Finding Cycles and Trees in Sublinear Time

Artur Czumaj Oded Goldreich Dana Ron C. Seshadhri Asaf Shapira
Christian Sohler

April 7, 2010

Abstract

We present sublinear-time (randomized) algorithms for finding simple cycles of length at least $k \geq 3$ and tree-minors in bounded-degree graphs. The complexity of these algorithms is related to the distance of the graph from being C_k-minor free (resp., free from having the corresponding tree-minor). In particular, if the graph is far (i.e., $\Omega(1)$-far) from being cycle-free, i.e. if one has to delete a constant fraction of edges to make it cycle-free, then the algorithm finds a cycle of polylogarithmic length in time $\widetilde{O}(\sqrt{N})$, where N denotes the number of vertices. This time complexity is optimal up to polylogarithmic factors.

The foregoing results are the outcome of our study of the complexity of *one-sided error* property testing algorithms in the bounded-degree graphs model. For example, we show that cycle-freeness of N-vertex graphs can be tested with one-sided error within time complexity $\widetilde{O}(\text{poly}(1/\epsilon) \cdot \sqrt{N})$. This matches the known $\Omega(\sqrt{N})$ query lower bound, and contrasts with the fact that any minor-free property admits a *two-sided error* tester of query complexity that only depends on the proximity parameter ϵ. For any constant $k \geq 3$, we extend this result to testing whether the input graph has a simple cycle of length at least k. On the other hand, for any fixed tree T, we show that T-minor freeness has a one-sided error tester of query complexity that only depends on the proximity parameter ϵ.

Our algorithm for finding cycles in bounded-degree graphs extends to general graphs, where distances are measured with respect to the actual number of edges. Such an extension is not possible with respect to finding tree-minors in $o(\sqrt{N})$ complexity.

Keywords: Sublinear-Time Algorithms, Property Testing, Bounded-Degree Graphs, One-Sided vs Two-Sided Error Probability,

Contents

1 Introduction

Consider the algorithmic problem of finding a (simple) cycle in a bounded degree graph (assuming one exists), where the aim is to find such a cycle in (randomized) sublinear time. In general, finding a cycle in sublinear time may not be possible, since the graph may contain only cycles of length $\Omega(n)$. This may also be the case if one needs to remove a constant *number* of the edges of the graph in order to make it cycle-free. But suppose one needs to remove a constant *fraction* of the graph's edges in order to make it cycle free. Can we then devise a sublinear time algorithm? One of our results in this paper is an affirmative answer to this question. Furthermore, the running time of that algorithm is (essentially) optimal.

1.1 Our main results

As we have mentioned above, we consider graphs of bounded degree d with N vertices. We say that a graph is ϵ-far from being cycle-free if one has to remove at least ϵdN edges from G in order to make it cycle free. We can now formally state our first result.

Theorem 1.1 *There exists a randomized algorithm that, on input an N-vertex graph G of constant degree bound d that is ϵ-far from being cycle-free, finds a simple cycle in G in expected time $\widetilde{O}(\mathrm{poly}(1/\epsilon) \cdot \sqrt{N})$. Furthermore, the cycle found has length* $\mathrm{poly}(\epsilon^{-1} \log N)$.

Using the connection to one-sided error property testing (detailed in Section 1.2), we infer that the algorithm of Theorem 1.1 is optimal; that is, no randomized $o(\sqrt{N})$-time algorithm can find cycles in (bounded-degree) graphs that are $\Omega(1)$-far from being cycle-free. Furthermore, one can not expect to find simple cycles of length $o(\log N)$, since such may not exist (even if the graph is far from being cycle-free). The result of Theorem 1.1 can be extended to finding a simple cycle of length at least k, for any fixed $k \geq 3$.

Theorem 1.2 *For every constant $k > 3$, there exists a randomized algorithm that, on input an N-vertex graph G of constant degree bound d that is ϵ-far from having no cycles of length at least k, finds a simple cycle of length at least k in G in expected time $\widetilde{O}(\mathrm{poly}(1/\epsilon) \cdot \sqrt{N})$. Furthermore, the cycle found has length* $\mathrm{poly}(\epsilon^{-1} \log N)$.

Again, the algorithm obtained is optimal. We note that cycles of length at least k in G correspond to C_k-minors of G. Here, C_k denotes the k-vertex cycle. An H-minor of G is obtained from G by a sequence of edge removals, vertex removals, and edge contractions. (A C_k-minor of G is a cycle in G of length larger than k.)

We next turn from finding cycles to finding tree-structures in graphs. Our main result deals with finding tree-minors. Consider the following interesting special case. For any constant k, we want to find a tree with at least k leaves. One of our results is a randomized algorithm that finds such trees in expected time that is polynomially related to k and to the distance of the input graph from a graph having no such trees. This problem corresponds to finding minors that are k-vertex stars.

Theorem 1.3 *For any fixed tree T, there exists a randomized algorithm that, on input an N-vertex graph G of constant degree bound d that is ϵ-far from being T-minor free, finds a T-minor in expected time that only depends on ϵ, where vertex manipulation operations are counted at unit cost.*

We highlight the fact that finding tree minors can be done within complexity that only depends on ϵ, whereas finding cycles requires $\Omega(\sqrt{N})$ time (also for constant $\epsilon > 0$). In fact, we show that Theorem 1.3 extends to any cycle-free graph H, and on the other hand we prove that *for any H that contains a cycle finding H-minors requires $\Omega(\sqrt{N})$ queries.*[1] Thus, we obtain the following characterization:

Corollary 1.4 *Finding H-minors in a constant degree graph that is ϵ-far from being H-minor free can be done in complexity that only depends on ϵ if and only if H is cycle-free.*

All our results are obtained via the connection to one-sided error property testing, and are actually easier to state in terms of property testing.

1.2 The property testing connection

Loosely speaking, property testing refers to sublinear time probabilistic algorithms for deciding whether a given object has a predetermined property or is far from any object having this property (see the surveys [Fis01, Ron08b, Ron08a]). Such algorithms, called testers, obtain local views of the object by making suitable queries; that is, the object is seen as a function and the tester gets oracle access to this function (and thus may be expected to work in time that is sublinear in the size of the object).

Randomization is essential to natural testers (i.e., testers of natural properties that have sublinear query-complexity) [GS07]. The same holds also for error probability, at least on some instances, but the question is whether a (small) error probability must appear on all instances. In particular, *should we allow* (small) *error probability both on instances that have the property and on instances that are far from having it?*[2]

Indeed, testers come in two basic flavors referring to the foregoing question: two-sided error testers allow (small) error probability both on instances that have the property and on instances that are far from having it, whereas one-sided error testers only allow (small) error probability on instances that are far from having the property. That is, in one-sided error testers, any instance that has the property is accepted with probability 1.

An important observation regarding one-sided error testers is that *whenever such a tester rejects some instance, it always has a certificate that this instance does not have the property, where this certificate is the partial view of the instance as obtained by the tester.* Indeed, in the case of one-sided error, rejecting an instance based on a specific partial view means that there exists no instance that has the property and is consistent with this partial view. Furthermore, in some cases (as those addressed in the current work), this partial view contains some natural structures (e.g., a cycle or a tree of interest).

Consider, for example, the case of testing cycle-freeness (with one-sided error). In this case, whenever the tester rejects, its partial view must contain a cycle. Thus, *any one-sided tester of cycle-freeness may be used for finding cycles in graphs that are far from being cycle-free.* A similar observation applies to finding T-minors, for any fixed tree T.

We mention that in most of the property testing literature, one-sided error is viewed as a secondary feature that some testers have and others may lack. The foregoing connection demonstrates the fundamental advantage of one-sided error testers over standard (two-sided error) testers. (Other advantages are discussed in Section 1.5.)

[1] This fact was mentioned in [BSS08].

[2] In any case, the basic paradigm of property testing allows arbitrary error in case the instance neither has the property nor is far from having it.

Lower bounds on the complexity of one-sided error testers that significantly exceeds the performance guarantees of known two-sided error testers have been observed, starting with [GGR98, Sec. 10.1.6]. However, so far, no study has been devoted to providing a *one-sided error tester of optimal complexity, in the case where the complexity significantly exceeds that of the corresponding two-sided error tester.*[3]

In contrast, our work is aimed at providing a one-sided error tester of (almost) optimal complexity, in a case in which this complexity significantly exceed the complexity of the corresponding two-sided error tester. For example, recall that Goldreich and Ron provided a two-sided error tester for cycle-freeness of $\mathrm{poly}(1/\epsilon)$ query complexity [GR02, Thm. 4.2], where ϵ denotes the desired proximity parameter (i.e., the tester distinguishes cycle-free graphs from graphs that are ϵ-far from being cycle-free). In contrast, [GR02, Prop. 4.3] asserts that cycle-freeness has no one-sided error tester that makes $o(\sqrt{N})$ queries (even for $\epsilon = 1/3$), where N denotes the number of vertices in the input graph. In that context, Theorem 1.1 is equivalent to

Theorem 1.5 *Cycle-freeness of constant degree N-vertex graphs can be tested with one-sided error within time complexity $\widetilde{O}(\mathrm{poly}(1/\epsilon) \cdot \sqrt{N})$. Furthermore, whenever the tester rejects, it outputs a simple cycle of length* $\mathrm{poly}(\epsilon^{-1} \log N)$.

On the other hand, by the foregoing discussion, whenever the tester asserted in Theorem 1.5 rejects, it is the case that it explored a subgraph that is not cycle-free. Moreover, the furthermore clause of Theorem 1.5 asserts that in this case the explored subgraph actually contains a simple cycle of length $\mathrm{poly}(\epsilon^{-1} \log N)$. Thus, Theorem 1.5 implies Theorem 1.1. Similarly, *Theorem 1.5 extends to testing C_k-minor freeness, for any $k \geq 3$, which in turn is equivalent to Theorem 1.1.* And, similarly, Theorem 1.3 is equivalent to the existence of a *tester for T-minor freeness of query complexity that only depends on the proximity parameter, for any tree T.*

1.3 Techniques

As stated at the end of Section 1.1, all our results are obtained via the study of the complexity of one-sided error testers for the corresponding properties.

An interesting feature of our testers for C_k-minor freeness is that they are all obtained by local reductions. Specifically, our cycle-freeness (i.e., C_3-minor freeness) tester is obtained by a randomized reduction to testing bipartiteness, whereas our C_k-minor freeness tester is obtained by a deterministic reduction to testing cycle-freeness.

1.3.1 Testing cycle-freeness

We mention that the two-sided error tester of [GR02] does not even try to find a simple cycle. It just estimates the number of edges in the graph and rejects if this estimate exceed the number of

[3]To the best of our knowledge, the case that seems closest is the discussion in [AS03, Sec. 2] that refers to the complexity of testing $K_{t,t}$-freeness in the *adjacency matrix model* (introduced in [GGR98]). Specifically, [AS03, Clm. 2.2] asserts a two-sided tester of $K_{t,t}$-freeness having query complexity $O(1/\epsilon)$, whereas [AS03, Clm. 2.3] (combined with [GT03, Thm. 2]) asserts that one-sided error testing of $K_{t,t}$-freeness requires $\Omega(\epsilon^{-t/4})$ queries, which (as noted at the end of [AS03, Sec. 2]) is tight up to a polynomial function (i.e., there exists two-sided tester of $K_{t,t}$-freeness having query complexity $\epsilon^{-O(t)} = \mathrm{poly}(\epsilon^{-t/4})$). It is telling that [AS03, Sec. 2] leaves the complexity of one-sided error testing undetermined (at the "polynomial slackness" level). Indeed, like other prior works that address this issue, their interest is in demonstrating the gap between the complexities of two-sided and one-sided error testing, and not in determining the latter.

edges that correspond to any forest that spans the set of connected components of the graph.[4] We also mention that a "girth versus edge-density" lower bound implies that any graph $G = ([N], E)$ that is ϵ-far from being cycle-free must have a simple cycle of length $O((\log N)/\epsilon)$.[5] [6] The problem, however, is finding such a cycle in sublinear time.

Our one-sided error tester of cycle-freeness finds a cycle in the original graph by randomly reducing this problem to the problem of finding an odd-length cycle in an auxiliary graph. Specifically, the input graph $G = ([N], E)$ is randomly transformed into an auxiliary graph such that each edge $e \in E$ is replaced, with probability 1/2 by a 2-vertex path (with an auxiliary vertex), and remains intact otherwise. Thus, with probability 1/2, each cycle in G is transformed into an odd-length cycle. Furthermore, we show that if G is ϵ-far from being cycle-free, then (w.h.p.) the resulting graph is $\Omega(\epsilon)$-far from being bipartite.

A crucial feature of the foregoing randomized reduction is that it is local in the sense that each operation on the transformed graph can be implemented by a constant number of operations on the original graph. Thus, we can emulate the execution of a bipartite tester (i.e., the one of [GR99]) on the transformed graph. This allows us to establish Theorem 1.5.

1.3.2 Testing C_k-minor freeness, for any $k > 3$

Recall that the set of C_k-minor-free graphs coincides with the set of graphs that have no simple cycle of length at least k. Theorem 1.2 is proved by a (local) reduction of testing C_k-minor-freeness to testing cycle-freeness. For example, in the case of $k = 4$ we replace each triangle by a 3-vertex star; that is, we omit the original edges of this triangle, and introduce an auxiliary vertex that is connected to the three corresponding vertices. We then prove that if the original graph is C_4-minor-free then the resulting graph is cycle-free, whereas if the original graph is ϵ-far from being C_4-minor-free then the resulting graph is $\Omega(\epsilon)$-far from being cycle-free.

For larger values of k, a more sophisticated local replacement is used. That is, replacing all small cycles by auxiliary vertices will not do. To illustrate the difficulty of dealing with $k > 4$, note that, unlike in the case $k = 4$, a C_k-minor free graph may contain cycles of length smaller than k that share some common edges, and so the simple replacement will not yield a cycle-free graph. (In contrast, note that a C_4-minor free graph can not contain a pair of triangles that share an edge (since such a subgraph contains a cycle of length 4).

1.3.3 Testing H-minor freeness, for any cycle-free H

The main challenge for this problem is testing T-minor freeness, where T is an arbitrary tree. The simple case in which T is a k-vertex star, for some $k \geq 2$, provides a good illustration to the underlying main idea. In this case we may select a random vertex and start a BFS at this vertex, stopping whenever either we encounter a layer with at least k vertices or we explored more

[4] Note that any cycle-free graph is a forest, and if the number of trees in this forest is t, then the difference between the number of vertices and the number of edges in the graph equals t. The two-sided error tester of [GR02] estimates the number of edges and the number of connected components in the graph, and conducts the adequate computation. The number of connected components is estimated by the number of connected components that have more than $O(1/\epsilon)$ vertices, whereas the latter number is approximated by exploring the neighborhood of a few randomly selected vertices.

[5] This is a consequence of the Moore bound - a graph of girth k has at most $n^{1+1/k}$ edges.

[6] Consider any connected component of G, denoted $G' = ([N'], E')$, that is ϵ-far from being cycle-free, and let d be the constant degree bound of G. Note that $|E'| \geq (1 + (d\epsilon/2)) \cdot N'$, because selecting an arbitrary spanning tree of G' and omitting all non-tree edges yields a cycle-free. The claim follows by recalling that any n-vertex graph that has at least $n^{1+(1/k)}$ edges has girth at most $2k$ (i.e., has a simple cycle of length at most $2k$).

than $4k/\epsilon$ layers (or we explored the entire connected component). In the first case, we found the desired minor and can safely reject, whereas in the second case we found a set of at least $4k/\epsilon$ vertices that is separated from the rest of the graph by less than dk edges. Thus, if the graph $G = ([N], E)$ contains at least $(1 - \epsilon/4) \cdot N$ start vertices that do not lead the algorithm to reject, then G can be decomposed to connected components that are each T-minor free by omitting at most $\epsilon dN/2$ edges (i.e., the edges that are incident at the $\epsilon N/4$ exceptional vertices and the edges of the aforementioned small cuts).

Needless to say, the case of a general tree T is much more complex, but the governing principle remains a tight relation between having few start vertices that contain a T-minor at their vicinity and the ability to decompose the graph to connected components with few edges between them. This relation is captured by the following result, which may be of independent interest.[7]

Theorem 1.6 *For every d and k there exists an $r = r(d, k)$ such that if the r-neighborhood of a vertex s in a graph of degree bound d does not contain a T-minor of some tree T with at most k vertices, then this neighborhood contains a set S that is separated from the rest of the graph by less than $\epsilon d|S|/4$ edges.*

In other words, if all "sub-neighborhoods" of the r-neighborhood of s are "expanding" (i.e., are not separated from the rest by small cuts), then this r-neighborhood contains a T-minor of every tree T with at most k vertices.

We reduce finding H-minors, where H is an arbitrary cycle-free graph (forest), to finding disjoint tree minors. Again, the reduction is local, and in this case it is almost straightforward, where the subtlety is related to the fact that we refer to one-sided error. Specifically, if H consists of the connected components $H_1, ..., H_m$, then it does not necessarily hold that G is H-minor free if and only if G is H_i-minor free for all $i \in [m]$. Still, this is "almost true" and so a small modification of the straightforward reduction will do.

