might have used running the exact algorithm. (In case our tester detects no trace of non-k-connectivity, we may next run our exact algorithm.) It seems that in some natural setting where *typical* objects are either good or very bad, we may gain a lot. Furthermore, *if* it is *guaranteed* that objects are either good (i.e., graphs are k-connected) or very bad (i.e., far from being k-connected) then we may not even need the exact algorithm at all. The gain in such a setting is enormous.

Alternatively, we may be forced to take a decision, without having time to run an exact algorithm, while given the option of modifying the graph in the future, at a cost proportional to the number of added/omitted edges. For example, suppose you are given a graph which represents some design problem, where k-connectivity corresponds to a good design and changes in the design correspond to edge additions/omissions. Using a k-connectivity tester you always accept a good design, and reject with high probability designs which will cost a lot to modify. You may still accept bad designs, but then you know that it will not cost you much to modify them later. In this respect we mention the existence of efficient algorithms for determining a minimum set of edges to be added to a graph in order to make it k-connected [WN87, NGM97, Gab91, Ben95, NI97, ?].

1.3 Testing connectivity to the rest of the graph

Our algorithm for testing k-edge-connectivity, for $k \geq 2$, uses a subroutine which may be of independent interest. To describe it, suppose that you are given as input a vertex that resides in a small "component" which is disconnected from the rest of the graph by a cut of at most k edges. Your task is to find such a component, within complexity which depends only on the size of the component. As above, you are allowed oracle queries of the form "what is the i^{th} neighbor of vertex v".

Our algorithm finds the component containing the input vertex, within time cubic in the size of the component (independent of k and of the size of the entire graph). It is based on the underlying idea of the min-cut algorithm of Karger [Kar93]. For $k = 2$, we have an alternative algorithm which works in time linear in the size of the component, and for $k = 3$, we present an algorithm which works in quadratic time. We suggest the improvement of the complexity of the above task, for $k \geq 3$, as an open problem.

1.4 Subsequent work

As mentioned above, we show that in our model, any algorithm for testing whether an N-vertex graph is bipartite requires $\Omega(\sqrt{N})$ queries (where d and ϵ are constants). In follow-up work [GR99b], a bipartiteness tester is presented whose query and time complexities are $\sqrt{N} \cdot \text{poly}(\log N/\epsilon)$.

In [PR99] an alternative model for testing graph properties is studied, where the graphs are represented by incidence lists of *varying lengths*. In that model a graph is said to be ϵ-far from having a property, if the number of edges that need to be added or removed divided by the *number of edges in the graph* is more than ϵ. This model is more appropriate than ours for testing (sparse) graphs in which some vertices have very high degree D (e.g., $D = \Omega(N)$), but the average degree is $o(D)$ (e.g., a constant). Treating such graphs in our model will require setting $d = D$, but this may not be so meaningful in case there is a huge gap between the maximum degree and the average degree in the graph. Some of our algorithms can be extended to the varying-length incident-list model; see [PR99].

Errarata

In a preliminary version of this work [GR97], we claimed to have an algorithm for testing planarity that runs in time $\tilde{O}(d^4 \cdot \epsilon^{-1})$. The specific algorithm we had in mind had a fundamental flaw, which was discovered by an anonymous referee, whom we thank.[3]

Organization

In Section 2 we present the definitions used throughout the paper. Section 3 presents our algorithms for testing k-edge-connectivity (for $k \geq 1$). Testing algorithms for cycle-free, subgraph-free and Eulerian graphs are presented in Sections 4, 5 and 6, respectively. Our hardness results are presented in Section 7.

2 Definitions and Notation

We consider undirected graphs of bounded degree. We allow multiple edges but no self-loops. For a graph G, we denote by $\mathrm{V(G)}$ its vertex set and by $\mathrm{E(G)}$ its edge set. We assume, without loss of generality, that $\mathrm{V(G)} = [|\mathrm{V(G)}|] \stackrel{\text{def}}{=} \{1, \ldots, |\mathrm{V(G)}|\}$ and that for every vertex $v \in \mathrm{V(G)}$, the edges incident to v have distinct labels in $\{1, \ldots, d\}$. This labeling may be arbitrary and need not be consistent among neighboring vertices. Namely, $(u, v) \in \mathrm{E(G)}$ may be the i^{th} edge incident to u and the j^{th} edge incident to v, where $i \neq j$. In accordance with the above, we associate with a (bounded degree) graph G, a function $f_{\mathrm{G}} : \mathrm{V(G)} \times [d] \mapsto \mathrm{V(G)} \cup \{0\}$, where d is a bound on the degree of G. That is, $f_{\mathrm{G}}(v, i) = u$ if (u, v) is the i^{th} edge incident to v, and $f_{\mathrm{G}}(v, i) = 0$ if there is no such edge.

We consider property testing algorithms which are allowed queries to the above representation of a graph. That is, when referring to a graph G, the algorithm receives as ordinary inputs $|\mathrm{V(G)}|$ and a degree bound d, and is given oracle access to the function f_{G}.

Our measure of the (relative) distance between graphs depends on their degree bound. That is, the distance between two graphs G_1 and G_2 with degree bound d, where $\mathrm{V(G_1)} = \mathrm{V(G_2)} = [N]$, is

[3] Addendum (2008): Furthermore, as pointed out by Benjamini *et. al.* (STOC 2008), no one-sided error algorithm have have query complexity $o(\sqrt{N})$.

defined as follows:

$$\mathrm{dist}_d(\mathrm{G}_1, \mathrm{G}_2) \stackrel{\mathrm{def}}{=} \frac{|\{(v,i) : v \in [N],\ i \in [d] \text{ and } f_{\mathrm{G}_1}(v,i) \neq f_{\mathrm{G}_2}(v,i)\}|}{d \cdot N} \tag{1}$$

Note that for every two graphs G_1 and G_2, we have $0 \leq \mathrm{dist}_d(\mathrm{G}_1, \mathrm{G}_2) \leq 1$. This notation of distance is extended naturally to a set, $\mathcal{C}$, of N-vertex graphs with degree bound d; that is, $\mathrm{dist}_d(\mathrm{G}, \mathcal{C}) \stackrel{\mathrm{def}}{=} \min_{\mathrm{G}' \in \mathcal{C}}\{\mathrm{dist}_d(\mathrm{G}, \mathrm{G}')\}$. For a graph property Π, we let $\Pi_{N,d}$ denote the class of graphs with N vertices and degree bound d which have property Π. In case $\Pi_{N,d}$ is empty (for some Π, N, and d), we define $\mathrm{dist}(\mathrm{G}, \Pi_{N,d})$ to be 1 for every G.

Definition 2.1 *Let $\mathcal{A}$ be an algorithm which receives as input a size parameter $N \in \mathcal{N}$, a degree parameter $d \in \mathcal{N}$, and a distance parameter $0 < \epsilon \leq 1$. Fixing an arbitrary graph* G *with N vertices and degree bound d, the algorithm is also given oracle access to f_{G}. We say that $\mathcal{A}$ is a* property testing algorithm (or simply a testing algorithm) *for graph-property* Π, *if for every N, d, and ϵ and for every graph* G *with N vertices and maximum degree d, the following holds:*

- *if* G *has property* Π *then with probability at least $\frac{2}{3}$, algorithm $\mathcal{A}$* accepts G*;*
- *if* $\mathrm{dist}_d(\mathrm{G}, \Pi_{N,d}) > \epsilon$ *then with probability at least $\frac{2}{3}$, algorithm $\mathcal{A}$* rejects G.

In both cases, the probability is taken over the coin flips of $\mathcal{A}$. The query complexity *of $\mathcal{A}$ is a function of N, d, and ϵ bounding the number of queries made by $\mathcal{A}$ on input (N, d, ϵ) and oracle access to any f_{G}.*

We shall be interested in bounding both the query complexity and the running time of $\mathcal{A}$ as a function of N, d, and ϵ. In particular we try and achieve bounds which are polynomial in d, and $1/\epsilon$, and sub-linear in N. Actually, our query complexity will be independent of N and so is the running-time in a RAM model in which vertex names can be written, read and compared in constant time.

In the above definition we deviate from some traditions of having also a confidence parameter, denoted δ, and requiring the testing algorithm to be correct with probability at least $1-\delta$. Adopting these traditions seems justifiable in case one can derive better results than by merely repeating the basic procedure for $O(\log(1/\delta))$ times. Alas, this is not the case in the present work.

3 Testing k-Edge-Connectivity

Let $k \geq 1$ be an integer. A graph is said to be k–edge-connected if there are k edge-disjoint paths between every pair of vertices in the graph. An equivalent definition is that the subgraph resulting by omitting any $k-1$ edges from the graph, is connected. A graph that is 1–edge-connected, is simply referred to as connected. In this section we show the following.

Theorem 3.1 *For every $k \geq 1$ there exists a testing algorithm for k-edge-connectivity whose query complexity and running time are* $\mathrm{poly}(\frac{k}{\epsilon})$. *Specifically,*

- *For $k = 1, 2$ these complexities are $O\left(\frac{\log^2(1/(\epsilon d))}{\epsilon}\right)$.*
- *For $k = 3$ these complexities are $O\left(\frac{\log(1/(\epsilon d))}{\epsilon^2 \cdot d}\right)$.*

- *For $k \geq 4$ these complexities are* $O\left(\frac{k^3 \cdot \log(1/(\epsilon d))}{\epsilon^{3-\frac{2}{k}} \cdot d^{2-\frac{2}{k}}}\right)$.

Furthermore, the algorithms never reject a k–edge-connected graph.

We note that the above complexity bounds do not increase with the degree bound d. The reason is that the distance between graphs is measured as a fraction of $d \cdot N$; thus, d effects the number of operations as well as the distance and its effect on the latter is typically more substantial.

We start by describing and analyzing the algorithm for $k = 1$, and later show how it can be generalized to larger k. From now on we assume that $d \geq k$, since otherwise we would immediately reject the tested graph G simply because a graph with degree less than k cannot be k connected. In the case of $k = 1$ we may actually assume that $d \geq 2$ (since otherwise, except for $N \leq 2$, the graph cannot be connected).

3.1 Testing Connectivity

Our algorithm is based on the following simple observation concerning the connected components (i.e., the maximal connected subgraphs) of a graph.

Lemma 3.2 *Let $d \geq 2$. If a graph* G *is ϵ-far from the class of N-vertex connected graphs with maximum degree d, then it has more than $\frac{\epsilon}{4}dN$ connected components.*

Proof: Assume contrary to the claim that G has at most $\frac{\epsilon}{4}dN$ connected components. We will show that by adding and removing less than $\frac{\epsilon}{2}dN$ edges we can transform G into a connected graph G$'$ which has maximum degree at most d. This contradicts the hypothesis by which G is ϵ-far from the class of connected graphs with degree d. (Recall that according to our distance measure (Eq. (1)) every edge in the symmetric difference between graphs is counted twice).

Let $\mathrm{C}_1, \ldots, \mathrm{C}_\ell$ be the connected components of G. The easy case is when the sum of degrees in each C_i is at most $d \cdot |\mathrm{C}_i| - 2$. In this case, for every $i = 1, ..., \ell$, either C_i contains at least two vertices of degree $d-1$ or it contains at least one vertex of degree at most $d-2$. For simplicity, assume that the latter sub-case holds and let v_i be a vertex of degree at most $d-2$ in C_i. Then, for every $i = 1, ..., \ell - 1$, we may add the edge (v_i, v_{i+1}) to the graph, resulting in a connected graph. Furthermore, the degree of each vertex in the resulting graph is at most d (as we only increased the degrees of the v_i's). The argument extends to the other sub-case. That is, if $v_i, u_i \in \mathrm{C}_i$ have both degree $d-1$ then we connect some vertex of C_{i-1} to v_i and some vertex of C_{i+1} to u_i. In both sub-cases, we made the graph connected (and maintained the degree bound) by adding $\ell - 1 \leq \frac{\epsilon}{4}dN - 1 < \frac{\epsilon}{2}dN$ edges.

The above analysis used the case hypothesis by which the sum of degrees in each C_i is at most $d \cdot |\mathrm{C}_i| - 2$. But in general, this condition may not hold, and we need to do slightly more in order to make the graph connected *while maintaining the degree bound.* In particular, we remove edges within components (without disconnecting these components), so that we can add edges between components without violating the degree bound.

Suppose that for some connected component, C_i, the sum of degrees is greater than $d \cdot |\mathrm{C}_i| - 2$ (and hence we cannot add edges between C_i and $\mathrm{C}_{i\pm1}$ without violating the degree bound). Clearly, $|\mathrm{C}_i| \geq 2$ (or else C_i is an isolated vertex having degree $0 \leq d-2$). Let T_i be an arbitrary spanning tree of C_i. Since T_i contains at least two vertices, it has at least two leaves. By our assumption regarding C_i, at most one of its vertices has degree less than d. Thus, the tree T_i has a leaf which has degree $d \geq 2$ in G, and so this leaf has an incident edge in C_i which is not an edge in T_i. We can remove this edge from G without disconnecting C_i and get two vertices in C_i which have degree less than d. It follows that by removing at most one edge from each component and adding an edge between every C_i and C_{i+1}, we obtain a connected graph G' respecting the degree bound d. Since the symmetric difference between E(G) and E(G′) is bounded above by $2\ell - 1 < \frac{\epsilon d N}{2}$, we reached a contradiction and the claim follows. ■

As an immediate corollary we get:

Corollary 3.3 *If a graph* G *is* ϵ*-far from the class of* N*-vertex connected graphs of degree bound* $d \geq 2$*, then* G *has at least* $\frac{\epsilon d N}{8}$ *connected components each containing less than* $\frac{8}{\epsilon d}$ *vertices.*

Proof: By Lemma 3.2, G has at least $\frac{\epsilon d N}{4}$ connected components. The number of connected components containing at least $8/\epsilon d$ vertices is at most $\frac{N}{8/\epsilon d} = \frac{\epsilon d N}{8}$. So the remaining ones are at least $\frac{\epsilon d N}{4} - \frac{\epsilon d N}{8}$ in number, and each contains less than $8/\epsilon d$ vertices. ■

An implicit implication of Lemma 3.2 is that for $\epsilon \geq \frac{4}{d}$, *every* graph is ϵ-close to the class of connected graphs with degree bound d (as otherwise the lemma would imply the existence of an N-vertex graph with more than N connected components). Thus we may assume that $\epsilon < \frac{4}{d}$. By using the fact that each connected component contains at least one vertex we conclude that if G is ϵ-far from the class of connected graphs then the probability that a uniformly selected vertex belongs to a connected component which contains less than $\frac{8}{\epsilon d}$ vertices, is at least $\frac{\epsilon d N/8}{N} = \frac{\epsilon d}{8}$. Therefore, if we uniformly select $m = \frac{16}{\epsilon d}$ vertices, then the probability that no selected vertex belongs to a component of size less than $\frac{8}{\epsilon d}$ is bounded above by

$$\left(1 - \frac{\epsilon d}{8}\right)^m < e^{-\frac{\epsilon d}{8} \cdot m} = e^{-2} < \frac{1}{3}$$

(since $\frac{\epsilon d}{8} < 1$). On the other hand, once we select such a vertex, we may detect that it belongs to a small connected component at relatively low complexity (related to the size of the small connected component). This gives rise to the following testing algorithm, where we assume that $N \geq \frac{8}{\epsilon d}$ (since otherwise a connected graph having less than $\frac{8}{\epsilon d}$ vertices would be rejected in Step (2)). If $N < \frac{8}{\epsilon d}$, we can determine if the graph is connected by simply inspecting the whole graph (which takes time $O(Nd) = O(\epsilon^{-1})$).

Algorithm 3.4 (Connectivity Testing Algorithm):

1. *Uniformly and independently select* $m = \frac{16}{\epsilon d}$ *vertices in the graph;*[4]
2. *For each vertex* s *selected perform a Breadth First Search (BFS) starting from* s *until* $\frac{8}{\epsilon d}$ *vertices have been reached or no more new vertices can be reached (a small connected component has been found);*
3. *If any of the above searches finds a small connected component then output* REJECT*, otherwise output* ACCEPT.

[4]For sake of the analysis, in this and all other algorithms the vertices are selected independently, and so they are not necessarily distinct. However, if the number of graph vertices N is significantly larger than the sample size m, then with high probability they will in fact be distinct.

We note that the BFS is implemented in the obvious manner – by making queries of the form (v, i) *to* f_G.

Since a connected graph consists of a single component, the algorithm never rejects a connected graph. By the discussion preceding the algorithm and Corollary 3.3, if a graph is ϵ-far from connected then it is rejected with probability at least 2/3. The query complexity and running time of the algorithm are $m \cdot \frac{8}{\epsilon d} \cdot d = O\left(\frac{1}{\epsilon^2 d}\right)$. We note that the choice to perform a BFS is quite arbitrary, and that any other linear-time searching method (e.g., DFS) will do.

