Lecture Notes on Testing Graph Properties in the Bounded-Degree Graph Model

Oded Goldreich*

May 6, 2016

Summary: This lecture is devoted to testing graph properties in the bounded-degree graph model, where graphs are represented by their incidence lists (lumped together in an incidence function). The highlights of this lecture include:

1. Presenting lower and upper bounds on the complexity of testing `Bipartitness`; specifically, we present a $\mathrm{poly}(1/\epsilon) \cdot \widetilde{O}(\sqrt{k})$-time tester, and an $\Omega\sqrt{k})$ lower bound on the query complexity of any tester for `Bipartitness`.
2. Reviewing a quasi-$\mathrm{poly}(1/\epsilon)$-time tester for `Planarity`. The result extends to testing any minor-closed property (i.e., a graph property that is preserved under the omission of edges and vertices and under edge contraction).

We concluded this lecture with a taxonomy of known testers, organized according to their query complexity.

The current notes are based on many sources; see Section 7.1 for details.

Organization. The presentation of the various testers is organized by the algorithmic techniques that they utilize. These include local searches (see Section 2), random walks (see Section 4), and the implementation and utilization of partition oracles (see Section 5). In addition, the current notes includes a section on (query complexity) lower bounds (Section 3), which justifies the fact that the testers presented in Section 4 have higher complexity than those presented in Section 2.

Preliminaries. We assume that the reader is familiar with basic algorithmic techniques such as BFS and DFS (see, e.g., [12]). This will be important especially in Section 2.

Teaching note: Much of this chapter (e.g., Sections 6 and 7) is intended for optional independent reading. We recommend to base the actual teaching on Section 1 and selections from Sections 2–4. A very minimalistic choice includes Sections 2.3, 3.1 and 4.1. If time permits, we would also recommend including Section 2.4 (with a focus on Algorithm 10 and its analysis). Another recommendation consists of Sections 2.5 and 4.2 (along with Theorem 17 (which appears in Section 3.2)). We do share the temptation to cover also Section 5 in class, but think that teaching the material presented in prior sections should get higher priority.

*Department of Computer Science, Weizmann Institute of Science, Rehovot, Israel.

1 The Bounded-Degree Model: Definitions and issues

The study of property testing in the bounded-degree graph model is aimed at allowing the consideration of sparse graphs, which appear in numerous applications. The point is that the dense graph model, studied in the previous lecture, seems irrelevant to sparse graphs, both because the distance measure that underlies it deems all sparse graphs as close to one another, and because adjacency queries seems unsuitable for sparse graphs. Sticking to the paradigm of representing graphs as functions, where both the distance measure and the type of queries are determined by the representation, the following representation seemed the most natural choice. (Indeed, a conscious decision is made here not to capture, at this point (and in this model), sparse graphs that do not have constant (or low) maximum degree.)

The bounded-degree graph model refers to a fixed degree bound, denoted $d \geq 2$. An k-vertex graph $G = ([k], E)$, of maximum degree d, is represented in this model by a function $g : [k] \times [d] \to \{0, 1, ..., k\}$ such that $g(v, i) = u \in [k]$ if u is the i^{th} neighbor of v and $g(v, i) = 0$ if v has less than i neighbors. Hence, it is also adequate to refer to this model as the incidence function model. For simplicity, we assume here that the neighbors of vertex v appear in an arbitrary order in the sequence $g(v, 1), ..., g(v, \deg(v))$, where $\deg(v) \stackrel{\text{def}}{=} |\{i : g(v, i) \neq 0\}|$ is the degree of v. Also, we shall always assume that if $g(v, i) = u \in [k]$, then there exists $j \in [d]$ such that $g(u, j) = v$.

Distance between graphs is measured in terms of their aforementioned representation (i.e., as the fraction of (the number of) different array entries (over $n = d \cdot k$)), but occasionally we shall use the equivalent and more intuitive notion of the fraction of (the number of) edges over $dk/2$.

Recall that we are interested in *graph properties*, which are sets of graphs that are closed under isomorphism; that is, Π is a graph property if for every graph $G = ([k], E)$ and every permutation π of $[k]$ it holds that $G \in \Pi$ if and only if $\pi(G) \in \Pi$, where $\pi(G) \stackrel{\text{def}}{=} ([k], \{\{\pi(u), \pi(v)\} : \{u, v\} \in E\})$. We now spell out the meaning of property testing in this model.

Definition 1 (testing graph properties in the bounded-degree graph model):[1] *For a fixed d, a* tester *for a graph property Π is a probabilistic oracle machine that, on input parameters k and ϵ, and access to* (the incidence function of) *an k-vertex graph $G = ([k], E)$ of maximum degree d, outputs a binary verdict that satisfies the following two conditions.*

1. *If $G \in \Pi$, then the tester accepts with probability at least $2/3$.*
2. *If G is ϵ-far from Π, then the tester accepts with probability at most $1/3$, where G is ϵ-*far *from Π if for every k-vertex graph $G' = ([k], E') \in \Pi$ of maximum degree d it holds that the symmetric difference between E and E' has cardinality that is greater than $\epsilon \cdot dk/2$.* (Equivalently, we may say that G is ϵ-far from G' if for every $g : [k] \times [d] \to \{0, 1, ..., k\}$ and $g' : [k] \times [d] \to \{0, 1, ..., k\}$ that represent G and G', respectively, it holds that $|\{(v, i) : g(v, i) \neq g'(v, i)\}| > \epsilon \cdot dk$.)

If the tester accepts every graph in Π with probability 1, then we say that it has one-sided error*; otherwise, we say that it has* two-sided error*. A tester is called* non-adaptive *if it determines all its queries based solely on its internal coin tosses* (and the parameters k and ϵ)*; otherwise, it is called* adaptive.

[1] As in the dense graph model, we provide the tester with the number of vertices, denoted k, rather than with the size of the representation, denoted $n = d \cdot k$. The definition of a tester can be made even more uniform by providing the degree bound, denoted d, as an auxiliary parameter.

The query complexity of a tester is the number of queries it makes to any k-vertex graph, as a function of the parameters k and ϵ.[2] We say that a tester is efficient if it runs in time that is linear in its query complexity, where basic operations on elements of $[k]$ (and in particular, uniformly selecting an element in $[k]$) are counted at unit cost. Unless explicitly stated otherwise, the testers presented in this lecture are efficient.

On the degree bound d. As stated in Footnote 1, the degree bound, denoted d, may be viewed as an auxiliary parameter, and complexity bounds may be stated as a function of it too. Note that this parameter has two opposite effects. On the one hand, if our algorithm explores all neighbors of a given vertex, then its complexities increase linearly with d. On the other hand, (relative) distances are normalized by dk, which means that they decrease linearly with d, which in turn relaxes the requirements from a tester.

Degree queries. The model can be augmented by allowing also degree queries (i.e., query $v \in [k]$ is answered with the degree of v in the tested graph). Degree queries can be emulated by $\lceil \log(d+1) \rceil$ incidence queries, by performing a binary search (see Exercise 1).

Variants. Recall that we are using the convention by which the neighbors of v appear in an arbitrary order in the sequence $g(v,1),...,g(v,\deg(v))$, where $\deg(v)$ denotes the degree of v. In contrast to this convention, one may consider the following three variants on the model.

1. Sorted incidence functions: In this case for each $v \in [k]$, the sequence $g(v,1),...,g(v,\deg(v))$ is sorted; that is, for every $i \in [\deg(v)-1]$, it holds that $g(v,i) < g(v,i+1)$.

   This variant decreases the complexity of the task of finding whether two vertices are adjacent (by conducting a binary search on the incidence list of one vertex). Unfortunately, the two definitions of distance given in Definition 1 are no longer equivalent (since the Hamming distance between d-long sequences is not preserved when the sequences are sorted).[3]

2. Unaligned incidence functions: In this case it is no longer guaranteed that the $\deg(v)$ neighbors of v appear in the $\deg(v)$-long prefix of the sequence $g(v,1),...,g(v,d)$.

   This variant increases the complexity of tasks such as finding a neighbor of a given vertex or determining the degree of a given vertex.

3. Incidence-set functions: Here we represent the (degree d) graph $G = ([k],E)$, by $g : [k] \to \cup_{i=0}^{d} \binom{[k]}{i}$ such that $g(v)$ is the set of neighbors of vertex v.

   This variant decreases the complexity of tasks such as finding all neighbors of a given vertex (and less so w.r.t determining the degree of a given vertex). On the other hand, the two definitions of distance given in Definition 1 are no longer equivalent (since under this representation modifying the neighbor set costs one unit regardless of how much the set was modified).

We mention that none of the above variants is popular, and the first two variants seem a bit unnatural. Nevertheless, one may imagine applications in which these variants are adequate. In any case, it is legitimate to use these variants to fasciate the exposition.

[2] As in Footnote 1, we deviated from the convention of presenting the query complexity as a function of $n = dk$ and ϵ.

[3] Consider the sequences $(3,5,7,11)$ and $(13,5,7,11)$.

The role of adaptivity. We mention that adaptive queries are far more important in the bounded-degree graph model as compared to the dense graph model: Recall that in the dense graph model, adaptive queries could be replaced by non-adaptive queries at a moderate cost of squaring the number of queries. In contrast, in the bounded-degree graph model, there is a huge gap between the adaptive and non-adaptive query complexities of testing many natural graph properties. Specifically, as shown in Section 2, properties such as subgraph freeness, degree regularity, connectivity, and cycle-freeness can all be tested by using $\mathrm{poly}(d/\epsilon)$ adaptive queries, but (as shown next) each of them requires $\Omega(\sqrt{k})$ non-adaptive queries.

This lower bound follows as a special case of a result that asserts that any property that "is not determined by the vertex degree distribution" requires such complexity. We say that a property Π is not determined by the vertex degree distribution if there exists $\epsilon > 0$ such that for infinitely many $k \in \mathbb{N}$ there exists $(d_1, ..., d_k) \in \{0, 1, ..., d\}^k$ and two k-vertex graphs, one in Π and the other ϵ-far from Π, such that the degree of the i^{th} vertex in each of the graphs equals d_i. (If this is not the case, then we say that the property is determined by the vertex degree distribution.)[4]

Theorem 2 (limitation of non-adaptive queries (in the bounded-degree graph model)): *For any function $q' : (0, 1] \to \mathbb{N}$, if a graph property Π can be tested in $q(k, \epsilon) = o(\sqrt{k} \cdot q'(\epsilon))$ non-adaptive queries, then Π is determined by the vertex degree distribution.*

This result is quite tight, since triangle-freeness can be tested by $O(\sqrt{d^2k/\epsilon})$ non-adaptive queries (see Exercise 2).

Proof Sketch: Fix an $\epsilon > 0$ such that there exist an infinite sequence of pairs of graphs (G_1, G_0) that have the same number of vertices (denoted k) and the same degree sequence, although $G_1 \in \Pi$ and G_0 is ϵ-far from Π. We shall show that an algorithm of query complexity $o(\sqrt{k})$ cannot distinguish random isomorphic copies of these two k-vertex graphs.

We call a pair of queries (u, i) and (v, j) bad (for a graph) if either v or the j^{th} neighbor of v is the answer to the query (u, i) (i.e., if the answer to the query (u, i) equals either v or the answer to (v, j), assuming that the latter is not 0). Observe that if we take random isomorphic copies of both graphs, then the probability that q non-adaptive queries contain a bad pair of queries (for one of these copies) is at most $4 \cdot \binom{q}{2}/(k-1) < 2q^2/k$, since we may answer each query $(u, i) \in [k] \times [d]$ by selecting a random degree d_u (in the degree sequence) for u and answering with a random vertex if and only if $i \leq d_u$. This way of answering is consistent with both random copies, provided that the sequence of queries contain no bad pair. Hence, the distinguishing gap (w.r.t these random copies) of a non-adaptive algorithm that makes q queries is smaller than $2q^2/k$, and the theorem follows. ■

Non-adaptivity versus label-obliviousness. We note that a non-adaptive algorithm of $o(\sqrt{k})$ complexity cannot perform a local search on a k-vertex graph, since it can not find a neighbor of a neighbor of a given vertex. We wish to stress that a BFS from a given vertex to a given distance cannot be performed by a non-adaptive algorithm, although such a search is oblivious of the vertex labels. That is, obliviousness of the labels of vertices is fundamentally different from non-adaptivity; for example, the j^{th} neighbor of the i^{th} neighbor of v is a *label-oblivious* formulation, although it refers to the adaptive query $(g(v, i), j)$. Indeed, all "normal" graph algorithms as well as testers

[4] In that case, for every $\epsilon > 0$ and all but finitely many $k \in \mathbb{N}$, if two graphs have the same vertex degree distribution and one is in Π, then the other is ϵ-close to Π.

of graph properties are oblivious of vertex names, and in a sense this feature makes them "graph algorithms" (i.e., their operation is invariant under any relabeling of the graph's vertices).

On the difference in complexities among the different model. Another issue to notice is the difference between the query complexity of testing graph properties in the bounded-degree graph model as compared to the complexity of testing the same properties in the dense graph model. A few examples follows:

- Whereas `Bipartiteness` has a poly$(1/\epsilon)$-time tester in the dense graph model, it has no $o(\sqrt{k})$-query tester in the bounded-degree graph model. Furthermore, for $t \geq 3$, the dense graph model has a poly$(1/\epsilon)$-query tester for t-`Colorability`, but this property has no $o(k)$-query tester in the bounded-degree graph model.
- Whereas `triangle-freeness` has no poly$(1/\epsilon)$-query tester in the dense graph model, it has a $O(1/\epsilon)$-query tester in the bounded-degree graph model.
- Whereas `Connectivity` (and even "t-connectivity") is trivial in the dense graph model, it is far from being so in the bounded-degree graph model (although poly$(1/\epsilon)$-query testers do exist here too).

These examples and more will be discuss in the subsequent sections.

2 Testing by a local search

In this section we present relatively simple testers for subgraph freeness, degree regularity, connectivity, and cycle-freeness, where the latter tester has two-sided error. These poly$(1/\epsilon)$-query testers (as well as those for higher levels of connectivity) are based on conducting a small number of very local searches, but the parameters of these searchers and their goals vary from one case to another.

2.1 Testing subgraph freeness

Testing subgraph freeness (e.g., triangle-freeness), when the subgraph is not bipartite, is quite a challenge in the dense graph model. Recall that even testing triangle-freeness (in that model) involves the invocation of the Regularity Lemma. In contrast, we will present a relatively simple tester for the same properties in the current model (i.e., the bounded-degree graph model). Let us first recall the definition that we refer to.

Definition 3 (subgraph freeness): *Let H be a fixed graph. A graph $G = (V, E)$ is H-*free *if G contains no subgraph that is isomorphic to H.*

We shall focus on the case that H is connected, although the other case can be handled similarly (yielding similar, but not identical results).[5] Let $\mathrm{rd}(H)$ denote the radius of H; that is, $\mathrm{rd}(H)$ is the smallest integer r such that there exists a vertex v in H such that all vertices in H are at distance at most r from v. Such a vertex v is called a center of H, and indeed H may have several centers (e.g., consider the case that H is a clique).

[5] If H is composed of the connected components $H_1, ..., H_m$, then Algorithm 4.1 can be modified so to select uniformly $v_1, ..., v_m \in [k]$ and start a BFS from each of them. See Exercise 3.

Theorem 4 (testing subgraph freeness (in the bounded-degree graph model)): *Let $H = ([t], F)$ be a fixed* (connected) *graph of radius $r = \mathrm{rd}(H)$. Then, H-freeness has a* (one-sided error) *proximity-oblivious tester of query complexity $2d^{r+1}$ and linear detection probability. Furthermore, the time complexity of this tester is at most $(2d)^{rt}$.*

Proof: We consider the following natural algorithm.

Algorithm 4.1 (testing H-freeness): *On input parameters d and k and oracle access to the incidence function of a k-vertex graph $G = ([k], E)$, which has maximum degree d, the algorithm proceeds as follows.*

1. *Uniformly selects a vertex $v \in [k]$.*
2. *Conducts a BFS of depth at most r starting from v.*
3. *Accept if and only if the explored subgraph is H-free.*

Step 2 is implemented by querying the incidence function, and so the query complexity of this algorithm is upper-bounded by $\sum_{i=0}^{r} d^i \cdot d < 2d^{r+1}$. Step 3 can be implemented by checking all possible mappings of H to the explored graph, and so the time complexity of Algorithm 4.1 is upper-bounded by $\binom{2d^r}{t} \cdot (t!) < (2d)^{rt}$.

Algorithm 4.1 never rejects a graph that is H-free, since H-freeness is preserved by subgraphs of the original graph. (Algorithm 4.1 can be modified to check induced subgraph freeness, while noting that this property is preserved by induced subgraphs of the original graph.) It is left to analyze the detection probability of Algorithm 4.1.

Claim 4.2 (the detection probability of Algorithm 4.1): *If $G = ([k], E)$ is at distance δ from being G-free, then Algorithm 4.1 rejects it with probability at least $\delta/2$.*

Proof: A vertex $v \in [k]$ is called detecting if it is a center of a copy of H that resides in G. Then, G must have at least $\delta k/2$ detecting vertices, since omitting all edges that are incident at detecting vertices makes the graph H-free. The claim follows. ■

This completes the proof of the theorem. ■

2.2 Testing degree regularity

Testing degree regularity is somewhat easier in the bounded-degree graph model (as compared to the dense graph model), since determining the degree of a vertex is easier in this model. On the other hand, there is a small issue that arises here. If k is odd and we observe some vertex of odd degree, then we better reject, since a graph (with an odd number of vertices) in which almost all vertices are of odd degree is not closed to being regular (in the bounded-degree graph model, unlike in the dense graph model). This observation will be reflected in Step 4 of the following algorithm.

Algorithm 5 (testing degree regularity (in the bounded-degree graph model)): *On input parameters d, k and ϵ and oracle access to the incidence function of a k-vertex graph $G = ([k], E)$, which has maximum degree d, the algorithm proceeds as follows.*

1. *Uniformly selects a set of $O(1/\epsilon)$ vertices.*

2. *Determines the degree of each of the selected vertices.*

3. *If these degrees are not all the same, then the algorithm rejects.*

4. *If this same degree is odd and k is odd, then the algorithm rejects.*

Otherwise, the algorithm accepts.