1.4 Another perspective: Finding arbitrary forbidden minors

Our results may be viewed as progress in resolving an open problem, posed by Benjamini, Schramm, and Shapira [BSS08], that refers to one-sided error testing of H-minor-freeness, for any finite graph H (or even a finite family of such graphs).[8] Specifically, Benjamini *et al.* [BSS08] proved that, for any H, the property of being H-minor-free can be tested within query complexity that only depends on the proximity parameter,[9] *when allowing two-sided error.* They conjectured that for any non-forest H, there exists an H-minor-freeness tester with query complexity $O(\sqrt{N})$. They also mention that the $\Omega(\sqrt{N})$ lower bound of [GR] holds for testing any non-forest H.

Our results (essentially) resolve this question in the affirmative for the special case of $H = C_k$, for every $k \geq 3$.

[7]We mention that the problem of finding small trees in locally expanding graphs has been studied before (cf., e.g. [FP87]). However, our Theorem 1.6 seems incomparable, since we seek specific *tree minors* rather than specific trees, whereas our expansion condition is very weak.

[8]Recall that the graph G has an H-minor if H can be obtained from G through a series of vertex removals, edge removals, and edge contractions (see, e.g., Lovász [Lov06]). The graph G is H-minor free, if it contains no H-minor. Also recall that the set of cycle-free graphs coincides with the set of C_3-minor-free graphs, where C_k denote the k-cycle graph (i.e., a graph consisting of a simple cycle of length k).

[9]The query complexity obtained in [BSS08] is triple-exponential in $1/\epsilon$. The complexity was recently improved to exponential in $1/\epsilon$ [HKNO09],

Theorem 1.7 (see Theorems 1.1 and 1.2): *For any constant $k \geq 3$, there exists a one-sided error tester of time complexity $\widetilde{O}(\mathrm{poly}(d^k/\epsilon) \cdot \sqrt{N})$ for testing whether a constant degree N-vertex graph is C_k-minor-free. Furthermore, whenever the tester rejects, it outputs a simple cycle of length at least k (and at most $\mathrm{poly}(\epsilon^{-1} \log N)$).*

We note that finding cycles seems the "hard" part of finding minors; that is, cycles are the source of the $\Omega(\sqrt{N})$ query lower bound. Recall that [GR02, Prop. 4.3] establishes an $\Omega(\sqrt{N})$ query lower bound for any algorithm that finds C_3-minors (or, in other words, a one-sided property tester for cycle-freeness). Although it was mentioned in [BSS08] that this lower bound may extend to H-minor-freeness, for any H that contains a cycle, a proof of this fact has not appeared so far. We present a proof of this fact, thus establishing *an $\Omega(\sqrt{N})$ query lower bound for any algorithm that finds minors that contain cycles.* This stands in contrast to the following result (cf. Corollary 1.4).

Theorem 1.8 (see Theorem 1.3): *For any fixed cycle-free graph H, there exists a one-sided error tester of query complexity that only depends on the proximity parameter for testing whether a given constant degree graph is H-minor-free. Furthermore, whenever the tester rejects, it outputs an H-minor of the input graph.*

The query complexity of the foregoing tester is exponential in $(16d/\epsilon)^{O(k)}$, where k denotes the number of vertices in H and ϵ is the proximity parameter. We mention that better complexity can be achieved in some special cases (e.g., stars and depth-two trees, see Section 7).

A wider perspective on finding forbidden minors. The first result dealing with graph minors is the well known Kuratowski-Wagner theorem [Kur30, Wag37] that states that any non-planar graph contains a K_5 or $K_{3,3}$ minor. Consider a property $\mathcal{P}$ such that if $G \in \mathcal{P}$, then, for any minor H of G, $H \in \mathcal{P}$. Such a property is *minor-closed*. It was conjectured by Wagner that for *any* minor-closed property $\mathcal{P}$, there is a finite set of graphs $\mathcal{H}_\mathcal{P}$ such that $G \in \mathcal{P}$ iff G is H-minor free, for all $H \in \mathcal{H}_\mathcal{P}$. Robertson and Seymour had a long series of deep papers, which culminated in the proof of this conjecture [RS04], called the Graph-Minor Theorem. From an algorithmic perspective, one of the milestones in this series was a polynomial time algorithm that checked H-minor freeness, for any graph H [RS95].

It is natural to ask this algorithmic question, from a sublinear perspective. Do we really need to look at the whole graph to find a forbidden minor? Suppose we are given a graph G that is far from being minor-free, say a (small) constant fraction of the edges needs to be removed to make G minor-free. Then, can we find an H-minor by looking at a sublinear portion of the graph? This would imply the interesting combinatorial statement that such a graph contains sublinear sized H-minors. This paper is the first investigation into this problem. Indeed, one-sided property testers can be thought of as sublinear time minor finders (see Section 1.2).

1.5 Further reflections regarding one-sided error

The relative power of two-sided versus one-sided error randomized decision procedures has been the focus of considerable study in many settings, including in property testing. Indeed, in any setting, one-sided error procedures offer the advantage of never rejecting yes-instances. However, as we already saw in Section 1.2, this advantage has a special appeal in the context of property testing, since it yields algorithms for very efficiently finding some desired structures (whenever the graph is far from being "free of them"). Additional benefits of one-sided error testers are discussed next.

Firstly, we note that property testing is asymmetric in nature: It refers to distinguishing objects that *perfectly* satisfy a predetermined property from objects that are *far* from satisfying this property. Indeed, property testing is a relaxation of the original decision task (which refers to distinguishing objects that satisfy the property from objects that do not satisfy it), where the relaxation is applied to one type of instances but not to the other. In this context, it is natural to apply the probabilistic relaxation also to one type of instances (i.e., the far-away instances) but not to the other.

Secondly, we note that one of the main applications of property testers is their potential use as a preliminary "fast but crude" decision step, which when coupled with an exact (but slow) decision procedure yields a procedure that is always correct and often very fast. That is, we envision using a property tester as a "sieve" that rejects "on the spot" (i.e., "fast") very bad instances (i.e., those that are far from satisfying this property), while passing the rest of the instances for further examination. In such a context, we can afford passing very bad instances for further examination (since all this means is a waste of time), but we cannot afford failing a good instance.

Lastly, we consider the relationship between property testing and local structures in the tested property. Intuitively, the existence of a property tester means that a global structure (i.e., distance of the object to the property) is reflected in (or co-related with) a local structure (i.e., the part of the object being probed by the tester). In the general case (of two-sided error), this co-relation is statistical, whereas in the case of one-sided error this correlation is actually a ("robust") characterization.

The last aspect is particularly clear in the current study. Firstly, the notion of local structure is most appealing in the bounded-degree model, where it refers to graph neighborhoods. Secondly, the different types of local structures underlying the two-sided and one-sided error testers is most striking in the case of cycle-freeness. The two-sided error tester of [GR02] relies on the fact that distance from cycle-freeness in connected graphs is reflected by the difference between the number of edges and the number of vertices, whereas these numbers can be estimated (with two-sided error) by sampling the graph's vertices. Note that such *estimates* cannot yield a characterization (let alone a robust one) of the cycle-free graphs. In contrast, our one-sided error tester relies on the fact that distance from cycle-freeness is reflected in the density of short simple cycles in the graph, whereas such cycles can be found by an appropriate randomized exploration of the graph. Indeed, this yields a (robust) characterization of the set of cycle-free graphs (i.e., a graph is cycle-free iff it contains no simple cycle, and the farther the graph is from being cycle-free the shorter and more abundant these cycles are).

1.6 The general (unbounded-degree) graph model

Our algorithm for finding cycles in bounded-degree graphs (i.e., Theorem 1.1) extends to the general graphs model (i.e., the model in [PR02]), where distances are measured with respect to the actual number of edges (see Section 8).[10] This follows by an alternative presentation of the basic randomized reduction, which may be viewed as reducing cycle-freeness to a generalization of 2-colorability. In this generalization, edges of the graph are labeled by either `eq` or `neq`, and a legal 2-coloring (of the vertices) is one in which every two vertices that are connected by an edge

[10] Algorithms in this model use the same type of incidence queries as in the main (bounded-degree) model we consider. The difference is that a graph $G = ([N], E)$ is said to be ϵ-far from H-minor-freeness if $\epsilon|E|$ edges (rather than $\epsilon dN/2$ edges) must be removed from G in order to obtain an H-minor-free subgraph. The point is that the number of edges is related to the average degree of G rather than to its degree (upper) bound, which may be significantly smaller. Thus, distances under this model are possibly larger, and thus the testing requirement is possibly harder.

labeled eq (resp. neq) are assigned the same color (resp., opposite colors). We also observe that the Bipartite testers of [GR99, KKR04] extend to this generalization of 2-colorability.

We mention that analogous extensions do not work for testing C_k-minor freeness, for $k > 3$, nor for testing tree-minor-freeness. In fact, we show that, in the general graph model, it is not possible to find tree-minors (or even test freeness with two-sided error) by using $o(\sqrt{N})$ queries.

1.7 Organization

Section 2 contains a formal statement of the relevant definitions and terminology. The testers of C_k-minor freeness are presented in Sections 3–5. Our main result (i.e., the one-sided error tester of cycle-freeness) is presented in Section 3. The reduction of testing C_k-minor freeness to testing cycle-freeness is presented in Section 5, but Section 4 provides an adequate warm-up by treating the case of $k = 4$.

In Section 6, we prove the lower bound claimed in [BSS08] regarding the query complexity of one-sided error testing H-minor freeness, when H contains a cycle. In contrast, in Section 7 we consider the case that H is cycle-free, and present the improved testers for H-minor freeness in this case (i.e., when H is a forest).

Finally, in Section 8 we consider the unbounded-degree model, discussed in Section 1.6.

2 Preliminaries

This work refers to the bounded-degree model (introduced in [GR02]). This model refers to a fixed degree bound, denoted d. An N-vertex graph $G = ([N], E)$ (of maximum degree d) is represented in this model by a function $g : [N] \times [d] \to \{0, 1, ..., N\}$ such that $g(v, i) = u \in [N]$ if u is the i^{th} neighbor of v and $g(v, i) = 0$ if v has less than i neighbors. Testing in this model is captured by the general definition of property testing of functions, when applied to functions of the foregoing type and considering only graph properties (i.e., properties that are preserved under isomorphism). That is, saying that a tester has oracle access to a graph G means that it is given oracle access to the corresponding function g.

Definition 2.1 (testers in the bounded-degree model): *Let $d \in \mathsf{N}$ be fixed and Π be a property of graphs with maximum degree at most d. We denote the restriction of Π to N-vertex graphs by Π_N. A randomized oracle machine T is called a* tester *for Π if the following two conditions hold:*

1. *For every $N \in \mathsf{N}$ and $\epsilon \in [0, 1]$, on input (N, ϵ) and when given oracle access to any $G \in \Pi_N$ the machine T accepts with probability at least 2/3; that is, $\Pr[T^G(N, \epsilon) = 1] \geq 2/3$.*

2. *For every $N \in \mathsf{N}$ and $\epsilon \in [0, 1]$, and every N-vertex graph G that is ϵ-far from Π_N, it holds that $\Pr[T^G(N, \epsilon) = 1] \leq 1/3$, where $G = ([N], E)$* is ϵ-far from *Π_N if for every $G' = ([N], E') \in \Pi_N$ it holds that the symmetric difference of E and E' contains more than $\epsilon \cdot dN/2$ elements.*[11]

In case the first condition holds with probability 1, we say that T has one-sided error*. Otherwise, we say that T has* two-sided error.

[11] Alternatively, representing G by $g : [N] \times [d] \to \{0, 1, ..., N\}$ (resp., G' by $g' : [N] \times [d] \to \{0, 1, ..., N\}$) we may require that $\Pr_{x \in [N] \times [d]}[g(x) \neq g'(x)] > \epsilon$. Note that in this case, for each G we should consider all legitimate representations of G' as a function g'.

Throughout our study, the degree bound $d \geq 3$ is a constant,[12] and sometimes O/Omega-notions hide a dependence on d. The query and time complexities of testers are stated as functions of the graph size, N, and the proximity parameter, ϵ.

Notation. For a graph $G = ([N], E)$, we denote the set of neighbors of $v \in [N]$ (in G) by $\Gamma_G(v)$; that is, $\Gamma_G(v) = \{u \in [N] : \{u, v\} \in E\}$.

Terminology. We stress that by a cycle in a graph $G = ([N], E)$ we mean a sequence of vertices $(v_1, \ldots, v_t, v_{t+1})$ such that $v_1 = v_{t+1}$ and for every $i \in [t]$ it holds that $\{v_i, v_{i+1}\} \in E$; that is, (u, v, w, v, u) (or even (u, v, u)) is considered a cycle. A simple cycle is a cycle as above in which $t \geq 3$ and $|\{v_i : i \in [t]\}| = t$.

3 Testing Cycle-Freeness

As stated in the introduction, we reduce testing cycle-freeness to testing bipartiteness. Recall that we consider bounded-degree graphs, where the degree bound d is assumed to be a constant (for the general case, see Appendix 8). We stress that the reduction is randomized and local (i.e., operations in the resulting graph are easily implemented via operations in the original graph). Wishing to avoid a general definition of (randomized) local reductions, we explicitly present the tester obtained by it.

For a fixed graph $G = ([N], E)$ and function $\tau : E \to \{1, 2\}$, we denote by G_τ the graph obtained from G by replacing each edge $e \in E$ such that $\tau(e) = 2$ by a 2-edge path (with an auxiliary intermediate vertex). Each edge $e \in E$ such that $\tau(e) = 1$ remains an edge in G_τ. That is, the graph $G_\tau = (V_\tau, E_\tau)$ is defined as follows:

$$V_\tau \stackrel{\text{def}}{=} [N] \cup \{a_e : e \in E \wedge \tau(e) = 2\} \quad (1)$$

$$E_\tau \stackrel{\text{def}}{=} \{e : e \in E \wedge \tau(e) = 1\} \cup \{\{u, a_e\}, \{a_e, v\} : e = \{u, v\} \in E \wedge \tau(e) = 2\} \quad (2)$$

We now turn to the tester itself. The tester emulates the execution of the bipartiteness testing algorithm [GR99] on G_τ by performing queries to G. The bipartiteness testing algorithm performs two types of operations: selecting a vertex uniformly at random and taking random walks by querying vertices on their neighbors. Thus the execution of the tester boils down to emulating these operations, as described next. We use $\perp$ to denote "null". If we query for the i^{th} neighbor of vertex v and no such neighbor exists, we get $\perp$ as an answer.

Algorithm 3.1 (the cycle-freeness tester): *Given input graph $G = ([N], E)$, the tester selects uniformly at random a function $\tau : E \to \{1, 2\}$ and invokes a bipartite tester, denoted T, on the graph G_τ, emulating its operations as follows.*

1. *If T wishes to select a random vertex in G_τ, then the tester selects uniformly a vertex $v \in [N]$, outputs v with probability $1/(d+1)$, and otherwise selects each neighbor u of v with probability $1/(2(d+1))$ and outputs $a_{\{u,v\}}$ if $\tau(\{u, v\}) = 2$. Indeed, this process is guaranteed to output a uniformly distributed vertex with probability at least $1/(d+1)$, and in case of failure it is repeated* (up to $O(\log N)$ times).

[12] There is little point in considering $d \leq 2$, because in this case the problems we consider are either trivial (i.e., for $d = 1$) or very easy (i.e., for $d = 2$). Specifically, for $d = 2$, one can test C_k-minor-freeness by selecting a random vertex and exploring its $k/2$-neighborhood.

2. *If T queries for the i^{th} neighbor of vertex $v \in [N]$, then the tester queries for the i^{th} neighbor of v, and answers accordingly. That is, if the answer was $\perp$, then $\perp$ is given as answer to T, whereas if the answer was u, then u is given to T if $\tau(\{u,v\}) = 1$ and $a_{\{u,v\}}$ is given otherwise.*

   Finally, if T queries for the i^{th} neighbor of a vertex $a_{\{u,v\}}$ and $u < v$ then the tester answer with u if $i = 1$, with v if $i = 2$, and with $\perp$ if $i > 2$.

When T halts, the current tester halts with the same verdict.

Furthermore, if the bipartite tester provides an odd-length cycle in G_τ, then we can easily obtain a corresponding cycle in G (by contracting the 2-vertex paths that appear on it into single edges). We note that the random function $\tau : E \to \{1,2\}$ can be selected "on the fly" (i.e., whenever we need the value of $\tau(e)$, if this value is still undefined then we select it uniformly in $\{1,2\}$ and store it for possible future use).

Using the bipartite tester of [GR99] in the role of T, we obtain an algorithm of the desired complexity that always accepts a cycle free graph (see below). Our analysis is thus focused on the case that G is not cycle-free.