The complexity of the Connectivity Tester can be improved by applying Corollary 3.3 more carefully. Above, when analyzing the probability that the algorithm selects a vertex in a small component, we considered the extreme case in which the component consists of a single vertex. On the other hand, when analyzing the complexity of scanning the component, we considered the extreme case in which the component consists of $\Theta(1/\epsilon d)$ vertices. Instead, suppose that all components in the conclusion of Corollary 3.3 were of the same size, denoted s. Then the probability that a vertex in such a component is selected is at least $s \cdot \frac{\epsilon d N/8}{N} = \frac{s\epsilon d}{8}$, which means that it suffices to set $m = O(1/(s\epsilon d))$ in Step (1) of the algorithm above, and that in Step (2) it suffices to look for $s+1$ vertices. Thus, the overall complexity would be $O(1/\epsilon)$, provided that such s exists and is given to the algorithm. Since the latter assumption does not hold, we use a relaxed generalization of the above idea: That is, suppose that G has at least $L \stackrel{\text{def}}{=} \frac{\epsilon d N}{8}$ connected components each of size at most $\frac{8}{\epsilon d} - 1$. Then, (as we show in Lemma 3.6), there exists an $i \le \ell \stackrel{\text{def}}{=} \log(8/\epsilon d)$ (where throughout the paper $\log(\cdot) = \log_2(\cdot)$), so that G has at least $\frac{L}{\ell}$ connected components of size ranging between 2^{i-1} and $2^i - 1$. We do not know this i, but we may try them all. This suggests the following improved algorithm, where here we assume that $N > \frac{16 \cdot \log(8/(\epsilon d))}{\epsilon \cdot d}$ (and for smaller N we simply inspect the whole graph).

Algorithm 3.5 (Connectivity Testing Algorithm – Improved Version):

1. *For* $i = 1$ *to* $\log(8/(\epsilon d))$ *do:*
   (a) *Uniformly and independently select* $m_i = \frac{32 \cdot \log(8/(\epsilon d))}{2^i \cdot \epsilon \cdot d}$ *vertices in* G*;*
   (b) *For each vertex* s *selected, perform a BFS starting from* s *until* 2^i *vertices have been reached or no new vertices can be reached.*
2. *If any of the above searches finds a small connected component then output* REJECT*, otherwise output* ACCEPT.

Lemma 3.6 *If* G *is* ϵ*-far from the class of connected graphs with maximum degree* d *then Algorithm 3.5 rejects it with probability at least* $\frac{2}{3}$*. The query complexity and running time of the algorithm are* $O\left(\frac{\log^2(1/(\epsilon d))}{\epsilon}\right)$.

Proof: Let B_i be the set of connected components in G which contain at most $2^i - 1$ vertices and at least 2^{i-1} vertices. Let $\ell \stackrel{\text{def}}{=} \lfloor \log(8/\epsilon d) \rfloor$. By Corollary 3.3 we know that $\sum_{i=1}^{\ell} |B_i| \ge \frac{\epsilon d N}{8}$. Hence, there exists an $i \in \{1, 2, ..., \ell\}$ so that $|B_i| \ge \frac{\epsilon d N}{8 \cdot \ell}$. Thus, the number of vertices residing in components belonging to B_i is at least $2^{i-1} \cdot |B_i|$. It follows that the probability that a uniformly selected vertex resides in one of these components is at least

$$\frac{2^{i-1} \cdot |B_i|}{N} \ge \frac{\epsilon \cdot d \cdot 2^i}{16 \cdot \ell} = \frac{2}{m_i}$$

(where m_i is as defined in Step (1a) of Algorithm 3.5). Thus, with probability at least $1-(1-\frac{2}{m_i})^{m_i} > 1-e^{-2} > \frac{2}{3}$, a vertex s belonging to a component in B_i is selected in iteration i of Step (2), and the BFS starting from s will discover a small connected component leading to the rejection of G. The query complexity and running-time of the algorithm are bounded by $\sum_{i=1}^{\ell} m_i \cdot 2^i \cdot d = O\left(\frac{\log^2(1/(\epsilon d))}{\epsilon}\right)$. ■

The first part (i.e., $k=1$) of Item 1 in Theorem 3.1 follows from Lemma 3.6 and the fact that Algorithm 3.5 never rejects a connected graph (having more than $\frac{16\cdot\log(8/(\epsilon d))}{\epsilon\cdot d}$ vertices).

3.2 Testing k-Connectivity for $k > 1$

The structure of the testing algorithm for k-Connectivity where $k > 1$ is similar to the structure of the Connectivity Tester (i.e., case $k = 1$): We uniformly select a set of vertices and for each of these vertices we test if it belongs to a small component of the graph which has a certain property (i.e., is separated from the rest of the graph by an edge-cut of size less than k). Similarly to the $k = 1$ case, we show that if a graph is ϵ-far from being k–connected then it has many such components. In addition, we present an efficient procedure for recognizing such a component given a vertex which resides in it.

3.2.1 The Combinatorics

A subset of vertices $\mathrm{S} \subseteq \mathrm{V}$ is said to be k–edge-connected if there are k edge-disjoint paths between each pair of vertices in S. We stress that, in case $k \geq 3$, these paths may go through vertices not in S and that any singleton (a subset containing a single vertex) is defined to be k–edge-connected. The k–edge-connected classes of a graph G are maximal subsets of V(G) which are k–edge-connected, and each vertex in V(G) resides in exactly one such class. In the remainder of this subsection, whenever we say k–connected we mean k–edge-connected, and a k-class is a k–connected class.

We start by assuming that the graphs we test for k-connectivity are $(k-1)$–connected. We later (in Sec. 3.2.6) remove this assumption. In Appendix A we describe in more detail the structure of $(k-1)$–connected graphs in terms of their k-classes. Here we only state the facts necessary for our algorithms. Let G be a $(k-1)$–connected graph. Then we can define an auxiliary graph $\mathrm{T_G}$ [DW98] (based on the *cactus* structure of [DKL76]), which is a tree, such that for every k-class in G there is a corresponding (unique) node in $\mathrm{T_G}$. The tree $\mathrm{T_G}$ might include additional auxiliary nodes, but they are not leaves and we shall not be interested in them here. If G is k–connected, then $\mathrm{T_G}$ consists of a single node, corresponding to the vertex set of G. Otherwise, $\mathrm{T_G}$ has at least two leaves. The leaves of $\mathrm{T_G}$ play a central role in our algorithm. Each leaf corresponds to a k-class C of G which is separated from the rest of the graph by a cut of size $k-1$. (Recall that G is assumed to be $(k-1)$–connected.) As we show below, for every leaf class C, given a vertex $v \in \mathrm{C}$, we can efficiently identify that v belongs to a leaf class. For $k = 2$ this can be done deterministically within query and time complexity $O(|\mathrm{C}| \cdot d)$. For $k = 3$ this can be done deterministically within query and time complexity $O(|\mathrm{C}|^2 \cdot d)$. For $k \geq 4$, we present a randomized algorithm with query and time complexity $O(|\mathrm{C}|^3 \cdot d)$. The analysis of our algorithm relies on the following lemma which directly follows from Lemma A.4 (see Appendix A).

Lemma 3.7 *Let* G *be a* $(k-1)$*–connected graph that is* ϵ*-far from the class of* k*–connected graphs with maximum degree* $d \geq k$*. Suppose that either* $d \geq k+1$ *or* $k \cdot |\mathrm{V(G)}|$ *is even.*[5] *Then,* $\mathrm{T_G}$ *has*

[5] The reason for this technical requirement is to rule out the pathological case in which $d (= k)$ and $|\mathrm{V}(G)|$ are both

at least $\frac{\epsilon}{8}d|\mathrm{V(G)}|$ *leaves.*

Proof: Note that by the technical condition (in the lemma), either $d > k$ or $dN = kN$ is even, where $N \stackrel{\rm def}{=} |\mathrm{V(G)}|$. Assume towards contradiction that $\mathrm{T_G}$ has $L < \frac{\epsilon}{8}dN$ leaves. Then by Lemma A.4, G can be transformed into a k-connected graph G′ by removing and adding at most $4L < \frac{\epsilon}{2}dN$ edges. Furthermore, the maximum degree of G′ is $\max(k, d) = d$. This contradicts the hypothesis that G is ϵ-far from the class of k–connected graphs with maximum degree d. ■

Corollary 3.8 *Let* G *be a* $(k-1)$*–connected graph that is* ϵ*-far from the class of* k*–connected graphs with maximum degree* $d \geq k$*. Suppose that either* $d \geq k+1$ *or* $k \cdot |\mathrm{V(G)}|$ *is even. Then* $\mathrm{T_G}$ *has at least* $\frac{\epsilon}{16}d|\mathrm{V(G)}|$ *leaves each containing at most* $\frac{16}{\epsilon d}$ *vertices.*

3.2.2 The Basic Algorithm

Corollary 3.8 suggests the following algorithm, where the implementation of Step (2) is discussed subsequently. As was shown for the $k = 1$ case, the algorithm below can be modified to save a factor of $\tilde{\Theta}(1/\epsilon d)$ in its query complexity and running time, but for sake of simplicity we describe the less efficient algorithm. We also assume that the number of vertices N in G is greater than $\frac{16}{\epsilon d}$, (since otherwise a k-connected graph having less than $\frac{16}{\epsilon d}$ vertices would be rejected in Step (2)). If $N < \frac{16}{\epsilon d}$, we can decide if the graph is k–connected by observing the whole graph and running an algorithm for finding a minimum cut (in deterministic time $\tilde{O}(Ndk)$ [Gab95] or probabilistically in time $O(Nd\log^3 N)$ [Kar96], which here means $O(\epsilon^{-1}\log^3(1/\epsilon d))$).

Algorithm 3.9 (k-Connectivity Testing Algorithm – Basic version): *Recall, here we assume that the input graph is* $(k-1)$*-connected.*

1. *Uniformly and independently select* $m = \frac{32}{\epsilon d}$ *vertices;*
2. *For each vertex* s *selected, check whether* s *belongs to a* k*-class leaf which has at most* $\frac{16}{\epsilon d}$ *vertices.*
3. *If any leaf class is discovered then output* REJECT*, otherwise output* ACCEPT*.*

Our procedures for checking whether a given vertex belongs to a small k-class leaf always return the correct answer in case the vertex does not belong to such a leaf. Hence, a k-connected graph is always accepted. For $k = 2, 3$ the procedures also return a correct answer whenever the given vertex belongs to a small k-class leaf, and for $k \geq 4$ a correct answer is returned with probability at least 5/6. Hence, if the graph is ϵ-far from being k-connected, there may be two sources for the probability that it is erroneously accepted: By Corollary 3.8, the probability that no vertex s belonging to a small k-class leaf is selected in Step 1 is at most $(1-(\epsilon d)/16)^m < e^{-2} < 1/6$. For $k \geq 4$ we need to add the probability that the procedure for identifying a k-class leaf fails given such a vertex, obtaining the total of at most 1/3 error probability.

As said above, this algorithm can be modified analogously to the improved version of the connectivity tester, yielding

---

odd in which case it is not possible to transform G into a k–connected graph with maximum degree d by performing edge modifications. In other words, the class of k–connected graphs with max-degree k where k and $|\mathrm{V(G)}|$ are odd is empty. Clearly, this pathological case is easily detected by the algorithm.

Lemma 3.10 *Algorithm 3.9 runs in time*

$$O\left(\frac{\log(1/(\epsilon d))}{\epsilon d}\right) \cdot \sum_{i=1}^{\log(16/(\epsilon d))} \frac{\mathcal{T}_k(d, 2^i)}{2^i}$$

where $\mathcal{T}_k(d,n)$ is the time needed to implement the identification of a k-class leaf of size at most n on a graph with degree at most d (i.e., Step (2)). *It always accept a k–connected graph and rejects with probability at least $\frac{2}{3}$ any graph that is $(k-1)$–connected but ϵ-far from the class of k–connected with maximum degree d.*

In the following three subsection, we present such (k-class leaf) identification algorithms for the three cases $k = 2$, $k = 3$ and $k \geq 4$. The running-time bounds are $\mathcal{T}_2(d,n) = O(nd)$, $\mathcal{T}_3(d,n) = O(n^2 d)$, and $\mathcal{T}_k(d,n) = O(n^{3-\frac{2}{k}} d)$, respectively, where d is the degree bound (or actually the maximum degree of vertices in the class).

3.2.3 Identifying a 2-class Leaf

Given a vertex s and an integer n, the following Identification Procedure can be used to determine whether s belongs to a 2–connected class of size at most n which is a leaf in $\mathrm{T_G}$. Note that the upper bound, n, on the size of the class is determined by our higher level algorithm (for testing 2-connectivity) when calling the identification procedure. We use the following notation: for a subset $\mathrm{S} \subseteq \mathrm{V}$, we let $\overline{\mathrm{S}} \stackrel{\text{def}}{=} \mathrm{V} \setminus \mathrm{S}$.

Algorithm 3.11 (2-Class Leaf Identification Procedure): *On input a vertex s, and a bound n.*

1. *Starting from s, perform a Depth First Search (DFS) until $n+1$ vertices have been reached. Let T be the directed tree defined by the search, and let $\mathrm{E(T)}$ be its tree edges.*
2. *Starting once again from s, perform another search (using either DFS or BFS) until n vertices are reached or no new vertices can be reached. This search is restricted as follows: If (u, v) is an edge in T, where u is the parent of v, then (u, v) cannot be used to get from u to v in the second search (but can be used to get from v to u). Let S_2 be the set of vertices reached.*
3. *If there is a single edge with one end-point in S_2 and the other outside of S_2 (i.e. $(\mathrm{S}_2, \overline{\mathrm{S}_2})$ is a cut of size 1), then declare S_2 as the 2-class leaf (to which s belongs). Else announce failure to detect a small 2-class leaf containing s.*

Clearly, the query complexity and running time of the procedure are $O(nd)$. Since the procedure always checks if it has found a cut of size 1, it will never identify a 2-class leaf when given a vertex s belonging to a 2-connected graph (of size greater than n). Thus, we only need to prove that if s resides in a a 2-class leaf of size at most n then the above procedure will indeed detect this.

Lemma 3.12 *Let G be a connected graph, C a 2-class in G of size at most n which is a leaf in $\mathrm{T_G}$, and s a vertex in C. Then the above procedure terminates with $\mathrm{S}_2 = \mathrm{C}$.*

Proof: Since C is a 2-class, there exists a single edge (u, v) so that $u \in \mathrm{C}$ and $v \in \overline{\mathrm{C}}$. The first DFS terminates after seeing $n+1$ vertices, which means it must reach vertices of $\overline{\mathrm{C}}$, which in turn is possible only by traversing the single edge (u, v) from $u \in \mathrm{C}$ to $v \in \overline{\mathrm{C}}$. Thus, (u, v) must be a edge in T (with u being the parent). This ensures that the second search will never exit C. In other words, $S_2 \subseteq C$. What needs to be shown is that the second search reaches *every* vertex in C (i.e., $S_2 = C$), and hence the cut $(C, \overline{C})$ is discovered.

Assume contrary to this claim, that $X \stackrel{\text{def}}{=} C \setminus S_2$ is non-empty. Let $(u_1, v_1), \ldots, (u_\ell, v_\ell)$ be the set of edges crossing the cut (S_2, X), where $(\forall i)$ $u_i \in S_2$ and $v_i \in X$. Since C is 2–connected, there must be at least two edges in the cut (S_2, X). By our assumption that no vertex in X is reached in the second search, it follows that for every i, (u_i, v_i) is an edge in the DFS-tree T, and furthermore, u_i is the parent of v_i. Without loss of generality, let v_1 be the first vertex in X reached in the DSF defining T. Since C is 2–connected there must be a path between v_1 and v_2 which does not use the edge (u_1, v_1). There are two cases.

1. In case the path does not contain vertices in S_2, we reach a contradiction to T being a DFS-tree (since v_2 must be reached before the DFS backtracks from v_1 and hence $u_2 \to v_2$ cannot be a tree edge).

2. Otherwise, there must be a cut edge between some vertex, $v \in X$, in the DFS-subtree rooted at v_1 and a vertex, u, in S_2. By the structure of the DFS-tree, this cannot be a DFS-tree edge from u to v (as v must be reached before the DFS backtracks from v_1), contradicting our hypothesis about the cut edges.

■

3.2.4 Identifying a 3-class Leaf

Given a vertex s and a size bound n, we first perform a DFS until $n+1$ vertices are discovered. Next, for each edge e in this DFS-tree (which contains n edges), we "omit" e from the graph and invoke the 2-class leaf identification (of the previous subsection) on the residual graph.

Algorithm 3.13 (3-Class Leaf Identification Procedure): *On input a vertex s, and a bound n.*

1. *Starting from s, perform a Depth First Search (DFS) on* G *until $n+1$ vertices have been reached. Let* T *be the corresponding DFS-tree.*

2. *For each $e \in E(T)$, invoke the 2-Class Leaf Identification Procedure on the graph obtained by omitting e from* G *(that is, the edge e is not traversed at any step of the procedure.) In all these invocations, the input pair is (s, n) as above.*

3. *If a 2-class is identified in any of these invocations, output it as the desired 3-class. Otherwise announce failure to detect a small 3-class leaf containing s.*

Clearly, the above works in time $O(n \cdot nd)$, and never identifies a 3-class leaf when the graph G is 3-connected (and has more than n vertices). Identification of small 3-class leaves follows from Lemma 3.12.