Step 2 is implemented by a binary search on the incidence list of each selected vertex, and so the query (and time) complexity of this algorithm is $O(\epsilon^{-1} \log d)$. Evidently, Algorithm 4.1 never rejects a regular graph (where rejection in Step 4 is justified by noting that if a k-vertex graph is d'-regular, then $d'k$ is even).[6] The analysis of Algorithm 4.1 is based on the local-vs-global claim that was proved in the analysis of the degree-regularity tester for the dense graph model. This claim is restated next.

Claim 5.1 (local-vs-global distance to degree regularity): *Let $d' < k$ and $d'k/2$ be natural numbers, and let $d_G(v)$ denote the degree of vertex v in the graph $G = ([k], E)$. If $\sum_{v \in [k]} |d_G(v) - d'| \leq \epsilon' \cdot B$, then there exists a d'-regular k-vertex graph $G' = ([k], E')$ such that the symmetric difference between E and E' is at most $3\epsilon' B$.*

In the previous lecture, this claim was stated with $B = k^2$ and the bound on the symmetric difference was stated in terms of distance in the dense graph model (i.e., in units of $k^2/2$). Nevertheless, since that claim was stated for any $\epsilon' > 0$, it immediately yields Claim 5.1. Using Claim 5.1, we establish the following

Claim 5.2 (analysis of Algorithm 5): *If $G = ([k], E)$ is ϵ-far from being regular, then Algorithm 4.1 rejects with probability at least $2/3$.*

Proof: Let d' denote the degree of the first vertex selected in Step 1 of the algorithm. (Indeed, we may modify the algorithm so that the first vertex is selected arbitrarily.) If $d'k$ is odd, then the algorithm always rejects (in Step 4, if it reaches Step 4 at all), and so we may assume that $d'k$ is even.

Combining the claim's hypothesis with Claim 5.1, we infer that $\sum_{v \in [k]} |d_G(v) - d'| > \epsilon \cdot dk/6$. (This is the case since the symmetric difference between E and the edge set of any d'-regular k-vertex graph is greater than $\epsilon dk/2$.)[7] It follows that $|\{v \in [k] : d_G(v) \neq d'\}| > \epsilon k/6$, and the claim follows (since at least one of the vertices having degree different than d' is selected, w.h.p., and in this case Step 3 rejects). ■

The proof of Claim 5.2, reveals that selecting two vertices (one arbitrarily and the other at random) and determining their degrees, will do for obtaining a proximity oblivious tester. Hence, we get.

Theorem 6 (testing degree regularity (in the bounded-degree graph model)): *Degree regularity has a* (one-sided error) *proximity-oblivious tester of* (query and) *time complexity $2\lceil \log d + 1 \rceil$ and linear detection probability.*

[6] Recall that, in every graph, the sum of vertex degrees is even.

[7] Hence, using $B = dk/2$ and $\epsilon' = \epsilon/3$, we infer that $\sum_{v \in [k]} |d_G(v) - d'| \leq \epsilon' \cdot B$ is impossible, because it would yield a symmetric difference of at most $3\epsilon' B = \epsilon dk/2$.

Proof: The tester is a version of Algorithm 4.1 that selects only two vertices in Step 1. As noted in the proof of Claim 5.2, if G is δ-far from being regular and d' is the degree of the first vertex, then either $d'k$ is odd (in which case Step 4 guarantees rejection) or at least $\delta k/6$ vertices have degree different from d'. Hence, this algorithm will reject G with probability at least $\delta/6$. ■

Testing whether a graph is Eulerian. Recall that a graph is called Eulerian if all its vertices have even degree. (Note that we do not require here that the graph be connected.) We can easily test if a graph is Eulerian by sampling a random vertex and determining its degree, but again the analysis is not trivial because we need to preserve the degree bound (and the simplicity) of the graph. That is, we need to show that if few vertices of a graph of maximum degree d have odd degree, then this graph is close to a (simple) Eulerian graph *of maximum degree* d. This is not trivial since the degree bound may prevent us from connecting pairs of vertices that have odd degree (whereas arbitrarily omitting edges incident at vertices of currently odd degree is a bad idea).[8] Nevertheless, Exercise 4 shows that if a graph $G = ([k], E)$ has maximum degree d, and k' of its vertices have odd degree, then there exists a k-vertex Eulerian graph $G' = ([k], E')$ of maximum degree d such that the symmetric difference between E and E' is at most $3k'/2$.

2.3 Testing connectivity

The tester for `Connectivity` is based on the following observation.

Proposition 7 (distance from connectivity versus number of connected components): *Let $G = ([k], E)$ be a graph of maximum degree $d \geq 2$ that has m connected components. Then, there exists a connected graph $G' = ([k], E')$ of maximum degree d such that the symmetric difference between E and E' is at most $2m - 1$.*

(The non-trivial aspect of this proposition is the preservation of the degree bound. Omitting this restriction allows to present a connected graph $G'' = ([k], E'')$ such that the symmetric difference between E and E'' equals $m - 1$, which is optimal.)

Proof: We would like to add $m - 1$ edges between these m connected components so that the resulting graph is connected, but this may not be possible due to the degree bound. Specifically, we say that a k'-vertex connected component is saturated if the sum of its vertex degrees is at least $k' \cdot d - 1$, and call it unsaturated otherwise. Note that each saturated connected component can be made unsaturated by omitting a single edge, while preserving its connectivity. This can be seen by noting that such a connected component has a spanning tree (which consists of $k' - 1$ edges), implying that it has at least $(k'd - 1) - (k' - 1) > 0$ non-tree edges that can all be omitted without harming the connectivity.

Hence, by omitting at most m edges, we make all m connected components unsaturated, and now we can connect them by adding $m - 1$ edges (while preserving the degree bound). Specifically, we connect these components by ordering them arbitrarily, and connecting each pair of consecutive components by a single edge (using vertices of degree lower than d). Hence, we increase the sum of the vertex degrees in each component by at most two units, and we can afford doing so because the components are (now) unsaturated. ■

[8] Since the other endpoint of the edge may have even degree, and such a sequence of omissions may result in too many modifications (see the case of a long path).

Towards a tester. Proposition 7 implies that a graph that is ϵ-far from being connected has more than $\epsilon dk/4$ connected components. The next observation, which is pivotal to the tester, is that many of these connected components are small. Specifically, if there are k' connected components of size (i.e., number of vertices) at most s, then $k'+(k/s) > \epsilon dk/4$. For example, there must be at least $\epsilon dk/8$ connected components of size at most $8/(d\epsilon)$. Hence, selecting at random $O(1/\epsilon d)$ vertices and conducting a "truncated BFS" from each of them so that the BFS is suspended once more than $8/(d\epsilon)$ vertices are encountered, yields a tester for `Connectivity`. The time (and query) complexity this tester is $O(1/\epsilon d) \cdot O(d/d\epsilon) = O(1/d\epsilon^2)$. But using Levin's economical work investment strategy (see previous lecture), we can do better.[9]

Theorem 8 (testing connectivity (in the bounded-degree graph model)): *Connectivitry has a* (one-sided error) *tester of time* (and query) *complexity* $\widetilde{O}(1/\epsilon)$.

Proof: For sake of self-containment, we provide a full analysis of application of Levin's economical work investment strategy to this context. Fixing a graph $G = ([k], E)$ that is ϵ-far from being connected, for every $i = 0, ..., \ell \stackrel{\text{def}}{=} \log(9/d\epsilon)$, we denote by B'_i the set of vertices that reside in connected components of size at most $\lfloor 8/(2^i d\epsilon) \rfloor$ and at least $\lfloor 8/(2^{i+1} d\epsilon) \rfloor + 1$.[10]

Recall that G must have more that $\epsilon dk/4$ connected components, whereas there are at most $k/(8/d\epsilon)$ connected components of size larger than $8/d\epsilon$. Furthermore, all other vertices of G (i.e., those residing in connected components of size at most $8/d\epsilon$) are in $\cup_{i=0}^{\ell} B'_i$, since there are no connected components of size at most $8/(2^\ell d\epsilon) < 1$. On the other hand, the number of connected components that contain vertices of B'_i is at most $\frac{|B'_i|}{8/(2^{i+1} d\epsilon)}$, since each of these connected components has size that is larger than $8/(2^{i+1} d\epsilon)$. Combining these facts, we get

$$\sum_{i=0}^{\ell} \frac{|B'_i|}{8/(2^{i+1} d\epsilon)} > \frac{\epsilon dk}{4} - \frac{\epsilon dk}{8} \tag{1}$$

since the l.h.s of Eq. (1) represents an upper bound on the number of connected components of size at most $8/d\epsilon$, whereas the r.h.s represents a lower bound on that number. Noting that Eq. (1) simplifies to $\sum_{i=0}^{\ell} 2^{i+1}|B'_i| > k$, it follows that there exists $i \in \{0, 1, ..., \ell\}$ such that $|B'_i| = \Omega(2^{-i}k/\ell)$, whereas every corresponding connected can be explored in time $d \cdot (8/(2^i d\epsilon))$, since each of these connected components has size that is at most $8/(2^i d\epsilon)$. This leads to the following tester, where we assume that $8/(d\epsilon) < k$ (since otherwise we can retrieve the entire graph in time $dk = O(1/\epsilon)$).

The actual tester. For $i = 0, 1, ..., \ell$, perform the following steps.

1. Select at random $O(2^i \ell)$ vertices.
2. For each of these vertices, denoted v, perform a (BFS or DFS) search starting at v, suspending the execution if more than $8/(2^i d\epsilon)$ vertices were encountered in this search (or if the search scanned the entire connected component).

   Note that this search is implemented in time $8/(2^i \epsilon)$.

[9] We get an improvement only when $\epsilon = o(1/d)$, whereas when $\epsilon = \omega(1/d)$ we are actually worse. But, the case of $\epsilon > 4/d$ is trivial, since (in the current context of the bounded-degree graph model) every graph is $4/d$-close to being connected.

[10] In terms of the previous lecture, we may view such vertices as having quality in $[2^{i-3}d\epsilon, 2^{i-2}d\epsilon)$, and as requiring work investment $O(1/2^i\epsilon)$.

3. If any of these searches detected a connected component of size at most $8/(2^i d\epsilon)$, then the tester rejects. (Here we rely on $8/(d\epsilon) < k$.)

If none of these searchers detected a connected component that is smaller than k, then the tester accepts. Note that any linear-time search can be used in Step 2, and in such a case the overall time complexity of the tester is $\sum_{i=0}^{\ell} O(2^i \ell) \cdot 8/(2^i \epsilon) = O(\ell^2/\epsilon)$.

By its construction, this tester always accepts a connected graph, whereas a graph that is ϵ-far from being connected is rejected with high probability, because there exists an $i \in \{0, 1, ..., \ell\}$ such that $|B'_i| = \Omega(2^{-i}k/\ell)$, which implies that a vertex residing in a connected component of size at most $8/(2^i d\epsilon)$ was selected, w.h.p., in Step 1 (of iteration i), fully explore in Step 2, and causing rejection in Step 3. ■

Testing whether a graph is connected and Eulerian. Testing whether a graph is connected and Eulerian reduces to testing that it has both properties. This reduction relies on the fact that if G is ϵ-close to both properties, then it is $O(\epsilon)$-close to their intersection (see Exercise 5).

2.4 Testing t-connectivity (overview and one detail)

There are two different natural notions that generalize the notion of connectivity.

***t*-edge connectivity:** A graph is t-edge connected if there are t *edge-disjoint* paths between every pair of vertices.

***t*-vertex connectivity:** A graph is t-vertex connected if there are t *vertex-disjoint* paths between every pair of vertices.[11]

Clearly, t-vertex connectivity implies t-edge connectivity, and for $t = 1$ both notions coincides with the notion of connectivity. The connectivity level of a graph cannot exceed the (minimum) degree of its vertices, which means that we shall focus on $t \leq d$. All these t-connectivity properties can be tested in poly$(1/\epsilon)$-time, where the polynomial may depend on t.

Theorem 9 (testing t-connectivity, in the bounded-degree graph model):

- *For every $t \geq 2$, testing t-*`edge connectivity` *can be performed in time $\widetilde{O}(t^3/\epsilon^{c_t})$, where $c_t = \min(3, t-1)$.*
- *For every $t \geq 2$, testing t-*`vertex connectivity` *can be performed in time $\widetilde{O}((t/d\epsilon)^t)$.*

Testing t-connectivity generalize two ideas that appear in the tester for `Connectivity`: One main idea, which was conspicuous in the base case (of $t = 1$), is that distance from t-connectivity implies the existence of many small t-connected components. Furthermore, one can establish the existence of many small t-connected components that can be disconnected *from the rest of the graph* by omitting less than t edges (resp., vertices). This strengthening is important, because such small (and "isolatable") components seem easier to detect. The second idea, which was obvious and transparent in the base case, is that these small t-connected components can be detected.

Detailing the first idea, in the current context, requires getting into the structure of (the connections among the t-connected components of) graphs that are not t-connected, which we wish to

[11] Needless to say, the notion of vertex-disjoint paths excludes the end-points of these paths.

avoid. As for the second idea, we focus on the case of edge-connectivity, since the case of vertex-connectivity is more involved. We use the known fact that *a graph is t-edge connected if and only if it contains no cut of less than t edges* (i.e., for every partition of its vertex set into (V_1, V_2), there are at least t edges having one endpoint in each side). Now, suppose that you are given a vertex v that resides in a set S of size at most s such that the subgraph of $G = ([k], E)$ induced by S is t-connected and the cut $(S, [k] \setminus S)$ contains less than t edges.[12] *Can you find S within complexity that is related to s and unrelated to k?*

The rest of this section is devoted to the study of the foregoing problem, which is of independent interest. Recall that the task was easy for $t = 1$; that is, when given v, the connected component containing the vertex v can be found in time that is linearly related to its size (by invoking a BFS or a DFS at vertex v). In the case of a general $t > 1$, things are less obvious. Still, we may proceed (recursively) as follow.

1. Invoke a DFS at the vertex v and suspend its execution as soon as more than s vertices are encountered.
2. If the DFS detected a connected component of size at most s, then output it.
3. Otherwise, for each edge e in the DFS-tree constructed in Step 1, invoke the procedure on the graph $G' = ([k], E \setminus \{e\})$ with the same start vertex v but *with connectivity parameter* $t - 1$. If any of these (recursive) invocations returns a cut with less than $t - 1$ edges, then return this cut.

The reader may verify that this recursive procedure finds the desired cut in time $O(s^{t-1} \cdot ds) = O(ds^t)$, where the key observation is that the desired cut (is either empty or) must contain an edge of the DFS tree. Another good exercise (Exercise 6) is handling the case of $t = 2$ in time $O(ds)$, which yields an upper bound of $O(ds^{t-1})$ for $t \geq 3$. But using randomization yields an improvement on the foregoing bound.

Algorithm 10 (finding small t-edge connected components): *On input parameters t, d, k and s, a vertex $v \in [k]$ and oracle access to $G = ([k], E)$, the algorithm proceeds in iterations, starting with $S' = \{v\}$. In each iteration the algorirthm performs the following steps.*

1. *If the cut $C' = (S', [k] \setminus S')$ contains at most $t - 1$ edges, then output S'.*
2. *Otherwise, assign uniformly distributed random weights in $[0, 1]$ to every edge in the cut C' that was not assign a weight before.*
3. *Select an edge $(u, w) \in C'$ of minimum weight, and add w to S'* (i.e., $S' \leftarrow S' \cup \{w\}$).
4. *If $|S'| > s$, then halt with no output.* (Otherwise, proceed to the next iteration.)

Whenever Algorithm 10 outputs a set S', it is the case that $|S'| \leq s$ and the cut $(S', [k] \setminus S')$ has less than t edges. It is also apparent that Algorithm 10 makes at most ds queries (and runs in $\widetilde{O}(ds)$ time), but the question is what is the probability that it outputs a set at all. While a naive guess may be that the answer is $\Theta(t/ds)^{t-1}$, the correct answer is much better.[13]

[12] The edge $\{u, w\}$ is said to reside in the cut $(S, [k] \setminus S)$ if $(u, w) \in (S, [k] \setminus S)$. We shall often associate edges of the cut (i.e., the edges contained in the cut) with the corresponding ordered pairs $\{(u, w) \in (S, [k] \setminus S) : \{u, w\} \in E\}$.

[13] The naive guess is based on considering at the probability that the $t-1$ edges of the cut are assigned the heaviest weights among all edges that are incident at S.

Theorem 11 (analysis of Algorithm 10): *Suppose that v resides in a set S of size at most s such that the subgraph of $G = ([k], E)$ induced by S is t-connected and the cut $(S, [k] \setminus S)$ contains less than t edges. Then, Algorithm 10 outputs S with probability at least $\Omega(s^{-2(t-1)/t}/t)$.*

Hence, we obtain a randomized algorithm that succeeds with probability at least 2/3 by invoking Algorithm 10 for $O(s^{2(t-1)/t}/t) = o(s^2)$ times, which means that the total running time of the resulting algorithm is $o(ds^3)$.

Proof Sketch: As a mental experiment, we assume that weighrs are assigned (at random) to all edges of the graph, and we consider the weight of edges in the cut $C = (S, [k] \setminus S)$ as well as the weight of edges in the lightest spanning tree of the subgraph of G induced by S, denoted G_S. (One may assume that all weights are distinct, since we use infinite precision in this mental experiment.)[14] Using induction on the construction of the set S', one can prove the folllowing (see Exercise 7).

Claim 11.1 (a sufficient condition for success): *If the weight of the each edge in the cut C is larger than the weight of each edge in the lightest spanning tree of G_S, then Algorithm 10 outputs S.*

The complementary claim assert that this sufficient condition is satisfied with probability at least $\Omega(s^{-2(t-1)/t})$.