Lemma 3.2 (analysis of the reduction):

1. *If G is cycle-free then, for every choice of $\tau : E \to \{1,2\}$, the graph G_τ is bipartite.*
2. *If G is not cycle-free then, with probability at least $1/2$ over the random choice of $\tau : E \to \{1,2\}$, the graph G_τ is not bipartite.*
3. *There exists a universal constant $c > 0$ such that if G is ϵ-far from being cycle free then, with probability at least $1 - \exp(-c\epsilon dN)$ over the random choice of $\tau : E \to \{1,2\}$, the graph G_τ is $c \cdot \epsilon$-far from being bipartite.*

Proof: The first item follows from the fact that if G is cycle-free then, for every $\tau : E \to \{1,2\}$, the graph G_τ is also cycle-free, and thus bipartite. The second item follows by observing that any cycle in G is transformed with probability $1/2$ to an odd-length cycle in G_τ. Turning to the last item, we consider an arbitrary graph G that is not cycle-free. Denoting by Δ the actual number of edges (not its fraction) that needs to be omitted from G in order to obtain a cycle-free graph, we shall show that (with probability at least $1 - \exp(-c\epsilon dN)$) the number of edges that needs to be omitted from G_τ in order to obtain a bipartite graph is at least $c \cdot \Delta$.

We may assume, without loss of generality, that the graph G is connected, or else we apply the claim separately to each connected component that is not cycle-free. We may also assume that G has no vertices of degree 1, since truncating such vertices maintains the value of Δ (i.e., the absolute distance from being cycle-free) as well as the (distribution of) the number of edges that have to be removed to make G_τ bipartite. Finally, except in the case that G is a simple cycle, we may also assume that there are no vertices of degree 2, since we can contract paths that only contain intermediate vertices of degree 2 to a single edge, while again preserving Δ as well as the (distribution of) the number of edges that have to be removed to make G_τ bipartite.[13]

[13] The latter assertion follows from the fact that the distribution of the parity of the path-lengths in G_τ is maintained (i.e., both the original path and the contracted path in G_τ have odd/even length with probability $1/2$. We also mention that the contracted graph G may contain self-loops and parallel edges, but the rest of the argument holds in this case too. We stress that the contracted graph is merely a mental experiment for proving the current lemma.

In light of the forgoing, we consider a connected graph $G = ([N], E)$ in which each vertex has degree at least 3. It follows that $\Delta = |E| - (N-1) > N/2$. We shall prove that, with high probability over the choice of τ, more than $c \cdot \Delta$ edges must be omitted from the graph G_τ in order to obtain a bipartite graph.

For each $E' \subset E$ of size $c\Delta$, we consider the probability that G'_τ is bipartite, where G'_τ denotes the graph obtained from G_τ by omitting the edges of G_τ that replace the edges in E' (or alternatively applying the randomized reduction to the graph $G' = ([N], E \setminus E')$).[14] Note that G_τ is at (absolute) distance at most $c\Delta$ from being bipartite if and only if there exists a set E' of size $c\Delta$ such that G'_τ is bipartite. Thus, we have

$$\begin{aligned} p &\stackrel{\text{def}}{=} \Pr_\tau[\exists E' \subset E \text{ such that } |E'| = c\Delta \text{ and } G'_\tau \text{ is bipartite}] \\ &\leq \sum_{E' \subset E : |E'| = c\Delta} \Pr_\tau[G'_\tau \text{ is bipartite}] \\ &\leq \binom{|E|}{c\Delta} \cdot 2^{N-1} \cdot 2^{-(|E| - c\Delta)} \end{aligned}$$

where the second inequality is due to considering all possible 2-partitions of $[N]$ and noting that for each edge e in $E \setminus E'$ and each 2-partition π, with probability $1/2$ over the choice of $\tau(e) \in \{1, 2\}$ a violation is cause by e. Specifically, if $\pi(u) = \pi(v)$ and $\tau(\{u, v\}) = 1$, then the edge $\{u, v\}$ violates the 2-partition π, and ditto if $\pi(u) \neq \pi(v)$ and $\tau(\{u, v\}) = 2$. Note that the hypothesis that G is (connected and is) at (absolute) distance Δ from being cycle-free implies that $|E| = (N-1) + \Delta$. Now, substituting $|E|$ by $(N-1) + \Delta$ and using $\Delta \geq N/2$ (and $c < 1/2$), we get

$$\begin{aligned} p &< \binom{N + \Delta}{c\Delta} \cdot 2^{-(1-c)\Delta} \\ &< \binom{3\Delta}{c\Delta} \cdot 2^{-\Delta/2} \end{aligned}$$

which vanishes exponentially in Δ provided that $c > 0$ is a sufficiently small constant. ■

Conclusion. Combining Lemma 3.2 with the straightforward observations preceding it, we conclude that *Algorithm 3.1 is a one-sided error tester for cycle-freeness, and its complexity is* $\widetilde{O}(\mathrm{poly}(1/\epsilon) \cdot \sqrt{N})$. This establishes Theorem 1.5.

4 Testing C_4-Minor-Freeness

As a warm-up towards testing C_k-minor-freeness, for any $k \geq 3$, we present the treatment of the special case of $k = 4$. We actually reduce the task of *testing C_4-minor-freeness* to the task of *testing C_3-minor-freeness*. The reduction is summarized in the following construction.

Construction 4.1 (the reduction): *Given a graph $G = ([N], E)$ (of max degree d), we* (locally) *construct the auxiliary graph $G' = ([N] \cup T, E')$ such that T contains the vertex $\bigtriangledown_{u,v,w}$ (referred to as a "triangle" vertex), if and only if $\{u, v\}, \{v, w\}, \{w, u\} \in E$ and*

$$E' = \left(E \setminus \left(\bigcup_{u,v,w : \bigtriangledown_{u,v,w} \in T} \{u, v\}\right)\right) \cup \left\{\{u, \bigtriangledown_{u,v,w}\} : \bigtriangledown_{u,v,w} \in T\right\}. \qquad (3)$$

[14]Note however that if G'_τ is bipartite then G_τ can be made bipartite by omitting $|E'|$ (rather $2|E'|$) edges, since it suffices to omit a single edge from each path in G_τ that replaced an edge in E'.

Specifically, the set of neighbors of $v \in [N]$ in G', denoted $\Gamma_{G'}(v)$, consists of the following elements of $[N] \cup T$.

1. *Neighbors of v in G that do not reside in G on a triangle together with v; that is, $u \in \Gamma_G(v)$ is in $\Gamma_{G'}(v)$ if and only if $\Gamma_G(u) \cap \Gamma_G(v) = \emptyset$.*

2. *Each triangle that contains v in G; that is, $\bigtriangledown_{u,v,w}$ is in $\Gamma_{G'}(v)$ if and only if $u, w \in \Gamma_G(v)$ and $\{w, u\} \in E$.*

The set of neighbors of $\bigtriangledown_{u,v,w} \in T$ equals $\{u, v, w\}$. Noting that $d + \binom{d}{2} \leq d^2$, we view G' as a graph of maximal degree d^2.

For an illustration of Construction 4.1 see Figure 1. Note that given any $v \in [N]$, we can easily

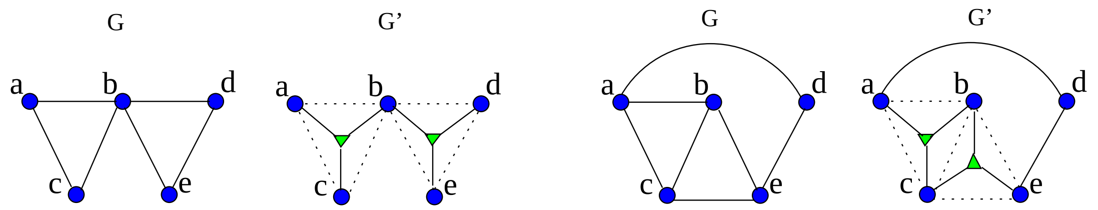

Figure 1: An illustration for Construction 4.1. On the left, G is C_4-minor free, and indeed G' is cycle-free; while on the right, G is not C_4-minor free, and G' contains cycles (but no cycles of length 3 (triangles).)

determine its neighbors in G' by checking the foregoing conditions. Similarly, for every u, v, w, we can easily determine whether $\bigtriangledown_{u,v,w}$ is in G'. Lastly, note that we can select a vertex of G' uniformly by using the following procedure.

1. Select uniformly $v \in [N]$.

2. Select one of the following two instructions at random with equal probability.

   (a) (Generating a vertex of G):
   Output v with probability d^{-2}.

   (b) (Generating a triangle):
   Select uniformly $u, w \in \Gamma_G(v)$. If $\{u, w\} \in E$, then output $\bigtriangledown_{u,v,w}$ with probability $p_v = d^{-2} \cdot |\Gamma_G(v)|^2/6$.

   In all the other cases, there is no output.

Thus, this process outputs each vertex of G with probability $N^{-1} \cdot 0.5 \cdot d^{-2} = d^{-2}/2N$, and outputs each $\bigtriangledown_{u,v,w} \in T$ with probability $\sum_{x \in \{u,v,w\}} N^{-1} \cdot 0.5 \cdot 2|\Gamma_G(x)|^{-2} \cdot p_x = d^{-2}/2N$. Since there are at least N vertices in G', the probability that the process does not output *any* vertex in G' is at most $(1 - d^{-2})$. If we repeat the process $\Theta(\log N)$ times (recall that d is assumed to be a constant), then the probability that we get no output is $1/\text{poly}(N)$. Since the total size of the sample needed is $o(N)$, by a union bound, the probability that this occurs at any step of the algorithm, is negligible, and this can be accounted for in the one-sided error probability by letting the algorithm accept in case sampling fails.

Algorithm 4.2 (the C_4-minor-freeness tester): *Given input graph $G = ([N], E)$, the tester emulates the execution of Algorithm 3.1 on the graph $G' = ([N] \cup T, E')$ as defined in Construction 4.1. In the emulation, vertices of G' are selected at random and their neighbors are explored on the fly, as detailed above.*

The analysis of Algorithm 4.2 reduces to an analysis of Construction 4.1.

Claim 4.3 *If G is C_4-minor-free, then G' is cycle-free.*

Proof: We first give a high-level idea of the proof and then give a detailed argument. By the hypothesis, the only simple cycles in G are triangles, and they are replaced in G' by stars centered at auxiliary vertices. Specifically, the triangle $\{u, v, w\}$ (i.e., the edges $\{u, v\}, \{v, w\}, \{w, u\}$) is replaced by a star-tree centered at $\bigtriangledown_{u,v,w}$ and having the leaves u, v, w. Note that this replacement can form no simple cycles in G', because the simple paths in G' correspond to simple paths in G (where the sub-path v— $\bigtriangledown_{u,v,w}$ —w corresponds to the edge v—w).

The corresponding detailed argument proceeds as follows. Assume, contrary to the claim, that there exists a simple cycle $\psi' = v_1$—v_2–$\cdots$–v_t—$v_{t+1} = v_1$ in G'. Consider replacing each length-2 subpath u—$\bigtriangledown_{u,w,x}$—w in ψ' by the edge (in G) between u and w (where this edge exists because u and w belong to a common triangle and $u \neq w$). Since, by construction of G', there are no edges in G' between *triangle* vertices, this way we obtain a cycle in G, which we denote by ψ. We next show that ψ is a simple cycle of length greater than 3, and we reach a contradiction to the hypothesis that G is C_4-minor-free.

We first verify that the length of ψ is greater than 2. This is true because otherwise, the cycle ψ' is either of the form u—$\bigtriangledown_{u,w,x}$—w—u, or it is of the form u—$\bigtriangledown_{u,w,x_1}$—$w$—$\bigtriangledown_{u,w,x_2}$—$u$. In the first case ψ' contains an edge $\{w, u\}$ of a triangle in G, which is not possible by construction of G'. In the second case, since ψ' is simple (so that $x_1 \neq x_2$), there is a simple 4-cycle u—x_1—w—x_2—u in G (contradicting the hypothesis that G is C_4-minor-free). It follows that ψ is a simple cycle and it remains to verify that its length is greater than 3.

Suppose that the length of ψ is 3, that is, $\psi = u$—w—v—u is a triangle in G. It follows that none of the edges $\{u, w\}, \{w, v\}, \{v, u\}$ belong to G' and therefore, $\psi' = u$— $\bigtriangledown_{u,w,x_1}$ —w— $\bigtriangledown_{w,v,x_2}$ —v— $\bigtriangledown_{v,u,x_3}$ —u, where the triangles are distinct and hence at least one of them does not equal $\bigtriangledown_{u,w,v}$. But this implies that there exists a simple 4-cycle in G (contradicting the hypothesis that G is C_4-minor-free). ■

Claim 4.4 *If G is ϵ-far from being C_4-minor-free, then G' is $\Omega(\epsilon)$-far from being cycle-free, where the Omega-notation hides a polynomial in d.*

Proof: Suppose that G' is δ-close to being cycle-free, where the distance refers to the degree bound of G', which is d^2. Let R' be a set of at most $\delta \cdot d^2 \cdot (N + |T|)/2$ edges such that removing R' from G' yields a cycle-free graph, $([N] \cup T, E' \setminus R')$. Let $R \subseteq E$ be a set of edges that consists of (1) all edges of E that are in R', and (2) each edge $\{u, v\} \in E$ such that $\{u, \bigtriangledown_{u,v,w}\}$ is in R'. Hence, $|R| \leq 2|R'| < \delta \cdot d^4 N$, where we use $|T| \leq \binom{d}{2} \cdot N$. We next prove that removing R from G yields a graph that is C_4-minor-free, and it follows that G is $2d^2\delta$-close to being C_4-minor-free.

Assume, contrary to the claim, that for some $t \geq 4$ there exists a *simple* cycle v_1—v_2–$\cdots$–v_t—v_1 in the resulting graph (i.e., in the graph $([N], E \backslash R)$). We consider the corresponding (not necessarily simple) cycle in the graph $([N] \cup T, E' \setminus R')$:

Case 1: If the edge $\{v_i, v_{i+1}\} \in E \setminus R$ is not a part of any triangle in G, then $\{v_i, v_{i+1}\} \in E' \setminus R'$, because $\{v_i, v_{i+1}\}$ is an edge of G' and it cannot be in R' (since this would imply that

$\{v_i, v_{i+1}\} \in R$). In this case, we just use the edge $\{v_i, v_{i+1}\}$ on the cycle in the graph $([N] \cup T, E' \setminus R')$.

Case 2: If the edge $\{v_i, v_{i+1}\} \in E \setminus R$ is part of a triangle v_i, v_{i+1}, w (in G), then $\{v_i, \bigtriangledown_{v_i,v_{i+1},w}\} \in E' \setminus R'$ and $\{v_{i+1}, \bigtriangledown_{v_i,v_{i+1},w}\} \in E' \setminus R'$, because both pairs are edges of G' and cannot be in R' (since this would imply that $\{v_i, v_{i+1}\} \in R$). In this case, we replace the edge $\{v_i, v_{i+1}\} \in E \setminus R$ by the length-two-path v_i— $\bigtriangledown_{v_i,v_{i+1},w}$ —v_{i+1} (in the graph $([N] \cup T, E' \setminus R')$).

Observe that the "triangle" vertices used in Case (2) need not be distinct, but they can collide only when they refer to three consecutive vertices on the original t-cycle (i.e., if $\bigtriangledown_{v_i,v_{i+1},w_1} = \bigtriangledown_{v_j,v_{j+1},w_2}$, for $i < j$, then $v_j = v_{i+1}$ must hold, and $w_1 = v_{j+1} = v_{i+2}$ follows). Such collisions can be eliminated at the cost of omitting a single "non-triangle" vertex (i.e., the path v_i— $\bigtriangledown_{v_i,v_{i+1},v_{i+2}}$ —v_{i+1}— $\bigtriangledown_{v_i,v_{i+1},v_{i+2}}$ —v_{i+2} is replaced by the path v_i— $\bigtriangledown_{v_i,v_{i+1},v_{i+2}}$ —v_{i+2}). Thus, we derive a simple cycle of length at least $t \geq 4$ in the graph $([N] \cup T, E' \setminus R')$ (since we have a "triangle" vertex per each omitted "non-triangle" vertex). This contradicts the hypothesis that $([N] \cup T, E' \setminus R')$ is cycle-free, and so the claim follows. ■

Conclusion. Combining Claims 4.3 and 4.4, we conclude that *there exists a one-sided error tester of complexity is $\widetilde{O}(\mathrm{poly}(1/\epsilon) \cdot \sqrt{N})$ for C_4-minor-freeness.*

5 Testing C_k-Minor-Freeness, for any $k \geq 4$

In this section we show that, for any $k \geq 4$, the task of *testing C_k-minor-freeness* reduces to the task of *testing C_3-minor-freeness*. The reduction extends the ideas underlying the reduction of *testing C_4-minor-freeness* to *testing C_3-minor-freeness* (as presented in Section 4).