Lemma 3.14 *Let* G *be a 2–connected graph,* C *a 3-class leaf of* T_G *with at most n vertices, and s an arbitrary vertex in* C*. Then the above search process terminates in finding the cut* $(C, \overline{C})$*.*

Proof: Clearly the initial DFS must cross an edge of the cut $(C, \overline{C})$, and so its DFS-tree has at least one cut edge. When this cut edge is omitted from the graph, the cut $(C, \overline{C})$ contains a single edge in the resulting graph, denoted G'. While the removal of this edge might decrease the connectivity of the vertices in C (which was 3 in G), they are at least 2–connected in G'. Invoking Lemma 3.12, we are done. ■

3.2.5 Identifying a k-class Leaf

The following applies to any $k \geq 2$, but for $k = 2,3$ we have described more efficient procedures (above). Our algorithm for finding leaf k-classes ($k \geq 2$) is based on Karger's Contraction Algorithm [Kar93] which is a randomized algorithm for finding a minimum cut in a graph.

Algorithm 3.15 (k-Class Leaf Identification Procedure): *Given a vertex s and a size bound n, the following randomized search process is performed $\Theta(n^{2-\frac{2}{k}})$ times, or until a cut $(\mathrm{S}, \overline{\mathrm{S}})$ of size less than k is found:*

> Random search process: *Starting from the singleton set $\{s\}$, the algorithm maintains the set, denoted S, of vertices it has visited. In each step, as long as $|\mathrm{S}| < n$ and the cut $(\mathrm{S}, \overline{\mathrm{S}})$ has size at least k, the algorithm selects at random (as specified below) an edge to traverse among the cut edges in $(\mathrm{S}, \overline{\mathrm{S}})$ and adds the new vertex reached to S. In case the cut $(\mathrm{S}, \overline{\mathrm{S}})$ has size less than k, we declare S to be a k-class leaf. If $|\mathrm{S}| = n$ the we complete the current search. Otherwise, we proceed to the next step.*

In case none of the $\Theta(n^{2-\frac{2}{k}})$ invocations of the above process has detected a k-class leaf, we announce failure to detect such a k-class.

Clearly, the query complexity and running time of Algorithm 3.15 are $O(n^{2-\frac{2}{k}} \cdot nd)$. If the graph is k-connected (and has size greater than n), then for every possible starting vertex s, the algorithm will announce failure to detect a k class of size at most n. Below we show that if s belongs to a k-class leaf of size at most n, then the probability that any (independent) invocation of the random search process succeeds is $\Theta\left(n^{-(2-\frac{2}{k})}\right)$. Since the random search process is invoked $c \cdot n^{2-\frac{2}{k}}$ times (for some constant c), for a sufficiently large constant c, the algorithm detects that s belongs to a k-class leaf with probability at least $5/6$. But before actually lower bounding the success probability of the random search process, we have to fully specify the process (i.e., the random selection of cut edges in the current $(\mathrm{S}, \overline{\mathrm{S}})$). Let C be the k-class leaf that s belongs to (where $|\mathrm{C}| \leq n$). Then we are interested in a random process for which the probability that an edge in $(\mathrm{C}, \overline{\mathrm{C}})$ is selected before all edges within C are selected is as small as possible.

A natural idea is to select, in each step, an edge uniformly in the current $(\mathrm{S}, \overline{\mathrm{S}})$; but this does not work well.[6] Instead, we think of uniformly and independently assigning each edge in the graph a cost in $[0,1]$. Then, at each step of the algorithm, we select the edge with lowest cost in the current $(\mathrm{S}, \overline{\mathrm{S}})$. This is implemented as follows: Whenever a new vertex is added to S, its incident edges that were not yet assigned costs are each assigned a random cost uniformly in $[0,1]$. Thus, whenever we need to select an edge from the current cut $(\mathrm{S}, \overline{\mathrm{S}})$, all edges in the cut have costs, and we select the edge with lowest cost (just as in the mental experiment in which all graph edges are assigned uniform costs at the beginning).

Lemma 3.16 *Let G be a $(k-1)$-connected graph, C a k-class leaf of $\mathrm{T_G}$ with at most n vertices, and s an arbitrary vertex in C. Then, with probability $\Theta\left(n^{-(2-\frac{2}{k})}\right)$, the random search process succeeds in finding the cut $(\mathrm{C}, \overline{\mathrm{C}})$.*

[6] Consider the case $k = 2$ and a graph containing a cycle of n-vertices connected to the rest of the graph by a single edge, denoted $e = (v, u)$. Thus, the cycle is separated from the rest of the graph by a single cut edge e. Suppose we start the random search at the cycle-node, denoted v, incident to e. Then, at each step until e is selected (i.e., u joins S), the current cut $(\mathrm{S}, \overline{\mathrm{S}})$ has 3 edges and e is one of them. Thus, the probability that e is selected in each step is $1/3$. It follows that the probability that all edges on the cycle are selected before e is selected (so that the random search process detects the cycle as a k-class leaf), equals $(2/3)^n$.

Proof: Assume first that instead of assigning the edges costs in an online manner as described above, all edges in the graph are assigned random costs off-line (as in the motivating "mental experiment"). We may think of our algorithm as simply revealing these costs as it proceeds. Consider any assignment of costs to all edges in the graph. A spanning tree, T, of the subgraph induced by C is said to be cheaper than the cut if the cost of every edge in T is smaller than the cost of any of the cut edges between C and $\overline{\mathrm{C}}$.

Claim 3.16.1: Suppose that C contains a spanning tree that is cheaper than the cut $(\mathrm{C}, \overline{\mathrm{C}})$. Then the search process succeeds in finding $(\mathrm{C}, \overline{\mathrm{C}})$.

Comment: The above claim presents a sufficient but NOT necessary condition for the success of the search process. For example, the search may expand S by an edge with cost greater than any cut-edge in case S is not incident to any cut-edge.

Proof of Claim 3.16.1: We prove, by induction on the size of the current S, that $\mathrm{S} \subseteq \mathrm{C}$. Specifically, at each step there is a tree-edge in the current cut $(\mathrm{S}, \overline{\mathrm{S}})$. Since this edge has lower cost than any edge in $(\mathrm{C}, \overline{\mathrm{C}})$, it follows that in this step the search cannot traverse an edge of $(\mathrm{C}, \overline{\mathrm{C}})$. Using the fact that $|\mathrm{C}| \leq n$, it follows that the search terminates with $\mathrm{S} = \mathrm{C}$. □

Thus, all we need is to lower bound the probability that C contains a cheaper-than-the-cut spanning tree. This is done by using Karger's analysis of his contraction algorithm (for finding a minimum cut) [Kar93]. Details follow.

Claim 3.16.2: Suppose that each edge is independently assigned a uniformly distributed cost in $[0, 1]$. Then, with probability at least $\Theta\left(n^{-(2-\frac{2}{k})}\right)$, C contains a spanning tree which is cheaper than the cut.

Proof of Claim 3.16.2: We start by considering an auxiliary graph G′, in which all of $\overline{\mathrm{C}}$ is represented by an auxiliary vertex, denoted x. That is, $\mathrm{V}(\mathrm{G}') = \mathrm{C} \cup \{x\}$ and $\mathrm{E}(\mathrm{G}')$ contains all edges internal to C and an edge (u, x) for every edge (u, v) such that $u \in \mathrm{C}$ and $v \in \overline{\mathrm{C}}$. Since C is a k–connected class in G, the graph G′ has a single minimum cut of size $k - 1$; that is, the cut $(\mathrm{C}, \{x\})$.

We now turn to Karger's analysis of his Contraction Algorithm. *Contraction* is an operation performed on a pair of vertices connected by an edge. When two vertices u and v are contracted, they are merged into a single vertex, w, where for each edge (u, z) such that $z \neq v$, we have an edge (w, z), and similarly for each edge (v, z') (such that $z' \neq u$). Thus, multiple edges are allowed, but there are no self-loops. Given a graph as input, the Contraction Algorithm performs the following process until two vertices remain: It selects an edge at random from the current graph (which is initially the original graph), and contracts its endpoints (resulting in a new graph which is smaller).[7] An alternative presentation is to assign all edges uniformly chosen costs in $[0, 1]$ and to contract the cheapest edge at each step. Karger shows that the probability that the algorithm never contracts a min-cut edge is at least $2n^{-2}$. In our case, this means that with probability at least $2n^{-2}$, Karger's algorithm does not contract an edge incident to x, which implies that C has a spanning tree cheaper than the cut $(\mathrm{C}, \{x\})$.

To obtain the better bound (i.e., $\Theta\left(n^{-(2-\frac{2}{k})}\right)$) claimed above, we reproduce Karger's analysis [Kar93]. We consider an $(n+1)$-vertex graph with min-cut of size $c = k - 1$ and such that, except for one vertex (i.e., x), the degree of every vertex in the residual graph at any step of the

[7]Note that this is not the same as randomly selecting an edge between the set of vertices previously merged (S) and the rest of the graph $\overline{\mathrm{S}}$, as here we allow the selection of any edge in the graph at each step.

Contraction Algorithm is at least $D \geq k$. The degree of x remains $k-1$, provided none of its edges was contracted. Hence, for $i = 1, ..., n-1$, at the i^{th} step of the algorithm, the probability of choosing to contract a cut edge is at most $\frac{c}{(c+(n-(i-1))\cdot D)/2}$ (i.e., the size of the cut divided by a lower bound on the number of current edges). The probability no cut edge is contracted in any step of the algorithm is at least

$$\prod_{i=1}^{n-1}\left(1-\frac{2c}{c+(n-(i-1))D}\right) = \prod_{i=0}^{n-2}\frac{(n-i)D-c}{(n-i)D+c} = \prod_{j=2}^{n}\frac{j-(c/D)}{j+(c/D)} > \Theta(n)^{-2c/D} \qquad (2)$$

where the strict inequality is due to elementary algebraic manipulations (see Appendix B). In our case, since all cuts in G' other than the minimum cut $(\mathrm{C}, \{x\})$ have size at least k, we can set $c = k-1$, $D = k$, and the claim follows. □

Combining Claims 3.16.1 and 3.16.2, Lemma 3.16 follows. ■

3.2.6 Testing k-Connectivity of Graphs that are not $(k-1)$–connected

So far we have assumed that the graph being tested (for k-connectivity) is $(k-1)$-connected. In this section we remove this assumption and show that (a slight modification of) Algorithm 3.9, with distance parameter set to $\epsilon/O(k)$, rejects with probability at least 2/3 any graph that is ϵ-far from being k-connected. This yields the general tester for k-connectivity asserted in Theorem 3.1.

Let us consider first what happens when we run Algorithm 3.9 on an $(i-1)$–connected graph which is ϵ-far from being i–connected, where $i \leq k$. In this case, by Corollary 3.8 the auxiliary graph $\mathrm{T_G}$ (corresponding to the i-classes of the graph) has at least $\frac{\epsilon}{16}dN$ i-class leaves each containing at most $\frac{16}{\epsilon d}$ vertices. Hence, with probability at least $1-(1-\frac{\epsilon d}{16})^{\frac{32}{\epsilon d}} > 1-\epsilon^{-2} > 5/6$ a vertex belonging to such a class is selected. We next observe that the Identification Procedure for k-class leaves is such that when invoked inside a small i-class leaf it detects a cut of size $i-1 < k$ (with probability at least 5/6). (We stress that this holds also for $i = 1$ (with probability 1), in which case this means that the algorithm detects a small connected component.) Furthermore, the more efficient Identification procedures for 2-class leaves (resp., 3-class leaves) can be easily modified so that they detect small connected component (resp., small 2-class leaf), when the start vertex resides in such a component (resp., class). Specifically, in Step (1) of the 2-Class procedure, one should declare detection in case less than $n+1$ vertices are found in the initial DFS. The 3-Class procedure is modified analogously. Hence, with probability at least $\frac{2}{3}$, the algorithm will detect a small i-class leaf and will reject.

However, in general the situation may be more complex: Although the graph may be ϵ-far from being k–connected, it may be the case that there exists no i so that the graph is an $(i-1)$–connected graph and ϵ-far from being i–connected. Intuitively, the k-connectivity tester should reject such graphs also with probability at least 2/3; but the question is how to prove this intuition. Let $\mathrm{G}_0 \stackrel{\text{def}}{=} \mathrm{G}$ be a graph that is ϵ-far from being k–connected, and for $i = 1, \ldots, k$, let G_i be an i–connected graph (with maximum degree d) that is closest to G_{i-1}. By definition of the G_i's there exists an i such that G_{i-1} (which is $(i-1)$–connected), is ϵ/k-far from being i–connected (since otherwise we would reach contradiction to G being ϵ-far from k-connected). Now, if the algorithm were to run on this G_{i-1} (with distance parameter ϵ/k) then it would reject with probability at least 2/3. The problem, however, is that the algorithm runs on G. It is tempting to think that nothing can go wrong, but there are two issues to take care to: Firstly, even if G_{i-1} can be obtained from $\mathrm{G} = \mathrm{G}_0$ only by adding edges (so that G is a subgraph of G_{i-1}), it has to be shown that if the algorithm rejects a graph it will also reject any subgraph of it. Secondly, it may not be the case that G_{i-1} can be obtained from G by just adding edges (since maintaining the degree bound may cause us to omit edges as well – see proof of Lemma A.4). We start by addressing the second problem. The following lemma allows us to simplify the analysis by considering the distance of the graph to the class of i–connected graphs rather than to the class of i–connected graphs with degree bound d. We stress that the minimum distance to the former class (which has no degree bound) is obtained by only adding edges.

Lemma 3.17 *Let* G *be a graph that is ϵ-far from the class of k-connected graphs with maximum degree d, where either kN is even or $d \geq k+1$.*[8] *Then the minimum number of edges which must be added to* G *in order to transform it into a k-connected graph* (without any bound on its degree)*, is at least $\frac{1}{26}\epsilon dN$.*

Proof: Assume, contrary to the claim that in order to transform G into a k-connected graph it suffices to augment it with $m < \frac{1}{26}\epsilon dN$ edges. We next show that by adding and removing at most $13m < \frac{1}{2}\epsilon dN$ edges we can transform G into a k-connected graph which has maximum degree d, in contradiction to the hypothesis.

Let G_k be a k-connected graph which results from augmenting G with m edges. Some of the vertices in G_k might have degree larger than d. Hence we define the *excess* of G_k (with respect to the degree bound d) as $\sum_{v,\, \deg(v)>d}(\deg(v) - d)$. Since G has maximum degree d, and G_k was obtained by augmenting G with m edges, the excess of G_k is at most $2m$. We now show how by performing at most $12m$ edge modifications to G_k, we can obtain a k-connected graph with excess 0 (i.e., maximum degree at most d). Thus, we transform G (via G_k) into a k-connected graph with degree bound d by modifying at most $m + 12m$ edges. At each step of the following process we decrease the excess of the graph while retaining its k-connectivity.

While the excess of the graph is non-zero, do:

Case 1: *There is an edge (u,v) such that* $\deg(u) > d$ *and* $\deg(v) > k$. In this case we start by removing the edge (u, v) from the graph. If the graph remains k-connected, no additional modification is needed. Otherwise (the graph becomes $(k-1)$-connected), by Lemma A.2 (in Appendix A), the auxiliary tree of the graph consists of a simple path, with u belonging to one k-class leaf, and v to the other. Since v now has degree at least k, it cannot be a singleton leaf (because leaves have exactly $k-1$ edges going out of them). The same holds for u which now has degree at least $d \geq k$. We can thus apply Lemma A.3 on the two leaf k-classes, and obtain a k-connected graph at the cost of 4 edge modifications. Thus, we have decreased the excess by at least 1, at the cost of $1 + 4 = 5$ edge modifications.

Case 2: *For every vertex u such that* $\deg(u) > d$*, all of u's neighbors have degree k.* (Recall that no vertex may have degree lower than k since the graph is k-connected.) We consider two subcases.

Case 2.a: *There exist two vertices, u_1 and u_2, so that* $\deg(u_i) > d$ *and all neighbors of u_i have degree k.* Then there must exist two vertices $v_1 \neq v_2$ such that v_1 is a neighbor of u_1 and v_2 is a neighbor of u_2. (If u_1 and u_2 only had a single (common) neighbor, or had edges between themselves, this would contradict the hypothesis that they both only have degree k neighbors.) We add an edge between v_1 and v_2, increasing their degree to

[8] Recall that the technical condition (i.e., either kN is even or $d \geq k+1$) is required as otherwise the class of k-connected graph with maximum degree d is empty.

$k+1$, and then apply Case 1 twice; that is, to the edges (u_i, v_i), for $i = 1, 2$. We have decreased the excess of the graph by 2, at a cost of $1 + 2 \cdot 5 = 11$ edge modifications.

Case 2.b: *There exist a single vertex, u, with degree greater than d* (and all its neighbors have degree $k \leq d$). Here we further consider two subcases.