Claim 11.2 (the main claim): *For natural numbers $t' < t < |S|$, suppose that the cut C has t' edges* (and recall that G_S is t-edge connected). *Then, with probability at least $\Omega(s^{-2t'/t}/t)$, the weight of each edge in the cut C is larger than the weight of each edge in the lightest spanning tree of G_S.*

Towards proving Claim 11.2, it is instructive to consider an auxiliary graph $G' = (S \cup \{x\}, E')$, in which $[k] \setminus S$ is contracted into a single vertex, denoted x. In this graph, x has degree t', whereas all other vertices have degree at least t (since otherwise G_S can not be t-edge connected). The proof of Claim 11.2 can be reduced to the analysis of Karger's edge-contraction algorithm [28], when this algorithm is applied to G'. The edge-contraction algorithm proceeds in iterations, until the multi-graph (which may contain parallel edges) contains exactly two vertices, and it refers to random edge-weights as assigned above. In each iteration, the algorithm chooses the edge $e = \{u, w\}$ of minimum weight, and contracts it, which means that it merges its endpoints into a single vertex that "takes over" the edges of both these endpoints (but not the edges between them).[15] That is, every edge that was incident at either u or w (but not incident at both) becomes incident to the "contracted vertex" (which may cause the appearance of multiple edges, but not of self-loops).

The proof of Claim 11.2 is reduced to the analysis of Karger's edge-contraction algorithm by observing that *if Karger's algorithm does not contract an edge incident at the vertex x, then G_S contains a spanning tree with edges that are each lighter than any edge in the cut C.*[16] Hence,

[14] In the actual algorithms, weights may be chosen in multiples of $1/(ds)^4$, adding an error term of $1/(ds)^2$ (for the case of a possible collision).

[15] Note that, in advanced iterations, there may be edges that are parallel to the edge e.

[16] Indeed, the set of edges contracted by Karger's algorithm (together with the lightest remaining edge) form a spanning tree of the graph G'. Furthermore, if the last edge is incident at x, then the contracted edges form a spanning tree of G_S such that each edge in that tree is lighter than any edge incident at x. Recall that this spanning tree is actually the lightest one (e.g., it is found in a process that corresponds to Kruskal's algorithm for finding a minimum weight spanning tree).

Claim 11.2 follows by lower-bounding the probability that none of the iterations of the Karger's algorithm (applied to G') contacts an edge incident at x. This event occurs if and only if, at each iteration, the current graph contains an edge that is lighter than any edge in C.

The key observation is that the probability that an edge incident at x is contacted in the i^{th} iteration (conditioned on no such edge being contracted in prior iterations) is at most

$$\frac{t'}{((|S|-i+1)\cdot t + t')/2}.$$

This observation is proved as follows.

- At the beginning of the i^{th} iteration (assuming that no edge incident at x was contracted in prior iterations), the graph consists of $|S|-(i-1)$ vertices of degree at least t and a single vertex (i.e., x) of degree t'. The former claim follows from the fact that each vertex corresponds to a subset of S, and by the hypothesis the cut between this subset and the rest of S has at least t edges.

  Hence, the number of edges in the current graph is at least $m \stackrel{\text{def}}{=} ((|S|-i+1)\cdot t + t')/2$.

- The conditioning that *no edge incident at x was contracted in prior iterations* can be interpreted as saying that all edges in the current graph have weights that are larger than the weight of the edges contracted in prior iterations. But if the weight of the edge contracted in the last iteration is ω, then we can think of the weights of the current edges as being uniformly distributed in $[\omega, 1]$. Indeed, we may think that the weights of all current edges are re-selected uniformly at random in the interval $[\omega, 1]$.

  Hence, the probability that an edge incident at x has minimum weight is at most t'/m.

Hence, the probability that we never contracted an edge incident at x is at least

$$\begin{aligned}\prod_{i=1}^{|S|}\left(1-\frac{t'}{(t'+(|S|-i+1)\cdot t)/2}\right) &= \prod_{i=1}^{|S|}\frac{(|S|-(i-1))\cdot t - t'}{(|S|-(i-1))\cdot t + t'} \\ &= \prod_{j=1}^{|S|}\frac{j-(t'/t)}{j+(t'/t)}\end{aligned}$$

Hence, it suffices to lower-bound $\prod_{j=1}^{s}\frac{j-\alpha}{j+\alpha}$, where $\alpha \in [0,1)$. For starters (or as a motivation), note that $\prod_{j=2}^{s}\frac{j-\alpha}{j+\alpha}$ is lower-bounded by $\prod_{j=2}^{s}\frac{j-1}{j+1} = \frac{2}{s\cdot(s+1)}$. In general, using $\prod_{j=2}^{s}\frac{j-\alpha}{j+\alpha} = \Omega(s^{-2\alpha})$, the claim follows (since $\prod_{j=1}^{s}\frac{j-\alpha}{j+\alpha} = \Omega((1-\alpha)\cdot s^{-2\alpha})$), and so does the theorem. [17] ■

[17] For our purpose, it suffices to erstablish the claim for rational α, since here $\alpha = t'/t$. Indeed, we lower-bound $\prod_{j=2}^{s}\frac{j-(t'/t)}{j+(t'/t)}$ by using

$$\begin{aligned}\left(\prod_{j=2}^{s}\frac{j-(t'/t)}{j+(t'/t)}\right)^t &= \prod_{j=2}^{s}\left(\frac{jt-t'}{jt+t'}\right)^t \\ &> \prod_{j=2}^{s}\prod_{i=1}^{t}\frac{(j-1)t+i-t'}{(j-1)t+i+t'}\end{aligned}$$

2.5 Testing cycle-freeness (with two-sided error)

The tester for `Cycle-freeness` is based on the following well-known observation, which generalizes the even more well known fact by which a connected k-vertex graph is a cycle-free if and only it has $k-1$ edges.

Proposition 12 (the number of connected components in a cycle-free graph): *Let $G=([k],E)$ be a graph with m connected components. Then, G is cycle-free if and only if it has $k-m$ edges.*

This proposition follows immediately by considering the number of edges in each connected component of G. Specifically, letting k_i denotes the number of vertices in the i^{th} connected component, we observe that G is cycle-free if for every $i \in [m]$ the i^{th} connected component has exactly $k_i - 1$ edges.

Proposition 12 suggests that cycle-freeness can be tested by comparing the number of edges in the graph to the number of connected components. Estimating the number of edges is quite straightforward, but how can we estimate the number of connected components? The key idea is that it suffices to estimate the number of small connected components, whereas the number of large connected components is small and therefore can be ignored.

The number of small connected components is estimated by repeating the following experiment an adequate number of times: Select uniformly a random vertex $v \in [k]$, perform a truncated search starting at v and suspending the search if too many vertices are encountered, and use as estimator the reciprocal of the size of the (small) connected component that was fully visited in this searcher (using zero as estimator in case that the search was suspended before the component was fully visited). Hence, if a small connected component has size s, then its contribution to the expected value of this experiment is $\frac{s}{k} \cdot \frac{1}{s}$, where the first factor represents the probability that a vertex residing in this component was selected and the second factor represents its contribution in such a case. Repeating the experiment for a sufficient number of times, we obtain the following algorithm.

Algorithm 13 (two-sided error tester for cycle-freeness (in the bounded-degree graph model)): *On input parameters d, k and ϵ and oracle access to the incidence function of a k-vertex graph $G=([k],E)$, which has maximum degree d, the algorithm proceeds as follows.*

1. *Using $O(1/\epsilon^2)$ random queries* (in $[k] \times [d]$)*, the algorithm estimates the number of edges up to $\pm\epsilon dk/6$. Let $\widetilde{e}$ denote this estimate.*[18]

$$\begin{aligned} &= \prod_{i=1}^{st-t} \frac{t-t'+i}{t+t'+i} \\ &= \frac{\prod_{i=1}^{2t'}(t-t'+i)}{\prod_{i=1}^{2t'}(st-t'+i)} \\ &> \frac{(2t'/3)^{2t'}}{(st+t')^{2t'}} \,. \end{aligned}$$

Hence, $\prod_{j=2}^{s} \frac{j-(t'/t)}{j+(t'/t)} = \Omega(t'/st)^{2t'/t} = \Omega(1/s)^{2t'/t}$, and the claim follows.

[18] **Advanced comment:** One can reduce the number of queries used in this step to $O(\max(1/\epsilon, 1/d\epsilon^2))$ by assuming that $|E| \leq m \stackrel{\text{def}}{=} \max(2k, \epsilon dk/2)$, since in this case we seek a multiplicative approximation factor of $1 \pm \frac{\epsilon dk/6}{m}$ for an event that occurs with probability smaller than m/dk, and a random sample of $O((m/dk)^{-1} \cdot (\epsilon dk/m)^{-2})$ pairs will do. The foregoing assumption can be justified by augmenting the algorithm with a step that checks this condition (and rejects if $|E| \leq \max(2k, \epsilon dk/2)$ seems not to hold, since this indicates that $|E| > k$). Such a check can be implemented using $O(1/\epsilon)$ queries (see Exercise 8).

2. *Estimates the number of connected components up to* $\pm\epsilon dk/6$ *by selecting at random* $t = O(1/d\epsilon)^2$ *start vertices,* $v_1, ..., v_t$*, and incrementing the counter by* k/s_i *if the search started at* v_i *encountered* $s_i < \ell \stackrel{\text{def}}{=} 12/(d\epsilon)$ *vertices* (and by zero otherwise)*. That is, for each* $i \in [t]$*, the algorithm proceeds as follows:*

   (a) *Performs a linear-time* (e.g., BFS or DFS) *search starting at* v_i*, and suspending the search if more than* ℓ *vertices are encountered in it.*
   Hence, this scan involves $O(\ell \cdot d) = O(1/\epsilon)$ *queries.*

   (b) *If the entire connected component is scanned and its size is* s_i*, then the counter is incremented by* k/s_i*.*

   Divide the accumulated sum by t*, and denote the result by* $\widetilde{m}$*.*

3. *If* $\widetilde{e} \geq k - \widetilde{m} + \epsilon dk/3$*, then reject. Otherwise, accept.*

The query complexity of Algorithm 13 is $O(1/\epsilon^2) + t \cdot O(1/\epsilon) = O(1/\epsilon^2) + O(1/d^2\epsilon^3)$. The algorithm may err (with small probability) both in the case that the graph is cycle-free and in the case it is far from being cycle-free, where the source of error probability is the estimates that are performed in Steps 1 and 2.

Note that only connected components of size at most $\ell = 12/d\epsilon$ contribute to the estimate $\widetilde{m}$, whereas $\widetilde{m}$ is supposed to estimate the number of all connected components. However, since the number of the larger (than $12/d\epsilon$) connected components is at most $(d\epsilon/12) \cdot k$, we can ignore their contribution. Details follow.

Claim 14 (analysis of Algorithm 13): *Algorithm 13 is a* (two-sided error) *tester for* `Cycle-freeness`*.*

Proof Sketch: The following analysis presumes that the samples used in Steps 1 and 2 provides the stated estimates, with high probability. (This is fact is easy to establish using an additive Chernoff bound, while noting that the desired estimates are, respectively, an $\Omega(\epsilon)$ and an $\Omega(d\epsilon)$ fraction of the range.)[19] When analyzing Step 2, let $m' \geq m - \epsilon dk/12$ denote the number of small connected components, and prove that (w.h.p) $|\widetilde{m} - m'| \leq \epsilon dk/12$. (Representing the contribution of the i^{th} search by the random variable ζ_i, note that $\mathbb{E}[\zeta_i] = m'$).[20]

If $G = ([k], E)$ is cycle free and has m connected components, then $|E| = k - m$. In this case, with high probability it holds that $\widetilde{e} \leq |E| + \epsilon dk/6 = k - m + \epsilon dk/6$, whereas $\widetilde{m} \leq m' + \epsilon dk/12 \leq m + \epsilon dk/12$ (since $m' \leq m$ by definition). Hence, $\widetilde{e} + \widetilde{m} \leq (k - m + \epsilon dk/6) + (m + \epsilon dk/12) < k + \epsilon dk/3$, and Algorithm 13 accepts.

On the other hand, if $G = ([k], E)$ has m connected and is ϵ-far from being cycle-free, then $|E| \geq k - m + \epsilon dk/2$ (since otherwise G can be made cycle-free by omitting at most $\epsilon dk/2$ edges). Now, with high probability, it holds that $\widetilde{e} \geq |E| - \epsilon dk/12 \geq k - m + 5\epsilon dk/12$, whereas $\widetilde{m} \geq m' - \epsilon dk/12 \geq m - \epsilon dk/6$ (since $m' > m - \epsilon dk/12$, because $m - m'$ represents the number of large (i.e., larger than $12/d\epsilon$) connected components). In this case, $\widetilde{e} + \widetilde{m} \geq (k - m + 5\epsilon dk/12) + (m - \epsilon dk/6) > k + \epsilon dk/3$, and Algorithm 13 rejects. ■

[19] In Step 1, each random query, which is effectively answered by a value in $\{0, 1\}$, is an unbiased estimator of $|\{(v, i) \in [k] \times [d] : g(v, i) \neq 0\}|/dk$, and we consider the probability that the average of $O(1/\epsilon^2)$ such estimators deviates from the correct value by more than $\epsilon/6$. In Step 2, each search returns a value in $[0, k]$ that is an unbiased estimator of the number of small connected components, and we consider the probability that the average of $O(1/d\epsilon)^2$ such estimators deviates from the correct value by more than $\epsilon dk/12$.

[20] For $j \in [m']$, let C_j denote the vertex set of the j^{th} connected component. Then, $\mathbb{E}[\zeta_i] = \sum_{j \in [m']} \sum_{v_i \in C_j} \frac{1}{k} \cdot \frac{k}{|C_j|}$.

Improving over Algorithm 13. Recall that Step 2(a) of Algorithm 13 performs a search aimed at detecting small connected components towards estimating their number, where a connected component is defined as small if it has size at most $\ell = 12/d\epsilon$. But when upper-bounding the cost of such a search, we used $\ell \cdot d$ as a bound. This fails to capitalize on the fact that *if we encountered more edges than vertices in the current search, then we found a cycle in the current connected component*. Hence, it is begging to suspend the search in such a case, and reject. Note that the modified algorithm has complexity $O(1/\epsilon^2) + O(t \cdot \ell) = O(1/\epsilon^2) + O(1/d\epsilon)^3$, whereas its verdicts are at least as reliable as those of the original algorithm: On the one hand, graphs that are cycle-free are accepted by the modified algorithm with the same probability as they are accepted by Algorithm 13, since the modification has no effect in this case. On the other hand, graphs that are not cycle-free are rejected by the modified algorithm with probability that is lower bounded by the probability that they are rejected by by Algorithm 13, since the modification can only increase the rejection probability. Hence, we get:

Theorem 15 (an alternative two-sided error tester for cycle-freeness): *Testing* `Cycle-freeness` (in the bounded-degree graph model) *can be performed in time* $O(\epsilon^{-2} + d^{-3} \cdot \epsilon^{-3})$.

Recall that the tester establishing Theorem 15 has two-sided error probability. As we shall see in the next section, two-sided error probability is unavoidable for a tester for `Cycle-freeness` that has query complexity $\mathrm{poly}(1/\epsilon)$. Actually, two-sided error probability is unavoidable even for query complexity $f(\epsilon) \cdot o(\sqrt{k})$, for any function $f : (0, 1] \to \mathbb{N}$ (see Theorem 17).

3 Lower bounds

When $d \leq 2$ the graph consists of a collection of isolated paths and edges; actually, if $d = 1$, the graph consists of a collection of isolated edges (and isolated vertices). Since any graph property of interest is either trivial or easy to test in these cases, we focus on the case that $d \geq 3$.

Teaching note: This section relies on a lower bound technique presented in the lecture on the subject, and called the method of indistinguishable distributions. This technique is simple enough to pick-up on the fly, but it may be better to study the corresponding lecture notes first.

3.1 Bipartitness

In contrast to the situation in the dense graph model, in the the bounded degree graph model there exists no `Bipartite` tester of complexity that is independent of the graph size. This fact reflects the fact that being far from `Bipartiteness` does not require having constant size cycles of odd length. Actually, graphs that are far from being bipartite may lack odd-length cycles of sub-logarithmic length (see Exercise 9), and so testing `Bipartiteness` (at least with one-sided error probability) cannot be performed in sub-logarithmic (in k) query complexity. The stronger lower bound presented next goes beyond these existential considerations.

Theorem 16 (lower bound on the complexity of testing `Bipartiteness` (in the bounded-degree graph model)): *For proximity parameter* $\epsilon = 0.01$ *and any degree bound* $d \geq 3$*, testing* `Bipartiteness` *requires* $\Omega(\sqrt{k})$ *queries.*

Note that graphs that are 0.01-far from being bipartite do have odd-length cycles of logarithmic length (see Exercise 10).[21]

Proof Sketch: We shall focus on $d = 3$, and prove the lower bound using 3-regular graphs. For any (even) k, we consider the following two families of graphs:

1. The first family, denoted $\mathcal{G}_1$, consists of all 3-regular graphs that are composed of the union of a Hamiltonian cycle and a perfect matching (which does not match vertices that are adjacent on the cycle). That is, there are k edges forming a simple k-vertex cycle, and the other $k/2$ edges are a perfect matching.

2. The second family, denoted $\mathcal{G}_2$, is the same as the first *except* that the perfect matchings allowed are restricted such that the distance on the cycle between every two vertices that are connected by a perfect matching edge must be odd. Equivalently, labeling the vertices according to their location on the cycle (so that the i^{th} vertex is adjacent to the $i+1^{\text{st}}$ vertex, for every $i \in [k]$),[22] we requite that if $\{i, j\}$ is a perfect matching edge, then $i \not\equiv j \pmod 2$.

Clearly, all graphs in $\mathcal{G}_2$ are bipartite. It can be shown (see Claim 16.1) that *almost all graphs in $\mathcal{G}_1$ are far from being bipartite*. On the other hand, one can prove (see Claim 16.2) that *an algorithm that performs $o(\sqrt{k})$ queries cannot distinguish between a graph chosen randomly from $\mathcal{G}_2$* (which is always bipartite) *and a graph chosen randomly from $\mathcal{G}_1$* (which with high probability is far from bipartite). Loosely speaking, this is the case since in both cases the algorithm is unlikely to encounter a cycle (among the vertices that it has inspected).