The basic idea of the reduction is replacing simple cycles that have length smaller than k by stars. Actually, we replace certain subgraphs that contain such cycles by stars. We start by defining the class of (induced) subgraphs that we intend to replace by stars. These subgraphs (or rather their vertex sets) will be called *spots*. Below, the term 2-connectivity means *2-vertex connectivity*; that is, a graph is called 2-connected if every two vertices in the graph can be connected by two vertex-disjoint paths.

Definition 5.1 (spots): *A set $S \subseteq [N]$ is called a k-spot of the graph $G = ([N], E)$ if the following three conditions hold:*

1. *The subgraph induced by S, denoted G_S, contains no simple cycle of length $\geq k$; that is, G_S is C_k-minor-free.*

2. *The subgraph induced by S is 2-connected.*

3. *For every $u \neq v \in S$, either u and v are not connected by any path that is external to G_S or the length of every such external path is at least $\ell(k) \stackrel{\text{def}}{=} 2k$. Here, by a path* external to G_S *we mean a path that does not use any edge that is incident to a vertex in S* (i.e., all intermediate vertices of the path belong to $[N] \setminus S$).

For example, every 4-spot of G induces a triangle in G, whereas the set of possible subgraphs induced by 5-spots of G consists of the following graphs: the 4-cycle (i.e., C_4), the 4-cycle augmented by a chord, the 4-clique (i.e., K_4), and the graphs $K_{2,n}$ and $K'_{2,n}$ for every $n \geq 3$, where $K'_{2,n}$ is the graph $K_{2,n}$ augmented by a single edge that connects the two vertices on the small side.[15] (Indeed, in Section 4 we essentially used a relaxed notion of a 4-spot in which the third condition was not required.)

5.1 Some basic facts regarding spots

Since k is fixed throughout the rest of our discussion, we may omit it from the notations and refer to k-spots as spots. A few basic properties of spots are listed below.

Claim 5.2 *If S is a k-spot of G, then the diameter of G_S is smaller than $k/2$.*

It follows that $|S| < \sum_{i=0}^{k/2} d^i < 2d^{k/2} < d^{k-1}$ (since $k \geq 4$ and $d \geq 3$).[16]

Proof: Otherwise, consider $u, v \in S$ such that the distance between u and v in G_S is at least $k/2$. Since G_S is 2-connected, there exists a simple cycle in G_S that passes through both u and v, and it follows that this cycle has length at least k, which contradicts the hypothesis that G_S is C_k-minor-free. ■

Note that, for any spot S and every three distinct vertices $u, v, w \in S$, the subgraph G_S contains a simple path that goes from u to v via w. This hold by the very fact that G_S is 2-connected (i.e., the second condition in Definition 5.1). By Claim 5.2 the length of this path is less than d^{k-1}. As we shall show next, a much better bound follows by using the fact that G_S is C_k-minor-free (i.e., the first condition in Definition 5.1),

Claim 5.3 *For every k-spot S and distinct vertices $u, v, w \in S$, the subgraph G_S contains a simple path of length at most $2k-1$ that goes from u to v via w.*

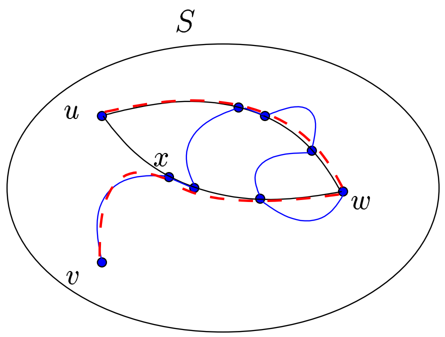

Figure 2: An illustration for the proof of Claim 5.3. The jotted line is the path between u and v that passes through w.

Proof: We just take a closer look at the standard proof that the fact that a graph is 2-connected implies the existence of a u–$\cdots$–w–$\cdots$–v path (for every three vertices u, v, w in the graph). The

[15] Recall that $K_{m,n}$ denotes the complete bipartite graph with m vertices on one side and n vertices on the other side; that is, $K_{m,n} = ([m+n], \{\{i, m+j\} : i \in [m], j \in [n]\})$.

[16] We mention that there may exists spots of size $d^{(k-1)/2}$. Consider, for example, a graph that consists of two copies of a depth $(d-1)$-ary tree of depth $(k-1)/2$ such that each vertex in one tree is connected to its mirror vertex in the second tree. To see that this graph is C_k-minor-free, consider the correspondence between cycles on this graphs and traversals of parts of the original tree, and note that simple cycles correspond to traversals in which each edge is used at most twice. Since such traversals have length at most twice the depth of the tree, the claim follows.

proof starts by considering two different vertex-disjoint u–$\cdots$–w paths, and an arbitrary path between v and w. In the current case (i.e., by C_k-minor-freeness), we may assume that the total length of the first two paths is smaller than k. Similarly, without loss of generality, the length of the third path is smaller than k. Proceeding as in the standard proof, we ask whether the third path (i.e., the v–$\cdots$–w path) intersects both the u–$\cdots$–w paths. If the answer is negative, then we are done (as we obtain the desired simple path by concatenating the path v–$\cdots$–w to the w–$\cdots$–u path that does not intersect it). Otherwise, let x be the "closest to v" vertex on the path v–$\cdots$–w that appear on either of the u–$\cdots$–w paths; that is, x is on one of the u–$\cdots$–w paths and the sub-path v–$\cdots$–x (of the path v–$\cdots$–w) contains no vertex from either the u–$\cdots$–w paths. Note that $x = v$ is possible (but $x = w$ is not), and assume, w.l.o.g., that x resides on the first u–$\cdots$–w path. Then, consider the path obtained by combining the following three path segments: (1) the segment v–$\cdots$–x of the path v–$\cdots$–w, (2) the segment x–$\cdots$–w of the first u–$\cdots$–w path, and (3) the second u–$\cdots$–w path. Note that the total length of this path is at most $2(k-1)$ (i.e., the total length of the three paths), and that the three segment do not intersect (since the v–$\cdots$–x segment does not intersect the x–$\cdots$–w segment nor the u–$\cdots$–w path by choice of v). For an illustration of the argument, see Figure 2. ■

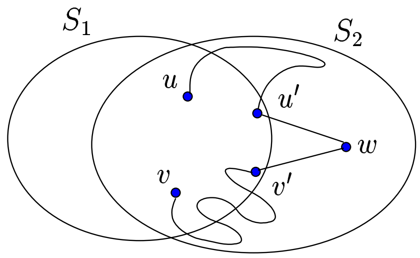

Figure 3: An illustration for the proof of Claim 5.4.

Claim 5.4 *If $S_1 \neq S_2$ are k-spots of G, then $|S_1 \cap S_2| \leq 1$.*

It follows that the number of spots in a graph G is upper-bounded by the number of edges in G, because every spot S that contains v must also contain at least two of v's neighbors whereas spots that contain v may not share any other vertex. Thus, vertex v may participate in at most $|\Gamma(v)|/2$ spots.

Proof: Otherwise, consider $u, v \in S_1 \cap S_2$ and $w \in S_2 \setminus S_1$. By Claim 5.3, the subgraph G_{S_2} contains a simple path of length at most $2k-1$ that goes from u to v via w. Let u' (resp., v') be the last (resp., first) vertex of S_1 that appears on this path before reaching w (resp., after leaving w). Then, we get a simple path (in G) from $u' \in S_1$ to $v' \in S_1 \setminus \{u'\}$ such that this path contains only intermediate vertices of $S_2 \setminus S_1$. Recalling that this path has length at most $2k-1$, we reach a contradiction to the hypothesis that S_1 is a k-spot (specifically to the third condition of Definition 5.1). For an illustration of the argument, see Figure 3. ■

Claim 5.5 *Each simple cycle in any C_k-minor-free graph G is a subset of some k-spot of G.*

Proof: Consider the following iterative process of constructing a spot S that contains the aforementioned cycle. Initially, we set S to equal the set of vertices that reside on this cycle. Clearly, this set S satisfies the first two conditions of the definition of a spot (i.e., Definition 5.1), which is an invariant that we shall maintain throughout the iterative process. If the current S satisfies also the third condition of the definition of a spot, then S is a spot and we are done. Otherwise, we consider a simple path external to S that connects two of its vertices; that is, the vertices $u, v \in S$. Adding this path to S we obtain a new set that satisfies Condition 1 (since G is C_k-minor-free). To see that the new set satisfies Condition 2, we need to show that there exist two disjoint paths between each pair of vertices that are not both in S.

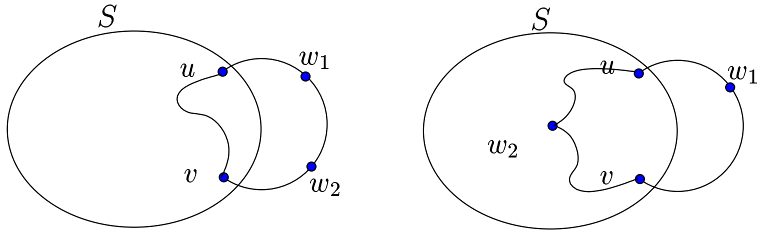

Figure 4: An illustration for the proof of Claim 5.5.

In the case that w_1 and w_2 are both new vertices (which reside on the aforementioned S-external path), we connect them by the direct path that resides outside of S as well as by a simple path that (wlog) connects w_1 to u (via the external path), connects u and v via S, and connects v and w_2 (via the external path). In the case that w_1 is new but $w_2 \in S$, we use the external path to connect w_1 to u and v, respectively, and use the fact that there are vertex disjoint paths in G_S that connect u and v to w_2. For an illustration see Figure 4. ■

5.2 The actual reduction

Using these facts, we are ready to present our reduction.

Construction 5.6 (the reduction): *Given a graph* $G = ([N], E)$ (of max degree d), *we* (locally) *construct the auxiliary graph* $G' = ([N] \cup \{\langle S \rangle : S \in \mathcal{S}\}, E')$ *such that* $\mathcal{S}$ *is the the set of all spots of* G *and*

$$E' = \left(E \setminus \left(\bigcup_{S \in \mathcal{S}} \{\{u, v\} : u, v \in S\}\right)\right) \cup \{\{v, \langle S \rangle\} : S \in \mathcal{S}, v \in S\}. \tag{4}$$

Specifically, the set of neighbors of $v \in [N]$ *in* G'*, denoted* $\Gamma_{G'}(v)$*, consists of the following elements of* $[N] \cup \{\langle S \rangle : S \in \mathcal{S}\}$.

1. *Neighbors of* v *in* G *that do not reside in any spot together with* v*; that is,* $u \in \Gamma_G(v)$ *is in* $\Gamma_{G'}(v)$ *if and only if* $\{u, v\}$ *is not a subset of any* $S \in \mathcal{S}$.
2. *Each spot that contains* v *in* G*; that is,* $\langle S \rangle$ *is in* $\Gamma_A(v)$ *if and only if* $S \in \mathcal{S}$ *and* $v \in S$.

For any $S \in \mathcal{S}$*, the set of neighbors of* $\langle S \rangle$ *in* G' *equals* S*. Recalling that each* $S \in \mathcal{S}$ *has size at most* d^{k-1}*, we view* G' *as a graph of maximal degree* d^{k-1}.

Observe that the set of spots that contain a vertex $v \in [N]$ is determined by the $(k + \ell(k))$-neighborhood of v in G, where the t-neighborhood of v contains all vertices that are at distance at most t from v. Thus, we can determine the set of neighbors of each vertex in G'. We note that the process of determining the spots that contain a vertex may fail if a cycle of length at least k is encountered. In such a case the algorithm can clearly reject. Lastly, note that we can select a vertex of G' uniformly by using the following procedure.

1. Select uniformly $v \in [N]$.
2. Select one of the following two instructions at random with equal probability.
   (a) (Generating a vertex of G):
   Output v with probability $1/d$.
   (b) (Generating a spot):
   Select uniformly a spot S that contain v (i.e., $S \in \mathcal{S}_v$), and output $\langle S \rangle$ with probability $p_v = \frac{|\mathcal{S}_v|}{d|S|}$, where $\mathcal{S}_v \stackrel{\text{def}}{=} \{S \in \mathcal{S} : v \in S\}$.

   In all the other cases, there is no output.

Thus, this process output each vertex of G with probability $N^{-1} \cdot 0.5 \cdot d^{-1} = 1/(2dN)$, and outputs each spot $\langle S \rangle \in \mathcal{S}$ with probability $\sum_{v \in S} N^{-1} \cdot 0.5 \cdot |\mathcal{S}_v|^{-1} \cdot p_v = 1/(2dN)$.

Algorithm 5.7 (the C_k-minor-freeness tester): *Given input graph $G = ([N], E)$, the tester emulates the execution of Algorithm 3.1 on the graph G' as defined in Construction 5.6. In the emulation, vertices of G' are selected at random and their neighbors are being explored on the fly, as detailed above.*

The analysis of Algorithm 5.7 reduces to an analysis of Construction 5.6.

Claim 5.8 (yes-instances): *If G is C_k-minor-free, then G' is cycle-free.*

Proof: Suppose, contrary to the claim, that v_1—v_2−$\cdots$−v_t—v_1 is a simple cycle in G'. We consider two cases.

Case 1: *All v_i's are vertices of G.* In this case, the edges $\{v_i, v_{i+1}\}$ in G' must be edges of G (since the only edges in G' that are not edges in G are incident to spot-vertices). On the other hand $t < k$ must hold, because G is C_k-minor-free. But this yields a contradiction, because, by Claim 5.5, the set $\{v_i : i \in [t]\}$ must be a subset of some spot of S, which means that none of the edges $\{v_i, v_{i+1}\}$ may exist in G'.

Case 2: *Some v_i represents a spot of G.* Let $v_i = \langle S \rangle$, for some $S \in \mathcal{S}$. Then $v_{i+1}, v_{i-1} \in S$. Now, consider a minimal sub-path of $v_{i+1}, ..., v_t, v_1, ..., v_{i-1}$ that starts in a vertex of S, denoted u, and ends in a vertex of S, denoted v. That is, we consider a sub-path that starts and ends in vertices of S, but has no intermediate vertices in S. This sub-path (in G') cannot consist of a single edge (because the edge $\{u, v\} \subset S$ cannot appear in G'), it cannot contain the vertex $\langle S \rangle$ (because $\langle S \rangle$ already appears as v_i), and it cannot be a 2-path that goes through another spot (because, by Claim 5.4, no other spot may contain both u and v). Since this path may not contain intermediate vertices in S, and since spot-vertices cannot be adjacent in G', it follows that this path must contain a vertex $w \in [N] \setminus S$. That is, we get a path in G' that goes from u to v via w, without passing through any vertex in S.

We now obtain a corresponding path in G; that is, a path in G that goes from u to v via w, without passing through any vertex in S. This is done by replacing any length-2 subpath u'—$\langle S' \rangle$—v' (in G') by a sub-path u'−$\cdots$−v' (in G) that does not pass through S, where the latter path exists by the fact that $u', v' \in S'$ are connected by vertex-disjoint paths (internal to S') such that their intersection with S contains at most a single vertex (see Claim 5.4). It follows that G itself contains a path between u and v that passes through w and does not pass through S, where $u, v \in S$ but $w \notin S$. Thus, G itself contains a simple (non-edge) path between u and v that does not pass through S (i.e., an external path). By the third condition in Definition 5.1, the length of this external path is at least $\ell(k) > k$, but this contradicts the hypothesis that G is C_k-minor-free (because u and v are connected in G_S and $\ell(k) \geq k$, yielding a simple cycle of length at least k).

The claim follows. ■

Claim 5.9 (no-instances): *If G is ϵ-far from being C_k-minor-free, then G' is $\Omega(\epsilon)$-far from being cycle-free, where the Omega-notation hides a d^k factor.*

Proof: Suppose that G' is δ-close to being cycle-free, where the distance refers to the degree bound of G', which is d^{k-1}. Recall that $|\mathcal{S}| \leq |E| \leq dN/2$. Let R' be a set of at most $\delta \cdot d^{k-1}(N+|\mathcal{S}|)/2 < \delta \cdot d^k N/2$ edges such that removing R' from G' yields a cycle-free graph. Let $R \subseteq E$ be a set of edges that consists of (1) all edges of E that are in R', and (2) each edge $\{v, w\} \in E$ such that $\{v, \langle S \rangle\}$ is in R'. Hence, $|R| \leq d|R'| < \delta \cdot d^{k+1}N/2$. We next prove that removing R from G yields a graph that is C_k-minor-free, and it follows that G is $\delta \cdot d^k$-close to being C_k-minor-free.

Suppose, contrary to the claim, that for $t \geq k$ there exists a simple cycle $v_1 — v_2 - \cdots - v_t — v_1$ in the resulting graph (i.e., in the graph $([N], E \setminus R)$). We first show that there exists a corresponding (not necessarily simple) cycle in $E' \setminus R'$. Specifically, for each $\{v_i, v_{i+1}\} \in E \setminus R$, we consider two cases.

Case 1: *This edge is not a subset of any spot in G.* In this case, $\{v_i, v_{i+1}\} \in E' \setminus R'$, because this edge is in E' and cannot be in R' (or else it would have been in R). So we just use this edge in the cycle (in $E' \setminus R'$).