(i) $\deg(u) > d + 1$. In such a case, we must remove at least two edges adjacent to u. Let $v_1 \neq v_2$ be any two neighbors of u (once again, the existence of two such distinct vertices follows from the hypothesis that all of u's neighbors have degree k). We now proceed as in Case 2.a, by adding an edge between v_1 and v_2 and then applying Case 1 to (u, v_1) and then to (u, v_2). We have decreased the excess of the graph by 2, at a cost of $1 + 2 \cdot 5 = 11$ edge modifications.

(ii) $\deg(u) = d + 1$. Let v be any neighbor of u (which, recall, must has degree $k \leq d$).
Claim: There exists a vertex (other than v), denoted w, with degree smaller than d. Before proving the claim, let us see how we complete the process in this case. First, we add an edge between v and w, raising the degree of v to $k+1$ (where the degree of w is now at most d). Applying Case 1 to the edge (u, v) we are done (at a cost of $1 + 5 = 6$ edge modifications).
Proof of Claim: Assume the claim does not hold. Then, except for u and v, all vertices in the graph have degree d. We show that this is not possible by using the lemma's technical assumptions by which either $d > k$ or $dN = kN$ is even. In case $d > k$, all neighbors of u other than v have degree $d > k$, contradicting the hypothesis that all of u's neighbors have degree k (and again, u must have such neighbors since $\deg(v) = k < d + 1 = \deg(u)$). In case $d = k$ we have that u has degree $d + 1$ and all other vertices in the graph have degree $k = d$, yielding a degree sum of $kN + 1$ which is odd (and hence impossible). The claim follows. □

Thus in all cases, a decrease of 1 unit in the excess of the graph is obtained at a cost of at most 6 edge modifications. Since the initial excess (of G_k) is at most $2m$, we obtain the desired graph via at most $2m \cdot 6 = 12m$ edge modifications (to G_k). The lemma follows. ■

Following the discussion above, we slightly modify Algorithm 3.9 so that in Step (2) (rather than looking for a k-class leaf) one looks for a small set of vertices which is separated from the rest of the graph by a cut of size $j < k$. Such a set will be called j-separated, and is called j-extreme if it contains no subset which is j'-separated for any $j' \leq j$. We also incorporate the change in parameters (i.e., replacing ϵ by $\epsilon/O(k)$). For sake of clarity, we reproduce the resulting algorithm below.

Algorithm 3.18 (k-Connectivity Testing Algorithm – General version):

1. *Uniformly and independently select $m = \frac{O(k)}{\epsilon d}$ vertices;*
2. *For each vertex s selected, check whether for some $j \leq k$, vertex s belongs to a j-extreme set containing at most $\frac{200k}{\epsilon d}$ vertices.*
3. *If any such separated set is discovered then output* REJECT, *otherwise output* ACCEPT.

Our procedures for Identifying k-class leaves are easily adapted to detect that a give vertex belongs to a j-extreme set for some $j < k$ (see details below). But first let us verify that Algorithm 3.18 constitutes a tester for k-connectivity. Clearly, Algorithm 3.18 always accept a k–connected graph (having more than $\frac{200k}{\epsilon d}$ vertices). On the other hand, using Lemma 3.17 and observing that the rejecting probability of Algorithm 3.18 can only increase when we remove edges from the graph, we prove

Lemma 3.19 *Algorithm 3.18 rejects with probability at least $\frac{2}{3}$ any graph that is ϵ-far from the class of k–connected graphs with maximum degree d.*

Proof: Let G be ϵ-far from the class of k–connected graphs with maximum degree d. By Lemma 3.17, at least $m \geq \frac{\epsilon dN}{26}$ edges must be added to G in order to make it k–connected. For every $i \geq 1$, let us denote by m_i the minimum number of edges that should be added to G in order to make it i–connected, and let G_i denote an i–connected graph which results when adding such m_i edges to G. (We stress that G_i does not necessarily maintain the degree bound d.) Let $m_0 \stackrel{\text{def}}{=} 0$ and $\mathrm{G}_0 \stackrel{\text{def}}{=} \mathrm{G}$. Then, there must exist an $i \in \{1, ..., k\}$ so that $m_i - m_{i-1} \geq m/k$. Let us consider any such i and let $\epsilon' \stackrel{\text{def}}{=} \epsilon/(26k)$.

It follows that in order to transform G_{i-1} into an i–connected graph, we must augment it with at least $(m/k) = \epsilon' dN$ edges. By applying Lemma A.2, it follows that the auxiliary tree of G_{i-1} has a least $\frac{1}{2}\epsilon' dN$ leaves (or else G_{i-1} can be transformed into a k-connected graph by adding at most $\epsilon' dN - 1$ edges).[9] Following the argument in Corollary 3.3, at least $\frac{1}{4}\epsilon' dN$ of these leaves have each at most $\frac{4}{\epsilon' d} = \frac{104k}{\epsilon d}$ vertices. Thus, with probability at least $\frac{1}{4}\epsilon' d = \epsilon d/O(k)$, a uniformly selected vertex resides in such a component. Thus, if we were to run Algorithm 3.18 on G_{i-1} then the algorithm would reject with probability at least 2/3. What is left to show is that the rejection probability of the algorithm on input graph G, which is a subgraph of G_{i-1}, is not smaller. The key observation is that if a vertex, s, belongs to some i-class leaf, C, of G_{i-1} then for $j \leq i$ vertex s must belong to some j-extreme set $\mathrm{C}' \subseteq \mathrm{C}$ of G (which is a subgraph of G_{i-1}). It follows that the number of small (disjoint) extreme sets in G is lower bounded by the number of i-class leaves in G_{i-1}, and the lemma follows. ■

Detecting extreme sets: Algorithm 3.15 (for detecting k-class leaves) actually detect j-extreme sets, for any $j \leq k$. This can be verified by going over the proof of Lemma 3.16 and noting that it relies only on the hypothesis that the relevant set (in that case the k-class leaf) is in fact j-extreme (there for $j = k$). It follows that a single iteration of the random search process started in a j-extreme set of size j will detect the set with probability at least $O(n^{2-\frac{2}{j}})$ if $j \geq 2$ and probability 1 otherwise (for $j = 1$ which means that the set is a connected component). Algorithm 3.11 (resp., Algorithm 3.13) for detecting 2-class (resp., 3-class) leaves actually detects 2-extreme (resp., 3-extreme) sets. But we need to modify it a little so that it may detect j-extreme sets, for any $j \leq 2$ (resp., $j \leq 3$). These modifications were already discussed in the beginning of the current subsection.

Finally, to derive Theorem 3.1, we modify Algorithm 3.18 analogously to the way Algorithm 3.4 was modified to obtain Algorithm 3.5. Observe that our analysis of the execution of the algorithm on graphs which are far from being k-connected only refers to a collection of disjoint extreme sets. For any such set S (which is j-extreme for some $j \leq k$), the probability that a uniformly selected vertex resides in it equals $\frac{|S|}{N}$. Moreover, on input a vertex in S and a size bound $n \geq |S|$ the cut $(S, \overline{S})$ is detected with high probability (say with probability at least 0.9) within time $\mathcal{T}_k(d, n)$,

[9] Note that since we don't require the resulting graph to maintain the degree bound, this simpler lemma suffices (and we don't need the more sophisticated Lemma 3.7, which in turn relies on Lemma A.4).

where $\mathcal{T}_k(d,n)$ denotes the running time of our procedures for identifying j-extreme sets for $j \leq k$ (analogously to the definition in Lemma 3.10). Using an analysis as in the proof of Lemma 3.6 the complexities asserted in Theorem 3.1 follow.

4 Testing if a Graph is Cycle-Free (a Forest)

The testing algorithm described in this section is based on the following observation. Let G be the tested graph and $C_1, C_2, \ldots, C_k$ its connected components. By definition, if G is cycle-free then each of its components is a tree. In such a case, each C_i has $|C_i| - 1$ edges, and the total number of edges in G is $N - k$. On the other hand, if G is far from being cycle-free then it has many more edges within its components, where these edges create cycles inside the components. Each such "superfluous" edge resides either in a small component or in a big component (where the notions of small and big are made precise in the formal analysis of the algorithm). If there are many extra edges residing in small components, then (due to the degree bound) there must be many vertices that belong to such small components. In this case, if we uniformly select a large enough number of vertices, with high probability we obtain such a vertex, and we can detect that its component has extra edges (i.e., contains cycles), by performing a search. Otherwise, there are many extra edges residing in big components (whom we cannot exhaustively search). In this case we consider the subgraph of G that consists of all big components and detect a discrepancy between its edge count and its vertex count. Since here the number of components is relatively small it cannot account for this discrepancy.
The above discussion suggest the following algorithm.

Algorithm 4.1 (Cycle-Freeness Testing Algorithm):

1. *Uniformly and independently select* $\ell = \Theta\left(\frac{1}{\epsilon^2}\right)$ *vertices;*
2. *For each vertex s selected, perform a BFS starting from s until* $\frac{8}{\epsilon d}$ *vertices are reached or no more new vertices can be reached (s belongs to a small connected component);*
3. *If any of the above searches found a cycle then output* REJECT *(otherwise continue);*
4. *Let* $\hat{n}$ *be the number of vertices in the sample which belong to connected components of size greater than* $\frac{8}{\epsilon d}$*, and let* $\hat{m}$ *be half the sum of their degrees. If* $\frac{\hat{m}-\hat{n}}{\ell} \geq \frac{\epsilon d}{16}$ *then output* REJECT*, otherwise output* ACCEPT*.*

Theorem 4.2 *Algorithm 4.1 is a testing algorithm for the Cycle-Free property whose query complexity and running time are* $O\left(\frac{1}{\epsilon^3} + \frac{d}{\epsilon^2}\right)$.

Proof: Since each BFS takes time $O(1/(\epsilon d) \cdot d) = O(1/\epsilon)$, and $\ell = O(1/\epsilon^2)$ such searches are performed, Steps 1-3 of the algorithm takes $O(1/\epsilon^3)$ time. Step 4 takes at most $\ell \cdot d = O(d/\epsilon^2)$ time, and we obtain the complexity bounds stated in the lemma. We now turn to establish that Algorithm 4.1 is indeed a tester for cycle-freeness. We start with the quality of the approximations performed in Step (4).

We say that a component is *small* if it contains less than $\frac{8}{\epsilon d}$ vertices, otherwise it is *big*. Let us denote by t the number of *big* components. We first establish that with probability at least $\frac{2}{3}$ both estimates done in Step (4) are accurate to within $(\epsilon d)/32$. Let N' be the number of vertices belonging to big components, and let M' be the number of edges in big components. For $i = 1, \ldots, \ell$, let χ_i be a 0-1 random variable that equals 1 if and only if the i^{th} vertex selected belongs to a big component. Then $\hat{n} = \sum_i \chi_i$, and the expected value of $\frac{\hat{n}}{\ell}$ is $\frac{N'}{N}$. By a Chernoff bound, since $\ell = \Theta(1/\epsilon^2)$, then for an appropriate constant in the $\Theta(\cdot)$ notation, with probability at least 5/6, we have $\left|\frac{\hat{n}}{\ell} - \frac{N'}{N}\right| < \frac{\epsilon}{32}$. Similarly, for $i = 1, \ldots, \ell$, let ψ_i be a random variable taking values between 0 and d, that equals the degree of the i^{th} vertex selected if it belongs to a big component, and 0 otherwise. Then $\hat{m} = \frac{1}{2}\sum_i \psi_i$, and the expected value of $\frac{\hat{m}}{\ell}$ is $\frac{M'}{N}$. Applying a Chernoff bound once again (while noting that the range of the random variables is $[0, d]$) we obtain that with probability at least 5/6, $\left|\frac{\hat{m}}{\ell} - \frac{M'}{N}\right| < \frac{\epsilon d}{32}$. From this point on we assume that these estimates in fact hold, so that $\left|\frac{\hat{m}-\hat{n}}{\ell} - \frac{M'-N'}{N}\right| < \frac{\epsilon d}{16}$. The probability (of at most 1/3) that these estimates are not within these bounds accounts for the probability that the testing algorithm fails.

In case G is cycle-free, the algorithm never rejects in Step (2). Furthermore, in this case we have $M' - N' = -t \leq 0$, and so by our assumption on the estimates $\hat{n}$ and $\hat{m}$, $\frac{\hat{m}-\hat{n}}{\ell} < \frac{\epsilon d}{16}$, so that the algorithm accepts in Step (4).

We now consider the case that G is ϵ-far from cycle-free. For any connected component in G having n vertices and m edges, we define $m - (n-1) \geq 0$ to be the number of *superfluous edges in the component.* Since G is ϵ-far from cycle-free the total number of superfluous edges is at least $\frac{1}{2}\epsilon dN$. We consider two cases:

Case 1: *There are* $\frac{\epsilon dN}{4}$ *superfluous edges inside small components.* Consider a (small) component having s superfluous edges. Then using the degree bound d, this component must contain at least $2s/d$ vertices. Thus, the total number of vertices in small components which contain superfluous edges is at least $\frac{\epsilon N}{2}$. Recall that if a connected component has a superfluous edge then it necessarily has a cycle. Hence, in this case a cycle is detected in Step (2) with probability at least $1 - (1 - \frac{\epsilon}{2})^\ell > \frac{2}{3}$.

Case 2: *There are* $\frac{\epsilon dN}{4}$ *superfluous edges inside big components.* Recall that t denotes the number of big components, and N' (resp., M') the number of vertices (resp., edges) in them. By definition of superfluous edges, we have $M' - (N' - t) \geq \frac{\epsilon dN}{4}$. Since $t \leq \frac{N}{8/(\epsilon d)} = \frac{\epsilon dN}{8}$, we get that $\frac{M'-N'}{N} \geq \frac{\epsilon d}{8}$. By our assumption on the estimates $\hat{n}$ and $\hat{m}$, we obtain $\frac{\hat{m}-\hat{n}}{\ell} > \frac{\epsilon dN}{16}$ so that the algorithm rejects in Step (4).

■

Remark: The above tester has two-sided error probability. The next proposition, whose proof is provided at the end of Subsection 7.1, asserts that this is unavoidable if one allows only $o(\sqrt{N})$ many queries.

Proposition 4.3 *Any algorithm for testing cycle-freeness that always accept cycle-free graphs must make* $\Omega(\sqrt{N})$ *queries.*

5 Testing Subgraph Freeness

Two graphs, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, are called isomorphic if there is a 1-1 and onto mapping $\pi : V_1 \to V_2$ so that $(u, v) \in E_1$ iff $(\pi(u), \pi(v)) \in E_2$. A graph G is H-free, if no subgraph of G is isomorphic to H; that is, for every 1-1 mapping $\phi : \mathrm{V(H)} \to \mathrm{V(G)}$ there exist $u, v \in \mathrm{V(H)}$ so that $(u, v) \in \mathrm{E(H)}$ but $(\phi(u), \phi(v)) \in \mathrm{E(G)}$.

A natural algorithm for testing H-freeness consists of selecting a vertex at random and checking if it participates in a subgraph of G which is isomorphic to H. Let diam(H) denote the diameter of H (where the diameter of a connected graph is the largest distance between any pair of vertices in the graph). Then starting at a random vertex, we should just search G up to distance diam(H).

Algorithm 5.1 (H-freeness Testing Algorithm):

1. *Uniformly and independently select* $m = \Theta\left(\frac{1}{\epsilon}\right)$ *vertices in* G*;*
2. *For each vertex s chosen, perform a BFS starting from s up to depth* diam(H)*.*
3. *If any of the above searches found a subgraph isomorphic to* H *then output* REJECT*, otherwise output* ACCEPT*.*

Theorem 5.2 *Algorithm 5.1 is a testing algorithm for the* H*-freeness property whose query complexity and running time are* $O\left(\frac{d^{\mathrm{diam(H)}}}{\epsilon}\right)$ *and* $O\left(\frac{d^{\mathrm{diam(H)}\cdot|\mathrm{V(H)}|+1}\cdot|\mathrm{V(H)}|}{\epsilon}\right)$*, respectively.*

Proof: Clearly, if G is H-free it is accepted with probability 1. Since in each search at most $d^{\mathrm{diam(H)}}$ queries are asked (as diam(H) is the depth of the BFS), the algorithm's query complexity is $O\left(\frac{d^{\mathrm{diam(H)}}}{\epsilon}\right)$. Let R denote the subgraph of G reached during the BFS in Step 2. Then, the third step of the algorithm (i.e., looking for a subgraph isomorphic to H) can be implemented by trying all possible 1-1 mappings of H into R, and for each such mapping checking if the induced subgraph contains the edges of H. Thus, the time complexity is bounded by $|\mathrm{V(R)}|^{|\mathrm{V(H)}|} \cdot d|\mathrm{V(H)}|$. Since $|\mathrm{V(R)}| \leq d^{\mathrm{diam(H)}}$ the bound in the theorem follows.