Claim 16.1 (almost all graphs in $\mathcal{G}_1$ are far from being bipartite): *All but an exponentially vanishing fraction of the graphs in $\mathcal{G}_1$ are 0.01-far from being bipartite.*

Proof: We consider a uniformly distributed graph in $\mathcal{G}_1$, and upper-bound the probability that it can be made bipartite by omitting $0.01 \cdot dk/2 = 0.015k$ of its edges. We shall actually consider an omission of $0.015k$ of its (Hamiltonian) cycle edges and $0.015k$ of the matching edges. For each of the possible $\binom{k}{0.015k} < 2^{0.1124k}$ choices of $0.015k$ cycle edges, we consider all $2^{0.015k-1}$ legal 2-colorings of the resulting collection of $0.015k - 1$ paths. Now, we lower-bound the probability that the random perfect matching does not have more than $0.015k$ edges that violate this fixed 2-coloring. This is done by selecting these $k/2$ edges in iterations, while noting that in the $i + 1^{\text{st}}$ iteration a violating edge is selected with probability at least

$$\min_{j \in \{0,\ldots,k-2i\}} \left\{ \frac{\binom{j}{2} + \binom{k-2i-j}{2}}{\binom{k-2i}{2}} \right\} \geq \frac{2 \cdot \binom{(k-2i)/2}{2}}{\binom{k-2i}{2}} \approx \frac{1}{2}$$

where the approximation holds for any $i \leq (k/2) - \omega(1)$. Hence, the probability that less than $0.015k$ violating edges occur is less that $e^{-0.485^2 \cdot k/2 + O(1)}$. Using a union bound, the claim follows. ■

Claim 16.2 (indistinguishability by $o(\sqrt{k})$-query algorithms): *An algorithm that performs q queries can distinguish between a graph chosen randomly from $\mathcal{G}_1$ and a graph chosen randomly from $\mathcal{G}_2$ with gap of at most q^2/k.*

[21] A weaker bound (i.e., odd-length cycles of polylogarithmic length) follows directly from Theorem 21.

[22] Indeed, we identify the $k + 1^{\text{st}}$ vertex with the first one.

Proof: We shall assume, to the benefit of the algorithm, that the incidence function g is "nice" in the sense that for every vertex v it holds that $g(v,1)$ is successor of v on the (Hamiltonian) cycle, whereas $g(v,2)$ is the predecessor of v on that cycle (which means that $g(v,3)$ is the vertex matched to v by the perfect matching). We assume, without loss of generality, that the algorithm does not make queries for which it knows the answers (e.g., after making the query $g(v,1)$, it does not make the query $g(g(v,1),2)$). Recall that we use $d = 3$ and 3-regular graphs; hence, the queries of the algorithm correspond to edges (i.e., the query (v,i) corresponds to the edge $\{v, g(v,i)\}$). These conventions merely facilitate the verification of the key observation that appears next.

We consider an iterative process of generating a randomly distributed graph in $\mathcal{G}_1$ (resp., in $\mathcal{G}_2$) by answering queries of the algorithm, while keeping track of the "knowledge graph" of the algorithm (at each point), where the knowledge graph is defined as the subgraph consisting of the edges that correspond to the algorithm's queries so far. The key distinction is between vertices that are in the knowledge graph (i.e., vertices that have appeared either in a previous query of the algorithm or as a previous answer provided to it) and those that are not in this graph. The *key observation* is that *as long as the knowledge graph of the algorithm is cycle-free and contains relatively few edges, both generation processes* (i.e., the one constructing a random element of $\mathcal{G}_1$ and the one constructing a random element of $\mathcal{G}_2$) *behave in a very similar manner*. Actually, each of these processes answers the $i+1^{\text{st}}$ query with an old vertex (i.e., a vertex in the knowledge graph) with probability at most $\frac{2i}{k-1}$, and otherwise the answer is uniformly distributed among the labels that do not appear in the current knowledge graph. Hence, the distinguishing gap of the algorithm is upper-bounded by the probability that at least one of the q queries is answered by an old vertex, and the claim follows. ■

This completes the proof of the theorem. ■

3.2 Applications to other properties

The proof of Theorem 16 can be adapted to yield hardness results for two natural testing problems, which seem unrelated to testing `Bipartitness`.

Application to testing cycle-freeness. Recall that, in Section 2.5, we presented *two-sided error* testers of query complexity $\text{poly}(1/\epsilon)$ for `Cycle-freeness`. We now show that the two-sided error was inherent to these testers, since `Cycle-freeness` does not have a *one-sided error* tester of complexity that depends on the proximity parameter only.

Theorem 17 (lower bound on the query complexity of one-sided error testers for `Cycle-freeness`): *For any degree bound $d \geq 3$, every one-sided error $(1/d)$-tester for* `Cycle-freeness` *has query complexity $\Omega(\sqrt{k})$.*

Proof: We use any of the two families of graphs presented in the proof of Theorem 16, while noting that each of these graphs is 1/3-far from being cycle-free (since it has $0.5k+1$ superfluous edges). Hence, any 1/3-tester for `Cycle-freeness` is required to reject each of these graphs with probability at least 2/3. On the other hand, the proof of Claim 16.2 actually establishes that a q-query machine sees a cycle in a random graph (drawn from any of these families), with probability at most q^2/k. Hence, with probability at least $\frac{2}{3} - \frac{1}{2} > 0$, a tester of query complexity $\sqrt{k/2}$ rejects some graph without seeing a cycle in the subgraph that it has explored, which means that this tester cannot have one-sided error. ■

Teaching note: The rest of Section 3 is intended for optional independent reading.

Application to testing expansion. Fixing a constant $c > 0$, we say that the graph $G = ([k], E)$ is c-expanding if, for every set $S \subset [k]$ of cardinality at most $k/2$, it holds that $|\Gamma^+(S)| \geq c \cdot |S|$, where

$$\Gamma^+(S) \stackrel{\text{def}}{=} \{u \in ([k] \setminus S) : \exists v \in S \text{ s.t. } \{u, v\} \in E\} \tag{2}$$

denotes the set of vertices that are not in S but neighbor some vertices in S. One can show that, for sufficiently small constant $c > 0$ and all sufficiently large k, with very high probability, a random 3-regular graph is c-expanding.

Theorem 18 (lower bound on the query complexity testing c-expansion): *For sufficiently small constant $c > 0$ and any degree bound $d \geq 3$, every (c/d)-tester for c-*`expansion` *has query complexity $\Omega(\sqrt{k})$.*

Proof Sketch: We start with the family $\mathcal{G}_1$ presented in the proof of Theorem 16, and show (see Claim 18.1) that, with very high probability, a uniformly distributed (in $\mathcal{G}_1$) graph is c-expanding. We then show that a $o(\sqrt{k})$-query algorithm cannot distinguish a uniformly distributed k-vertex graph (drawn from $\mathcal{G}_1$) from a k-vertex graph that consists of two isolated $k/2$-vertex graphs drawn from (the $k/2$-vertex version of) $\mathcal{G}_1$. The theorem follows by noting that the latter graphs are far from being expanding (in any reasonable sense of that term), since the vertices of the first $k/2$-vertex graph neighbor no vertex in the second $k/2$-vertex graph.

Claim 18.1 (almost all graphs in $\mathcal{G}_1$ are expanding): *For sufficiently small constant $c > 0$, with very high probability, a uniformly distributed* (in $\mathcal{G}_1$) *graph is c-expanding.*

Proof Sketch: Using a (carefully executed) union bound, we upper-bound the probability that there exists a set S of size at most $k/2$ such that $|\Gamma^+(S)| < c \cdot |S|$. Specifically, for every set $S \subseteq [k]$, we consider the random variable X_S that represents the size of $\Gamma^+(S)$ in a graph drawn at random (from $\mathcal{G}_1$). The union bound is based on a partition of the possible sets S to two classes.

1. Sets S such that the subgraph induced by S on the graph consisting only of the edges of the Hamiltonian cycle has at least $c \cdot |S|$ connected components.

   In this case, $\mathbf{Pr}[X_S \geq c \cdot |S|] = 1$, merely by virtue of the cycle edges.[23] Hence, sets of this type contribute nothing to the probability that there exists a set S of size at most $k/2$ such that $|\Gamma^+(S)| < c \cdot |S|$.

2. Sets S that have less than $c \cdot |S|$ such connected components.

   We first observe that the number of such sets of size s is at most $2 \cdot \sum_{i \in [cs]} \binom{k}{2i}$, since each choice of $2i$ vertices determine two possible i-long sequences of disjoint sectors of the cycle. Note that $2 \cdot \sum_{i \in [cs]} \binom{k}{2i} = \exp(H_2(2cs/k) \cdot k)$, where H_2 is the binary entropy function.[24] Next, fo each such set S, we show that

   $$\mathbf{Pr}[X_S < c \cdot |S|] \leq \binom{s}{cs} \prod_{i \in [(1-c)s/2]} \frac{s - 2(i-1)}{k - 2(i-1)} = 2^{H_2(c) \cdot s} \cdot (s/k)^{\Omega(s)},$$

[23] Note that, when "going around the cycle", the last vertex in each of the aforementioned connected components neighbors a distinct vertex not in S.

[24] That is, $H_2 : [0,1] \to [0,1]$ such that $H_2(p) = p \log(1/p) + (1-p) \log(1/(1-p))$.

where the inequality is due to the probability that $s - cs$ specific vertices in S are matched to vertices of S (by the perfect matching).[25] Taking a union bound over all relevant sets S, we obtain the probability bound $\exp(H_2(2cs/k) \cdot k + H_2(c) \cdot s - \Omega(s \log(k/s))$, which equals $\exp(-\Omega(s))$ when $c > 0$ is sufficiently small.

The claim follows. ■

Claim 18.2 (indistinguishability by $o(\sqrt{k})$-query algorithms): *Let $\mathcal{G}'_1$ denote the set of k-vertex graphs that consist of two isolated $k/2$-vertex graphs taken from the $k/2$-vertex version of $\mathcal{G}_1$. Then, a q-query can distinguish between a graph chosen uniformly at random in $\mathcal{G}_1$ and a graph chosen uniformly in $\mathcal{G}'_1$ with gap of at most q^2/k.*

Claim 18.2 follows by noting that the argument used in the proof of Claim 16.2 extends to $\mathcal{G}'_1$; that is, a q-query algorithm re-visited an old vertex (i.e., obtain an answer that is already in its knowledge graph) when inspecting a random graph drawn uniformly from $\mathcal{G}'_1$, with probability at most q^2/k. Note that as long as no old vertex is re-visited, the two distributions of answers are identical. Using Claim 18.1, the theorem follows. ■

3.3 Linear lower bounds

While the $\Omega\sqrt{k})$ lower bounds capitalize on the difficulty of detecting a cycle in the graph (or, equivalently, on the difficulty of reaching the same vertex in two non-trivially different ways), this strategy is unlikely to work for obtaining higher lower bounds. Indeed, different methods are used for obtaining results of the following type.

Theorem 19 (lower bound on the complexity of testing `3-Colorability` (in the bounded-degree graph model)): *For some proximity parameter $\epsilon > 0$ and a degree bound d, testing* `3-Colorability` *requires $\Omega(k)$ queries.*

The proof of Theorem 19 can be found in [7]. Here we only sketch an alternative proof, also due to [7], that only applies to the one-sided error case. However, in this case, the proof exhibits a general trade-off between $\epsilon \in (0, 1/3)$ and $d \geq 3$ (and the constant that is hidden in the Ω-noation). We note that 1/3-testing `3-Colorability` is trivial, since every graph is 1/3-close to being 3-colorable.[26]

Proof outline for the one-sided error case: Let $\epsilon : \mathbb{N} \to (0, 1/3)$ and $\rho : \mathbb{N} \to (0, 1)$. The basic idea is that, for every $d \geq 3$, the exist d-regular k-vertex graphs that, one the one hand, are $\epsilon(d)$-far from being 3-colorable but, on the other hand, all their $\rho(d) \cdot k$-vertex induced subgraphs are 3-colorable. Such a graph must be rejected with probability at least 2/3 by any $\epsilon(d)$-tester, but if this tester rejects without seeing a subgraph that is not 3-colorable, then it is not of the one-sided error type (because it would reject with positive probability a graph that consists solely

[25] Denoting this set of vertices by S', we upper-bound the latter probability by considering an iterative process in which we pick for each vertex in S' a matched vertex, and continue only if the matched vertex is in S. We halt the process after $(s - cs)/2$ steps, although we should continue till all vertices in S' are matched.

[26] Note that a random assignment of three colors to the vertices of the graph $G = ([k], E)$ is expected to have exactly $|E|/3$ monochromatic edges.

of that subgraph). Hence, all that is left is to show the existence of graphs with the aforementioned property.

We shall show that such graphs exist, by showing that a random d-regular graph satisfies the aforementioned property, with very high probability. Actually, it will be instructive to consider a random d-regular multi-graph (which may contain parallel edges), and note that if it satisfies the property, then so does the graph obtained from it by omitting parallel edges (which are extremely few in number).

Claim 19.1 (a random d-regular graph is far from being 3-colorable): *Suppose that $G = ([k], E)$ is generated by take the union of d perfect matching of the elements of $[k]$. If $d = \Omega(((1/3) - \epsilon)^{-2})$, then, with probability at least $1 - \exp(-\Omega(dk))$, the graph G is ϵ-far from 3-colorable.*

The proof of Claim 19.1 is rather technical and can be found in [7]. The first step is taking a union bound over all 3^k possible 3-partitions of $[k]$, denoted (V_1, V_2, V_3), and upper-bounding the probability that less than $\epsilon dk/2$ of the edges have endpoints in the same V_i. Analyzing the latter event would have been easy if the $dk/2$ edges were selected independent of one another. In such a case, we would have had $dk/2$ independent events, each succeeding with probability $\sum_{i\in[3]} \binom{|V_i|}{2}/\binom{k}{2} < 1/3$, and (by a Chernoff bound) the probability of having less than $\epsilon dk/2$ successes is $\exp(-\Omega(((1/3) - \epsilon)^2 dk))$.

Claim 19.2 (a random d-regular graph has large 3-colorable subgraphs): *Let G be as in Claim 19.1. For $\rho = \mathrm{poly}(1/d)$, with probability at least $1 - \exp(-\Omega(dk))$, the subgraph of G induced by any set of ρk vertices is 3-colorable.*

The proof of Claim 19.2 reduces to showing that for any set S of at most ρk vertices, the subgraph induced by S, denoted G_S, contains a vertex of degree less than three. Again, the actual proof is technical (see [7]).[27] The claim follows by considering a minimal set S of size at most ρk such that G_S is not 3-colorable, and reaching a contradiction by using the fact that this set contains a vertex v of degree at most two in G_S (since, by minimality, $G_{S\setminus\{v\}}$ is 3-colorable, but then contradiction is reached by extending this 3-coloring to G_S). ■

4 Testing by random walks

As noted at the beginning of Section 3, we focus on the case of $d \geq 3$, since when $d \leq 2$ the graph consists of a collection of isolated paths and edges and any graph property of interest are either trivial or easy to test in that case.

The algorithms presented in this section are based on taking "random walks on the input graph" (defined momentarily). The intuition is that such walks may provide information that

[27] **Advanced comment:** We upper bound the probability that a subgraph induced by a set of $s \leq \rho k$ vertices has no vertex of degree lower than three. We actually upper-bound the probability that such a set has at least $m = 3s/2 < k/4$ edges, by a union bound on all $\binom{k}{s} = O(k/s)^s$ possible choices of this set of vertices. For each such choice of s vertices, we upper-bound the probability that the induced subgraph has m edges by $\binom{\binom{s}{2}}{m} \cdot \prod_{i=0}^{m-1}(d/(k-2i))$, which is upper-bounded by $O(s^2/m)^m \cdot (2d/k)^m = O(s^2 d/mk)^m$. Hence, the union bound gives

$$\begin{aligned} O(k/s)^s \cdot O(s^2 d/mk)^m &= O(k/s)^s \cdot O(s^2 d/sk)^{1.5s} \\ &= O(sd^3/k)^{0.5s} \end{aligned}$$

and the claim follows.

extends beyond what can be deduced based on local searches. It is not *a priori* clear whether this additional information may be beneficial to our (testing) goals, but for sure taking a random walk is a natural thing to try if one wants to get beyond local searches and still maintain sublinear complexity.

By a random walk of length ℓ on a graph $G = ([k], E)$ we mean a path $(v_0, ..., v_\ell)$ in G selected at random such that v_0 is uniformly distributed in $[k]$ and v_i is uniformly distributed among the neighbors of v_{i-1}.

4.1 Testing `Bipartiteness`

The executive summary is that the lower bound of Theorem 16 is essentially tight; that is, for every constant $\epsilon > 0$, `Bipartiteness` can be ϵ-tested in $\widetilde{O}(\sqrt{k})$ queries. Furthermore, the following algorithm constitutes a `Bipartite` tester of running time $\mathrm{poly}((\log k)/\epsilon) \cdot \sqrt{k}$. Essentially, the algorithm selects a random start vertex, takes $\widetilde{O}(\sqrt{k})$ random walks from it, each of $\mathrm{poly}(\epsilon^{-1} \log k)$-length, and rejects if an only if the subgraph encountered in these walks is bipartite.

The natural question is why does this algorithm reject graphs that are far away from being bipartite. The intuitive answer is as follows. Fixing a start vertex s, if many vertices are reached by an odd-length random walk from s with about the same probability as by an even-length random walk from s, then (with high probability) an odd-length cycle will be formed in the explored subgraph, and the algorithm will reject. Otherwise, we can color each vertex according to the more frequent parity of the random walk in which the vertex is reached, and infer that there are relatively few monochromatic edges. Hence, if the graph is far from being bipartite, then the algorithm will reject with high probability. This intuition will be implemented in Claims 21.3 and 21.2, respectively. But before doing so, let us spell out the algorithm.

Algorithm 20 (testing `Bipartiteness` (in the bounded-degree graph model)): *On input d, k, ϵ and oracle access to an incidence function for an k-vertex graph, $G = ([k], E)$, of degree bound d, repeat the following steps $t \stackrel{\text{def}}{=} \Theta(\frac{1}{\epsilon})$ times:*

1. *Uniformly select s in $[k]$.*
2. (Try to find an odd-length cycle through vertex s):
   (a) *Perform $m \stackrel{\text{def}}{=} \mathrm{poly}((\log k)/\epsilon) \cdot \sqrt{k}$ random walks starting from s, each of length $\ell \stackrel{\text{def}}{=} \mathrm{poly}((\log k)/\epsilon)$.*
   (b) *Let R_0* (respectively, *R_1*) *denote the set of vertices reached from s in an even* (respectively, odd) *number of steps in any of these walks. That is, assuming that ℓ is even, for every such walk $(s = v_0, v_1, ..., v_\ell)$, place $v_0, v_2, ..., v_\ell$ in R_0 and place $v_1, v_3, ..., v_{\ell-1}$ in R_1.*
   (c) *If $R_0 \cap R_1$ is not empty, then* reject.