Case 2: *This edge is a subset of a spot S in G.* In this case, $\{v_i, \langle S \rangle\}, \{v_{i+1}, \langle S \rangle\} \in E' \setminus R'$, because both these edges are in E' and cannot be in R' (or else $\{v_i, v_{i+1}\}$ would have been in R). In this case, we replace the edge $\{v_i, v_{i+1}\} \in E \setminus R$ by the length-two-path $v_i — \langle S \rangle — v_{i+1}$.

Thus, we obtain a cycle in $([N] \cup \{\langle S \rangle : S \in \mathcal{S}\}, E' \setminus R')$ that contains the vertices $v_1, ..., v_t \in [N]$ as well as (possibly) some elements in $\{\langle S \rangle : S \in \mathcal{S}\}$. Since the latter elements may appear in multiple copies, the foregoing cycle is not necessarily simple. Note that a simple cycle in $([N] \cup \{\langle S \rangle : S \in \mathcal{S}\}, E' \setminus R')$ yields a contradiction to the hypothesis that this graph is cycle-free, and thus establishes our claim that the graph $([N], E \setminus R)$ is C_k-minor-free. We obtain a simple cycle, in two steps, as follows.

First, we replace every maximal sub-path of the form $v_i — \langle S \rangle — v_{i+1} — \langle S \rangle - \cdots - \langle S \rangle — v_j$, where $j \neq i$ (or else S contains a t-cycle for $t \geq k$), by a length-two path $v_i — \langle S \rangle — v_j$. If the resulting cycle contain distinct spot (representative) vertices, then we are done (since we obtain a simple cycle). Otherwise, we obtain a cycle of the form

$$u_1 - \cdots - u_{t_1} — \langle S_1 \rangle — u_{t_1+1} - \cdots - u_{t_1+t_2} — \langle S_2 \rangle — u_{t_1+t_2+1} - \cdots - u_{t_1+t_2+t_3} — \langle S_3 \rangle \cdots \langle S_m \rangle — u_1$$

where the u_i's are all distinct and adjacent S_i's are distinct (but non-adjacent S_i's may be identical). Next, we consider a sub-path of the foregoing cycle such that the endpoints of this sub-path are two copies of the same spot S and no other spot appears more than once on this sub-path. This sub-path cannot have length two (because adjacent S_i's are distinct), which means that it is actually a simple cycle, and we are done. ■

Conclusion. Combining Claims 5.8 and 5.9 with the straightforward observations preceding it, we conclude that *Algorithm 5.7 is a one-sided error tester for C_k-minor-freeness, and its complexity is* $\widetilde{O}(\mathrm{poly}(d^k/\epsilon)\cdot\sqrt{N})$. This establishes Theorem 1.2.

6 Proof of the Lower Bound

Recall that Goldreich and Ron proved a $\Omega(\sqrt{N})$ query lower bound on the complexity of one-sided error testers for cycle-freeness [GR02, Prop. 4.3]. As stated in the introduction, Benjamini, Schramm, and Shapira [BSS08] mentioned that this lower bound may hold for testing H-minor freeness, for any H that is not a forest. This is indeed the case, as proved next.

Theorem 6.1 *For any fixed H that contains a simple cycle, the query complexity of* one-sided error *testing of H-minor freeness is $\Omega(\sqrt{N})$.*

Indeed, as can been seen easily in the case that H is a single edge, the lower bound does not hold in case H contains no simple cycles. A general study of testing H-minor freeness for any cycle-free H is initiated in Section 7.

Proof: Following the proof of [GR02, Prop. 4.3], we show that for sufficiently large N, with high probability, the random N-vertex graphs considered in [GR02, Sec. 7] are far from being H-minor free. Once this is done, the theorem follows, because it was shown in [GR02, Sec. 7] that a probabilistic machine that makes $o(\sqrt{N})$ queries is unlikely to find a cycle in such a random graph (and so it must accept as otherwise it is not a one-sided error tester). Also note that it suffices to show that, for any fixed k and sufficiently large N, with high probability, such a random graph is far from being K_k-minor free, because containing a minor of the K_k implies containing a minor of any k-vertex graph H.

The random graphs considered in [GR02, Sec. 7] are graphs uniformly chosen in the family $\mathcal{G}_N$ (which is denoted $\mathcal{G}_1^N$ in [GR02]). Each (N-vertex) graph in $\mathcal{G}_N$ consists of the union of a simple N-vertex (Hamiltonian) cycle and a perfect matching of these N vertices. (Indeed, each graph in $\mathcal{G}_N$ is 3-regular.) Furthermore, the cycle is fixed to be $(1, 2, ..., N, 1)$ and so a random graph in $\mathcal{G}_N$ corresponds to a random choice of a perfect matching. Our aim is to prove that, with high probability, such a random graph is far from being K_k-minor free. We start with an overview of this proof.

Fixing a sufficiently small constant value $\epsilon > 0$ (i.e., $\epsilon \ll 1/3k$), we partition the cycle to k equal-length segments (i.e., $(1, 2, ..., (N/k))$, $((N/k)+1, (N/k)+2, ..., (2N/k))$, ..., $(((k-1)N/k)+1, ((k-1)N/k)+2, ..., N))$. Focusing on the subgraph induced by each segment, we first prove that, with high probability, omitting at most $3\epsilon N/2$ edges from it yield a graph that has a connected component that contains most of the vertices (i.e., more than $N/2k$ vertices). Next, we prove that for every pair of connected components, with high probability, there are more than $3\epsilon N/2$ edges going from one component to the other one. Contracting each of these k connected components, we get a copy of K_k that survives the omission of $3\epsilon N/2$ edges. We now turn to the actual proof.

We consider a process in which a graph is uniformly selected in $\mathcal{G}_N$, and then $3\epsilon N/2$ edges are (adversarially) omitted from it. Our aim is to show that, with high probability, the resulting graph contains a K_k-minor. We shall actually consider a worse process in which $3\epsilon N/2$ edges are omitted from the Hamiltonian cycle and $3\epsilon N/2$ edges are omitted from the matching. We shall show that, for any choice of $3\epsilon N/2$ edges from the Hamiltonian cycle, with overwhelming high probability, the residual process (i.e., selecting a random perfect matching and (adversarially) omitting $3\epsilon N/2$ matching edges) yields a graph that contains a K_k-minor.

Using $\epsilon < 1/k^5$ and setting $\ell = \Theta(k/\epsilon)$, we further partition each of the large k segments into $(N/k)/\ell$ small segments, each of length ℓ. Note that omitting any $3\epsilon N/2$ edges (of the Hamiltonian cycle), leaves all but at most $3\epsilon N/2 < N/4\ell k$ of these ℓ-segments intact. Fixing any choice of these $3\epsilon N/2$ omitted edges, we consider an auxiliary (random) graph that represents the matching edges going between the ℓ-segments. That is, this auxiliary graph has a vertex set that equals the set of the intact ℓ-segments, and with (possibly multiple) edges connecting two ℓ-segments if and only if these segments contain vertices that are matched in the original graph. The main technical fact (proved below) is that *for every two disjoint $(N/4\ell k)$-sets of ℓ-segments, with probability at least* $1 - \exp(-\Omega(N/k^4))$, *there exist at least $2\epsilon N$ edges going between these sets.* Applying a union bound over all possible choices of these two sets, we infer that, with probability at least $1 - \binom{N/\ell}{N/4\ell k}^2 \cdot \exp(-\Omega(N/k^4)) > 1 - \exp(-\Omega(N/k^4))$, for every two disjoint $(N/4\ell k)$-sets of segments there exist at least $2\epsilon N$ edges going between these sets. In this case, after omitting any set of $3\epsilon N/2$ matching edges, the auxiliary graph contains connected components that cover more than half the vertices associated with each large segment and there are edges between each pair of these connected components. Applying a union bound over all the possible choices of $3\epsilon N/2$ cycle edges, the theorem follows.

Thus, it is left to prove the aforementioned technical fact. Let S_1 and S_2 be disjoint sets of ℓ-segments such that $|S_1| = |S_2| = N/4\ell k$. We need to prove that, with probability at least $1 - \exp(-\Omega(N/k^4))$, there exist at least $2\epsilon N$ matching edges going between S_1 and S_2. Note that each random matching edge connects S_1 and S_2 with probability $(4k)^{-2}$, and if these events were mutually independent then the fact would follow by the Chernoff bound. However, these events are not independent, yet setting an adequate Martingale and using Azuma's Inequality the fact follows just as well. Details are omitted. ■

7 Testing Tree-Minor Freeness

As noted in Section 6, the $\Omega(\sqrt{N})$ lower bound of Theorem 6.1 does not hold in the case the forbidden minor is a tree. This is easiest to see in the case that the forbidden minor is a single edge. We show that, for any cycle-free graph H, the set of H-minor free graphs can be tested with one-sided error with query complexity independent of the input graph's size (and only depends on the proximity parameter and on H).

To begin, we provide a reduction of the case where H is a forest to the case where H is a tree. Actually, this reduction works for any H (regardless of cycle-freeness) allowing to focus on the connected components of H. Next, we turn to two special cases (which are easy to handle): the case that H is a k-path and the case that H is a k-star. Since these cases correspond to the two possible extremes, it is tempting to hope that all cases can be treated easily. We warn, however, that the extreme cases have simple characterizations, which are not available in non-extreme cases. Nevertheless, the case of stars provides some intuition towards the more complicated treatment of general trees. Further intuition can be obtained from the case of depth-two trees, treated in Section 7.5, where we also obtain better complexity than in the general case.

7.1 A reduction of unconnected H to connected H

Let H be a graph with connected components $H_1, ..., H_m$. Then, essentially (but not exactly), a graph G is H-minor free if and only if for some $i \in [m]$ the graph G is H_i-minor free; in other words, G has an H-minor if and only if for every $i \in [m]$ the graph G contains an H_i-minor. The alternative formulation reveals the small inaccuracy: it may be that the H_i-minors contained in G

are not disjoint (and in such a case G does not necessarily have an H-minor). Still, for our purposes (of studying one-sided error testers of sublinear query complexity), this problem can be overcome (as done next).

Indeed, we focus on one-sided error testers of sublinear query complexity. Given such testers for H_i-minor freeness, we present the following one-sided error tester for H-minor freeness.

Algorithm 7.1 (the H-minor-freeness tester for cycle-free H): *On input $G = ([N], E)$ and proximity parameter ϵ, set $G_0 = G$ and proceed in m iterations, as follows. For $i = 1$ to m,*

1. *Invoke the H_i-minor tester on input G_{i-1}, using error parameter $1/3m$ and proximity parameter $\epsilon/2$.*
2. *If the answer is positive then accept.*
3. *Otherwise, omit from G_{i-1} all vertices that were visited by the tester, obtaining a residual graph G_i.*

If all iterations rejected, then reject.

If Algorithm 7.1 rejects, then (by the one-sided error feature of the tests) the m exploration contain corresponding (disjoint) H_i-minors, and so G contains an H-minor. Thus, Algorithm 7.1 satisfies the one-sided error condition. On the other hand, if G is ϵ-far from being H-minor free, then, for every $i \in [m]$, the graph G must be ϵ-far from being H_i-minor free (because otherwise G is ϵ-close to an H_i-minor free graph, which in turn is H-minor free). Furthermore, for every $i \in [m]$, the graph G_{i-1} is $\epsilon/2$-far from being H_i-minor free, because G_{i-1} is obtained from G by omitting $o(N)$ edges (since all testers have sublinear query complexity). Thus, in each iteration i, with probability at least $1 - (1/3m)$, the corresponding tester rejects. It follows that Algorithm 7.1 rejects G with probability at least $2/3$ (as required). We thus get the following result.

Proposition 7.2 *Let H have connected components $H_1, ..., H_m$, and suppose that H_i-minor freeness can be tester by a one-sided error tester of query complexity $q_i(N, \epsilon)$. Suppose that $q_i(N, \epsilon)$ is monotonically non-decreasing with N. Then, H-minor freeness can be tester by a one-sided error tester of query complexity $q(N, \epsilon) = O(\log m) \cdot \sum_{i=1}^{n} q_i(N, \epsilon/2)$.*

(The $O(\log m)$ factor is due to error reduction that is employed on each of the testers.)

Detour. For sake of elegance, it would be nice to prove a similar reduction also for the case of two-sided error testers. Naturally, for testing H-minor freeness with two-sided error, we may just run all H_i-minor freeness tests (with error probability parameter set to $1/3m$) and accept if and only if at least one of these tests accepted (i.e., reject iff all these tests rejected). Clearly, if G is ϵ-far from being H-minor free, then, for every i, the graph G must be ϵ-far from being H_i-minor free (see above), and so in this case, with probability at least $2/3$, all tests will reject, and so will we. But what is missing is proving that if G is H-minor free, then the above tester accepts with high probability. (Indeed, it is not necessarily the case that if G is H-minor free then for some i it holds that G is H_i-minor free).

7.2 Testing that the graph contains no simple k-length path

Here we consider the special case where $H = P_k$, where P_k denotes the k-length path. Note that a graph G is P_k-minor free if and only if G contains no simple path of length k. Thus, we just search for such a path at random. Specifically, we select uniformly a start vertex and take a random k-step walk, rejecting if and only if the walk corresponds to a simple path. Clearly, we never reject a P_k-minor free graph.

Claim 7.3 *If G is ϵ-far from being a P_k-minor free graph, then we reject with probability at least $\epsilon/2d^k$.*

Thus, P_k-minor freeness can be tested by a one-sided error tester of query complexity $q \stackrel{\rm def}{=} O(d^k k/\epsilon)$ and time complexity $O(q \log N)$.

Proof: We call a vertex v bad if there is a simple path of length k starting at v. Let ρ denote the density of bad vertices in G. Then, on the one hand, we reject G with probability at least ρ/d^k. On the other hand, $\rho \geq \epsilon/2$, because omitting all bad vertices (or rather their incident edges) from G we obtain a graph that has no simple k-length paths. ■

7.3 Testing that the graph contains no k-star as a minor

Here we consider the special case where $H = T_k$, where T_k denotes the k-star (i.e., the $(k+1)$-vertex tree that has k leaves). The key observation here is a graph $G = ([N], E)$ is T_k-minor free if and only if for every set S such that G_S is connected it holds that the set S has less than k neighbors (in $[N] \setminus S$). Another important observation is that it suffices to consider sets S of size at most $4k/\epsilon$, because a set S of size $4k/\epsilon$ such that all its subsets satisfy the condition can be ignored (since G_S is T_k-minor free and has less than k edges to the rest of the graph). The latter reasoning will be the crux of the formal analysis that follows. Yet another important observation is that the complexity of searching for sets that violate the condition can be reduced by using a BFS, as in the following algorithm.

Algorithm 7.4 (the k-star-minor-freeness tester): *On input $G = ([N], E)$ and proximity parameter ϵ, proceed as follows.*

1. *Select uniformly a start vertex $s \in [N]$.*
2. *Perform a BFS starting at s and stopping as soon as either $4k/\epsilon$ layers were explored or a layer with at least k vertices was encountered.*

   Note that it may also be that the BFS terminates before either of these conditions hold; this can only happen if s resides in a connected component of size smaller than $4k^2/\epsilon$.
3. *Accept if and only if the explored graph is T_k-minor free.*

Clearly, Algorithm 7.4 never rejects a T_k-minor free graph. In analyzing its performance on inputs that are ϵ-far from T_k-minor free, we shall refer to a weaker rejection criterion that corresponds to the motivating discussion (i.e., the existence of small sets S that have k neighbors). Step 2 is only used in order to improve the complexities; it guarantees that Algorithm 7.4 has query complexity $q \stackrel{\rm def}{=} O(k^2/\epsilon)$, and by [RS95] the time complexity is $O(q^3 \log N)$. Thus, all that is left is to prove the following.

Claim 7.5 *If G is ϵ-far from being a T_k-minor free graph, then Algorithm 7.4 rejects with probability at least $\epsilon/4$.*

Thus, T_k-minor freeness can be tested by a one-sided error tester of query complexity $O(k^2/\epsilon^2)$ and time complexity $O(k^6\epsilon^{-4}\log N)$.

Proof: We call a vertex v bad if there exists a set $S \ni v$ such that (i) G_S is connected and has radius at most $4k/\epsilon$ from v (i.e., all vertices are at distance at most $4k/\epsilon$ from v), and (ii) the set S has at least k neighbors in G (i.e., $|\{u \in [N] \setminus S : \exists w \in S \text{ s.t. } \{u,w\} \in E\}| \geq k$). Note that if a bad vertex is chosen in Step 1, then Algorithm 7.4 rejects in Step 3 (because either a $4k/\epsilon$-step BFS of G starting at v reaches a layer with at least k vertices, or it reaches all vertices in the witness set S). Let ρ denote the density of bad vertices in G. By the above, Algorithm 7.4 rejects with probability at least ρ. We next show that G must be $(2\rho + (\epsilon/2))$-close to T_k-minor free, and so $\rho \geq \epsilon/4$ follows.