It remains to show that if G is ϵ-far from the class of H-free graphs then the Algorithm 5.1 rejects it with probability at least $\frac{2}{3}$. But this follows directly from the definition of ϵ-far: If G is ϵ-far from the class of H-free graphs then it contains at least $\frac{\epsilon}{2}dN$ edges that reside in subgraphs of G which are isomorphic to H. Since the degree of every vertex is at most d, there are at least ϵN vertices that reside in such subgraphs. Since the algorithm uniformly selects $\Theta\left(\frac{1}{\epsilon}\right)$ vertices, with probability 2/3 at least one of these vertices resides in such a subgraph, and this will be detected in the third step of the algorithm. ■

The above algorithm extends to testing whether the input graph G has no subgraph isomorphic to any of a fixed collection of graphs $\mathrm{H}_1, ..., \mathrm{H}_k$. Alternatively, we note that although, in general, property testing is not closed under intersection of properties [GGR98], closure does hold for monotone decreasing graph properties (such as H-freeness). That is,

Theorem 5.3 *Let* Π_1 *and* Π_2 *be two graph properties that are monotone decreasing; that is, if* $\mathrm{G} \in \Pi_i$ *then every subgraph of* G *is in* Π_i*. Suppose that* $\mathcal{A}_i$ *is an algorithm for testing property* Π_i *having failure probability* 1/6 *(rather than* 1/3*). Then an algorithm that on input graph* G *and distance parameter* ϵ *invokes both* $\mathcal{A}_i$*'s on* G *with distance parameter* $\epsilon/2$*, and accepts if and only if both accept, is a property tester for the conjunction of* Π_1 *and* Π_2*.*

We comment that the above theorem extends also to arbitrary properties that are monotone decreasing (i.e., classes of arbitrary functions that are not necessarily graph properties).

Proof: Let $\Pi_{1,2}$ denote the property that is defined by the conjunction of Π_1 and Π_2. Clearly, if G has property $\Pi_{1,2}$ then each of the two algorithms will reject it with probability at most 1/6, and hence the combined algorithm rejects with probability at most 1/3. The key claim is that, in case both properties are monotone decreasing, if G is ϵ-far from $\Pi_{1,2}$ then G must be either $\epsilon/2$-far from Π_1 or $\epsilon/2$-far from Π_2, in which case it is rejected by either $\mathcal{A}_1$ or $\mathcal{A}_2$ (with probability at least $5/6 > 2/3$). Suppose, on the contrary that G is $\epsilon' = \epsilon/2$-close to both Π_1 and Π_2. Let G_1 be a graph having property Π_1 that is at distance ϵ' from G, and let G_2 be a graph having property Π_2 that is at distance ϵ' from G. Consider a maximal graph, denoted G', which is a subgraph of the three graphs G, G_1 and G_2. Namely, $E(G') = E(G) \cap E(G_1) \cap E(G_2)$. By monotonicity of both properties, G' has property $\Pi_{1,2}$. By definition of G', $E(G') \subseteq E(G)$. Finally, any edge that appears in G and not in G' must be missing in either G_1 or G_2, and so is counted in their distances to G. This implies that

$$\mathrm{dist}_d(G, G') = \frac{2 \cdot |E(G) \setminus E(G')|}{Nd} \leq \frac{2 \cdot |E(G) \setminus E(G_1)| + 2 \cdot |E(G) \setminus E(G_2)|}{Nd} \leq 2\epsilon' = \epsilon$$

But this contradicts the fact that G is ϵ-far from $\Pi_{1,2}$, and the theorem follows. ∎

6 Testing if a Graph is Eulerian

A graph $G = (V, E)$ is *Eulerian* if there exists a path in the graph that traverses every edge in E exactly once. It is well known 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. The testing algorithm is quite straightforward. In addition to testing connectivity (as done in subsection 3.1), we sample vertices and reject whenever we see more than two vertices of odd degree.

Algorithm 6.1 (Eulerian Testing Algorithm):

1. *Invoke Algorithm 3.5 with distance parameter $\epsilon/2$, and* REJECT *if that algorithm rejects.*
2. *Uniformly and independently select $m = O(1/\epsilon d)$ vertices in the graph, determine the degree of each vertex, and* REJECT *if more than two different vertices have odd degree. Otherwise* ACCEPT.

   That is, initiate $S \leftarrow \emptyset$, and repeat the following steps m times.

   (a) Uniformly select a vertex v in the graph;

   (b) If the degree of v is odd then $S \leftarrow S \cup \{v\}$.

   If $|S| > 2$ the REJECT *else* ACCEPT.

Thus, we test the two properties whose conjunction yields the desired property. However, the analysis does not reduce to showing that each of the two sub-testers is valid – as property testing of a conjunction of two sub-properties does not reduce in general to the property testing of each of the two sub-properties [GGR98]. Nonetheless, the following lemma does establish the validity of our tester.

Lemma 6.2 *Let G be a graph that is ϵ-far from the class of Eulerian graphs with maximum degree d. Then, it either has more than $\frac{\epsilon}{8}dN$ connected components, or it has more than $\frac{\epsilon}{16}dN$ vertices with odd degree.*

Proof: Assume contrary to the claim that G has at most $\frac{\epsilon}{8}dN$ connected components, and at most $\frac{\epsilon}{16}dN$ vertices with odd degree. We now show that by adding and removing less than $\frac{\epsilon}{2}dN$ edges we can transform G into a Eulerian graph (while maintaining the degree bound).

First consider the case in which $d \geq 2$ is even, and hence all odd degree vertices have degree less than d. In such a case, we first pair all these vertices up and add an edge between every pair (using at most $\frac{\epsilon}{32}dN$ edges). Clearly, the number of connected components can only decrease in this process. At this point, all vertices have even degree, which in particular means that all (at most $\frac{\epsilon}{8}dN$) connected components either consist of a single vertex (with degree 0) or have a cycle in them. We can then remove one edge from each non-trivial component, and then connect all components in a cycle without raising the degree of any vertex above d. Specifically, in case the edge (u_i, v_i) was removed from the i^{th} component then we connect u_i (resp., v_i) to a vertex of the $i - 1^{\text{st}}$ (resp., $i + 1^{\text{st}}$) component. Thus, the resulting graph is connected and all its vertices have even degree. The total number of edge modifications is bounded by $\frac{\epsilon dN}{32} + 2 \cdot \frac{\epsilon dN}{8} < \frac{\epsilon dN}{2}$.

In case d is odd, we first remove a single incident edge from each vertex of degree d. Since there are at most $\frac{\epsilon}{16}dN$ vertices of odd degree, at most $\frac{\epsilon}{16}dN$ edges were removed. The number of vertices of odd degree cannot increase (as each edge omission flips the parity of the degrees of both end-points, and at least one of these degrees was odd). The number of connected component may increase by at most $\frac{\epsilon}{16}dN$, and so is now at most $\frac{3\epsilon}{16}dN$. The resulting graph has degree at most $d - 1$, which is even, and so we can apply the procedure of the even case (above). In this case, we obtain an Eulerian graph of degree at most $d - 1$ by making at most

$$\frac{\epsilon dN}{16} + \left(\frac{\epsilon dN}{32} + 2 \cdot \frac{3\epsilon dN}{16}\right) \; < \; \frac{\epsilon dN}{2}$$

edge modifications. ■

Theorem 6.3 *Algorithm 6.1 is a testing algorithm for the Eulerian property whose query complexity and running time are* $O\left(\frac{\log^2(1/(\epsilon d))}{\epsilon}\right)$.

Proof: Algorithm 6.1 has complexities as stated and clearly accepts any Eulerian graph. Now suppose it is given access to a graph that is ϵ-far from any Eulerian graph (with maximum degree d). Then, by Lemma 6.2, one of the following cases holds.

Case 1: *The graph has at least $\frac{\epsilon}{8}dN$ connected components.* By adapting the figures in the proof of Lemma 3.6 it follows that with probability at least 2/3, Algorithm 6.1 rejects in Step (1).

Case 2: *The graph has at least $\frac{\epsilon d}{16} \cdot N$ vertices of odd degree.* Thus, the probability that a uniformly selected vertex has odd degree is at least $\epsilon d/16$. With an appropriate choice of $m = O(1/\epsilon d)$, it follows that with probability at least 2/3 more than two odd degree vertices are seen in Step (2), and the algorithm rejects.

Thus, in both cases Algorithm 6.1 rejects with probability at least 2/3 as required. ■

7 Hardness Results

In this section we present several lower bounds on the query complexity and running time required for testing various properties.

7.1 Testing Bipartiteness

A graph is said to be *bipartite* if its set of vertices can be partitioned into two disjoint sets so that there are no *violating edges*. An edge is said to be violating with respect to a given partition (V_1, V_2), if both its endpoints are either in V_1 or in V_2. An equivalent characterization of bipartite graphs is that they contain no odd-length cycles. In this section we show that any algorithm for testing whether a graph is bipartite has query complexity $\Omega(\sqrt{N})$. This lower bound stands in contrast to a result on testing bipartiteness which is described in [GGR98]. In [GGR98] a graph is assumed to be represented by its $N \times N$ adjacency matrix, and the distance between two graphs is defined to be the fraction of entries on which their respective adjacency matrices differ. Thus, in this model, a testing algorithm for a certain graph property should distinguish between the case in which the graph has the property, and the case in which one must add and/or remove at least $\frac{1}{2}\epsilon N^2$ edges in order to transform the graph into a graph that has the property. In [GGR98] there is an algorithm for testing bipartiteness in this model whose query complexity and running time are $\mathrm{poly}(1/\epsilon)$. Recall that in the current paper, graphs are represented by incident lists of length d and distance is measured as (twice) the number of edge modifications divided by dN (rather than by N^2).

Theorem 7.1 *Testing Bipartiteness with distance parameter 0.01 requires $\frac{1}{4} \cdot \sqrt{N}$ queries.*

Proof: We describe two families of degree-3 N-vertex graphs that are hard to distinguish by any algorithm which makes less than $\sqrt{N}/4$ queries: A typical member of one family is 0.01-far from being bipartite, whereas all members of the second family are bipartite graphs. Specifically, we fix any testing algorithm that makes less than $\sqrt{N}/4$ queries, and consider its decision when given a graph uniformly selected in one of these families. The indistinguishability claim implies that on the average, such an algorithm will accept the random input graph, with about the same probability regardless of the family it was selected from. But this contradicts the requirement from a testing algorithm, since it should accept every member of the second family with probability at least 2/3 while for almost all members of the second family it is allowed acceptance probability smaller than 1/3.

We start with the construction of both families: Let N be an even integer.[10]

1. The first family, denoted $\mathcal{G}_1^N$, consists of all degree-3 graphs that are composed of the union of a Hamiltonian cycle and a perfect matching. That is, there are N edges connecting the vertices in a cycle, and the other $N/2$ edges are a perfect matching.

2. The second family, denoted $\mathcal{G}_2^N$, is the same as the first *except* that the perfect matchings allowed are restricted as follows: the distance on the cycle between every two vertices that are connected by an perfect matching edge must be odd.

In both cases we assume that the edges incident to any vertex are labeled in the following fixed manner: Each cycle edge is labeled 1 in one endpoint and 2 in the other. This labeling forms an orientation of the cycle. The matching edges are labeled 3.

Clearly, all graphs in $\mathcal{G}_2^N$ are bipartite as all cycles in the graph are of even length. We next prove that almost all graphs in $\mathcal{G}_1^N$ are far from being bipartite. Afterwards, we show that a testing

[10] For odd N, every graph (in both families) contains one degree-0 vertex, and the rest of the vertices are connected as in the even case.

algorithm that performs less than $\alpha\sqrt{N}$ queries (for some constant $\alpha < 1$) is not able to distinguish between a graph chosen randomly from $\mathcal{G}_2^N$ (which is always bipartite) and a graph chosen randomly from $\mathcal{G}_1^N$ (which with high probability is far from bipartite).

Lemma 7.2 *With probability at least* $1 - \exp(-\Omega(N))$*, a graph chosen randomly in* $\mathcal{G}_1^N$ *is* 0.01*-far from the class of bipartite graphs.*

Proof: We fix a certain ordering of the vertices on the cycle and consider all possible partitions of the graph vertices into two sets. We say that an edge (u, v) is a *violating* edge with respect to a partition $(\mathrm{V}_1, \mathrm{V}_2)$ if for $i \in \{1, 2\}$ both u and v belong to the same V_i. We show that with high probability (over the choice of the matching edges) all such partitions have at least $\frac{1}{64}N$ violating edges (and since $d = 3$, this implies that the graph is ϵ-far from bipartite for $\epsilon = \frac{2\cdot(N/64)}{dN} = \frac{1}{96}$).

Consider a particular partition $(\mathrm{V}_1, \mathrm{V}_2)$ of V. We consider two cases:

1. There are at least $\frac{1}{64}N$ violating cycle edges with respect to $(\mathrm{V}_1, \mathrm{V}_2)$. In this case we are done no matter how the matching edges are chosen.

2. There are less than $\frac{1}{64}N$ violating cycle edges. In this case we show that with probability at least $1 - \exp(-\frac{7}{32}N)$, over the choice of the matching edges, there are at least $\frac{1}{64}N$ violating matching edges with respect to $(\mathrm{V}_1, \mathrm{V}_2)$.

   We first observe that a random matching can be constructed by selected at each step any *arbitrary* vertex that is yet unmatched, and matching it with another unmatched vertex that is selected uniformly. Thus, assume without loss of generality that $|\mathrm{V}_1| \geq N/2$ and consider the following process for choosing a random matching. Starting from $j = 1$, select an arbitrary vertex v in V_j, and match it with a randomly chosen unmatched vertex u. In case $u \in \mathrm{V}_j$, the edge (v, u) is a violating edge with respect to $(\mathrm{V}_1, \mathrm{V}_2)$. If the number of unmatched vertices in V_j is smaller than the number of unmatched vertices in the other side of the partition then in the next step switch sides (i.e., let $j \leftarrow 3 - j$).

   By definition of the process, we always try to match a vertex from the side having more unmatched vertices. Hence, at each step we create a violating edge with probability at least $\frac{1}{2}$ (independent of the past events), and so the probability that less than $\frac{1}{64}N$ violating edges are created (in the $N/2$ steps) is upper bounded by the probability that when tossing $N/2$ unbiased coins, less than $N/64$ turn out `heads`. The probability of the latter event is

   $$\sum_{i=0}^{(N/64)-1} \binom{N/2}{i} \cdot 2^{-N/2} \quad < \quad 2^{(H(2/64)+o(1))\cdot\frac{N}{2}} \cdot 2^{-N/2} \quad < \quad 2^{-0.3N} \tag{3}$$

   where $H(p) \stackrel{\text{def}}{=} -p\log p - (1-p)\log(1-p)$ is the (binary) entropy function, and the first inequality follows from the bound $\binom{n}{k} \leq 2^{nH(k/n)}$ (see [CT91, Page 284]).

Given the above, we upper bound the probability that there exists a partition with less than $N/64$ violating edges, by summing, over all possible partitions $(\mathrm{V}_1, \mathrm{V}_2)$, the probability that $(\mathrm{V}_1, \mathrm{V}_2)$ has less than $N/64$ violating edges. We group all possible partitions into two categories corresponding to the above two cases. The contribution of each partition of the first category (i.e., Case 1) to the sum is zero, since by definition each of these partitions has at least $\frac{1}{64}N$ violating cycle edges. The contribution of each partition of the second category is at most $\exp(-\frac{7}{32}N)$. We multiply the latter bound by the number of partitions of the second category. The number of such partitions is computed by observing that, for any fixed $i \leq N$, each partition which has i violating cycle edges is determined by the choice of those i violating edges. Thus there are $\sum_{i=0}^{(N/64)-1} \binom{N}{i} < 2^{0.2N}$ partitions with less than $\frac{1}{64}N$ violating cycle edges. (We use $H(1/64) + o(1) < 0.2$.) Thus, the probability that there exists a partition with less than $N/64$ violating edges is upper bounded by $2^{0.2N} \cdot 2^{-0.3N} = \exp(-\Omega(N))$, and the lemma follows. ■

We now turn to showing that a testing algorithm which performs less than $\alpha\sqrt{N}$ queries (for some constant $\alpha < 1$) is not able to distinguish between a graph chosen randomly from $\mathcal{G}_2^N$ and a graph chosen randomly from $\mathcal{G}_1^N$

Notation. Let $\mathcal{A}$ be an algorithm for testing bipartiteness using $\ell = \ell(N)$ queries. Namely, $\mathcal{A}$ is a (possibly probabilistic) mapping from *query-answer histories* $[(q_1, a_1), \ldots, (q_t, a_t)]$ to $q_{t+1} \in \mathrm{V} \times \{1, 2, 3\}$, for every $t < \ell$, and to $\{\mathsf{accept}, \mathsf{reject}\}$, for $t = \ell$. A query q_t is a pair (v_t, i_t), where $v_t \in \mathrm{V}$ and $i_t \in \{1, 2, 3\}$, and an answer a_t is simply a vertex $u_t \in \mathrm{V}$. We assume that the mapping is defined only on histories which are consistent with some graph. Any query-answer history of length $t-1$ can be used to define a *knowledge* graph, $\mathrm{G}_{t-1}^{\mathrm{kn}}$, at time $t-1$ (i.e., before the t^{th} query). The vertex set of $\mathrm{G}_{t-1}^{\mathrm{kn}}$ contains all vertices which appear in the history (either in queries or as answers), and its edge set contains the edges between $v_{t'}$ and $a_{t'}$ for all $t' < t$ (with the appropriate labelings – $i_{t'}$ at vertex $v_{t'}$). Thus, $\mathrm{G}_{t-1}^{\mathrm{kn}}$ is a labeled subgraph of the labeled graph tested by $\mathcal{A}$.