If the algorithm did not reject in any of the foregoing t iterations, then it accepts.

The time (and query) complexity of Algorithm 20 is $t \cdot m \cdot \ell \cdot \log d = \mathrm{poly}(1/\epsilon) \cdot \widetilde{O}(\sqrt{k})$, where the $\log d$ factor is due to determining the degree of each vertex encountered in the random walk (before selecting a neighbor at random). It is evident that the algorithm always accepts a bipartite graph. Furthermore, Algorithm 20 can be easily modified so that in case of rejection it outputs

an odd-length cycle of length $\mathrm{poly}((\log k)/\epsilon)$, which constitutes a "witness" that the graph is not bipartite. The difficult part of the analysis is proving the following.

Theorem 21 (Algorithm 20 is a `Bipatitenss` tester (for the bounded-degree graph model)): *If the input graph is ϵ-far from being bipartite, then Algorithm 20 rejects with probability at least $2/3$.*

The proof of Theorem 21 is quite involved. We shall only provide a proof of the "rapid mixing" case, and hint at the ideas used towards extending this proof to the general case.

The special case of rapid mixing graphs. We consider the special case in which the input graph has a "rapid mixing" feature (defined next). It is convenient to modify the random walks so that at each step each neighbor is selected with probability $1/2d$, and otherwise (with probability at least $1/2$) the walk remains in the present vertex. Such a modified random walk is often called a lazy random walk. Indeed, using a lazy random walk, the next vertex on a walk can be selected at unit cost (rather than at $\log d$ cost, which is required for determining the degree of the current vertex).

We will consider a single execution of Step 2, starting from an arbitrary vertex, s, which is fixed for the rest of the discussion. (Indeed, in this special case it suffices to execute Step 2 once and the start vertex s may be arbitrary (i.e., need not be selected at random).) The rapid mixing feature that we assume here is that, for every vertex v, a lazy random walk of length ℓ starting at s reaches v with probability approximately $1/k$ (say, up-to a factor of 2).

Definition 21.1 (the rapid mixing feature): *Let $(v_1, ..., v_\ell) \leftarrow \mathcal{RW}_\ell$ be an ℓ-step lazy random walk* (on $G = ([k], E)$) *starting at $v_0 \stackrel{\text{def}}{=} s$; that is, for every $\{u, v\} \in E$ and every $i \in [\ell]$, it holds that*

$$\mathbf{Pr}_{(v_1,...,v_\ell)\leftarrow\mathcal{RW}_\ell}[v_i = v | v_{i-1} = u] = \frac{1}{2d} \tag{3}$$

$$\mathbf{Pr}_{(v_1,...,v_\ell)\leftarrow\mathcal{RW}_\ell}[v_i = u | v_{i-1} = u] = 1 - \frac{d_G(u)}{2d} \tag{4}$$

where $d_G(u)$ denotes the degree of u in G. Then, the graph G is said to be rapidly mixing *if, for every $v_0 \in [k]$, it holds that*

$$\frac{1}{2k} < \mathbf{Pr}_{(v_1,...,v_\ell)\leftarrow\mathcal{RW}_\ell}[v_\ell = v] < \frac{2}{k} \tag{5}$$

Note that if the graph is an expander, then it is rapidly mixing (since $\ell = \omega(\log k)$).

The key quantities in the analysis are the following probabilities that refer to the parity *of the length of a path obtained from the lazy random walk by omitting the self-loops* (transitions that remain at the current vertex). Let $p_0(v)$ (respectively, $p_1(v)$) denote the probability that a *lazy random walk of length ℓ, starting at s, reaches v while making an even* (respectively, *odd*) *number of real* (i.e., non-self-loop) *steps*. That is, for every $\sigma \in \{0, 1\}$ and $v \in [k]$,

$$p_\sigma(v) \stackrel{\text{def}}{=} \mathbf{Pr}_{(v_1,...,v_\ell)\leftarrow\mathcal{RW}_\ell}[v_\ell = v \ \wedge \ |\{i \in [\ell] : v_i \neq v_{i-1}\}| \equiv \sigma \pmod 2] \tag{6}$$

The path-parity of the walk $(v_1, ..., v_\ell)$ is defined as $|\{i \in [\ell] : v_i \neq v_{i-1}\}| \bmod 2$.

By the rapid mixing assumption (for every $v \in [k]$), it holds that

$$\frac{1}{2k} < p_0(v) + p_1(v) < \frac{2}{k}. \tag{7}$$

We consider two cases regarding the sum $\sum_{v\in[k]} p_0(v)p_1(v)$: If the sum is (relatively) "small", then we show that $[k]$ can be 2-partitioned so that there are relatively few edges between vertices that are placed in the same side, which implies that G is close to being bipartite. Otherwise (i.e., when the sum is not "small"), we show that, with significant probability, when Step 2 is started at vertex s, it is completed by rejecting G. These two cases are analyzed in the following two (corresponding) claims.

Claim 21.2 (a small sum implies closeness to being bipartite): *Suppose* $\sum_{v\in[k]} p_0(v)p_1(v) \leq 0.01\epsilon/k$. *Let* $V_1 \stackrel{\text{def}}{=} \{v \in [k] : p_0(v) < p_1(v)\}$ *and* $V_2 = [k] \setminus V_1$. *Then, the number of edges with both end-points in the same* V_σ *is bounded above by* $\epsilon dk/2$, *which implies that* G *is* ϵ*-close to being bipartite.*

Proof Sketch: Consider an edge $\{u, v\}$ such that both u and v are in the same V_σ, and assume, without loss of generality, that $\sigma = 1$. Then, by the (lower bound of the) *rapid mixing hypothesis*, both $p_1(v)$ and $p_1(u)$ are greater than $\frac{1}{2} \cdot \frac{1}{2k}$. Using the following two observations, we infer that $p_0(v) > \frac{1}{3d} \cdot p_1(u)$:

1. An $(\ell-1)$-step random walk of path-parity 1 ending at u is almost as likely as an ℓ-step random walk of path-parity 1 ending at u. That is, letting $p'_1(u)$ denote the probability that a $(\ell-1)$-step lazy random walk (starting at s) reaches v while making an odd number of real (i.e., non-self-loop) steps, it holds that $p'_1(u) \approx p_1(u)$.

   This can be shown by noting that if we take a random $(\ell-1)$-step walk of the type measured in $p'_1(u)$ and insert a "staying in place" step at a random location in it, then we obtain a distribution that is very close to the one measured in $p_1(u)$: see Exercise 11 for details.

2. If an $(\ell-1)$-step walk reaches u, then, with probability exactly $1/2d$, it continues to v in the next step; hence, $p_0(v) \geq p'_1(u)/2d$. Furthermore, $p_0(v) \geq \sum_{u\in V_1:\{u,v\}\in E} p'_1(u)/2d$, since the events that correspond to different u's are disjoint.

Thus, the edge $\{u, v\}$ contributes at least $\frac{p_1(u)}{3d} \cdot p_1(v) \geq \frac{(1/4k)^2}{3d}$ to the sum $\sum_{w\in[k]} p_0(w)p_1(w)$. More formally, we have

$$\begin{aligned}
\sum_{v\in[k]} p_0(v)p_1(v) &= \sum_{\sigma\in\{0,1\}} \sum_{v\in V_\sigma} p_{1-\sigma}(v)p_\sigma(v) \\
&\geq \sum_{\sigma\in\{0,1\}} \sum_{v\in V_\sigma} \sum_{u\in V_\sigma:\{u,v\}\in E} \frac{p_\sigma(u)}{3d} \cdot p_\sigma(v) \\
&\geq \sum_{\sigma\in\{0,1\}} |\{\{u, v\} \in E : u, v \in V_\sigma\}| \cdot \frac{(1/4k)^2}{3d}
\end{aligned}$$

It follows that we can have at most $\frac{0.01\epsilon/k}{1/(48dk^2)} < \epsilon dk/2$ such edges, and the claim follows. ■

Claim 21.3 (a large sum implies high rejection probability): *Suppose* $\sum_{v\in[k]} p_0(v)p_1(v) \geq 0.01\epsilon/k$, *and that Step 2 is executed with vertex* s. *Then, for* $m \geq 25\sqrt{k/\epsilon}$, *with probability at least* $2/3$, *the set* $R_0 \cap R_1$ *is not empty* (and rejection follows).

Proof: Consider the probability space defined by an execution of Step 2 (with start vertex s). For every $i \neq j$ such that $i, j \in [m]$, we define an indicator random variable $\zeta_{i,j}$ representing *the event that the vertex encountered in the ℓ^{th} step of the i^{th} walk equals the vertex encountered in the ℓ^{th} step of the j^{th} walk, and that the i^{th} walk has an even path-parity whereas the j^{th} has an odd path-parity.* (That is, $\zeta_{i,j} = 1$ if the foregoing event holds, and $\zeta_{i,j} = 0$ otherwise.) Recalling the definition of the $p_\sigma(v)$'s, observe that $\mathbf{Pr}[\zeta_{i,j}=1] = \sum_{v\in[k]} p_0(v)p_1(v)$. Hence,

$$\begin{aligned}\sum_{i\neq j} \mathbb{E}[\zeta_{i,j}] &= m(m-1)\cdot \sum_{v\in[k]} p_0(v)p_1(v) \\ &> \frac{600k}{\epsilon}\cdot \sum_{v\in[k]} p_0(v)p_1(v) \\ &\geq 6\end{aligned}$$

where the first inequality is due to the setting of m, and the second inequality is due to the claim's hypothesis. Note that $\mathbf{Pr}[|R_0 \cap R_1| > 0] \geq \mathbf{Pr}[\sum_{i\neq j} \zeta_{i,j} > 0]$, since whenever the event captured by the $\zeta_{i,j}$ holds it is the case that the common endpoint of the i^{th} and j^{th} paths is in $R_0 \cap R_1$.

Intuitively, the sum of the $\zeta_{i,j}$'s should be positive, with high probability, since the expected value of the sum is large enough and the $\zeta_{i,j}$'s are "sufficiently independent" (almost all pairs of $\zeta_{i,j}$'s are independent). The intuition is indeed correct, but proving it is less straightforward than it seems, since the $\zeta_{i,j}$'s are not pairwise independent.[28] Yet, since the sum of the covariances of the dependent $\zeta_{i,j}$'s is quite small, Chebyshev's Inequality is still very useful (cf. [3, Sec. 4.3]). Specifically, letting $\mu \stackrel{\text{def}}{=} \mathbb{E}[\zeta_{i,j}] = \sum_{v\in[k]} p_0(v)p_1(v)$, and $\overline{\zeta}_{i,j} \stackrel{\text{def}}{=} \zeta_{i,j} - \mu$, we get:

$$\begin{aligned}\mathbf{Pr}\left[\sum_{i\neq j} \zeta_{i,j} = 0\right] &< \frac{\mathbb{V}\left[\sum_{i\neq j} \zeta_{i,j}\right]}{(m(m-1)\cdot\mu)^2} \\ &= \frac{1}{m^2(m-1)^2\mu^2}\cdot \sum_{i_1\neq j_1, i_2\neq j_2} \mathbb{E}\left[\overline{\zeta}_{i_1,j_1}\overline{\zeta}_{i_2,j_2}\right]\end{aligned}$$

We partition the terms in the last sum according to the number of distinct elements such that, for $t \in \{2,3,4\}$, we let $(i_1, j_1, i_2, j_2) \in S_t \subseteq [m]^4$ if and only if $|\{i_1, j_1, i_2, j_2\}| = t$ (and $i_1 \neq j_1 \wedge i_2 \neq j_2$). Hence,

$$\mathbf{Pr}\left[\sum_{i\neq j} \zeta_{i,j} = 0\right] < \frac{1}{m^2(m-1)^2\mu^2}\cdot \sum_{t\in\{2,3,4\}} \sum_{(i_1,j_1,i_2,j_2)\in S_t} \mathbb{E}\left[\overline{\zeta}_{i_1,j_1}\overline{\zeta}_{i_2,j_2}\right] \tag{8}$$

Now, note that if $i_1 = j_2$ (resp., $i_2 = j_1$), then $\mathbb{E}[\overline{\zeta}_{i_1,j_1}\overline{\zeta}_{i_2,j_2}] \leq \mathbb{E}[\zeta_{i_1,j_1}\zeta_{i_2,j_2}] = 0$, where the equality is due to the fact that in this case $\zeta_{i_1,j_1} = 1$ and $\zeta_{i_2,j_2} = 1$ make conflicting requirements of the path-parity of walk number $i_1 = j_2$ (resp., $i_2 = j_1$).[29] Hence, rather than summing over the S_t's, we can sum over the following S'_t's defined such that $(i_1, j_1, i_2, j_2) \in S'_t \subseteq S_t$ if and only if $i_1 \neq j_2 \wedge i_2 \neq j_1$.

[28] Indeed, if the $\zeta_{i,j}$'s were pairwise independent, then a straightforward application of Chebyshev's Inequality would do.

[29] Recall that $\zeta_{i_1,j_1} = 1$ requires that the i_1^{th} walk has even path-parity, whereas $\zeta_{i_2,j_2} = 1$ requires that the j_2^{th} walk has odd path-parity, and these requirements conflict when $i_1 = j_2$. Ditto for the i_2^{th} and j_1^{th} walks when $i_2 = j_1$. We also used the inequality $\mathbb{E}[(X - \mathbb{E}[X])\cdot(Y - \mathbb{E}[Y])] \leq \mathbb{E}[XY]$, which holds for any non-negative random variables X and Y, and the equality $\mathbb{E}[XY] = \mathbf{Pr}[X = Y = 1]$, which holds for any 0-1 random variables.

Furthermore, the contribution of each element in $S'_4 = S_4$ to the sum is zero, since the four walks are independent and so $\mathbb{E}[\overline{\zeta}_{i_1,j_1}\overline{\zeta}_{i_2,j_2}] = \mathbb{E}[\overline{\zeta}_{i_1,j_1}] \cdot \mathbb{E}[\overline{\zeta}_{i_2,j_2}] = 0$. Plugging all of this into Eq. (8), we get

$$\begin{aligned}
\mathbf{Pr}\left[\sum_{i\neq j}\zeta_{i,j}=0\right] &< \frac{1}{m^2(m-1)^2\mu^2}\cdot\sum_{t\in\{2,3\}}\sum_{(i_1,j_1,i_2,j_2)\in S'_t}\mathbb{E}\left[\overline{\zeta}_{i_1,j_1}\overline{\zeta}_{i_2,j_2}\right] \\
&= \frac{1}{m^2(m-1)^2\mu^2}\cdot\left(\sum_{i\neq j}\mathbb{E}\left[\overline{\zeta}^2_{i,j}\right]+\sum_{i_1,i_2,i_3:|\{i_1,i_2,i_3\}|=3}\left(\mathbb{E}\left[\overline{\zeta}_{i_1,i_2}\overline{\zeta}_{i_1,i_3}\right]+\mathbb{E}\left[\overline{\zeta}_{i_1,i_2}\overline{\zeta}_{i_3,i_2}\right]\right)\right) \\
&< \frac{m(m-1)\cdot\mu+m(m-1)(m-2)\cdot(\mathbb{E}[\zeta_{1,2}\zeta_{1,3}]+\mathbb{E}[\zeta_{1,2}\zeta_{3,2}])}{m^2(m-1)^2\mu^2} \\
&< \frac{1}{(m-1)^2\mu}+\frac{1}{(m-1)\mu^2}\cdot(\mathbb{E}[\zeta_{1,2}\zeta_{1,3}]+\mathbb{E}[\zeta_{1,2}\zeta_{3,2}])
\end{aligned}$$

where in the second inequality we use $\mathbb{E}[\overline{\zeta}^2_{i,j}] \leq \mathbb{E}[\zeta^2_{i,j}] = \mu$ and $\mathbb{E}[\overline{\zeta}_{i_1,j_1}\overline{\zeta}_{i_2,j_2}] \leq \mathbb{E}[\zeta_{i_1,j_1}\zeta_{i_2,j_2}]$. For the second term, we observe that $\mathbb{E}[\zeta_{1,2}\zeta_{1,3}] = \mathbf{Pr}[\zeta_{1,2}=\zeta_{1,3}=1]$ is upper-bounded by $\mathbf{Pr}[\zeta_{1,2}=1] = \mu$ times the probability that the ℓ^{th} vertex of the third walk appears as the ℓ^{th} vertex of the first path, since $\zeta_{1,3}=1$ mandates the latter event. Using the (upper bound of the) *rapid mixing hypothesis*, we upper-bound the latter probability by $2/k$, and obtain $\mathbb{E}[\zeta_{1,2}\zeta_{2,3}] \leq \mu\cdot 2/k$. (Ditto for $\mathbb{E}[\zeta_{1,2}\zeta_{3,2}]$.) Hence,

$$\begin{aligned}
\mathbf{Pr}[|R_0\cap R_1|=0] &< \frac{1}{(m-1)^2\mu}+\frac{2}{(m-1)\mu^2}\cdot\frac{2\mu}{k} \\
&= \frac{1}{(m-1)^2\mu}+\frac{4}{(m-1)\mu k} \\
&< \frac{1}{3}
\end{aligned}$$

where the last inequality uses $\mu \geq 0.01\epsilon/k$ and $(m-1)^2 \geq 600k/\epsilon$ (along with $m > 2400/\epsilon$). The claim follows. ■

Beyond rapid mixing graphs (an overview). For starters, suppose that the graph consists of several connected components, each having the rapid mixing property. Then, we can apply the foregoing argument to each connected component separately. Note that, already in this case, it is important that we select a start vertex at random, since some of the connected components may be bipartite (e.g., it may be that only an $O(\epsilon)$ fraction of the vertices reside in connected components that are $\Omega(1)$-far from being bipartite). But otherwise, the extension is straightforward. We define a sum of the foregoing type (i.e., $\sum_v p_0(v)p_1(v)$) for each connected components, and argue as in Claims 21.2 and 21.3.