Let $G^{(0)}$ denote the graph obtained from G by omitting all the edges that are incident at bad vertices. Indeed, $G^{(0)}$ is 2ρ-close to G. The rest of our analysis proceed in iterations. If the current graph $G^{(i-1)}$ is T_k-minor free, then we are done. Otherwise, we pick an arbitrary vertex $s^{(i)}$ that resides in some T_k-minor. Since $s^{(i)}$ is not bad, it must reside in a connected component of $G^{(i-1)}$ that has radius at least $4k/\epsilon$ from $s^{(i)}$ (because otherwise the existence of a T_k-minor containing $s^{(i)}$ contradicts the hypothesis that v is not bad). Consider an arbitrary set $S^{(i)} \ni s^{(i)}$ of $4k/\epsilon$ vertices such that $G^{(i-1)}_{S^{(i)}}$ is connected. Since $s^{(i)}$ is not bad, it follows that $S^{(i)}$ has less than k neighbors (in $G^{(i-1)}$). We now obtain $G^{(i)}$ by omitting the (less than kd) edges of the cut $(S^{(i)}, [N] \setminus S^{(i)})$, and observe that $G^{(i)}_{S^{(i)}}$ is T_k-minor free (and that $S^{(i)}$ will not intersect with any future $S^{(j)}$). When the process ends, we have a T_k-minor free graph. In total, we omitted at most $tk \cdot d$ edges (from $G^{(0)}$), where $t \leq N/(4k/\epsilon)$ denotes the number of iteration. Noting that $tdk \leq (\epsilon/4)dN$, we conclude that $G^{(0)}$ is $\epsilon/2$-close to $G^{(t)}$ and thus G is $(2\rho + (\epsilon/2))$-close to T_k-minor free. ■

7.4 The general case: Testing T-minor freeness for any tree T

Following is a presentation of the main result of this section: a one-sided tester for T minor-freeness, where T is an arbitrary rooted tree with k vertices. The algorithm is an extension of the algorithm for stars: We perform a BFS from a random starting vertex (but for more levels) and check if we find a T-minor.

The analysis of this algorithm, in the current (general) case, is far more involved; nonetheless, the basic intuition remains the same. Suppose our procedure is typically unable to find a T-minor in G. We shall show that we can split up the graph into many small pieces, each being T-minor free and having few edges leaving it. Removing the few edges going between these pieces, we get a T-minor graph, which proves that G is close to being T-minor free.

The main challenge is to perform the foregoing decomposition. For that, we will define an auxiliary procedure, called `find`, that attempts to find T-minors. This procedure will not be used by our algorithm; it will be used solely in the analysis. But, first, let us detail the alleged tester (while assuming that $\epsilon \leq 1/2$, or else we set $\epsilon = 1/2$).

Algorithm 7.6 (the tree-minor-freeness tester): *Given as input a proximity parameter ϵ and given query access to a graph $G = ([N], E)$ with maximum degree at most d, set $D = k \cdot (16d/\epsilon)^{4k+2}$ and proceed as follows.*

1. *Select uniformly, independently at random, $8/\epsilon$ start vertices in $[N]$.*

2. *For each selected start vertex s, perform a BFS starting at s and stop as soon as D layers are explored (or the BFS reaches all the vertices of a connected component in G).*

3. *Accept if and only if all explored subgraphs are T-minor free.*

Clearly, Algorithm 7.6 never rejects a T-minor free graph. Its query complexity is exponential in D, and its time complexity is polynomial in its query complexity (by [RS95]). The correctness of the algorithm thus follows from the next lemma.

Lemma 7.7 *If G is ϵ-far from being a T-minor free graph, then Algorithm 7.6 rejects with probability at least* $2/3$.

As stated above, the heart of the proof of this lemma is a procedure called `find` that tries to find small T-minors. When invoked at a certain vertex and failing to find a small T-minor, the procedure provides us with a sort of "explanation for it failure" in the form of a sparse cut. Thus, if the graph G is accepted by the tester with high probability, then we can use this procedure to get the desired decomposition. As may be expected, the procedure `find` is designed by a (tedious, but not obvious) induction on the size of T. Following is an overview of our approach.

Consider the tree T and remove an edge to two trees T_1 and T_2. Let the roots of these trees be the endpoints of the edge removed. A T-minor can be broken up into a T_1-minor and T_2-minor with a path connecting the two respective roots. So, it seems that we should try to find "rooted minors", where we specify a vertex v that must be present in the connected component that is the root. Inductively, assume that we have a procedure `find` for T_1 and T_2. We can use `find` to get these minors and try to connect the roots by a path. The problem is that we have to get *disjoint* minors to get a T-minor. Suppose we find a T_1-minor in the original graph. Because we want to find a disjoint T_2-minor, we make the vertices in this minor a *forbidden* set F (and effectively remove them from G). This means that `find` is not allowed to use the vertices of F in the T_2-minor. But now, `find` may return a sparse cut, instead of T_2-minor, in the modified graph. This cut is only sparse in the modified graph (without F), but may be dense in the graph G. To get around this, we somehow need to ensure that whenever a cut is found, the number of vertices in the smaller side is much larger than $|F|$. Then, a sparse cut in the modified graph remains sparse in the original. We will give an indication of how this is done when we describe the parameters of `find`.

First, we introduce some definitions and notation. For a graph $H = (V(H), E(H))$ and a subset of vertices $S \subseteq V(H)$, we use the standard notation H_S to denote the subgraph of H that is induces by S.

Definition 7.8 (Distances) *Let $H = (V(H), E(H))$ be a fixed graph. For any pair of vertices $v, u \in V(H)$, let $\mathrm{dist}_H(v, u)$ be the shortest-path distance between u and v in H. Given a set of vertices $T \subset V(H)$ and a vertex $v \in V(H)$, let $\Delta_H(v, T) \stackrel{\text{def}}{=} \max_{u \in T}\{\mathrm{dist}_H(v, u)\}$. More generally, for two sets of vertices $S, T \subseteq V(H)$, let $\Delta_H(S, T) \stackrel{\text{def}}{=} \max_{u \in T} \min_{v \in S} \mathrm{dist}_H(v, u)$.*

Definition 7.9 (Sparse Cuts) *For a graph $H = (V(H), E(H))$ with degree bound d, a cut $(S, V(H) \setminus S)$ is* sparse *with respect to H, if the number of edges in $E(H)$ that cross the cut is at most $\epsilon |S| d/4$. For $V(H) = [N]$ we denote the cut $(S, [N] \setminus S)$ by cut(S).*

To differentiate from the original input graph G, the input graph to `find` will be the graph G'. We usually refer to cuts in G', and hence, in such cases we remove the explicit reference to G' (i.e., we shall say that *cut*(R) is sparse rather than say that it is sparse with respect to G').

The parameters of find: The procedure `find` takes as input a vertex v in a graph $G' = ([N], E')$, a set of vertices U containing v, a rooted tree T with k nodes, and a set of *forbidden* vertices F (not containing v). Let $f = \max\{|F|, k(16d/\epsilon)^{4k+2}\}$, and $G'' = G'_{[N]\setminus F}$. The procedure works under the conditions that U is disjoint from F, $|U| \geq 16f/\epsilon$, and $\Delta_{G''}(v, U) \leq (16/\epsilon)\ln(f/\epsilon)$.

The procedure $\texttt{find}(v, U, T, F)$ outputs a pair (σ, S) such that $\sigma \in \{\texttt{minor}, \texttt{cut}\}$ and $S \subseteq [N]\setminus F$, where there is a path in G'' between v and every vertex in S. It will be convenient to express quantities in terms of $\hat{k} = 4k - 2$.

The requirement from find: The output of $\texttt{find}(v, U, T, F)$ should satisfy the following conditions.

$\sigma = \texttt{minor}$. The graph G'_S contains a T-minor not involving F that is rooted at v (i.e., v resides in the connected component that is contracted to fit the root r of T). We have $\Delta_{G''}(v, S) \leq (16d/\epsilon)^{\hat{k}} \ln(f/\epsilon)$.

$\sigma = \texttt{cut}$. The cut $cut(S)$ is sparse and $\Delta_{G''}(v, S) \leq (16d/\epsilon)^{\hat{k}} \ln(f/\epsilon)$.

Intuitively, the set U acts as a kind of large buffer around v. This deals with the issue that we raised earlier. When we try to find a T_2-minor by making the vertices of the T_1-minor forbidden, we could get a sparse cut in this modified graph. The buffer U ensures that this cut contains sufficiently many vertices.

Claim 7.10 *There exists a procedure* `find` *that satisfies the foregoing properties.*

Before proving Claim 7.10 we state and prove some preliminary claims. In what follows, when we say we perform a BFS in a graph $H = (V(H), E(H))$ from a subset of vertices M, we mean the following. Consider the graph $H'(M)$ whose vertex set is $(V(H)\setminus M) \cup \{v(M)\}$ (so that M is replaced by a single vertex $v(M)$), and whose edge set is $\{(u,w) \in E(H) : u, w \in V(H)\setminus M\} \cup \{(u, v(M)) : u \notin M \text{ and } \exists w \in M \text{ s.t. } (u,w) \in E(H)\}$. A BFS from M in H corresponds to a BFS in $H'(M)$ that starts from $v(M)$.

Claim 7.11 *Let F and M be two disjoint subsets of vertices in G' such that $|M| \geq (8/\epsilon)|F|$. Suppose we perform a BFS up to depth t in $G'_{[N]\setminus F}$, starting from M, and let ℓ be the size of the last level reached. Then either there exists a subset of vertices R that are reached by the BFS and such that $cut(R)$ is sparse, or $\ell \geq |M| \cdot e^{(\epsilon/9)t}$.*

Proof: Consider some intermediate level in the BFS, and let R be the set of vertices reached up to that level (including it). Suppose that the next level has at most $\epsilon|R|/8$ vertices. All edges in $\texttt{cut}(R)$ are either incident to vertices in the next level (which contains at most $\epsilon|R|/8$ vertices) or to F. Since $|R| \geq |M| \geq 8|F|/\epsilon$, the size of the cut is at most $\epsilon|R|d/4$, and hence it is sparse.

Otherwise, the size of the levels keeps expanding by a factor of at least $(1 + \epsilon/8)$. Since the depth of the BFS is t, the size of the last level is at least $|M| \cdot (1 + \epsilon/8)^t \geq |M| \cdot e^{(\epsilon/9)t}$. ■

Definition 7.12 (Boundaries) *Given sets of vertices S and F, let $\partial_F(S)$ denote the* boundary *of S in $G'_{[N]\setminus F}$. That is, $\partial_F(S) \stackrel{\text{def}}{=} \{u \in S : \exists w \in [N]\setminus(S \cup F) \text{ s.t. } (u,w) \in E(G')\}$. We use $\overline{\partial}_F(S)$ to denote the set $S \setminus \partial_F(S)$.*

Claim 7.13 *Let F and M be two disjoint subsets of vertices such that $|M| \geq (8/\epsilon)|F|$, and let $\widetilde{F} = \overline{\partial}_F(M) \cup F$. There is a procedure that, given an integer parameter t, outputs one of the following:*

- *A set R such that the $cut(R)$ is sparse and $\Delta_{G'_{[N]\setminus F}}(\partial_F(M), R) \leq t$.*
- *A vertex $v \in \partial_F(M)$ and a set U_v disjoint from $\widetilde{F}$ such that $v \in U_v$, $|U_v| \geq e^{(\epsilon/9)t}$, and $\Delta_{G'_{[N]\setminus \tilde{F}}}(v, U_v) \leq t$*

Proof: We start by performing a BFS from M in $G'' = G'_{[N]\setminus F}$ up to depth t. By the definition of the BFS, all the vertices reached in levels $1, \ldots, t$ are disjoint from M and F. Applying Claim 7.11, in the process of this BFS either we find a sparse cut, thus satisfying the first condition, or the size of the last level is at least $|M| \cdot e^{(\epsilon/9)t}$. In the latter case, for each vertex $v \in \partial_F(M)$, perform a BFS in $G'_{[N]\setminus \widetilde{F}}$ up to depth t, and let U_v be the set of vertices reached. Since the last level of the original BFS is contained in $\bigcup_v U_v$, we have that $\sum_{v \in \partial_F(M)} |U_v| \geq |M| \cdot e^{(\epsilon/9)t}$. Therefore, there exists a vertex $v \in \partial_F(M)$ such that $|U_v| \geq |M| \cdot e^{(\epsilon/9)t}/|\partial_F(M)| \geq e^{(\epsilon/9)t}$. ■

With these tools in hand, we are ready to describe the procedure `find`.

Proof of Claim 7.10. We prove the claim by induction over the size of the tree T. For the base case, let T be a singleton vertex. Then, the procedure `find` just outputs the pair $(\texttt{minor}, U)$. Now for the induction step.

Take an edge e of T that is incident to the root r. Removing this edge gives us two trees T_1 and T_2 with roots r_1 and r_2 (these are the respective endpoints of e). We let T_1 be the tree still rooted at r (so that $r_1 = r$). Using subscripts to denote the respective size parameters of these trees, we have $\hat{k} = \hat{k}_1 + \hat{k}_2 + 2$ (recall that $\hat{k} = 4k - 2$). We also have that $\hat{k}_1, \hat{k}_2 \geq 2$.

We will describe the procedure $\texttt{find}(v, U, T, F)$ using the respective procedures for T_1 and T_2. We set $D_1 = (16d/\epsilon)^{\hat{k}_1} \ln(18f/\epsilon^2)$ (recall that $f = \max\{|F|, k(16d/\epsilon)^{4k+2}\}$). We will be dealing mainly with the graph $G'' = G'_{[N]\setminus F}$ and hence all our boundaries are in this graph. For ease of notation, for a set S, we shall use the shorthand $\partial(S)$ for $\partial_F(S)$ and $\overline{\partial}(S)$ for $\overline{\partial}_F(S)$. We also use the shorthand $\Delta(\cdot,\cdot)$ for $\Delta_{G''}(\cdot,\cdot)$ (and if distances are measured with respect to another graph then we'll state this explicitly). Recall that the procedure is required to work under the conditions that U is disjoint from F, $|U| \geq 16f/\epsilon$, and $\Delta(v, U) \leq (16/\epsilon)\ln(f/\epsilon)$. We may actually assume that $|U| = 16f/\epsilon$. Suppose this is not the case. Take the vertex in U farthest from v and remove it from U. We keep repeating this until $|U| = 16f/\epsilon$. Note that the upper bound on $\Delta(v, U)$ remains.

We now describe the steps of the procedure `find`. Refer to Figure 5 to understand the various steps of the procedure.