Overview. In what follows we describe two random processes, P_1 and P_2, which interact with an arbitrary algorithm $\mathcal{A}$, so that for $j \in \{1, 2\}$, P_j answers $\mathcal{A}$'s queries while constructing a random graph from $\mathcal{G}_j^N$. Thus, the interaction of P_j with $\mathcal{A}$ captures a (random) execution of $\mathcal{A}$ on a graph uniformly distributed in $\mathcal{G}_j^N$. (The fact that the input graph is randomly constructed "online" while the algorithm is making queries to it is immaterial; what is important is that the distribution over the graphs constructed is exactly uniform over $\mathcal{G}_j^N$.) For a fixed $\mathcal{A}$ that uses ℓ queries, and for $j \in \{1, 2\}$, let $D_j^{\mathcal{A}}$ denote the distribution on query-answer histories of length ℓ induced by the interaction of $\mathcal{A}$ and P_j. We show (below) that for any $\mathcal{A}$ that uses $\ell \leq \alpha\sqrt{N}$ queries, the statistical difference between $D_1^{\mathcal{A}}$ and $D_2^{\mathcal{A}}$ is at most $4\alpha^2$, where the statistical difference between distributions D_1 and D_2 is defined as

$$\frac{1}{2} \cdot \sum_{\alpha} |\mathrm{Prob}[D_1 = \alpha] - \mathrm{Prob}[D_2 = \alpha]| \;=\; \max_{f:\{0,1\}^* \mapsto \{0,1\}} |\mathrm{Prob}[f(D_1) = 1] - \mathrm{Prob}[f(D_2) = 1]| \quad (4)$$

In what follows we first define the two processes and prove that they in fact induce the desired (uniform) distribution over the respective classes of graphs. We then prove the bound mentioned above on the statistical difference between $D_1^{\mathcal{A}}$ and $D_2^{\mathcal{A}}$, and show that Theorem 7.1 follows by combining this bound with Lemma 7.2.

We start by defining P_1. The process has two stages. In the first stage, which goes on as long as the algorithm performs queries, the exact position of the vertices on the cycle is undetermined. However, each vertex that is introduced into the knowledge graph of the algorithm, following some query, is assigned the parity of its future position on the cycle (but this bit is NOT given to $\mathcal{A}$). That is, we think of the N positions on the cycle as being numbered from 0 to $N-1$, and a vertex which is assigned even (resp. odd) parity, will be allowed to be positioned only in even (resp. odd) cycle positions in the second stage. Thus, in this stage, the process essentially maintains the knowledge graph (which is extended according to the query-answer pairs), and keeps one additional bit per vertex. Observe that by our convention on the labeling of the edges, the knowledge graph maintained during the first stage can be viewed as "floating" (cycle) sections some of which are connected by arcs (the matching edges). In the second stage, all vertices in the final knowledge graph are positioned on the cycle randomly in a way that is consistent with the position-parity of the vertices, and so the knowledge graph edges that are labeled 1 or 2 coincide with cycle edges. Thus these sections "stop floating" and are restricted to fixed positions. Finally, all vertices that do not belong to the knowledge graph are randomly positioned on the remaining cycle positions and all unmatched vertices are randomly matched.

First Stage of P_1: Starting from $t = 1$, for each query $q_t = (v_t, i_t)$ of $\mathcal{A}$, process P_1 proceeds as follows:

1. If v_t belongs to $\mathrm{G}^{\mathrm{kn}}_{t-1}$ then there are three cases:
   (a) This edge already exists in the knowledge graph (i.e., there exists an edge (v_t, u) in $\mathrm{G}^{\mathrm{kn}}_{t-1}$ and this edge is labeled i_t at the endpoint v_t). In this case P_1 answers "u" (and the knowledge graph remains unchanged).
   (b) $i_t = 3$ and v_t is unmatched in $\mathrm{G}^{\mathrm{kn}}_{t-1}$ (i.e., there is no edge $(v_t, \cdot)$ in $\mathrm{G}^{\mathrm{kn}}_{t-1}$ that is labeled 3). In this case P_1 selects a random unmatched vertex $u \in \mathrm{V}$ (where u may belong to $\mathrm{G}^{\mathrm{kn}}_{t-1}$) and answers "$u$". If u did not belong to $\mathrm{G}^{\mathrm{kn}}_{t-1}$, then it is assigned a position-parity in the following manner: Let n_e be the number of vertices in $\mathrm{G}^{\mathrm{kn}}_{t-1}$ that were assigned even parity, and let n_o be the number that where assigned odd parity. Then u is assigned even parity with probability $\frac{(N/2)-n_e}{N-(n_e+n_o)}$ and odd parity otherwise. In any case, the edge (v_t, u) is added to the knowledge graph (with label 3).
   (c) $i_t \in \{1, 2\}$ and there is no edge incident to v_t in $\mathrm{G}^{\mathrm{kn}}_{t-1}$ which is labeled i_t (at v_t). Suppose, without loss of generality, that $i_t = 1$ and v_t has even parity. Let $X_{o,2}$ be the set of vertices in $\mathrm{G}^{\mathrm{kn}}_{t-1}$ which have odd parity, and do not have an incident edge labeled 2. Let $n_{o,2} \stackrel{\mathrm{def}}{=} |\mathrm{X}_{o,2}|$. Then P_1 first flips a coin with bias $\frac{n_{o,2}}{(N/2)-n_o+n_{o,2}}$ to decide whether to select a vertex in $X_{o,2}$. If so, it uniformly selects a vertex in $X_{o,2}$. Otherwise, it uniformly selects a vertex not in $\mathrm{G}^{\mathrm{kn}}_{t-1}$. In either case, let the selected vertex be u. Then the process answers "u", and if u does not belong to $\mathrm{G}^{\mathrm{kn}}_{t-1}$, it is assigned odd parity (i.e., parity opposite to v_t). In either case, the edge (v_t, u) is added to the knowledge graph (with label i_t at v_t).
2. If v_t does not belong to $\mathrm{G}^{\mathrm{kn}}_{t-1}$, process P_1 first assigns v_t parity as described in (1b) above, adds v_t to the knowledge graph, and next answers the query as in (1).

Second Stage of P_1: After all queries are answered, do the following:

1. Among all possible ways to embed $\mathrm{G}^{\mathrm{kn}}_{\ell}$ on the cycle, select one uniformly, where a possible embedding of $\mathrm{G}^{\mathrm{kn}}_{\ell}$ on the cycle must satisfy the following conditions.
   (a) Every vertex is assigned a cycle position (i.e., an integer in $\{0, \ldots, N-1\}$) with parity matching the vertex's parity bit.
   (b) Vertices connected by a cycle edge in $\mathrm{G}^{\mathrm{kn}}_{\ell}$ are assigned adjacent positions on the cycle. Furthermore, if v is assigned position j on the cycle, and v has an edge labeled "1" connecting it to u in $\mathrm{G}^{\mathrm{kn}}_{\ell}$, then u must be assigned position $(j+1) \pmod N$).
2. Next, randomly position all other vertices on the cycle,
3. Finally, match all unmatched vertices randomly.

Process P_2 is the same as P_1, except when randomly matching vertices in Step (1b) of the first stage and Step (3) of the second. Whereas process P_1 matches vertices at random (regardless of their position-parity), process P_2 may match two vertices only if they have opposite position-parity. The modification to the second stage of P_2 is self-evident (i.e., in Step (3) we randomly match the even-parity vertices with the odd-parity vertices). We also modify Step (1b) of the first stage – when choosing a vertex to match v_t, process P_2 only considers vertices in $\mathrm{G}^{\mathrm{kn}}_{t-1}$ that have opposite parity of v_t. Without loss of generality, assume v_t has even parity. Let $X_{o,3}$ be the set of vertices in $\mathrm{G}^{\mathrm{kn}}_{t-1}$ that have odd parity, and do not have an incident edge labeled 3. Let $n_{o,3} \stackrel{\mathrm{def}}{=} |\mathrm{X}_{o,3}|$. Then P_2 first flips a coin with bias $\frac{n_{o,3}}{(N/2)-n_o+n_{o,3}}$ to decide if to select a vertex in $X_{o,3}$. If so, it uniformly selects a vertex in $X_{o,3}$. Otherwise, it uniformly selects a vertex not in $\mathrm{G}^{\mathrm{kn}}_{t-1}$. The rest of the process, and in particular the assignment of parity to new vertices (i.e., Step (2)), remains unchanged.

We first show that the above two processes indeed generate a uniformly distributed graph in the corresponding family.

Lemma 7.3 *For every algorithm $\mathcal{A}$ and for each $j \in \{1,2\}$, the process P_j, when interacting with $\mathcal{A}$, uniformly generates graphs in $\mathcal{G}_j^N$.*

Proof: We'll prove this by induction on the number of queries, ℓ, that $\mathcal{A}$ performs. Since every probabilistic algorithm can be viewed as a distribution on deterministic algorithms, it suffices to prove the lemma for any deterministic algorithm $\mathcal{A}$. Also note that the (accept/reject) output of the algorithm is irrelevant to the claim and hence we view the algorithm only as a mapping from query-answer histories to queries.

The base case, $\ell = 0$ is clear since the knowledge graph is empty, and so in Stage 2 process P_j generates a random graph in $\mathcal{G}_j^N$ from scratch. Assuming the claim is true for $\ell - 1$, we prove it for ℓ. Let $\mathcal{A}$ be an algorithm that performs ℓ queries, and let $\mathcal{A}'$ be the algorithm defined by stopping $\mathcal{A}$ before it asks the ℓ^{th} query. By the induction hypothesis, we know that P_j when interacting with $\mathcal{A}'$ uniformly generates graphs in $\mathcal{G}_j^N$. We thus need to show that the same will be true if the second stage of P_j is performed following the ℓ^{th} query of $\mathcal{A}$. We need to consider the following cases, depending on the query $q_\ell = (v_\ell, i_\ell)$ of $\mathcal{A}$. We may assume without loss of generality that the answer to the query cannot be derived from the algorithm's knowledge graph, since this would be equivalent to asking no query (in which case the knowledge graph does not change and so the distribution on P_j's output after ℓ steps is identical to its output after $\ell - 1$ steps).

1. $i_\ell = 3$, and v_ℓ belongs to the algorithm's knowledge graph, $\mathrm{G}^{\mathrm{kn}}_{\ell-1}$. Consider first the process P_1 (when interacting with $\mathcal{A}'$). The probability that P_1 matches v_ℓ (in the second stage) to any vertex (either in $\mathrm{G}^{\mathrm{kn}}_{\ell-1}$ or not) is clearly independent of the exact ordering of the vertices on the cycle. Hence, by first answering this query and then performing the second stage of P_j we are only changing the order in which the final graph is constructed.

   In the case of P_2, the probability that P_2 matches v_ℓ to any vertex is still independent of the exact ordering of the vertices on the cycle, but it does depend on the parity of the vertices. In particular, assume without loss of generality that v_ℓ has even parity. Then in any possible matching done in the second stage following the interaction with $\mathcal{A}'$, the only vertices in $\mathrm{G}^{\mathrm{kn}}_{\ell-1}$ that v_ℓ can be matched to are vertices in $\mathrm{X}_{o,3}$. (Recall that $\mathrm{X}_{o,3}$ is the set of vertices assigned odd parity that do not have an incident edge labeled 3.) On the other hand, in any possible embedding of the vertices on the cycle, there are exactly $(N/2) - n_o$ vertices not in

$\mathrm{G}^{\mathrm{kn}}_{\ell-1}$ that have odd parity and thus may be matched to v_ℓ. (Recall that n_o is the number of vertices assigned odd parity.) This implies that v_ℓ is matched to some vertex in $\mathrm{X}_{o,3}$ with probability $\frac{|\mathrm{X}_{o,3}|}{|\mathrm{X}_{o,3}|+(N/2)-n_o}$, and to some vertex not in $\mathrm{G}^{\mathrm{kn}}_{\ell-1}$, with probability $\frac{(N/2)-n_o}{|\mathrm{X}_{o,3}|+(N/2)-n_o}$. Furthermore, conditioned on the event that v_ℓ is matched to a vertex in $\mathrm{X}_{o,3}$, this vertex is distributed uniformly in $\mathrm{X}_{o,3}$. Similarly, conditioned on the event that it is matched to a vertex not in $\mathrm{G}^{\mathrm{kn}}_{\ell-1}$, this vertex is uniformly distributed among vertices not in $\mathrm{G}^{\mathrm{kn}}_{\ell-1}$. But these probabilities are exactly as defined in Step (1b) of P_2.

Therefore, for both processes the induction step holds in this case.

2. $i_\ell = 3$, and v_ℓ does not belong to $\mathrm{G}^{\mathrm{kn}}_{\ell-1}$. This case is reduced to the previous one, provided that the parity of v_ℓ is chosen with the correct probability. In the second stage each vertex is assigned parity at random according to the proportion of missing vertices (with this parity). This is exactly the assignment rule of Step (2) in the first stage.

3. $i_\ell \in \{1,2\}$, and v_ℓ belongs to $\mathrm{G}^{\mathrm{kn}}_{\ell-1}$. Assume, without loss of generality, that $i_\ell = 1$ and v_ℓ has even parity. Clearly, in any embedding of $\mathrm{G}^{\mathrm{kn}}_{\ell-1}$ on the cycle, v_ℓ can be adjacent to a vertex u in $\mathrm{G}^{\mathrm{kn}}_{\ell-1}$ only if u belongs to $\mathrm{X}_{o,2}$ (as defined in the process). It is also clear that conditioned on the event that it is adjacent to a vertex in $\mathrm{G}^{\mathrm{kn}}_{\ell-1}$, this vertex is uniformly distributed in $\mathrm{X}_{o,2}$ (and similarly if it is not in the graph). Finally, since there should be exactly $N/2$ odd-parity vertices, and the total number of odd-parity vertices in $\mathrm{G}^{\mathrm{kn}}_{\ell-1}$ is n_o, the number of odd-parity vertices not in $\mathrm{G}^{\mathrm{kn}}_{\ell-1}$ (in any ordering of the vertices on the cycle) is $(N/2) - n_o$. Thus the probability that v_ℓ is adjacent to some $u \in X_{o,2}$ is $\frac{|\mathrm{X}_{o,2}|}{|\mathrm{X}_{o,2}|+(N/2)-n_o}$, and the probability that it is adjacent to some vertex outside the knowledge graph is $\frac{(N/2)-n_o}{|\mathrm{X}_{o,2}|+(N/2)-n_o}$, which is exactly as defined by the process. Hence the induction step holds in this case.

4. $i_\ell \in \{1,2\}$, and v_ℓ does not belong to the knowledge graph. This case is reduced to the previous one, provided that the parity of v_ℓ is chosen with the correct probability. The validity of the condition was already established in Case 2.

■

Finally we bound the statistical difference between the distributions of query-answer sequences induced by the interaction of $\mathcal{A}$ with the two processes. Recall that $D_j^{\mathcal{A}}$ denotes the distribution on query-answer histories (of length ℓ) induced by the interaction of $\mathcal{A}$ and P_j.

Lemma 7.4 *Let $\alpha < \frac{1}{2}$, $\ell \leq \alpha\sqrt{N}$ and $N \geq 8\ell$. Then, for every algorithm $\mathcal{A}$ that asks ℓ queries, the statistical distance between $D_1^{\mathcal{A}}$ and $D_2^{\mathcal{A}}$ is at most $4\alpha^2$. Furthermore, for both distributions, with probability at least $1-4\alpha^2$ the knowledge graph at time of termination of $\mathcal{A}$ contains no cycles.*

Proof: We assume without loss of generality that $\mathcal{A}$ does not ask queries whose answer can be derived from its knowledge graph, since those give it no new information. Under this assumption, we first prove the following.

Claim: Both in $D_1^{\mathcal{A}}$ and in $D_2^{\mathcal{A}}$, the total probability mass assigned to query-answer histories in which for some $t \leq \ell$ a vertex in $\mathrm{G}^{\mathrm{kn}}_{t-1}$ is returned as an answer to the t^{th} query is at most $4\alpha^2$.

Proof: We show that for every t the probability that the t^{th} answer is in $\mathrm{G}^{\mathrm{kn}}_{t-1}$ (i.e., there exist $t' < t$ such that $a_t = v_{t'}$ or $a_t = a_{t'}$) is at most $8(t-1)/N$. The claim directly follows (as described below). Fixing t, there are two cases in which the event $a_t \in \mathrm{G}^{\mathrm{kn}}_{t-1}$ might occur.