Intuitively, the same strategy should work also if these "strongly connected components" (which each have the rapid mixing property) are actually connected by relatively few edges, however things are less straightforward in this case. For starters, a random walk can exit such component (and enter a different component that is connected to it), and the definition of $p_\sigma(v)$ should be adapted accordingly. More importantly, we cannot assume that the graph has such a structure, but should rather impose an adequate structure on it. Indeed, this is the complicated part of the analysis.

Teaching note: The following three paragraphs provide additional hints regarding the ideas used towards extending the proof from the special case of rapid mixing to the general case. These paragraphs are terse and abstract and may be hard to follow. An illustration of the basic strategy appears in the guidelines of Exercise 10, which addresses a much weaker claim.

The proof in [22] refers to a more general sum of products; that is, $\sum_{u \in U} p_{\mathrm{odd}}(u) p_{\mathrm{even}}(u)$, where $U \subseteq [k]$ is an appropriate set of vertices, and $p_{\mathrm{odd}}(v)$ (respectively, $p_{\mathrm{even}}(v)$) is essentially the probability that an ℓ-step random walk (starting at s) passes through v after more than $\ell/2$ steps and the corresponding path to v has odd (respectively, even) parity. Note that these probabilities refers to the vertices visited in the last $\ell/2$ steps of the walk rather than to the last vertex visited in it, and this change is done in order to account for walks that leave U (and possibly return to it at a laster stage).

Much of the analysis in [22] goes into selecting the appropriate U (and an appropriate starting vertex s), and pasting together many such U's to cover all of $[k]$. Loosely speaking, U and s are selected so that there are few edges from U to the rest of the graph, and $p_{\mathrm{odd}}(u) + p_{\mathrm{even}}(u) \approx 1/\sqrt{k \cdot |U|}$, for every $u \in U$. The selection is based on the "combinatorial treatment of expansion" of Mihail [32]. Specifically, we use the contrapositive of the standard analysis, which asserts that rapid mixing occurs when all cuts are relatively large, to assert that the failure of rapid mixing yields small cuts that partition the graph so that vertices reached with relatively high probability (in a short random walk) are on one side and the rest of the graph is on the other side. The first set corresponds to the aforementioned U, and the cut is relatively small with respect to the size of U. A start vertex s for which the corresponding sum is big is shown to cause Step 2 to reject (when started with this s), whereas a small corresponding sum enables to 2-partition U while having few violating edges among the vertices in each part of U.

The actual argument of [22] proceeds in iterations. In each iteration a vertex s for which Step 2 accepts with high probability is fixed, and an appropriate set of remaining vertices, U, is found. The set U is then 2-partitioned so that there are few violating edges inside U. Since we want to paste all these partitions together, U may not contain vertices treated in previous iterations. This complicates the analysis, since it must refer to the part of G, denoted H, not treated in previous iterations. We consider walks over an (imaginary) Markov Chain representing the H-part of the walks performed by the algorithm on G. Statements about rapid mixing are made with respect to the Markov Chain, and linked to what happens in random walks performed on G. In particular, a subset U of H is determined so that the vertices in U are reached with probability $\approx 1/\sqrt{k \cdot |U|}$ (in the chain) and the cut between U and the rest of H is small. Linking the sum of products defined for the chain with the actual walks performed by the algorithm, we infer that U may be partitioned with few violating edges inside it. Edges to previously treated parts of the graphs are charged to these parts, and edges to the rest of $H \setminus U$ are accounted for by using the fact that this cut is small (relative to the size of U). A simplified version of this argument appears in the guideline for Exercise 10.

4.2 One-sided error tester for `Cycle-freeness`

Recall that, by Theorem 17, a one-sided error testers for `Cycle-freeness` requires $\Omega(\sqrt{k})$ queries. Here, we show that this lower bound can almost be met.

Theorem 22 (one-sided error tester for `Cycle-freeness`, in the bounded-degree graph model): `Cycle-freeness` *has a one-sided error tester of time* (and query) *complexity* $\mathrm{poly}(d/\epsilon) \cdot \widetilde{O}(\sqrt{k})$.

As in the case of the tester for `Bipartiteness`, the asserted tester can be modified so that in case of rejection it outputs a cycle of length $\mathrm{poly}((d \log k)/\epsilon)$. Hence, this one-sided error tester yields an algorithm for finding (relatively short cycles) in graphs that are ϵ-far from being cycle-free. See further discussion following the proof.

Proof: The proof is by a randomized (local) reduction of testing `Cycle-freeness` to testing `Bipartiteness`, where the notion of such a reduction was presented in the lecture on lower bound techniques and will be reviewed (and modified) below. But before doing so, let us provide some intuition.

Given a graph $G = ([k], E)$, we shall map it at random to a graph $G' = (V', E')$ by making, for each edge of G, a random choice on whether to keep this edge in G' or to replace it by a 2-path (with a new auxikiary vertex). Specifically, with probability 1/2, the edge $e = \{u, v\}$ is kept as is, and otherwise it is replaced by the edges $\{u, a_e\}$ and $\{a_e, v\}$, where a_e is an auxiliary vertex that is connected (only) to u and v. Formally, for any 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), and keeping the edge in G_τ otherwise (i.e., if $\tau(e) = 1$). That is, the graph $G_\tau = (V_\tau, E_\tau)$ is defined as follows:

$$\begin{aligned} V_\tau &\stackrel{\text{def}}{=} [k] \cup \{a_e : e \in E \wedge \tau(e) = 2\} \\ E_\tau &\stackrel{\text{def}}{=} \{e : e \in E \wedge \tau(e) = 1\} \cup \{\{u, a_e\}, \{a_e, v\} : e = \{u, v\} \in E \wedge \tau(e) = 2\}. \end{aligned}$$

Hence, G' is obtained by selecting $\tau : E \to \{1, 2\}$ uniformly at random, and letting $G' = G_\tau$.

Suppose that G is cycle-free. Then, for any choice of τ (i.e., with probability 1 over all possible choices of τ), the resulting graph G_τ is also cycle-free, which implies that G_τ is bipartite. On the other hand, if G is not cycle-free, then, *each of its cycles is mapped to an odd-length cycle* (in G_τ) *with probability* 1/2. Hence, with probability at least 1/2, the graph G_τ is not bipartite. However, we need to prove more than that in order to reduce testing `Cycle-freeness` to testing `Bipartitness`. Indeed, we shall show that if G is ϵ-far from `Cycle-freeness`, then, with high probability, the graph G_τ is $\Omega(\epsilon)$-far from `Bipartitness`.

Lemma 22.1 (analysis of the foregoing reduction): *Suppose that G is ϵ-far from* Cycle-freeness. *Then, with positive constant probability over the choice of τ, the graph G_τ is $\Omega(\epsilon/d)$-far from* Bipartitness.

The error probability can be made arbitrary small by invoking the reduction sufficiently many times, and considering a single graph composed of the graphs obtained in the various invocations. (This may reduce the constant hidden in the Ω-notation by a factor related to the constant success probability that is asserted in the Lemma 22.1.)

Proof Sketch: Let $\Delta \geq |E| - (k-1)$ denote the actual number of edges that should be omitted from G in order to obtain a cycle-free graph. We shall show that, with probability $1 - \exp(-\Omega(\Delta))$ over the choice of τ, the number of edges that should be omitted from G_τ in order to obtain a bipartite graph is $\Omega(\Delta)$.

The basic intuition is that the interesting case is when all vertices of G are of degree at least 3, in which case $\Delta > k/2$. This is so because vertices of degree 1 and 2 (in G) do not really matter: Vertices of degree 1 do not participate in any cycle, and their removal from the graph does not change Δ, while it reduces $|E|$ and k (by a similar amount). Vertices of degree 2 are intermediate

vertices on paths or cycles, and these paths or cycles act as a single edge, where in the anaylysis (which is a mental experiment) we allow also multiple edges and self-loops. (See more details at the end of the proof.)

The benefit of focusing on the case of $\Delta > k/2$ is that in this case we can afford to perform a union bound on all possible 2-partitions of the vertex-set of $G = ([k], E)$. Specifically, for each such partition, we show that, with probability at least $1 - 2^{-k-\Omega(\Delta)}$ over the choice of τ, there exist $m \stackrel{\text{def}}{=} \Omega(\Delta)$ edges that are inconsistet with that partition (under τ), where an edge $e = \{u, v\}$ is inconsistent with the partition (V_1, V_2) under τ *if either* $u, v \in V_i$ *and* $\tau(e) = 1$ *or* $(u, v) \in V_1 \times V_2$ *and* $\tau(e) = 2$. Applying a union bound, it follows that, with probability at least $1 - 2^{-\Omega(\Delta)}$, at least m edges should be omitted from G_τ in order to obtain a bipartite graph. Details follow.

We focus on the case that G is connected, where $\Delta = |E| - (k-1)$, leaving the general case to the reader (see Exercise 12). Suppose, indeed, that $\Delta > k/2$, and let $c > 0$ be a sufficiently small constant. Fixing any partition (V_1, V_2) of $[k]$, observe that the probability, over a random choice of τ, that the edges in E' are all consistent with (V_1, V_2) under τ equals $2^{-|E'|}$, since the value of τ on each edge that is consistent with (V_1, V_2) is uniquely determined (i.e., if $\{u, v\}$ is consistent with (V_1, V_2) under τ, then $\tau(\{u, v\}) = 2$ if u and v are in the same V_i, and $\tau(\{u, v\}) = 1$ otherwise). Using a union bound (on the relevant sets E') it follows that the probability over a random choice of τ, that less than $m = c \cdot \Delta$ edges of $G = ([k], E)$ are inconsistent with (V_1, V_2) under τ is at most

$$\binom{|E|}{m} \cdot 2^{-(|E|-m)}, \tag{9}$$

Using $|E| = k + \Delta - 1$ and $m = c \cdot \Delta$, Eq. (9) yields

$$\begin{aligned}
\binom{k+\Delta-1}{c\cdot\Delta} \cdot 2^{c\cdot\Delta-k-\Delta+1} &= 2^{-k+1} \cdot \binom{k+\Delta-1}{c\cdot\Delta} \cdot 2^{-(1-c)\cdot\Delta} \\
&< 2^{-k+1} \cdot \binom{3\Delta}{c\cdot\Delta} \cdot 2^{-(1-c)\cdot\Delta} \\
&\approx 2^{-k+1} \cdot 2^{H_2(c/3)\cdot 3\Delta} \cdot 2^{-(1-c)\cdot\Delta}
\end{aligned}$$

where the inequality is due to $\Delta > k/2$. Hence, any choice of $c > 0$ that satisfies $3H_2(c/3) + c < 1$ will do. (The foregoing argument assumes a sufficiently large k, but otherwise we can just use the fact that with probability at least $1/2$ the graph G_r is not bipartite.)

It is left to justify the focus on graphs G in which all vertices are of degree at least 3. Formally, we show that graphs G that do not satisfy this condition can be transformed into graphs that do satisfy this condition, while preserving Δ as well as (the distribution of) the number of edges that have to be removed from G_τ to make it bipartite. As hinted at the beginning of this proof, vertices of degree 1 are irrelevant and can be removed from the graph G (along with the edges that connects them to the rest of it). As for vertices of degree 2, we contract paths (and cycles) that only contain intermediate vertices of degree 2 to a single edge, while noting that the resulting graph may have parallel edges and self-loops. We note, however, that the foregoing argument is oblivious to this fact (i.e., it applies also to such non-simple graphs). The key observation is that the effect of applying the reduction to a t-path (resp., t-cycle) is identical to applying it to the resulting edge: In both cases, the reduction yields a path (resp., cycle) that has odd-length with probability exactly half. The lemma follows. ■

On the locality of the reduction. Lemma 22.1 asserts that the foregoing reduction preserves distances in the sense that instances that are far from one property are mapped (with high probability) to

instances that are far from the second property. (We also noted that instances that have the first property are mapped to instances that have the second property.) But this does not suffice for a reduction between the corresponding testing problems: Towards that end, we have to show that the reduction preserves the query complexity. Typically, this is done by showing that each query of the tester of the second property can be answered by few queries to the instance of the first problem.

All these conditions are summarized in the definition of randomized reductions that was presented in the lecture on lower bound techniques. Here, we reproduce this definition while adapting it to our current context.

Definition 22.2 (randomized local reductions, specialized): *Let Π_n and Π'_n be sets of functions that represent graph properties in the bounded-degree model as in Definition 1; that is, the function $g : D_n \to R_n$, where $D_n = [n/d] \times [d]$ and $R_n = \{0, 1, ..., n/d\}$, represents an n/d-vertex graph of degree bound d. A distribution of mappings $\mathcal{F}_n$ from the set of functions $\{f : D_n \to R_n\}$ to the set of functions $\{f' : D_{n'} \to R'_{n'}\}$ is called a* randomized q-local (ϵ, ϵ')-reduction of Π_n to $\Pi'_{n'}$ *if for every $f : D_n \to R_n$ the following conditions hold with probability at least $5/6$ when the mapping F_n is selected according to the distribution $\mathcal{F}_n$.*

1. Locality (local computation): *There exist randomized algorithms $Q_n : D_{n'} \to (D_n)^q$ and $V_n : D_{n'} \times R_n^q \to R'_{n'}$, which may depend on F_n, such that for every $e \in D_{n'}$ it holds that*

$$\mathbf{Pr}_{(e_1,...,e_q) \leftarrow Q_n(e)}[V_n(e, f(e_1), ..., f(e_q)) = (F_n(f))(e)] \;\geq\; 2/3. \tag{10}$$

2. Preservation of the properties: *If $f \in \Pi_n$, then $F_n(f) \in \Pi'_{n'}$.*

3. Partial preservation of distance to the properties: *If f is ϵ-far from Π_n, then $F_n(f)$ is ϵ'-far from $\Pi'_{n'}$.*

If Condition 2 holds for all F_n's and Eq. (10) *holds with probability 1, then the reduction is said to have one-sided error.*

Hence, if $f \in \Pi_n$ (resp., if f is ϵ-far from Π_n), then, with probability at least $5/6$, over the choice of F_n, Conditions 1 and 2 both hold (resp., Conditions 1 and 3 both hold).

As hinted above, Lemma 22.1 asserts that the randomized mapping from `Cycle-freeness` to `Bipartitness` satisfies Conditions 2 and 3 of Definition 22.2. Furthermore, the reduction has one-sided error (since Condition 2 holds for all F_n's), and so employing it preserves the one-sided error of the tester for `Bipartiteness`. Unfortunately, this randomized mapping does not seem to satisfy Condition 1. For starters, the number of vertices in the reduced graph, $G_\tau = (V_\tau, E_\tau)$, is not a fixed function of k (but is rather a random variable that varies with τ). In addition, there is no simple mapping between V_τ and $[|V_\tau|]$. This means that, formally speaking, the representation of G_τ does not fit Definition 1.

Nevertheless, we observe that the tester for `Bipartitness` that was presented in Section 4.1 does not make arbitrary queries. It rather performs two types of operations: (1) it selects uniformly a vertex in the graph, and (2) given a vertex name, it selects uniformly one of its neighbours. Hence, it suffices to locally implement both these operations. Firstly, we select uniformly $\tau : \binom{[k]}{2} \to \{1, 2\}$ (and use it rather than $\tau : E \to \{1, 2\}$ used in our description). Note that it is easy to determine the i^{th} neighbor of a vertex in G_τ by making the corresponding query to G (i.e., query its incidence

function g); specifically, if w is the i^{th} neighbor of v in G (i.e., $w = g(v,i)$), then the i^{th} neighbor of $v \in [k]$ in G_τ is w if $\tau(\{v,w\}) = 1$ and $a_{\{v,w\}}$ otherwise, whereas for $e = \{u,w\} \in E$ such that $\tau(e) = 2$ the neighbors of a_e in G_τ are u and w. Now, to select a random neighbor of $v \in V_\tau$ with uniform probability distribution, we just retreive all its neighbors (and select one at random).[30] Selecting at random a vertex in V_τ (with uniform probability distribution) is slightly more complex, and is done as follows (using the "repeated sampling" paradigm):

1. First, we select uniformly at random a name $w \in [k] \cup ([k] \times [d])$ of a potential vertex in V_τ. If $w \in [k]$, then we just output it (and are done).

2. Otherwise (i.e., $w \in ([k] \times [d])$), we let $w = (v,i)$ and query G for the i^{th} neighbor of v (i.e., we query its incidence function g).

   (a) If v has less than i neighbors (i.e., $g(v,i) = 0$), then we stop with no output.

   (b) Otherwise, letting $u = g(v,i)$ be the i^{th} neighbor of v, we check whether $u < v$ and $\tau(\{u,v\}) = 2$. If both conditions are satisfied, we output the vertex $a_{\{u,v\}}$, and otherwise we halt with no output.

Note that each vertex in V_τ is output with probability $1/(k+dk)$: A vertex $w \in [k]$ is output if and only if it was selected in Step 1, whereas a vertex $a_{\{u,v\}}$ is output in Step 2 if and only if $w = (v,i)$ was selected in Step 1 and it holds that $g(v,i) = u < v$ and $\tau(\{u,v\}) = 2$. (In particular, $a_{\{u,v\}}$ is output only if $\tau(\{u,v\}) = 2$, and in that case it is output if and only if (v,i) was chosen in Step 1, where $v > u$ and $g(v,i) = u$.) Indeed, with probability $1 - \frac{|V_\tau|}{k+dk} \leq 1 - \frac{1}{d+1}$, there is no output, but in such a case we just try again. We can stop trying after $(d+1) \cdot \log k$ attempts, which will just add an error probability of q/k to the error probability of the q-query tester that we emulate. ■

Digest. We have presented a *randomized* reduction of of `Cycle-freeness` to `Bipartiteness` that satisfies a *relaxed locality condition*. The relaxation that we used allows the vertex-set of the reduced graph to be arbitrary (rather than equal $[k']$ for some k' that is determined by k), but required an efficient way of sampling this vertex-set (and answering incidence queries with respect to it). This raises a couple of questions.