1. Initiate a BFS in the residual graph $G'' = G'_{[N]\setminus F}$ starting from U for $2D_1$ steps. Let A denote the set of all vertices reached (including U). We now invoke Claim 7.11 with F, $M := U$, and $t := 2D_1$. If we find a sparse cut $cut(R)$, then observe that $\Delta(v, R) \leq \Delta(v, U) + 2D_1 \leq (16d/\epsilon)^{\hat{k}} \ln(f/\epsilon)$. In this case, we output $(\texttt{cut}, R)$. Otherwise, the BFS reaches $2D_1$ levels, and $|A| \geq |U| \cdot e^{(\epsilon/2)D_1}$. We trivially bound $|A| \leq |U| \cdot e^{2D_1 \ln d}$, and continue to the next stage.
2. We invoke Claim 7.13 with F, $M := A$, and $t := (9/\epsilon)\ln(16|F_2|/\epsilon)$, where observe that $|F_2| = |\overline{\partial}(A) \cup F| \leq |A| \leq |U| \cdot e^{2D_1 \ln d}$. If we get a sparse cut $cut(R)$, then we have

$$\Delta(v, R) \quad \leq \quad \Delta(v, U) + \Delta(U, \partial(A)) + (9/\epsilon)\ln(16|A|/\epsilon)$$

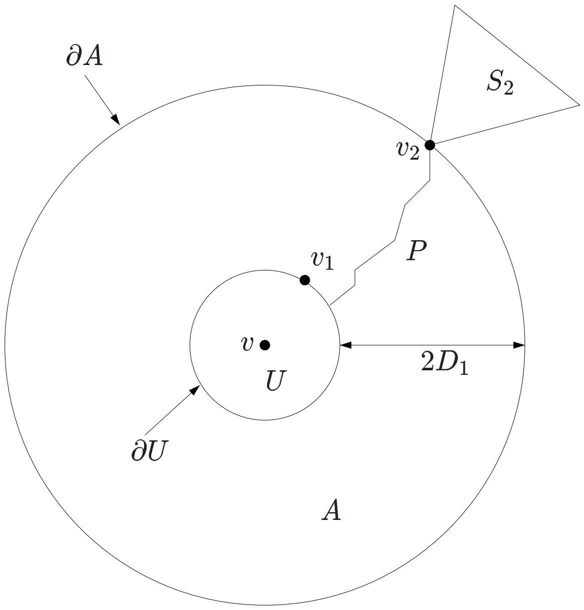

Figure 5: The various sets in `find`

$$
\begin{aligned}
&\leq (16/\epsilon)\ln(f/\epsilon) + 2D_1 + (9/\epsilon)\ln(16|U|/\epsilon) + 18D_1 \ln d/\epsilon \\
&\leq (64/\epsilon)\ln(f/\epsilon) + (18d/\epsilon)(16d/\epsilon)^{\hat{k}_1}\ln(18f/\epsilon^2) \\
&\leq (16d/\epsilon)^{\hat{k}}\ln(f/\epsilon)
\end{aligned}
$$

In this case we output $(\mathtt{cut}, R)$. Otherwise, we get a vertex $v_2 \in \partial(A)$ and a set U_2 such that $v_2 \in U_2$, $|U_2| \geq 16|F_2|/\epsilon$, and $\Delta_{G'_{[N]\setminus F_2}}(v_2, U_2) \leq (9/\epsilon)\ln(16|F_2|/\epsilon)$. Let $f_2 = \max\{|F_2|, k_2(16d/\epsilon)^{4k_2+2}\}$. Since $|F_2| \geq |U| = 16f/\epsilon$, we have that $f_2 = |F_2|$, so that $|U_2| \geq 16f_2/\epsilon$ and $\Delta_{G'_{[N]\setminus F_2}}(v_2, U_2) \leq (16/\epsilon)\ln(f_2/\epsilon)$. Therefore, the conditions for calling $\mathtt{find}(v_2, U_2, F_2, T_2)$ are met. Let (σ, S_2) be the output of this procedure. By the induction hypothesis:

$$
\begin{aligned}
\Delta(v, S_2) &\leq \Delta(v, U) + \Delta(U, \partial A) + \Delta(v_2, S_2) \\
&\leq 16\ln(f/\epsilon)/\epsilon + 2D_1 + (16d/\epsilon)^{\hat{k}_2}\ln(f_2/\epsilon) \\
&\leq 16\ln(f/\epsilon)/\epsilon + 2D_1 + (16d/\epsilon)^{\hat{k}_2}\cdot(\ln(|U|/\epsilon) + 2D_1\ln d) \\
&\leq 16\ln(f/\epsilon)/\epsilon + 2D_1 + (16d/\epsilon)^{\hat{k}_2}\cdot(\ln(16f/\epsilon^2) + 2D_1\ln d) \\
&\leq (16d/\epsilon)^{\hat{k}}\ln(f/\epsilon)
\end{aligned}
$$

If $(\mathtt{cut}, S_2)$ is output, then the main procedure also returns $(\mathtt{cut}, S_2)$. Otherwise, the set S_2 (disjoint from F_2) contains a T_2-minor such that v_2 belongs to the subset whose contraction corresponds to the root r_2 of T_2, and we continue to the next stage.

3. Consider the shortest path P from U to v_2 (in G''). By construction, $|P| \leq 2D_1$. Note that other than v_2, the path P is disjoint from S_2. We also have $f = \max\{|F|, k(16d/\epsilon)^{4k+2}\} \geq (16d/\epsilon)^{\hat{k}}\ln(f/\epsilon) \geq 2D_1$. Hence, $|F \cup P| \leq 2f$. Let $F' := F \cup P$, $F_1 := \overline{\partial}(U) \cup F'$ and

$f_1 = \max\{|F_1|, k_1(16d/\epsilon)^{4k_1+2}\}$. We invoke Claim 7.13 with F' as the forbidden set, $M := U$, and $t := (9/\epsilon)\ln(16f_1/\epsilon)$. Since $|F_1| \leq 16f/\epsilon + 2f$ (recall that we assumed without loss of generality that $|U| = 16f/\epsilon$), we have that $t = (9/\epsilon)\ln(16f_1/\epsilon) < (16d/\epsilon)^{\hat{k}_1}\ln(18f/\epsilon^2) = D_1$ If we get a sparse cut $cut(R)$, we output $(\mathtt{cut}, R)$.

Otherwise, we get a vertex $v_1 \in \partial(U)$ and a set $U_1 \ni v_1$ disjoint from F_1 such that $|U_1| \geq 16|f_1|/\epsilon$ and $\Delta_{G'_{[N]\setminus F_1}}(v_1, U_1) \leq t \leq 16\ln(f_1/\epsilon)/\epsilon$. We thus have the necessary conditions to call $\mathtt{find}(v_1, U_1, F_1, T_1)$. By the induction hypothesis, for the set S_1 returned, $\Delta_{G'_{[N]\setminus F_1}}(v_1, S_1) \leq D_1$. Hence $\Delta(v, S_1) \leq \Delta(v, U) + D_1 < 2D_1 \leq (16d/\epsilon)^{\hat{k}}\ln(f/\epsilon)$. If we get $(\mathtt{cut}, S_1)$, then the main procedure returns the same.

4. Otherwise, S_1 contains a T_1 minor such that v_1 belongs to a connected subset in S_1 whose contraction corresponds to r_1 (the root of T_1). Recall that we also have that S_2 contains a T_2 minor such that v_2 belongs to a connected subset in S_2 whose contraction corresponds to r_2 (the root of T_2) We next show how to construct a T-minor using U, S_1, S_2, and P. Note that all these sets are disjoint from F, and S_1 is disjoint from $S_2 \cup P$.

   Our aim is to connect v_1 to v_2 (in G'') by a path that is disjoint to $S_1 \cup S_2$. If this path contains v, we will get a T-minor rooted at v that involves no vertex of F. Take the path P in G'' that connects $\partial_F(U)$ to v_2. This path is disjoint to $S_1 \cup S_2$. The vertex v_1 is in $\partial_F(U)$ and v is connected to all of U in G''. We take a path from v to P and a path from v to v_1. This connects v_1 to v_2 (via v) in G'' and completes the construction of the T-minor.

■

Proof of Lemma 7.7. Recall that $D = k \cdot (16d/\epsilon)^{4k+2}$, and that Algorithm 7.7 performs a BFS from $8/\epsilon$ start vertices, up to depth D for each, and rejects if any of the subgraphs observed contains a T minor. We call a vertex v $\mathtt{bad}$ if its D-neighborhood (i.e., the subgraph induced by all vertices at distance at most D from v) contains a T-minor, and denote the fraction of bad vertices (in G) by ρ. We shall show that G is $(2\rho + \epsilon/2)$-close to being T-minor free. The lemma follows since this implies that if G is ϵ-far from being T-minor free, then $\rho > \epsilon/4$. In such a case, the probability that no bad vertex is selected as a start vertex by the algorithm is at most $(1 - \epsilon/4)^{8/\epsilon} < e^{-2} < 1/3$.

In order to prove that G is $(2\rho + \epsilon/2)$-close to being T-minor free, we will remove at most $(\rho + \epsilon/4)dN$ edges from G to make it T-minor free. We start by removing all edges incident to bad vertices, so that the number of edges removed at this stage is at most ρdN. Let the resulting graph be $G^{(0)}$. The rest of our analysis proceed in iterations. We have a current graph $G' = G^{(i-1)}$ where some connected components are marked "minor free". These components are certified to have no T-minor. If all the components are marked, then we are done. Otherwise, consider some unmarked component C. Suppose there is $v \in C$, such that $\Delta_{G'}(v, C) \leq D$. If C contains a T-minor, then v must be bad. This contradicts that fact that C is a connected component containing v. Therefore C has no T-minor, and can be marked. We proceed in this fashion till we get a component C that cannot be marked.

We take an arbitrary vertex $s^{(i)} \in C$ and observe that $\Delta_{G'}(s^{(i)}, C) > D$. Let F be initialized to $\emptyset$. We perform a BFS from $s^{(i)}$ up to depth $D_0 = (9/\epsilon)\ln(16f/\epsilon)$ steps, and invoke Claim 7.11 with $M = \{s^{(i)}\}$, $F = \emptyset$, and $t = D_0$. Suppose we get a set $S^{(i)}$ such that $cut(S^{(i)})$ is sparse. Since $\Delta_{G'}(s^{(i)}, S^{(i)}) = D_0 \leq D$, the subgraph $G_{S^{(i)}}$ cannot contain a T-minor. We remove all edges in the cut $cut(S^{(i)})$ and mark the connected components in $G_{S^{(i)}}$ as minor free. This gives us the graph $G^{(i)}$, and we continue with the next iteration.

Otherwise, the BFS gives a set U, such that $|U| \geq e^{(\epsilon/9)D_0} \geq 16f/\epsilon$, and $\Delta_{G'}(s^{(i)}, U) \leq D_0 \leq (16/\epsilon)\ln(f/\epsilon)$, and we call $\mathtt{find}(s^{(i)}, U, T, F)$. If it outputs $(\mathtt{minor}, S^{(i)})$, then v must be bad. This is a contradiction, and hence the output must be $(\texttt{cut}, S^{(i)})$. We have $\Delta_{G'}(s^{(i)}, S^{(i)}) \leq D$, where $cut(S^{(i)})$ is sparse. We proceed as before by removing all edges in $cut(S^{(i)})$ to get $G^{(i)}$.

When the process ends, we have a T-minor free graph. Since all the $S^{(i)}$'s considered are disjoint, in total, we omitted at most $\sum_i \epsilon d|S^{(i)}|/4 \leq \epsilon dN/4$ edges (from $G^{(0)}$), and thus G is $(2\rho + \epsilon/2)$-close to T-minor freeness. ■

The combinatorial core of the analysis. We observe that the features of `find` imply that, if a graph contain an "expanding neighborhood" (i.e., "locally looks as an expander"), then this neighborhood contains all possible tree-minors (of a related size). More formally,

Definition 7.14 *Let G be a graph of maximum degree d and s be a vertex of G. We say that the* R-neighborhood of s in G is ϵ-expanding *for every vertex set S such that $\Delta(v, S) \leq R$, it holds that the number of edges in the cut $(S, [N] \setminus S)$ is at least $\epsilon|S|d/4$.*

Theorem 7.15 *For any k and d, if the $k(16d/\epsilon)^{4k+2}$-neighborhood of s in G is ϵ-expanding, then this neighborhood contains a T-minor of any tree T of at most k vertices.*

Note that Lemma 7.7 can be derived from Theorem 7.15 similarly to the way it was derived from Claim 7.10. This is hardly surprising since Theorem 7.15 is closely related to Claim 7.10.

Proof: Indeed, the theorem follows from Claim 7.10, where the key observation is that `find` works well for any k-vertex tree T and that `find` may not return a sparse cut (because no such cut exists by the hypothesis). Specifically, set $F = \emptyset$ and let U be a set such that $|U| \geq 16f/\epsilon$ and $\Delta_G(v, U) \leq (16/\epsilon)\ln(f/\epsilon)$ (which exists since the said neighborhood contains no sparse cuts). Now, for any k-vertex tree T, we run $\texttt{find}(v, U, T, F)$ and get the output (σ, S), where $\sigma \neq \texttt{cut}$. Thus, we get the desired T-minor. ■

7.5 Testing T-minor freeness for any depth-two tree T

Let T be an arbitrary depth-two tree with k vertices; that is, T consists of a root, denoted r, and m stars, denoted $T_1, ..., T_m$, that are rooted at neighbors of r, where here we consider also the singleton vertex as a star (with 0 leaves). Denote the m corresponding roots by $r_1, ..., r_m$, and denote the number of leaves in these stars by $k_1, ..., k_m$ (i.e., $k = 1 + m + \sum_{i \in [m]} k_i$). The following algorithm is tailored for this tree T.

Algorithm 7.16 (tailored for the foregoing T): *On input $G = ([N], E)$ and proximity parameter ϵ, set $D = (5d^2k/\epsilon)^2$ and proceed as follows.*

1. *Select uniformly a start vertex $s \in [N]$.*
2. *Perform a BFS starting at s and stopping as soon as D layers are explored.*
3. *Accept if and only if the explored graph is T-minor free.*

Clearly, Algorithm 7.16 never rejects a T-minor free graph. Its query complexity is exponential in D, and its time complexity is polynomial in its query complexity (by [RS95]).

Lemma 7.17 *If G is ϵ-far from being a T-minor free graph, then Algorithm 7.16 rejects with probability at least $\epsilon/4$.*

Proof: We call a vertex v bad if its D-neighborhood (i.e., the vertices of distance at most D from v) contains a T-minor, and denote the fraction of bad vertices (in G) by ρ. As in the proof of Claim 7.5, it suffices to show that G is $(2\rho+(\epsilon/2))$-close to being T-minor free, and we again start by omitting all edges incident at bad vertices and considering the resulting graph, denoted $G^{(0)}$. Indeed, $G^{(0)}$ is 2ρ-close to G.

The rest of our analysis proceed in iterations. If the current graph $G^{(i-1)}$ is T-minor free, then we are done. Otherwise, we pick an arbitrary vertex $s^{(i)}$ that resides in (the root of) some T-minor. Since $s^{(i)}$ is not bad, it must reside in a connected component of $G^{(i-1)}$ that has radius at least D from $s^{(i)}$. We shall show how to identify a set $S^{(i)}$ such that $G^{(i-1)}_{S^{(i)}}$ has radius at most D and the cut $(S^{(i)},[N]\setminus S^{(i)})$ has less that $\epsilon d|S^{(i)}|/4$ edges. Omitting these cuts edges yields a graph $G^{(i)}$ such that $G^{(i)}_{S^{(i)}}$ is T-minor free (and $S^{(i)}$ will not intersect with any future $S^{(j)}$). When the process ends, we have a T-minor free graph. In total, we omitted at most $\sum_i \epsilon d|S^{(i)}|/4 \leq \epsilon dN/4$ edges (from $G^{(0)}$), and thus G is $(2\rho+(\epsilon/2))$-close to T-minor free.

The crux of the proof is indeed the process of identifying a suitable set $S'=S^{(i)}$ in $G' \stackrel{\rm def}{=} G^{(i-1)}$. The identification procedure is initiated at $s'=s^{(i)}$ and proceeds in two stages. In the first stage, the procedure tries to find either a set S_0 of size at least $4m/\epsilon$ such that the cut $(S_0,[N]\setminus S_0)$ has less than m edges or a set S_0 of size at most $4dm/\epsilon$ such that G'_{S_0} contains a m-star as a minor rooted at s'. (Clearly, in the first case we are done.) In the second case, we get to the second stage of the procedure, which explores G' (somewhat) beyond S_0 in an attempt to extend the m-star minor into a T-minor, but this attempt is bound to fail, and this failure will allow finding the desired cut. Loosely speaking, this second stage proceeds by trying to find disjoint T_j-minors, for $j=1,...,m$. This is done by invoking a "k'-star-minor finding" procedure, denoted $\mathtt{FS}_{k'}$, which generalizes the procedure that is described in the proof of Claim 7.5, The procedure $\mathtt{FS}_{k'}$ is invoked on a vertex, v, and a set of forbidden vertices, denoted F, and tries to either find a k'-star rooted at v in $G'_{[N]\setminus F}$ or find a good cut. Indeed, F will contain the set S_0 as well as adequate sets that will prevent the current search from entering any of the previously found star minors. We first provide a specification of $\mathtt{FS}$, and then turn to it actual implementation.

Specification of the procedure $\mathtt{FS}$. On input a vertex v and a forbidden set F, the procedure $\mathtt{FS}_{k'}$ outputs a triplet (σ,R',F') such that $\sigma\in\{\mathtt{minor},\mathtt{cut},\mathtt{free}\}$ and $F'\subseteq R'\subseteq[N]\setminus F$ such that $|F'|<dk'$ and $|R'|<(5dk'/\epsilon)\cdot(|F|+1)$. In addition, it always holds that all vertices of $G'_{R'}$ are connected to v, and one of the following cases holds.

$\sigma=\mathtt{minor}$. The graph $G'_{R'}$ contains a k'-star as a minor that is rooted at v (i.e., v resides in the connected component that is contracted to fit the root of the k'-star). Furthermore, all edges of the cut $(R'\setminus F',[N]\setminus(R'\setminus F'))$ are incident at $F\cup F'$.

$\sigma=\mathtt{cut}$. The cut $(R',[N]\setminus R')$ contains less that $\epsilon d|R'|/4$ edges.

$\sigma=\mathtt{free}$. All edges of the cut $(R',[N]\setminus R')$ are incident at F.

Let T' denote a generic k'-star, where we may assume that $k'\geq 1$.

Implementing the procedure $\mathtt{FS}$. Our aim is to either find a (relatively small) T'-minor or find a set with a relatively small cut from the rest of the graph. This is done by initiating a BFS in the residual graph $G'_{[N]\setminus F}$ starting at v, and stopping as soon as one of the following three cases occurs.

Case 1: *A layer containing at least k' vertices is found before $4(|F|+k')/\epsilon$ vertices are encountered.* In this case the procedure returns $(\mathtt{minor},R',F')$, where R' is the set of encountered vertices and F' is the set of vertices in the last BFS layer.

Note that in this case $G'_{R'}$ contains a T'-minor rooted at v, and that $|F'| < dk'$ (as otherwise the BFS would have terminated in a previous layer). Furthermore, by structure of the BFS, all edges of the cut $(R' \setminus F', [N] \setminus (R' \setminus F'))$ are incident at $F \cup F'$.