1. $i_t = 3$, and v_t is matched to a vertex in the knowledge graph G_{t-1}^{kn}. Since the number of vertices in G_{t-1}^{kn} is at most $2(t-1)$, this event occurs with probability at most $\frac{2(t-1)}{N-2(t-1)}$ when the process is P_1, and at most $\frac{2(t-1)}{(N/2)-2(t-1)}$ when the process is P_2.
2. $i_t \in \{1,2\}$ and a_t is chosen in G_{t-1}^{kn}. According to both processes this event occurs with probability less than $\frac{2(t-1)}{(N/2)-2(t-1)}$

Thus, in each of the cases, the probability that $a_t \in G_{t-1}^{kn}$ is at most $\frac{2(t-1)}{(N/2)-2(t-1)} < \frac{8(t-1)}{N}$ (as $N \geq 8t$). The probability that such an event occurs in any sequence of $\alpha\sqrt{N}$ queries, is at most $\sum_{t=1}^{\alpha\sqrt{N}} \frac{8(t-1)}{N} < 4\alpha^2$. $\square$

In particular, the Claim implies that with probability at least $1-4\alpha^2$, the knowledge graph of $\mathcal{A}$ contains no cycles. Observe that whenever any of these processes returns as an answer a vertex not in the current knowledge graph, this vertex is uniformly distributed among the vertices not in that graph. Since $\mathcal{A}$'s queries only depend on the preceding query-answer history, it follows that conditioned on the process not returning vertices in the current knowledge graph, the answers are distributed obliviously of the identity of the process. Lemma 7.4 follows. ■

Finishing up the proof of Theorem 7.1: We use the above three lemmas to show that $\mathcal{A}$ cannot be a tester for bipartiteness with distance parameter $\epsilon = 0.01$, and Theorem 7.1 follows. Setting $\alpha = 1/4$ and using Lemma 7.4, it follows that for any algorithm $\mathcal{A}$ which makes $\alpha\sqrt{N}$ queries, the statistical difference between $D_1^{\mathcal{A}}$ and $D_2^{\mathcal{A}}$ is at most $4\cdot(1/4)^2 = (1/4)$. By Lemma 7.3, $D_j^{\mathcal{A}}$ is distributed identically to the query-answer sequences in an execution of $\mathcal{A}$ on a uniformly distributed graph in $\mathcal{G}_j^N$. We start by observing that since all graphs in $\mathcal{G}_2^N$ are bipartite (and $\mathcal{A}$ is a bipartiteness tester),

$$\text{Prob}[\mathcal{A}(D_2^{\mathcal{A}}) = \mathsf{accept}] \;\geq\; \frac{2}{3} \tag{5}$$

Recall that $\mathcal{A}(D_2^{\mathcal{A}})$ denotes the final decision of $\mathcal{A}$ after interacting with process P_2, and this distribution is identical to the one in an execution of $\mathcal{A}$ on a uniformly distributed graph in $\mathcal{G}_2^N$. Combining Eq. (5) with the bound of $1/4$ on the statistical difference between $D_1^{\mathcal{A}}$ and $D_2^{\mathcal{A}}$, we have

$$\text{Prob}[\mathcal{A}(D_1^{\mathcal{A}}) = \mathsf{accept}] \;\geq\; \frac{2}{3} - \frac{1}{4} \;>\; 0.4 \tag{6}$$

But, by Lemma 7.2, more than 99% of the graphs in $\mathcal{G}_1^N$ are 0.01-far from bipartite and thus must be rejected. Thus, $\text{Prob}[\mathcal{A}(D_1^{\mathcal{A}}) = \mathsf{accept}] \;\leq 0.99\cdot\frac{1}{3} + 0.01 \;<\; 0.35$, in contradiction to Eq. (6). ■

Proof of Proposition 4.3: Consider either classes described in the proof of Theorem 7.1: A testing algorithm for cycle-freeness must reject a random graph in the class with probability 2/3 since such a graph is far from cycle free. However, if the algorithm asks only $o(\sqrt{N})$ queries then the probability it actually observes a cycle is negligible. Fixing any such sequence of coins where no cycle is detected, we observe that the algorithm will also reject a graph that consists only of the (partial) forest it has observed. Thus the algorithm has a non-zero rejecting probability on some cycle-free graphs. ■

7.2 Testing Whether a Graph is an Expander

The *neighbor set* of a set S of vertices of a graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$, denoted $\Gamma(\mathrm{S})$, is defined as follows:

$$\Gamma(\mathrm{S}) \stackrel{\mathrm{def}}{=} \mathrm{S} \cup \{u : (v,u) \in \mathrm{E}, v \in \mathrm{S}\}$$

A graph on N vertices is an (N, γ, δ)–expander if for every subset S of the vertices that has size at most γN, $|\Gamma(\mathrm{S})| \geq \delta|\mathrm{S}|$. Let us set $\gamma = \frac{1}{4}$ and $\delta = 1.2$, and simply refer to an $(N, \frac{1}{4}, 1.2)$–expander, as an expander. Here we show that

Theorem 7.5 *Testing whether a graph is an expander, with distance parameter $\epsilon = 0.01$, requires $\frac{1}{5} \cdot \sqrt{N}$ queries.*

Proof: Similarly to the lower bound for testing bipartiteness, we first describe two families of graphs where with extremely high probability, a graph chosen randomly in the first family is an expander, and every graph in the second family is far from being an expander. We then describe two processes which interact with a testing algorithm while constructing a random graph in one of the families, and show that the distributions induced on the query-answer sequences are very similar. For simplicity we assume that $N \equiv 0 \pmod 8$.

Let $d = 3$. It is well known (see [Pin73], [MR95, Thm. 5.6]) that if we randomly construct a graph by choosing d random perfect matchings to define its edge set, then with probability $1 - \exp(-\Omega(N))$, the resulting graph is an expander. The first family, $\mathcal{G}_1^N$, consists of all possible resulting graphs. A graph in the second family, $\mathcal{G}_2^N$, is constructed by first randomly partitioning the vertex set into 4 equal size subsets, and then choosing d random matchings inside each subset. Thus the four subsets are disconnected. Clearly, every graph in this family is $\frac{1}{60}$-far from being an expander, since in order to transform it into an expander we must connect each of the four subsets to at least $N/20$ vertices outside the subset. In both processes, each edge in the graph has the same label at both endpoints (i.e., corresponding to the index of the perfect matching to which the edge belongs).

The process P_1 for constructing a random graph in $\mathcal{G}_1^N$, while interacting with an algorithm $\mathcal{A}$, is completely straightforward. Let $q_t = (v_t, i_t)$ be $\mathcal{A}$'s t^{th} query. If the answer a_t is determined by the current knowledge graph, $\mathrm{G}_{t-1}^{\mathrm{kn}}$, then P_1 answers accordingly. Otherwise, it selects a random vertex u which does not have an incident edge labeled i_t, answers "u", and adds the edge (v, u) to the knowledge graph. (In case u does not belong to $\mathrm{G}_{t-1}^{\mathrm{kn}}$ it is of course added in.) When the interaction with $\mathcal{A}$ ends, P_1 randomly completes all d matchings.

Process P_2 is somewhat more complex. It maintains four subsets of vertices and coordinates its choice of matching edges with these growing subsets.

- Whenever algorithm $\mathcal{A}$ makes a query of the form (v, i) where v is not in the current knowledge graph, P_2 assigns it a *subset-id* in $\{1, 2, 3, 4\}$ with probability proportional to the number of vertices missing in each subset (P_2 starts with all subsets being empty). Specifically, let n_s be the number of vertices with subset-id s in the current knowledge graph, for $s = 1, 2, 3, 4$. Then the new vertex is assigned subset-id s with probability $\frac{(N/4) - n_s}{N - (n_1 + n_2 + n_3 + n_4)}$. The query is then processed as follows.

- To answer a query (v, i) when v is already in the current knowledge graph, P_2 matches it to either a vertex already assigned to the same subset as v or to an unassigned vertex. Specifically, suppose that v is already assigned to the s^{th} subset, and let $X_{s,i}$ denote the set of vertices which are assigned to the s^{th} subset but do not have an incident edge labeled i. Then with probability $\frac{|X_{s,i}|-1}{(N/4-n_s)+(|X_{s,i}|-1)}$ process P_2 matches v to a uniformly selected vertex, u, in $X_{s,i} \setminus \{v\}$. Otherwise, P_2 matches v to a uniformly selected vertex, u, which does not belong to the current knowledge graph, and assigns u to the s^{th} subset. In both cases P_2 answers with the selected vertex u, and the knowledge graph is augmented with the edge (v, u) labeled i.

It is easy to verify, using arguments similar to those in the proof of Lemma 7.3, that for both processes the distribution on the generated graphs is uniform in the respective graph family. Similarly to the bipartite lower bound, it remains to show that for any (not too long) query-answer history, the probability that we get an answer a_t which is a vertex in the knowledge graph (and not a uniformly distributed new vertex) is small. But this is easy to see. In the case of P_1, such a vertex is selected following the t^{th} query, with probability at most $\frac{2t}{N-2t}$. In the case of P_2, such a vertex is selected with probability at most $\frac{2t}{(N/4)-2t}$. The probability that such an event occurs in any sequence of $\alpha\sqrt{N}$ queries, is at most $\sum_{t=1}^{\alpha\sqrt{N}} \frac{8t}{N-8t}$ which is at most $8\alpha^2$, for every $N \geq 256$. ■

7.3 Vertex Cover and Dominating Set

It should come with little surprise that we cannot efficiently test graph properties which are related to hard-to-approximate problems on bounded-degree graphs.

Consider, for example, the class $\mathcal{C}_d^\rho$ of graphs with maximum degree d having a vertex cover of size ρN, for some constant $\rho > 0$. (A *vertex cover* of a graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ is a set $\mathrm{C} \subseteq \mathrm{V}$ so that every edge $e \in \mathrm{E}$ is incident to some vertex in C.) Let $\mathcal{A}$ be a property tester for $\mathcal{C}_d^\rho$ as in Definition 2.1. Namely, on input ϵ and d, and access to a graph with degree bounded by d, $\mathcal{A}$ accepts (with high probability) any graph in $\mathcal{C}_d^\rho$ but rejects (w.h.p.) any N-vertex graph (of degree $\leq d$) which requires modification of ϵdN edges in order to be in $\mathcal{C}_d^\rho$. We observe that it suffices to consider the number of edges omitted in the modification process, and that the number of omitted edges can be related to an increase in the vertex cover. Specifically,

Claim 7.6 *Suppose that $\mathcal{A}$ is a property tester for $\mathcal{C}_d^\rho$. Then, on distance parameter ϵ, algorithm $\mathcal{A}$ distinguishes between N-vertex graphs (of degree at most d) having a vertex cover of size $\rho \cdot N$ and N-vertex graphs (of degree at most d) having no vertex cover of size $\left(\rho + \frac{1}{2}\epsilon d\right) \cdot N$.*

Since distinguishing the two cases is NP-Hard for some constants d, ϵ and ρ [ALM$^+$98, PY91], we cannot expect $\mathcal{A}$ to have "reasonable" (e.g., polynomial in N) time-complexity.

Proof: By definition, the former graphs are in $\mathcal{C}_d^\rho$. It remains to see that any N-vertex graph having no vertex cover of size $\left(\rho + \frac{1}{2}\epsilon d\right) \cdot N$ requires the modification of more than $\frac{1}{2}\epsilon dN$ edges in order to put it in $\mathcal{C}_d^\rho$. Suppose that it suffices to omit m edges from a graph G in order to obtain a graph G′ in $\mathcal{C}_d^\rho$ (we don't care if edges were added in the process).[11] Then taking the ρN-vertex-cover of G′ and at most one endpoint of each of the m edges omitted from G, results in a vertex cover of G having size at most $\rho N + m$. Thus, we have $m > \frac{1}{2}\epsilon dN$. ■

[11] Actually, without loss of generality we may assume that no edges were added as they only make the task of covering harder.

Next, we consider the class $\mathcal{D}_d^\rho$ of graphs with maximum degree d having a dominating set of size ρN. (A *dominating set* of a graph $G = (V, E)$ is a set $D \subseteq V$ so that every vertex in V is either in D or adjacent to some vertex in D.) We observe that it suffices to consider the number of edges which need to be added to put the graph in $\mathcal{D}_d^\rho$. Specifically,

Claim 7.7 *Suppose that $\mathcal{A}$ is a property tester for $\mathcal{D}_d^\rho$. Then, on distance parameter ϵ, algorithm $\mathcal{A}$ distinguishes between N-vertex graphs (of degree at most d) having a dominating set of size $\rho \cdot N$ and N-vertex graphs (of degree at most d) having no dominating set of size $\left(\rho + \frac{1}{2}\epsilon d\right) \cdot N$.*

Again, since distinguishing the two cases is NP-Hard for some constants d, ϵ and ρ [ALM$^+$98, PY91], we cannot expect $\mathcal{A}$ to have "reasonable" time-complexity.

Proof: Again, the former graphs are in $\mathcal{D}_d^\rho$, and it remains to see that N-vertex graphs having no dominating set of size $\left(\rho + \frac{1}{2}\epsilon d\right) \cdot N$ require the modification of more than $\frac{1}{2}\epsilon dN$ edges in order to put them in $\mathcal{D}_d^\rho$. Suppose that it suffices to add m edges to a graph G, with maximum degree d, in order to obtain a graph G$'$ in $\mathcal{D}_d^\rho$ (we don't care if edges were omitted in the process).[12] Let S$'$ be a dominating set of size ρN of G$'$. Then S$'$ dominates all but at most m vertices in G (i.e., all vertices dominated in G$'$ except for those which are dominated due to the edges added to G). Adding these vertices to S$'$ we obtain a dominating set of size $|S'| + m$ of G, and thus $m > \frac{1}{2}\epsilon dN$. ■

We conclude by proving a lower bound on the query complexity of testers for the Vertex Cover Property, $\mathcal{C}_d^\rho$. Specifically,

Proposition 7.8 *Let $d = 3$, $\rho = 0.5$ and $\epsilon = 0.005$. Then testing whether a 3-regular N-vertex graph belongs to $\mathcal{C}_d^\rho$ or is ϵ-far from it requires $\Omega(\sqrt{N})$ queries.*

Proof: We use the families $\mathcal{G}_1^N$ and $\mathcal{G}_2^N$ presented in Subsection 7.1. By combining Lemmas 7.3 and 7.4, an algorithm which makes $o(\sqrt{N})$ queries can not distinguish graphs uniformly chosen in $\mathcal{G}_1^N$ from graphs uniformly chosen in $\mathcal{G}_2^N$. It is easy to see that graphs in $\mathcal{G}_2^N$ have a vertex cover of size $N/2$ (e.g., all vertices with odd locations on the cycle). It remains to show that, with very high probability, a graph chosen uniformly in $\mathcal{G}_1^N$ has no vertex cover of size $0.51 \cdot N$. By Claim 7.6, it follows that an algorithm which makes $o(\sqrt{N})$ queries cannot test $\mathcal{C}_3^{0.5}$ on distance parameter $2 \cdot 0.01/3 > 0.005$.

As in the proof of Lemma 7.2, we fix an ordering of the vertices on the cycle, and consider the probability over the random choice of a perfect matching, that the resulting graph has a vertex cover of size $0.51 \cdot N$. We observe that such a potential vertex cover, denoted C, must cover all cycle edges. This allows us to upper bound the number of potential vertex covers (of size $0.51 \cdot N$) which we should consider. In such a vertex cover, C, each vertex not in C must be adjacent (on the cycle) to vertices in C. Let $w_1, \ldots, w_{0.51 \cdot N}$ be the vertices in a generic cover, ordered according to their relative position on the cycle. Then a specific cover C is determined by whether w_1 is the first vertex on the cycle or the second, and by which of the vertices among $w_1, \ldots, w_{0.51 \cdot N}$ are followed by a vertex not in C. Thus, the number of possible sets of size $0.51N$ which cover the cycle edges is at most

$$2 \cdot \binom{0.51N}{0.49N} \le 2^{H(49/51) \cdot 0.51N + 1} < 2^{0.122N}$$

[12] Here we cannot assume that the modification of G into G$'$ consists only of the addition of edges, since we may be forced to omit edges in order to satisfy the degree bound. Nevertheless, this fact does not effect the proof.

where recall that $H(p) \stackrel{\text{def}}{=} -p\log p - (1-p)\log(1-p)$, and that the first inequality follows from the bound $\binom{n}{k} \leq 2^{nH(k/n)}$ (see [CT91, Page 284]). On the other hand, for every fixed C as above, the probability that C covers the matching edges is upper bounded by the probability that the first $0.4N$ edges selected have each an endpoint in C. Consider the selection of the $i+1^{\text{st}}$ edge. The probability that both its end-points are not in C is at least $(\frac{0.49N-i}{N-2i})^2$ (using the hypothesis that all prior edges had an end-point in C). Define $f(x) \stackrel{\text{def}}{=} \frac{0.49-x}{1-2x}$, and observe that this function is monotonically decreasing in $[0, 0.5]$. Thus, the probability that C covers the matching edges is upper bounded by

$$\prod_{i=0}^{0.4N}\left(1-f(i/N)^2\right) \;<\; \left(1-f(0.4)^2\right)^{0.4N} \;<\; 2^{-0.131N}$$

We conclude that the probability that a graph chosen uniformly in $\mathcal{G}_1^N$ has a vertex cover of size $0.51 \cdot N$ is smaller that $2^{0.122N} \cdot 2^{-0.131N} = \exp(-\Omega(N))$. The proposition follows. ■

Acknowledgments

We thank Yefim Dinitz, Shimon Even, and David Karger for helpful discussions. We are most grateful to an anonymous referee for very useful comments.