Open Problem 23 (cleaner local reductions of `Cycle-freeness` to `Bipartiteness`): *In both items, we refer to $q = \text{poly}(d \log k)$ and seek reductions that have one-sided error.*

1. *Does there exist a* non-relaxed (randomized) *q-local reduction of* `Cycle-freeness` *to* `Bipartiteness`*? That is, a reduction satisfying Definition 22.2.*

2. *Does there exist a* (relaxed) deterministic *q-local reduction of* `Cycle-freeness` *to* `Bipartiteness`*?*

In fact, $q = \text{poly}((d/\epsilon) \log k)$ will be interesting too.

[30] Alternatlely (and in fact less wastefully), we can first determine the number of its neighbors, denoted d_v, and then select uniformly $i \in [d_v]$ (and answer with the i^{th} neighbor of v). Yet another alternative is to just use, in the algorithm, the version of a random walk that was used in the analysis (i.e., just select uniformly $i \in [2d]$, use the i^{th} neighbor of v if such exists, and stay in place otherwise).

Perspective: Finding substructures and one-sided error testers. As stated in the beginning of this section, the one-sided error tester for `Cycle-freeness` yields a sublinear time algorithm that finds (relatively small) cycles in a (bounded-degree graph) that is far from being cycle-free. Likewise, the one-sided error tester for `Bipartitness` yields a sublinear time algorithm that finds (relatively small) odd-length cycles in a (bounded-degree graph) that is far from being bipartite (i.e., far from lacking odd-length cycles). The correspondence between one-sided error tester and sublinear time algorithms that find certain structures in the input arises whenever the property can be characterized as the set of objects that lack this type of structure. This fact provides additional motivation for the interest in one-sided error (rather than general) testers, a motivation that goes beyond the natural desire to avoid error probability in the case that the object is perfectly fine (i.e., the object has the property).[31]

In light of the foregoing perspective, we mention a few additional results regarding finding structures in bounded degree graphs.

Theorem 24 (finding of cycles and trees (in the bounded-degree graph model)):

1. *For every $\ell \geq 3$, there exists a $\mathrm{poly}(d^\ell/\epsilon) \cdot \widetilde{O}(\sqrt{k})$-time algorithm that finds simple cycles of length at least ℓ in k-vertex graphs that are ϵ-far from lacking such cycles.*

2. *For every $\ell \geq 3$, there exists an $O(\ell^3/\epsilon)$-time algorithm that finds trees that have at least ℓ leaves in k-vertex graphs that are ϵ-far from lacking such trees.*

Indeed, Part 1 generalizes Theorem 22, which refers to the case $\ell = 3$, whereas Part 2 extends the trivial algorithm that finds edges in a graph that has many edges (and corresponds to the case of $\ell = 2$).

5 Testing by implementing and utilizing partition oracles

Cycle-free graphs may be viewed as graphs that have no K_3-minor, where the graph H is a minor of the graph G if H can be obtained from G by a sequence of edge removal, vertex removal, and edge contraction operations. (Contracting the edge $\{u, v\}$ means that u and v are replaced by a single vertex that is incident to all vertices that were incident to either u or v.) We say that G is H-minor free if H is not a minor of G. Thus, a graph is cycle-free if and only if it is K_3-minor free, where K_k denotes the k-vertex clique.

The notion of minor freeness extends to sets of graphs; that is, for a set of graphs $\mathcal{H}$, the graph G is $\mathcal{H}$-minor free if no element of $\mathcal{H}$ is a minor of G. Recall that a graph G is planar if and only if it is $\{K_5, K_{3,3}\}$-free, where $K_{3,3}$ denotes the biclique having three vertices on each side.

A graph property is minor-closed if it is closed under removal of edges, removal of vertices, and edge contraction. Note that, for every finite set of graphs $\mathcal{H}$, the property of being $\mathcal{H}$-minor free is minor-closed. On the other hand, the celebrated theorem of Robertson and Seymour (see [41]) asserts that any minor-close property equals the set of $\mathcal{H}$-minor free graphs, for some finite set of graphs $\mathcal{H}$. The main result presented in this section is the following.

[31] Note the analogy to the notion of "perfect completeness" in the setting of probabilistic proof systems [14, Chap. 9].

Theorem 25 (testing minor-close graph properties (in the bounded-degree graph model)):[32] *Any minor-closed property can be tested in query* (and time) *complexity that is quasi-polynomial in* $1/\epsilon$ (i.e., $\exp(\mathrm{poly}(\log(1/\epsilon)))$). *Actually, the bound is* $(d/\epsilon)^{O(\log(1/\epsilon))}$.

We mention that this tester has two-sided error, which is unavoidable for any tester of query complexity $o(\sqrt{k})$, except for the case that the forbidden minors are all cycle-free. Before turning to the proof of Theorem 25, we state the begging question of whether the bound in Theorem 25 can be improved to a polynomial.

Open Problem 26 (improving the upper bound of Theorem 25): *Can any minor-closed property be tested in query* (and time) *complexity that is polynomial in* d/ϵ*? What about the special case of* `Planarity`*?*

The proof of Theorem 25 (as well as several related studies in this area) evolves around the local construction and utilization of a *partition oracle*. Loosely speaking, such an oracle provides a partition of the input graph $G = ([k], E)$ into small components with few edges connecting different components. Specifically, for given parameters $\epsilon > 0$ and $t \in \mathbb{N}$, such a partition oracle of a graph $G = ([k], E)$ is a function $P : [k] \to \cup_{i \in [t]} \binom{[k]}{i}$ such that (1) vertex v resides in $P(v)$ (which has size at most t); (2) the $P(v)$'s form a partition of $[k]$; (3) the subgraph of G induced by each $P(v)$ is connected; and (4) the total number of edges among different $P(v)$'s is at most ϵk.

Definition 27 (partition oracles): *We say that* $P : [k] \to 2^{[k]}$ *is an* (ϵ, t)-partition *of the graph* $G = ([k], E)$ *if the following conditions hold.*

1. *For every* $v \in [k]$*, vertex* v *is in the set* $P(v)$*, and* $|P(v)| \leq t$*.*
2. *The sets* $P(v)$*'s are a partition of* $[k]$*; that is, for every* $v, u \in [k]$*, the sets* $P(v)$ *and* $P(u)$ *are either identical or disjoint.*[33]
3. *For every* $v \in [k]$*, the subgraph of* G *induced by* $P(v)$ *is connected.*
4. *The number of edges among the different* $P(v)$*'s is at most* ϵk*; that is,* $|\{\{u, w\} \in E : P(u) \neq P(w)\}| \leq \epsilon k$*.*

Note that Conditions 1–3 are quite local (i.e., they refer to individual $P(v)$'s or to pairs of $P(v)$'s), whereas Condition 4 is global. As shown next, if we are given access to a partition oracle P for a graph G, then we can test whether G has a predetermined minor-close property. Of course, in the standard model, we are only given oracle access to the (incidence representation of the) graph G; so the next item on the agenda will be to implement a partition oracle when given oracle access only to G. But let us first show the testing consequence.

Theorem 28 (testing minor-close graph properties by using a partition oracle): *Let* Π *be a minor-closed property and supposed that we are given oracle access to a graph* $G = ([k], E)$*, represented by its incidence function* $g : [k] \times [d] \to \{0, 1, ..., k\}$*, as well as to an* $(d\epsilon/4, t)$*-partition oracle* $P : [k] \to 2^{[k]}$ *of* G*. Then, using* $O(td/\epsilon)$ *queries to* g *and to* P*, we can distinguish the case that*

[32] This result is due to [30], improving over [26], which improved upon [6]: The query complexity obtained in [6] is triple-exponential in $1/\epsilon$, and in [26] it is exponential in $\mathrm{poly}(1/\epsilon)$.

[33] The fact that $\cup_{v \in [k]} P(v) = [k]$ follows from Condition 1.

$G \in \Pi$ from the case that G is ϵ-far from Π. Actually, we accept each graph in Π with probability at least 0.9, and reject with probability at least 0.9 any graph that is ϵ-far from Π. Furthermore, a graph that is ϵ-far from Π is rejected with probability at least 0.9 even if $P : [k] \to 2^{[k]}$ only satisfies Conditions 1–3 of Definition 27.

The furthermore clause is important because our implementation of the partition oracle is guaranteed to satisfy Condition 4 (with high probability) only when the input graph is in Π. Hence, it is important that the foregoing procedure rejects graphs that are far from Π also when the partition oracle does not satisfy Condition 4. We mention that our implementation of the partition oracle always satisfies Conditions 1–3.

Proof: Let $G' = ([k], E')$ be the graph obtained from $G = ([k], E)$ by omitting all edges that have endpoints in different $P(i)$'s; that is, $E' \stackrel{\text{def}}{=} \{\{u, v\} \in E : P(u) = P(v)\}$. Since Π is closed under omission of edges, it follows that $G \in \Pi$ implies $G' \in \Pi$. On the other hand, if G is ϵ-far from Π, then either $|E \setminus E'| \geq \epsilon dk/4$ (i.e., G' is $\epsilon/2$-far from G) or G' is $\epsilon/2$-far from Π. Hence, it suffices to estimate the size of $E \setminus E'$ and to test whether G' is in Π.

The key observation is that when given oracle access to P and G, it is easy to emulate oracle access to G'. Specifically, letting $g'(v, i) = g(v, i)$ if $P(v) = P(g(v, i))$ and $g'(v, i) = 0$ otherwise (where $P(0) = \emptyset$), we obtain an "unaligned" incidence function of G' (see variants at the beginning of Section 1). Hence, the neighbours of v in G' can be found by making at most d queries to g and $d + 1$ queries to P.

The next observation is that testing whether G' has property Π reduces to checking whether a random connected component of G' has this property. Specifically, selecting $O(1/\epsilon)$ random vertices, and exploring the connected component in which they reside, will do.[34] The cost of each exploration amounts to td queries, and so the query complexity is as claimed.

Lastly, we turn to the task of estimating the size of $E \setminus E'$; that is, estimating the probability that $\{v, g(v, i)\} \in E \setminus E'$ when (v, i) is uniformly distributed in $[k] \times [d]$. Selecting $O(1/\epsilon)$ random pairs $(v, i) \in [k] \times [d]$, allows to distinguish the case that $|E \setminus E'| \leq \epsilon dk/4$ from the case that $|E \setminus E'| > 3\epsilon dk/8$. Hence, our actual algorithm proceeds as follows.

1. Using $O(1/\epsilon)$ random pairs $(v, i) \in [k] \times [d]$, the algorithm estimates $|E \setminus E'|$ up to an additive deviation of $\epsilon dk/16$. If the estimate is greater than $5\epsilon dk/16$, then it rejects. Otherwise, it continute to the next step.

2. The algorithm invokes the foregoing tester for Π on input G' using proximity parameter $\epsilon/4$. The queries made by this tester are answered by emulating G' as outlined in the penultimate paragraph (i.e., by using oracle queries to G and P).

Now, if $G \in \Pi$ and P is an $(\epsilon d/4, t)$-partition oracle of it, then with high probability the algorithm continues to Step 2, and in that case it always accepts. On the other hand, if G is ϵ-far from Π, then there are two cases to consider. The first case is that the partition defined by P yields a graph $G' = ([k], E')$ such that $|E \setminus E'| > 3\epsilon dk/8$. In this case, with high probability, Step 1 rejects. The second case is that $|E \setminus E'| \leq 3\epsilon dk/8$ (i.e., G' is $(3\epsilon/4)$-close to G), which implies that G' is $\epsilon/4$-far from Π. In this case, with high probability, Step 2 rejects. We stress that the last assertion does not refer to Condition 4 (and it holds also if P does not satisfy this condition). The claim follows. ■

[34] If less than $\epsilon k/4$ vertices reside in connected components that are not in Π, then the graph is $\epsilon/2$-close to Π (since a graph can be placed in Π by omitting all its edges).

Implementing a partition oracle. In light of Theorem 28, we now focus on the task of implementing (or rather emulating) partition oracles. Since the implementations that we use are randomized, it is crucial that the same randomness (denoted ω) is used in all invocations of the machine emulating the oracle. In other words, each choice of internal coin tosses for this machine yields a function $f : [k] \to 2^{[k]}$, and, with high probability (over these choices), this function is a good partition oracle (i.e., it satisfies Definition 27). The latter condition (i.e., yielding a good partition oracle) is required to hold only if the input graph has a predetermined property Π (which in our application is the property being tested). In contrast, we require that f always satisfies Conditions 1–3 of Definition 27 (even if the graph does not have bthe property Π).

Definition 29 (implementing a partition oracle): *We say that the oracle machine M* emulates an (ϵ, t)-partition oracle for graphs having property Π *if the following two conditions hold.*

1. *For any possible outcome ω of M's internal coin tosses, when given oracle access to any bounded-degree graph $G = ([k], E)$, the answers provided by M to all possible inputs $v \in [k]$ correspond to a function $P(v) \stackrel{\text{def}}{=} M^G(k, \omega; v)$ that satisfies Conditions 1–3 of Definition 27.*
2. *For any graph $G = ([k], E) \in \Pi$, with probability at least 0.9 over all possible choices of ω, the function $P(v) \stackrel{\text{def}}{=} M^G(k, \omega; v)$ is an (ϵ, t)-partition; that is, it also satisfies Condition 4 of Definition 27.*

Typically, t is a function of ϵ. Actually, any graph G that satisfies a minor-close property has $(\epsilon, O(d/\epsilon)^2)$-partition (for every $\epsilon > 0$);[35] the problem is finding such partitions "locally" or rather implementing corresponding partition oracles. A crucial aspect of such implementations is the number of queries that they make to their own oracle (i.e., the number of queries that M makes to G). In particular, the overhead created by utilizing such an implementation in Theorem 28 is linear in the query (resp., time) complexity of the implementation, which (like t) will be a function of ϵ only (i.e., independent of the size of the graph). Typically, the query complexity of the implementation will be larger than the parameter t, and in these cases it will dominate the complexity of the algorithm that is provided by Theorem 28.

We shall present two implementation of a partition oracle for any minor-free property Π. The first (and simpler) implementation has query complexity that is exponential in $\text{poly}(1/\epsilon)$, whereas the second implementation (which builds on the first) has quasi-polynomial (in $1/\epsilon$) complexity.

5.1 The simpler implementation

We first present the algorithm as a linear-time algorithm that gets a graph as input and generates an (ϵ, t)-partition of it, but the reader may notice that all operations are relatively local. This means that it will be relatively easy to convert this algorithm into an oracle machine that on input $v \in [k]$ makes relatively few queries to $G = ([k], E)$ and returns $P(v)$, where P is an (ϵ, t)-partition of G. In the following description, we shall assume that the graph G is H-minor free, for some fixed graph H. (This assumption will be removed when using Theorem 28.)

The algorithm proceeds in iterations, starting with the trivial partition P_0 in which each vertex is in a part of its own (i.e., $P_0(v) = \{v\}$ for every $v \in [k]$), and "coarsening" the partition in each iteration (i.e., in each iteration, each set of the new partition is a union of sets in the prior

[35] See Alon *et al.* [2], as detailed in [30, Cor. 2].

partition). Note that P_0 satisfies Conditions 1–3 of Definition 27, but violates Condition 4 (unless G is very sparse). Our goal, in each iteration, is to reduce the number of edges between different parts of the current partition, while preserving Conditions 1–3, where the non-trivial issue is preserving Condition 3 (i.e., the subgraph of G induced by $P(v)$ is connected).

The natural way of preserving Condition 3 is using *edge contractions*, which means replacing two adjacent vertices u and v by a new vertex, denoted $a_{u,v}$, and connecting the edges incident at u and v to $a_{u,v}$ (while omitting the edge $\{u,v\}$). Note that we should do so without violating Condition 1 (i.e., each $P(v)$ has size at most t). (Indeed, we shall keep track both of the origoinal graph and of the currently contracted graph, where vertices of the contracted graph correspond to sets of the current partition of the original graph.)

Of course, a key question is which edges to contract (in each iteration). This is a non-trivial question because there is a tension between the number of inter-parts edges that are removed by contraction and the size of the parts (which also effects the locality of the procedure). Note that contracting an edge between u and v removes all the parallel edges between these two vertices, where parallel edges are created by the contraction process (e.g., if both u and v are connected to w, then the contraction of the edge $\{u,v\}$ will form two parallel edges between the resulting vertex and w). Hence, it is a good idea to contract an edge that has many parallel edges (which can be represented as a single edge of corresponding weight). On the other hand, we should avoid contracting a set of edges that span a large subgraph. Looking ahead to a local implementation, it is natural that each vertex will contract an edge incident at it that has the largest number of parallel edges, but we should avoid a long path of contractions (by some "symmetry breaking" mechanism).[36]

It will be instructive to consider both the current partition (to be denoted P_i) of the original graph G and the currently contracted graph (to be denoted G_i). Starting with $G_0 \equiv G$, in the i^{th} iteration we shall contract some edges of G_{i-1} obtaining a graph G_i and a corresponding partition P_i such that each set in P_i corresponds to a vertex in G_i (i.e., the vertex set of G_i equals the set $\{P_i(v) : v \in [k]\}$).[37] Actually, we shall represent parallel edges by a single edge of corresponding weight (which represents the number of pareallel edges). Hence, the graph G_i will have weighted edges, whereas all edges in G_0 have weight 1. The weight of edges in G_i will equal the number of edges in G that connect the corresponding parts (i.e., the weight of $\{U,V\}$ in G_i equals the number of edges in G that connect vertices in U and vertices in V). Finally, we get to the contraction rule itself: *Each vertex in G_{i-1} selects uniformly at random a value in $\{\mathtt{L},\mathtt{C}\}$, and the edge $\{U,V\}$ is contracted if and only if vertex U selected $\mathtt{L}$, vertex V selected $\mathtt{C}$, and $\{U,V\}$ is the heaviest edge incident at U, where ties are broken arbitrarily.*

That is, for every vertex U in G_{i-1}, which corresponds to a set in P_{i-1} (which is a set of vertices in G), we denote by $h_i(U)$ the neighbor of U in G_{i-1} such that the edge $\{U,h_i(U)\}$ is heaviest among all the edges incident at U (in G_{i-1}). (In other words, the number of edges in G between U and $h_i(U)$ is the largest among the number of edges (in G) between $U = P_{i-1}(u)$ and any other $P_{i-1}(v)$.) Letting $r_i(U)$ denote the random choice of U, *we contract the edge $\{U,h_i(U)\}$ if and only if $r_i(U) = \mathtt{L}$ and $r_i(h_i(U)) = \mathtt{C}$* (see Figure 1, which depicts edges as directed from U

[36] The term "symmetry breaking" is used because we may have a long path of vertices such that the local view of each of them is identical (when ignoring vertex labels). In what follows, these potential symmetries are broken by assigning random binary values to the vertices and using a non-symmetric contraction rule that refers to these random values.