Case 2: *The search encountered at least* $4(|F| + k')/\epsilon$ *vertices, while Case 1 does not hold.* In this case the procedure returns $(\mathtt{cut}, R', \emptyset)$, where R' is the set of encountered vertices.

Note that in this case the cut $(R', [N] \setminus R')$ contains less than $(|F| + k') \cdot d \le \epsilon d|R'|/4$ edges.

Case 3: *The search cannot be extended any further, while Cases 1 and 2 do not hold.* In this case the procedure returns $(\mathtt{free}, R', \emptyset)$, where R' is the set of encountered vertices.

Note that in this case the cut $(R', [N] \setminus R')$ contains only edges that are incident at F.

In all cases $|R'| \le 4d(|F| + k')/\epsilon < 4dk'(|F| + 1)/\epsilon$, because if more than $4d(|F| + k')/\epsilon$ vertices are encountered then either Case 1 or Case 2 holds. Thus, this implementation satisfies the specification. We note that the above description applies also in case $k' \in \{0, 1\}$, where $k' = 0$ is trivial[17] (i.e., always return $(\mathtt{minor}, \{v\}, \{v\})$) and $k' = 1$ is almost trivial (i.e., return $(\mathtt{minor}, \{v, w\}, \Gamma_{G'}(v) \setminus F)$ if v has a neighbor w in $G'_{[N]\setminus F}$ and $(\mathtt{free}, \{v\}, \emptyset)$ otherwise).

Using the star finding procedure $\mathtt{FS}$, we now turn to the main identification procedure, which is invoked on input vertex $s' = s^{(i)}$ and aims at finding an adequate set $S' = S^{(i)}$. Recall that r denotes the root of T, and $r_1, ..., r_m$ denote the roots of the subtrees $T_1, ..., T_m$, where T_j is a k_j-star. The main procedure operates as follows.

1. It initiates a BFS in the graph G' starting at s', stopping as soon as at least $B = 4dk/\epsilon$ vertices are encountered. Let S_0 denote the set of encountered vertices. Note that $|S_0| \ge B$ must hold, because $s' = s^{(i)}$ resides in root of some T-minor having radius greater than D.

   Note that it holds that $|S_0| < dB$ (because otherwise we would have stopped at the previous BFS-layer).

2. Let F_0 denote the last layer in the BFS performed in the previous step. If $|F_0| < m$, then we just use S_0 as the desired set (i.e., let $S^{(i)} = S_0$).

   Note that, in this case, the cut $(S_0, [N] \setminus S_0)$ contains less than $m \cdot d$ edges, whereas by the case hypothesis $|S_0| \ge B > 4m/\epsilon$. So the conditions regarding this set are satisfied.

   We continue to the next step only if $|F_0| \ge m$.

3. (The purpose of the current step is to generate calls to $\mathtt{FS}$ that will eventually lead to returning a set as in the second output case (i.e., $\mathtt{cut}$), which can serve as $S^{(i)}$ (see above). The presentation, however, pretends that we attempt to find a T-minor as in the first output case (i.e., $\mathtt{minor}$). Observing that $S_0 \setminus F_0$ can serve as a contraction of the root of T, we attempt to find disjoint sets S_j that contain T_j-minors rooted at some $v_j \in F_0$.)

   For $j = 1, ..., m$, we try to find S_j as follows. Let $F' = \bigcup_{a \in [j-1]} F_a$ and $V' = \{v_1, ..., v_{j-1}\}$. For every $v \in F_0 \setminus V'$, we proceed as follows.

   We invoke $\mathtt{FS}_{k_j}$, letting $(\sigma, X, Y) \leftarrow \mathtt{FS}_{k_j}((F_0 \setminus \{v\}) \cup F', v)$.
   We note that $|X| \le (5dk_j/\epsilon) \cdot (|F_0| + |F'| + 1)$ and $|Y| \le dk_j$. Recall that $|F_0| < |S_0| < dB = 4d^2k/\epsilon$ and $|F'| = \sum_{a \in [j-1]} |F_a| < d \sum_{a \in [j-1]} k_a < d(k - m)$, where $k = 1 + m + \sum_{a \in [m]} k_a$. Thus, $|X| < (5d^2k/\epsilon)^2$.

[17] Actually, this case never occurs; that is, we never invoke $\mathtt{FS}_0$. The case $k' = 1$ may occur, but we could have avoided it too, but a direct treatment.

We consider the following three cases regarding σ.

$\sigma =$ `minor`. In this case we set $v_j \leftarrow v$ and $(S_j, F_j) \leftarrow (X, Y)$, and proceed to the next value of j (i.e., $j \leftarrow j+1$); see comment below.

Note that $|S_j| < (5dk/\epsilon)^2$. In fact, the same upper bound can be proved for $\sum_{a=0}^{j} |S_a|$.

Note that this case cannot occur when $j = m$, because this would yield a small T-minor rooted in s' in contradiction to the hypothesis that $s' = s^{(i)}$ is not bad.

$\sigma =$ `cut`. In this case we just use X as the desired set (i.e., let $S^{(i)} = X$).

Note that, by the specification of `FS`, the cut $(S^{(i)}, [N] \setminus S^{(i)})$ contains relatively few edges.

$\sigma =$ `free`. In this case we do nothing, and continue to the next candidate v.

Note that we halted with a desired cut if either Step 2 found such a cut or any of the invocations of `FS` returned an `cut`-value. Furthermore, as noted, it cannot be the case that in Step 3 we obtained a `minor`-value for each $j \in [m]$. Thus, we remain with the case that, for some $j \in [m]$, all invocations of `FS` returned a `free`-value. In this case, we let X' be the union of all sets X that were returned in the corresponding $|F_0| - (j-1)$ invocations, and use $S_0 \cup X'$ as the desired set (i.e., let $S^{(i)} = S_0 \cup X'$).

In this case, the size of the cut $(S^{(i)}, [N] \setminus S^{(i)})$ is at most $d \cdot |F'| < d^2 k$, because for each X all edges of the cut $(X, [N] \setminus X)$ are incident at $F_0 \cup F' \subseteq S_0 \cup F'$. Thus, the cut is sufficiently small, because $|S^{(i)}| \geq |S_0| \geq B = 4dk/\epsilon$. On the other hand, the size of $S_0 \cup X'$ is at most $|F_0| \cdot (5dk/\epsilon) \cdot |F'| < (5dk/\epsilon)^2$.

This completes the description of the operation of the procedure I as well as the showing that it satisfies its specification. It follows that for any $s^{(i)}$ that reside in the root of some T-minor in $G^{(i-1)}$, we obtain a set $S^{(i)}$ such that the cut $(S^{(i)}, [N] \setminus S^{(i)})$ has less than $4d|S^{(i)}|/\epsilon$ edges. Using the fact $|S^{(i)}| < D$, it follows that $G^{(i-1)}_{S^{(i)}}$ is T-minor free, and the lemma follows. ■

8 The unbounded-degree graph model

In this section we consider testing cycle-freeness in what we shall refer to as the unbounded-degree incidence-lists model [PR02]. In this model, the maximum degree d may be as large as $N-1$, so there is effectively no degree-bound, and a graph G is represented by a function $g : [N] \times [N-1] \to \{0, \ldots, N\}$. Similarly to the bounded-degree model, the algorithm may ask for the identity of the i^{th} neighbor of a vertex v, for any $v \in [N]$ and $i \in [N-1]$ of its choice, by querying the function g. (If v has less than i neighbors, then the answer returned is '0'). For the sake of simplicity, we assume that the algorithm can also query the degree of any vertex of its choice (where such a query can, of course, be replaced by $O(\log N)$ neighbor queries).

The main and crucial difference between the unbounded-degree model and the bounded-degree model is in the *distance measure between graphs*. Rather than measuring distance between graphs in terms of the size of the domain of g, as done in the bounded-degree model, we measure it with respect to the number of edges $|E|$ in $G = ([N], E)$. That is, we shall say that a graph G is ϵ-far from being cycle-free (in the unbounded-degree model), if the number of edges that must be removed in order to make it cycle-free is greater than $\epsilon|E|$. Letting d_{avg} denote the average degree in G, this is equivalent to saying that the number of edges in G is greater than $(N-1) + \epsilon d_{\text{avg}} N/2$.

We note that while the bounded-degree model is appropriate for testing graphs in which the maximum degree is of the same order as the average degree (and in particular constant-degree graphs), the unbounded-degree model is appropriate for testing graphs in which the maximum degree may be much higher than the average degree. We mention that the model considered in [KKR04] (see also Section 8.3) also allows adjacency queries (as in [GGR98]), but such queries are useless when the degree is smaller than $\sqrt{N}$.

8.1 Testing cycle-freeness

In this subsection, we show that the result of Theorem 1.5 (and thus also Theorem 1.1) extends to the unbounded-degree (incidence lists) model. This will be done by viewing the randomized reduction that underlies Algorithm 3.1 in a slightly different way, which actually yields an alternative tester (which is closely related to but different from Algorithm 3.1). We then show that this algorithm extends easily to the unbounded-degree model.

The pivot of our exposition is the following generalization of 2-colorability in which edges of the graph are labeled by either `eq` or `neq`. That is, an instance of this problem is a graph $G = ([N], E)$ along with a labeling $\mu : E \to \{\mathtt{eq}, \mathtt{neq}\}$. We say that $\chi : [N] \to \{0, 1\}$ is a legal 2-coloring of this instance if for every $\{u, v\} \in E$ it holds that $\chi(u) = \chi(v)$ if and only if $\mu(\{u, v\}) = \mathtt{eq}$. That is, a legal 2-coloring (of the vertices) is one in which every two vertices that are connected by an edge labeled `eq` (resp. `neq`) are assigned the same color (resp., opposite colors). Note that the standard notion of 2-colorability corresponds to the case in which all edges are labeled `neq`.

We observe that the Bipartite testers of [GR99] and [KKR04] can be extended to test this generalization of 2-colorability.[18] All that is needed is to define edges labeled `neq` as having even length (say, length zero or two), whereas edges labeled `eq` are defined as having odd length (say, length one). Modulo this definition, the entire analysis of [GR99] remains intact. Specifically, all references in [GR99] to the length of paths and cycles are re-interpreted as referring to the foregoing definition. In particular, an odd length cycles (under this label-dependent definition of length) indicates that the graph cannot be 2-colored (under the corresponding labeling of edges), whereas the non-existence of odd length cycles enables such a 2-coloring. (The same holds for [KKR04], which operates by a (local) reduction to [GR99].)

Lastly, we observe that the randomized reduction that underlies Algorithm 3.1 can be viewed as a randomized reduction of cycle-freeness to generalized 2-coloring, while keeping the graph intact. Specifically, the graph $G = ([N], E)$ is mapped to a random instance of the generalized 2-coloring problem such that the graph equals G itself and the labeling is selected uniformly among all possible $\mu : E \to \{\mathtt{eq}, \mathtt{neq}\}$. Invoking the generalized 2-coloring tester (derived from [KKR04]) on the resulting instance, we are done. (Indeed, in this case, unlike in the case of Algorithm 3.1, the emulation of the generalized 2-coloring tester is straightforward.)

8.2 Testing tree-minor-freeness

In contrast to Section 8.1, we show that the result of Theorem 1.3 cannot be extended to the unbounded-degree model. This follows by considering an N-vertex graph G that consists of a cycle of length $N - \sqrt{N}$ and a clique of size $\sqrt{N}$ (i.e., $G = C_{N-\sqrt{N}} + K_{\sqrt{N}}$). Denoting the 3-star by T_3, note that G is $\Omega(1)$-far from being T_3-minor-free (since we must omit $\sqrt{N} - 3$ edges from each vertex of the $\sqrt{N}$-clique in order to eliminate all copies of T_3 itself). On the other hand, no $o(\sqrt{N})$-query tester can find a T_3-minor in a random isomorphic copy of G, except with probability

[18] A similar observation refers to the k-colorability testers of [GGR98], which operate in the dense graph model. Thus, for every $k \geq 2$, the foregoing generalization of k-colorability can be tested in the dense graph model by using $\mathrm{poly}(1/\epsilon)$ queries.

$o(1)$, Furthermore, any algorithm of query complexity $o(\sqrt{N})$ cannot distinguish a random copy of G from a random copy of a N-vertex graph that consists of a cycle of length $N - \sqrt{N}$ and $\sqrt{N}$ isolated vertices.

We mention that an $O(\sqrt{N})$-query one-sided tester for T_k-minor-freeness does exist for any k (where T_k denotes the k-star). This tester may be obtained by combining the tester for the bounded-degree model (for $d = k$, as presented in Section 7.3) with an $O(\sqrt{N})$-query procedure for finding a vertex of degree at least k. Clearly, if the former tester ever sees a vertex of degree at least k, then the combined tester rejects; otherwise, the analysis of Section 7.3 applies to the graph induced by the low degree vertices. Thus, we should only worry about the case that distance from being T_k-minor-free is mostly due to vertices of degree at least k. In this case (i.e., at least $\epsilon|E|/2$ edges are incident at vertices of degree at least k), sampling a random edge and checking the degree of its endpoints will do, whereas such sampling can be performed using $\widetilde{O}(\sqrt{N})$ queries (see [KKR04]).

8.3 Testing with adjacency queries

Here we consider an augmentation of the model with adjacency queries. This augmentation was first considered in [KKR04], and it was shown to be useful (for testing bipartitness) when the average degree, $d_{\rm avg}$, exceeds $\sqrt{N}$. We observe that the same holds with respect to testing cycle-freeness (see details below). We also stress that in the bare model (i.e., without adjacency queries) the results presented in Section 8.1 are optimal.

We note that the reduction presented in Section 8.1 remains valid, except that in this case the generalized 2-coloring tester (derived from [KKR04]) may use adjacency queries. In this case, the resulting cycle-freeness tester will have complexity $\min(\widetilde{O}(\sqrt{N}), \widetilde{O}(N)/d_{\rm avg}) \cdot {\rm poly}(1/\epsilon)$ (just as the 2-coloring tester of [KKR04]).

References

[AS03] N. Alon and A. Shapira. Testing satisfiability. *Journal of Algorithms*, 47:87–103, 2003.

[BSS08] I. Benjamini, O. Schramm, and A. Shapira. Every minor-closed property of sparse graphs is testable. In *Proceedings of the Fourtieth Annual ACM Symposium on the Theory of Computing*, pages 393–402, 2008.

[Fis01] E. Fischer. The art of uninformed decisions: A primer to property testing. *Bulletin of the European Association for Theoretical Computer Science*, 75:97–126, 2001.

[FP87] J. Friedman and N. Pippenger. Expanding graphs contain all small trees. *Combinatorica*, 7:71–76, 1987.

[GGR98] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4):653–750, 1998.

[GR99] O. Goldreich and D. Ron. A sublinear bipartite tester for bounded degree graphs. *Combinatorica*, 19(3):335–373, 1999.

[GR02] O. Goldreich and D. Ron. Property testing in bounded degree graphs. *Algorithmica*, pages 302–343, 2002.

[GS07] O. Goldreich and O. Sheffet. On the randomness complexity of property testing. In *Proceedings of the Eleventh International Workshop on Randomization and Computation (RANDOM)*, pages 296–310, 2007.

[GT03] O. Goldreich and L. Trevisan. Three theorems regarding testing graph properties. *Random Structures and Algorithms*, 23(1):23–57, 2003.

[HKNO09] A. Hassidim, J. Kelner, H. Nguyen, and K. Onak. Local graph partitions for approximation and testing. In *Proceedings of the Fiftieth Annual Symposium on Foundations of Computer Science (FOCS)*, 2009.

[KKR04] T. Kaufman, M. Krivelevich, and D. Ron. Tight bounds for testing bipartiteness in general graphs. *SIAM Journal on Computing*, 33(6):1441–1483, 2004.

[Kur30] K. Kuratowski. Sur le problème des courbes gauches en topologie. *Fundamenta Mathematica*, 15:271–283, 1930.

[Lov06] L. Lovász. Graph minor theory. *Bulletin of the American Mathematical Society*, 43(1):75–86, 2006.

[PR02] M. Parnas and D. Ron. Testing the diameter of graphs. *Random Structures and Algorithms*, 20(2):165–183, 2002.

[Ron08a] D. Ron. Property testing: A learning theory perspective. *Foundations and Trends in Machine Learning*, 1(3):307–402, 2008.

[Ron08b] D. Ron. Some techniques in property testing. Presentation. Available from `http://www.eng.tau.ac.il/~danar/talks.html`, 2008.

[RS95] N. Robertson and P. D. Seymour. Graph minors. XIII. The disjoint paths problem. *Journal of Combinatorial Theory Series B*, 63(1):65–110, 1995.

[RS04] N. Robertson and P. D. Seymour. Graph minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory Series B*, 92(1):325–357, 2004.

[Wag37] K. Wagner. Über eine eigenschaft der ebenen komplexe. *Mathematische Annalen*, 114:570–590, 1937.