References

[ALM$^+$98] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and intractability of approximation problems. *Journal of the Association for Computing Machinery*, 45(3):501–555, 1998.

[AS98] S. Arora and S. Safra. Probabilistic checkable proofs: A new characterization of NP. *Journal of the Association for Computing Machinery*, 45(1):70–122, 1998.

[Ben95] A. Benczur. A representation of cuts within 6/5 times the edge connectivity with applications. In *Proceedings of the Thirty-Sixth Annual Symposium on Foundations of Computer Science*, pages 92–101, 1995.

[BFL91] L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1(1):3–40, 1991.

[BFLS91] L. Babai, L. Fortnow, L. Levin, and M. Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*, pages 21–31, 1991.

[BGS98] M. Bellare, O. Goldreich, and M. Sudan. Free bits, PCPs and non-approximability – towards tight results. *SIAM Journal on Computing*, 27(3):804–915, 1998.

[BLR93] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47:549–595, 1993.

[CT91] T. Cover and J. Thomas. *Elements of Information Theory*. Wiley, 1991.

[DKL76] E. A. Dinic, A. V. Karazanov, and M. V. Lomonosov. On the structure of the system of minimum edge cuts in a graph. *Studies in Discrete Optimizations*, pages 290–306, 1976. In Russian.

[DW98] Y. Dinitz and J. Westbrook. Maintaining the classes of 4-edge-connectivity in a graph on-line. *Algorithmica*, 20(3):242–276, 1998.

[Eve79] S. Even. *Graph Algorithms*. Computer Science Press, 1979.

[FGL$^+$96] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Approximating clique is almost NP-complete. *Journal of the Association for Computing Machinery*, 43(2):268–292, 1996.

[Gab91] H. Gabow. Applications of a poset representation to edge connectivity and graph rigidity. In *Proceedings of the Thirty-Second Annual Symposium on Foundations of Computer Science*, pages 812–821, 1991.

[Gab95] H. Gabow. A matroid approach to finding edge connectivity and packing arborescences. *Journal of Computer and System Sciences*, 50(2):259–273, 1995.

[GGR98] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the Association for Computing Machinery*, 45(4):653–750, 1998. An extended abstract appeared in the proceedings of FOCS96.

[GLR+91] P. Gemmell, R. Lipton, R. Rubinfeld, M. Sudan, and A. Wigderson. Self-testing/correcting for polynomials and for approximate functions. In *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*, pages 32–42, 1991.

[GR97] O. Goldreich and D. Ron. Property testing in bounded degree graphs. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing*, pages 406–415, 1997.

[GR99a] O. Goldreich and D. Ron. Property testing in bounded degree graphs. Available from `http://www.eng.tau.ac.il/~danar`, 1999.

[GR99b] O. Goldreich and D. Ron. A sublinear bipartite tester for bounded degree graphs. *Combinatorica*, 19(3):335–373, 1999.

[Hås96] J. Håstad. Testing of the long code and hardness for clique. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 11–19, 1996. To appear in *Acta Mathematica*.

[Kar93] D. Karger. Global min-cuts in $\mathcal{RNC}$ and other ramifications of a simple mincut algorithm. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 21–30, 1993.

[Kar95] D. Karger. *Random Sampling in Graph Optimization Problems*. PhD thesis, Stanford University, 1995. Available from `http://theory.lcs.mit.edu/~karger`.

[Kar96] D. Karger. Minimum cuts in near-linear time. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 56–63, 1996.

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[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.

[NI97] H. Nagamochi and T. Ibaraki. Deterministic $\tilde{O}(nm)$ time edge-splitting in undirected graphs. *Journal of Combinatorial Optimization*, 1(1):5–46, 1997.

[Pin73] M. Pinsker. On the complexity of a concentrator. In *7th International Teletraffic Conference*, pages 318/1–318/4, 1973.

[PR99] M. Parnas and D. Ron. Testing the diameter of graphs. In *Proceedings of Random99*, pages 85–96, 1999.

[PY91] C.H. Papadimitriou and M. Yanakakis. Optimization, approximation and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.

[RS96] R. Rubinfeld and M. Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, 1996.

[WN87] T. Watanabe and A. Nakamura. Edge-connectivity augmentation problems. *Journal of Computer and System Sciences*, 35:96–144, 1987.

A Background on Edge-Connectivity

In this appendix we recall some known facts regarding the structure of the k-edge-connected classes of a $(k-1)$-connected graph. Whereas the structure of the 2-classes of a connected graph is well-known and relatively simple (cf., [Eve79]), the (k-connected class) structure of $(k-1)$-connected graphs becomes slightly more complex when $k \geq 3$. We thus refrain from describing in detail this structure and merely state the facts which we need. The interested reader is referred to [DW98] for more details.

A.1 The auxiliary tree of a $(k-1)$–connected graph

We emphasize that the graphs below are not necessarily simple; that is, parallel edges are allowed.

Fact A.1 (cf., [DW98]): *Let $k > 1$ be an integer and G be a $(k-1)$-connected graph. Then there exists an auxiliary graph, $\mathrm{T_G}$, which is a tree such that:*

- *Each k-connected class in G corresponds to a unique node in $\mathrm{T_G}$.*
- *In addition to nodes corresponding to k-connected classes, there are two types of auxiliary nodes:* empty *nodes, and* cycle *nodes (the latter exist only for odd k). The neighbors of a cycle node in $\mathrm{T_G}$ are said to* belong *to a common cycle, and we associate a cyclic order with them. (Since $\mathrm{T_G}$ is a tree, any two cycles can have at most one common node.)*
- *All leaves of the auxiliary tree $\mathrm{T_G}$ correspond to k-connected classes of G. Furthermore, there are exactly $k-1$ edges (in G) going out from each of these classes.*

For example, when $k = 2$, all nodes of the auxiliary tree correspond to 2-classes, and the edges in the auxiliary tree correspond to graph edges which are known as *bridges*. Bridges are edges connecting vertices in different 2-classes of the graph, and their removal disconnects the graph. In the case of $k = 3$, the auxiliary tree includes cycle nodes (but no empty nodes). If $\mathrm{C}_1 \ldots, \mathrm{C}_\ell$ are neighbors of a cycle node Cy, then this means that there is a single graph edge between some vertex in C_i and some vertex in $\mathrm{C}_{i+1 \bmod \ell}$, for every i.

Before stating the next lemma we need to define the notion of *squeezing a cycle*. Let Cy be a cycle node in $\mathrm{T_G}$, and let its neighbors be $\mathrm{C}_1, \ldots, \mathrm{C}_t$ (where their indices corresponds to their ordering around the cycle). Then the result of *squeezing* Cy at C_i and C_j is the merging of C_i and C_j into a new node C_k, with one of the following changes to the cycle:

1. In case C_i and C_j are adjacent on the cycle, then we have two subcases.
   (a) If $t > 3$ then the merged node C_k is connected by a single edge to the cycle node Cy (and all other nodes belonging to the cycle remain that way).
   (b) If $t = 3$ (i.e., there was only one additional node on the cycle), then Cy is removed, and the additional node is connected by a tree edge to C_k.
2. In case C_i and C_j are separated by at least one node on the cycle then $t \geq 4$, and we have three subcases.
   (a) If $t = 4$ (and so C_i and C_j are separated by a single node in each cycle direction), then we put a tree edge between each of these intermediate nodes and C_k, and the cycle disappears.

(b) If $t > 4$ and C_i and C_j are separated by a single node C_ℓ on one of the cycle directions, then we put a tree edge between C_ℓ and C_k, and C_k belongs to a single cycle with all the rest of the (at least 2) nodes which were previously on the cycle.

(c) Otherwise ($t > 4$ and at least two nodes separate C_i and C_j in each direction), then we get two cycles, where C_k belongs to both, and the other nodes are partitioned among the cycles according to their relative position with respect to C_i and C_j.

Lemma A.2 (cf., [DW98]): *Let* G *be a* $(k-1)$*-connected graph, and* $\mathrm{T_G}$ *be its auxiliary tree. Suppose that we augment* G *by an edge with endpoints in the* k*-connected classes* C_1 *and* C_2*, respectively. Then the classes residing on the simple path between* C_1 *and* C_2 *in* $\mathrm{T_G}$ *form a* k*-connected class in the augmented graph, and all classes in* G *which do not reside on the path remain distinct* k*-classes in the augmented graph. In case the path passes through nodes* C_i *and* C_j *which belong to the same cycle* Cy*, then* Cy *is squeezed at* C_i *and* C_j*.*

A related lemma which we need follows. In what follows, when we refer to an edge as *being in a class* we mean that it connects two vertices belonging to the class.

Lemma A.3 *Let* G *be a* $(k-1)$*-connected graph,* $\mathrm{T_G}$ *be its auxiliary tree, and* C_1*,* C_2 *two* (k-connected) *classes of* G *each containing at least one edge. Suppose that we omit a single edge from each* C_i *and add two edges so to maintain the vertex degrees of* G*; Specifically, if the edges* (u_1, v_1) *and* (u_2, v_2) *were omitted from* C_1 *and* C_2 *respectively, then we either add the edges* (u_1, u_2) *and* (v_1, v_2)*, or the edges* (u_1, v_2) *and* (v_1, u_2)*. As a result, the classes residing on the simple path between* C_1 *and* C_2 *in* $\mathrm{T_G}$ *form a* k*-connected class in the augmented graph, and all classes in* G *which do not reside on the path remain distinct* k*-classes in the augmented graph.*

We note that this lemma can be proven (*private communication with Y. Dinitz, December 1996*) using the Circumference Theorem in [DKL76], but we provide a direct proof for completeness.

Proof: Let $I_1, ..., I_t$ be the (intermediate) k-classes residing on the path between C_1 and C_2 in the tree $\mathrm{T_G}$. (We do not exclude the case $t = 0$.)

Consider what happens when we omit the edge (u_i, v_i) from C_i. Either C_i remains a k-class, or it breaks into several k-classes, denoted $\mathrm{C}_i^1, ..., \mathrm{C}_i^{q_i}$. It follows from Lemma A.2 that in the latter case the classes $\mathrm{C}_i^1, ..., \mathrm{C}_i^{q_i}$ correspond to a path on the auxiliary tree of the modified graph, so that the vertex u_i resides in C_i^1, and vertex v_i resides in $\mathrm{C}_i^{q_i}$. (Any other restructuring is ruled out by Lemma A.2, since if we now add the edge (u_i, v_i) back, we must regain the k-class C_i.) Thus, the I_j's and the C_i^j's reside on a sub-tree of the auxiliary tree of the modified graph so that the only leaves in this sub-tree are among the "extreme" C_i^j's (i.e., C_1^1, $\mathrm{C}_1^{q_1}$, C_2^1, and $\mathrm{C}_2^{q_2}$).

Consider first the simpler case of $t \geq 1$. The existence of intermediate nodes guarantees that none of the C_1^j's may belong to the same cycle as a C_2^j. In this case, we may use either pairs of edges suggested in the lemma to join the four classes in two pairs and collapse the entire sub-tree into a single node. That is, suppose we add the edges (u_1, u_2) and (v_1, v_2). Then, by Lemma A.2 the first (resp., second) added edge will cause the collapse of all classes on the path between C_1^1 and C_2^1 (resp., $\mathrm{C}_1^{q_1}$ and $\mathrm{C}_2^{q_2}$). Since these are the only leaves on the sub-tree, the claim follows. A similar argument can be applied as long as C_1^1, $\mathrm{C}_1^{q_1}$, C_2^1, and $\mathrm{C}_2^{q_2}$ do not belong to the same cycle.

It remains to deal with the case in which C_1^1, $\mathrm{C}_1^{q_1}$, C_2^1, and $\mathrm{C}_2^{q_2}$ all belong to the same cycle. Here we must be careful in choosing which two edges to add. Assume, w.l.o.g., that indeed their order on the cycle is as above. Then it is essential that we add the edges (u_1, u_2) and (v_1, v_2) (i.e., connecting C_1^1 to C_2^1 and $\mathrm{C}_1^{q_1}$ to $\mathrm{C}_2^{q_2}$) in a *crossing* fashion, so as to insure that the two invocation of Lemma A.2 will cause the collapse of the four classes into one class. The lemma follows. ■

A.2 Distance from k-connectivity versus number of leaves

Using Lemma A.2, it is easy to transform any $(k-1)$-connected graph G into a k-connected graph G$'$ by adding at most $L-1$ edges, where L is the number of leaves in the auxiliary tree of G. This follows by observing that each application of the lemma reduces the number of leaves by one. However, this process (especially if applied obliviously) may result in a graph G$'$ which violates the degree bound. Thus, we use a slightly more complicated argument which utilizes Lemmas A.2 and A.3.

Lemma A.4 *Let* G *be a $(k-1)$-connected graph, whose auxiliary graph,* $\mathrm{T_G}$*, has L leaves. Then by removing and adding at most $4L$ edges to* G *we can transform it into a k-connected graph* G$'$*. Furthermore, suppose that the maximum degree of* G *is d then the maximum degree of* G$'$ *is upper bounded by* $\max\{d,k\}$ *if either $d > k$ or dN is even, and by $k+1$ otherwise.*

We note that there might be a way to save a constant factor in the number of edges added and removed from G when transforming it into a k-connected graph (while respecting the degree bound).

Proof: We first use Lemma A.2 to collapse all leaves in $\mathrm{T_G}$ which correspond to singleton classes (i.e., classes consisting of a single vertex of G). These vertices have degree $k-1$ and so we can match them in pairs and add a single edge between each pair. At this point we may be left with a single unmatched vertex/leaf, which we deal with later. Call the resulting graph $\mathrm{G_1}$ and its auxiliary tree $\mathrm{T_1}$. The number of leaves in $\mathrm{T_1}$ is at most $L-i$, where i is the number of pairs matched above. All leaves in $\mathrm{T_1}$ (except for possibly a unique singleton) can be now collapsed using Lemma A.3. The number of edge modifications in this stage is at most $4(L-i-1)$. The resulting graph, $\mathrm{G_2}$, has degree at most $d' \stackrel{\rm def}{=} \max\{d,k\}$. In case $\mathrm{G_2}$ is k-connected we are done.

Otherwise, $\mathrm{G_2}$ consists of a singleton which is connected to a k-connected class containing all other vertices. In case some vertex in the large class has degree lower than d' we connect it to the singleton and conclude as per Lemma A.2. Otherwise (i.e., all vertices in the large class have degree d'), we need to distinguish two subcases. In case $k < d'$ we simply omit one edge internal to the large class and connect its endpoints to the singleton. It can be seen that this makes the graph k-connected and that all vertices have degree at most d'. Finally, if $d' = k$ a parity argument shows that $d'N$ must be odd (as otherwise the sum of degrees, $(N-1)d' + (k-1) = Nd' - 1$, is odd). In this case we are allowed to add an edge and increase the degree of the resulting graph to $d'+1 = k+1$.

The total number of modifications is thus $i + 4(L-i-1) + 3 < 4L$, and the lemma follows. ■

B Proof of Inequality (2)

Our aim is to prove that for any integers $c \le D$ and $n \ge 2$,

$$p \stackrel{\rm def}{=} \prod_{j=2}^{n} \frac{j-(c/D)}{j+(c/D)} > \Theta(n)^{-2c/D}$$

A proof that $p = \Omega(n^{-2c/D})$, for constant c, D, can be derived from Karger's Ph.D. Thesis [Kar95] (see proof of Corollary 4.7.5 which refers to an exercise in Knuth Vol. 1). An alternative proof follows.

We first observe that for every $i > 0$

$$\frac{jD - c}{jD + c} > \frac{jD - c - i}{jD + c - i} \tag{7}$$

Using Eq. (7), we have

$$\begin{aligned}
p^D &= \left(\prod_{j=2}^{n} \frac{jD - c}{jD + c)} \right)^D \\
&> \prod_{i=0}^{D-1} \prod_{j=2}^{n} \frac{jD - c - i}{jD + c - i} \\
&= \prod_{k=2D-(D-1)}^{nD} \frac{k - c}{k + c} \\
&= \frac{(D - c + 1) \cdot (D - c + 2) \cdots (D + c)}{(nD - c + 1) \cdot (nD - c + 2) \cdots (nD + c)} \\
&> \frac{(D/O(1))^{2c}}{(nD + c)^{2c}}
\end{aligned}$$

Thus, using $c \leq D$ and $n \geq 2$, we get

$$\begin{aligned}
p &> \left(\frac{1/O(1)}{n + (c/D)} \right)^{2c/D} \\
&> \left(\frac{1}{\Theta(n)} \right)^{2c/D}
\end{aligned}$$