[37] Thus, G_0 has the vertex set $\{\{v\} : v \in [k]\} \equiv [k]$, and its edges are pairs of singletons of the form $\{\{u\},\{v\}\}$ such that $\{u,v\}$ is an edge of G.

to $h_i(U)$).

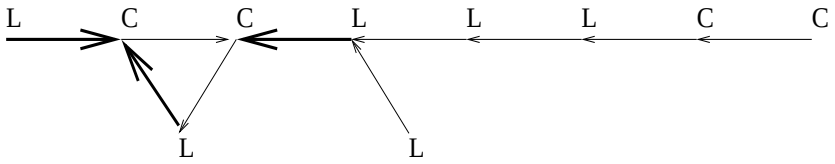

Figure 1: The directed graph of heaviest edges and a random choice of values for the vertices. The edges are directed from each vertiex to its heaviest neighbor, and the edges that will be contracted are shown by wider arrows.

Note that the set of heaviest edges incident at the various vertices defines a directed graph in which there is a single edge directed from each vertex of G_{i-1} to its heaviest neighbor. Hence, each vertex in this directed graph has out-degree 1. Among these directed edges, only edges directed from a vertex that chose `L` (for leaf) to a vertex that chose `C` (for center) are contracted. Thus, the set of contracted edges correspond to a collection of stars in this directed graph, where the center of each star is a vertex that chose `C` and the star's other vertices chose `L`. (Hence, the random values were used in order to "break" the directed graph, which may have directed paths of unbounded length, into components of very small diameter.)

The vertices of G_i are the sets obtained by the contraction of all the stars (or rather all the star edges) as well as all vertices that did not participate in any contraction. That is, if $U = P_{i-1}(u) \neq P_{i-1}(v) = h_i(U)$ and $r_i(U) = \texttt{L} \neq r_i(h_i(U)) = \texttt{C}$, then $P_i(u) \supseteq U \cup h_i(U) = P_{i-1}(u) \cup P_{i-1}(v)$. On the other hand, if $U = P_{i-1}(u) \neq P_{i-1}(v) = V$ do not satisfy the above condition (i.e., either $V \neq h_i(U)$ or $r_i(U) \neq \texttt{L}$ or $r_i(V) = \texttt{C}$), then U and V will not be merged in G_i (i.e., no set in P_i contains both U and V). In general, $P_i(u) \supseteq P_{i-1}(u)$ is the union of all sets $P_{i-1}(v)$ such that the edge $\{P_{i-1}(u), P_{i-1}(v)\}$ was contracted in the i^{th} iteration. Hence, the weight of each edge $\{P_i(u), P_i(v)\}$ of G_i is the sum of the weight of the corresponding edges in G_{i-1} (i.e., the weight of the edge between the vertices that were contracted into $P_i(v)$ and $P_i(u)$, resp.), which equals the number of edges in G between $P_i(v)$ and $P_i(u)$. Note that the weight of contracted edges disappears, since they represent edges of G with both endpoints in the same part of P_i. Hence, we are interested in the rate at which the total weight of the edges in the graphs G_i's decreases during the iterations.

Lemma 30 (the effect of an iteration): *Let $G = ([k], E)$ be a graph of maximum degree d, and consider the foregoing iterative process. Then, for every $i \geq 1$ the following holds.*

1. *For every $v \in [k]$, let $\Gamma_{i-1}(v)$ denote the set of all vertices $u \in [k]$ such that some vertex of $P_{i-1}(u)$ neighbors some vertex of $P_{i-1}(v)$ in G; that is, $\Gamma_{i-1}(v)$ consists of all vertices of G that reside in some vertex $U \in 2^{[k]}$ of G_{i-1} that neighbors vertex $P_{i-1}(v)$ in G_{i-1}. Then, $|P_i(v)| \leq |P_{i-1}(v)| + \sum_{u \in \Gamma_{i-1}(v)} |P_{i-1}(u)|$. Hence, $\max_{v \in G_i}\{|P_i(v)|\} \leq (d+1) \cdot \max_{w \in G_{i-1}}\{|P_{i-1}(w)|^2\}$.*
2. *Supposed that for some fixed graph H, the graph G is H-minor free. Then, with constant probability, the total weight of the edges in G_i is a constant factor smaller than the total weight of the edges in G_{i-1}. This holds even if the values of the r_i's are selected in a pairwise independent manner.*

We stress that Item 1 holds for any graph $G = ([k], E)$ of maximum degree d.

Proof Sketch: The proof of the main part of Item 1 is straightforward, and its second part follows since $|\Gamma_{i-1}(v)| \leq |P_{i-1}(v)| \cdot d$. Item 2 relies on the hypothesis that the graph G is H-minor free, for some fixed graph H, and the unspecified constants depend on H. Specifically, we shall rely on the fact that, for every H, *there exists a constant d_H such that every H-minor free graph has a vertex of degree at most d_H* (see [30] and the references therein).[38] Using this fact, it can be shown (see Exercise 13) that the graph of heaviest edges in G_{i-1} has total weight that is at least a $1/d_H$ fraction of the total weight of the edges of G_{i-1}.

Focusing on the set of heaviest edges, note that if the r_i's are pairwise independent and uniformly distributed in $\{\mathtt{L}, \mathtt{C}\}$, then each heavy edge is contracted with probability 1/4. Hence, the expected total weight of the edges contracted in iteration i is at least a $1/4d_H$ fraction of the total weight of the edges in G_{i-1}, and the claim follows (by Markov inequality). ■

Local implementation. We next show how to emulate oracle access to P_i, when given oracle access to P_{i-1} and to G as well as to r_i. (Note that we do not emulate oracle access to G_i, nor do we use oracle access to G_{i-1}, although this could be done too; the graphs G_i's are only used in the analysis.)

Algorithm 31 (emulating P_i based on P_{i-1}): *Let $\Gamma(v) = \{u : \{u, v\} \in E\}$ denote the set of neighbors of v in G. On input $v \in [k]$, we find $P_i(v)$ as follows.*

1. *Using an oracle call to P_{i-1}, we obtain $V \leftarrow P_{i-1}(v)$.*
2. *Using $d \cdot |V|$ oracle calls to G, we obtain $U \leftarrow \cup_{v \in V} \Gamma(v) \setminus V$. Along the way, we also obtain all edges between V and U.*
3. *Using $|U| \leq d|V|$ oracle calls to P_{i-1}, we obtain the collection of sets $\{P_{i-1}(u) : u \in U\}$, which is part of the partition P_{i-1}. We denote these sets by $U_1, ..., U_m$, where $m \leq |U|$.*

   Using the information gathered in Step 2, we compute, for each $j \in [m]$, the weight the edge $\{V, U_j\}$, which is an edge of G_{i-1}; indeed, this weight equals $\sum_{v \in V} |\Gamma(v) \cap U_j|$. This determines $h_i(V)$.
4. *Using $d \cdot |U|$ oracle calls to G, we obtain $W \leftarrow \cup_{u \in U} \Gamma(u) \setminus (V \cap U)$. Along the way, we also obtain all edges between U and W.*
5. *Using $|W| \leq d|U|$ oracle calls to P_{i-1}, we obtain $\{W_1, ..., W_{m'}\} \leftarrow \{P_{i-1}(w) : w \in W\}$.*

   Using the information gathered in Step 4, we compute, for each $j \in [m]$ and $j' \in [m']$, the weight the edge $\{U_j, W_{j'}\}$, which is an edge of G_{i-1}; indeed, this weight equals $\sum_{u \in U_j} |\Gamma(u) \cap W_{j'}|$. This determines $h_i(U_j)$ for all $j \in [m]$.
6. *Using $m + 1$ oracle queries to r_i, we obtain $r_i(V)$ and $r_i(U_1), ..., r_i(U_m)$. This, together with $h_i(V)$ and $h_i(U_1), ..., h_i(U_m)$, determines $P_i(v)$; that is, $P_i(v) = V \cup \bigcup_{j \in J} U_j$, where $j \in J$ if either $h_i(V) = U_j$ and $r_i(V) = \mathtt{L} \neq r_i(U_j)$ or $h_i(U_j) = V$ and $r_i(U_j) = \mathtt{L} \neq r_i(V)$.*

[38] Specifically, by [30, Fact 1] the edges of each such graph can be partitioned into d_H forests. Noting that each forest has average degree smaller than 1, the claim follows.

Hence, Algorithm 31 makes $O(d|V| + d|U| + |W|) = O(d^2 \cdot |P_{i-1}(v)|)$ oracle calls to P_{i-1} and G. The orcale calls to $r_i : 2^{[k]} \to \{\mathtt{L}, \mathtt{C}\}$ can be implemented by orcale calls to a random function $s_i : [k] \to \{\mathtt{L}, \mathtt{C}\}$ that assigns pairwise independent values to elements of $[k]$, which in turn can be implemented using $2\log_2 k$ random bits.[39] (The latter comment matters only for bounding the time complexity of Algorithm 31.)

We now consider what happens when Algorithm 31 is iterated $\ell \stackrel{\text{def}}{=} O(\log(d/\epsilon))$ times, where in the i^{th} itertation we use it to emulate P_i when using oracle access to P_{i-1} (as well as to G and r_i). Using Item 1 of Lemma 30, it follows that queries to P_i can be implemented at the cost of making $\text{poly}(d)^{2^i}$ queries to G, since the size of the sets in P_j is $\text{poly}(d)^{2^j}$ (and so the size of the recursion tree is $\prod_{j=1}^{i-1} \text{poly}(d)^{2^j} = \text{poly}(d)^{2^i}$). On the other hand, by Item 2 of Lemma 30, with very high probability, after ℓ iterations, the resulting graph has less than $(1 - \Omega(1))^{\Omega(\ell)} \cdot dk = \epsilon \cdot k$ edges, provided that G is H-minor free.[40] This means that, in this case, we can use P_ℓ as the desired partition, since in this case, with high probability, P_ℓ is an (ϵ, t)-partition for $t = \text{poly}(d)^{2^\ell} = \text{poly}(d)^{\text{poly}(d/\epsilon)}$. Recall that for any graph G of maximum degree d, the number of queries requires to emulate P_ℓ is of the same magnitude (as t).[41]

Reducing the size of sets. The size of the partitions generated by the foregoing iterations can be reduced considerablly, while essentially maintaining the query complexity. This can be done by emplying a partitioning algorithm to each set in the aforementioned partition.

Theorem 32 (finding good partitions in polynomial time):[42] *For every fixed graph H, there exists a polynomial-time algorithm that, for every $\epsilon > 0$, finds a $(\epsilon, O(d/\epsilon)^2)$-partition in any given H-minor free graph.*

Hence, when given an (ϵ, t)-partition oracle $P : [k] \to 2^{[k]}$ of an H-minor free graph $G = ([k], E)$, we can emulate an $O(2\epsilon, O(d/\epsilon)^2)$-partition oracle as follows. On input $v \in [k]$, we first retreive the set $P(v)$, next we retreive the subgraph of G induced by $P(v)$, and finally we invoke the partitioning algorithm of Theorem 32 on this subgraph (and answer with the set containing v).

5.2 The better implementation

Invoking the algorithm of Theorem 32 *on the final partition* generated by ($\ell = O(\log(d/\epsilon))$ iterations of) Algorithm 31 does reduce the size of the final sets in the partition, but it does not (and cannot) improve the complexity of generating the partition. The key to improving the said complexity is invoking the algorithm of Theorem 32 *after each iteration of Algorithm 31.*

This means that, in each iteration, we first decrease the number of edges between the sets of the current partition by a constant factor (by employing Algorithm 31), and then increase it by an additive term of $\epsilon' k$ (for an adequate constant $\epsilon' > 0$, by employing the algorithm of Theorem 32).

[39] Specifically, let $r_i(X)$ equal $s_i(x)$ such that $x \in [k]$ is the smallest element in $X \subseteq [k]$. Hence, *if the values that s_i assigns to elements of $[k]$ are pairwise independent and uniformly distributed in $\{\mathtt{L}, \mathtt{C}\}$, then so are the values assigned by r_i to sets in a fixed partition of $[k]$.* Next, let $s_i(x)$ be the least significant bit of $s' + xs''$, where s' and s'' are uniformly distributed in $\mathbb{Z}_{2^\ell}$ and $\ell = \lceil \log_2 k \rceil$. Indeed, here we view $x \in [k]$ as an element of $\mathbb{Z}_{2^\ell}$.

[40] Indeed, $(1 - \Omega(1))^{\Omega(\ell)} = \exp(-\Omega(\ell)) < \epsilon/d$.

[41] Recall that, when seeking an ϵ-tester, Theorem 28 is invoked with a purported $(\epsilon d/4, t)$-partition oracle. Hence, the resulting tester has query complexity $\text{poly}(d)^{\text{poly}(1/\epsilon)}$.

[42] This result is based on a separator theorem of Alon *et al.* [2]; see [30, Cor. 2] for details.

(This is slightly inaccurate since the decrease only occurs with constant probability, but this is good enough.) So we lose a little in terms of the progress made in each iteration, but we gain in maintaining the sets relatively small (i.e., we enter each iteration with a partition having sets of size $\mathrm{poly}(d/\epsilon)$). In particular, while in Section 5.1 the size of the sets in the partition was squared in each iteration, here these sets remain smaller than some fixed quantity (i.e., $\mathrm{poly}(d/\epsilon)$). It follows that the query complexity of implementing the final partition is only $\mathrm{poly}(d/\epsilon)^{O(\log(d/\epsilon))}$. Details follows.

Algorithm 33 (emulating P_i based on P_{i-1}, revised): *On input $v \in [k]$, we find $P_i(v)$ as follows.*

1. *Invoking Algorithm 31, denoted A, we obtain the part in which v reside, which we denote $P'_i(v)$; that is, $P'_i(v) \leftarrow A^{P_{i-1},G,r_i}(v)$. Recall that invoking Algorithm 31 requires providing it with oracle access to P_{i-1}, G and r_i.*

2. *Using oracle access to G construct the subgraph induced by $P'(i)$.[43] Invoking the algorithm of Theorem 32, with an error parameter $\epsilon' = \Theta(\epsilon)$, we obtain an $(\epsilon', O(d/\epsilon')^2)$-partition of this induced subgraph, and let $P_i(v)$ be the part containing v. Recall that this algorithm is invoked on a graph that we hold in hand, so no queries are needed here.*

Hence, Algorithm 33 makes $O(d^2 \cdot |P_{i-1}(v)|)$ oracle calls to P_{i-1} and G, and the oracle calls to r_i are implemented as in Section 5.1. The point is that $|P_i(v)| = O(d/\epsilon)^2$ for all $i \in [\ell]$ (and $v \in [k]$), and so the complexity of emulating the final partition is merely $(O(d^2) \cdot O(d/\epsilon)^2)^\ell$, since this is an upper bound on the size of the recursion tree.

It is left to analyze the quality of the final partition, when assuming that the input graph G is H-minor free. Let us denote by Z_i the number of edges in G_i, which is the graph that corresponds to the partition P_i; that is, Z_i is a random variable that depends on $r_1, ..., r_i$. Letting Z'_i denote the number of edges in the graph that corresponds to the partition P'_i, note that the proof of Lemma 30 actually establishes that $\mathbb{E}[Z'_i | Z_{i-1} = z] \leq c_H \cdot z$, where $c_H = (1 - (1/4d_H))$ is a constant that depends on H. Hence, $\mathbb{E}[Z_i] \leq c_H \cdot \mathbb{E}[Z_{i-1}] + \epsilon' k$, which implies $\mathbb{E}[Z_\ell] < c_H^\ell \cdot dk + (1 - c_H)^{-1} \cdot \epsilon' k$.[44] Letting $\ell = O(\log(d/\epsilon))$ such that $c_H^\ell < \epsilon/20d$, and $\epsilon' = (1 - c_H) \cdot \epsilon/20$, we have $\mathbb{E}[Z_\ell] < \epsilon k/10$. It follows that $\mathbf{Pr}[Z_\ell \geq \epsilon k] < 1/10$. Hence, we obtain:

Theorem 34 (implementing a partition oracle for a H-minor free graph): *For every fixed H, there exists an efficient mapping from $\epsilon > 0$ to M_ϵ such that M_ϵ is an $\exp(O(\log(d/\epsilon))^2)$-time oracle machine that emulates an $(\epsilon, O(d/\epsilon)^2)$-partition oracle for H-minor free graphs of maximum degree d.*

Recall that the notion of emulating a partition oracle for a property (as stated in Definition 29) mandates that, given oracle access to any graph of maximum degree d, (1) machine M_ϵ always implements a function that satisfies Conditions 1–3 of Definition 27, and that (2) when the graph has the property then, with probability at least 0.9, machine M_ϵ emulates an $(\epsilon, O(d/\epsilon)^2)$-partition oracle for it. Combining Theorems 28 and 34, we finally establish Theorem 25.

[43] Actually, this subgraph is implicit in the information gathered by Algorithm 31.

[44] Here we use $\sum_{i=0}^{\ell-1} c_H^i < \sum_{i \geq 0} c_H^i = 1/(1 - c_H)$.

Proof of Theorem 25. Invoking Theorem 34, while setting the approximation parameter to $\epsilon d/4$, we obtain an implementon of a $(d\epsilon/4, O(1/\epsilon^2))$-partition oracle of query (and time) complexity $\exp(O(\log(1/\epsilon))^2)$. Invoking Theorem 28, this yields an ϵ-tester of query complexity $\text{poly}(1/\epsilon)^{O(\log(1/\epsilon)}$. d for any H-minor freeness property. (Note that H-minor feree graphs are accepted with probability at least 0.9^2, whereas graphs that are ϵ-far from being H-minor free are rejected with probability at least 0.9.)[45] Using the fact that these properties are monotone, the same holds for the intersection of several such properties (see general result in the first lecture).

6 A Taxonomy of the known results

We first mention that, also in the current model, graph properties of arbitrary query complexity are known: Specifically, in this model, *graph properties* (in $\mathcal{NP}$) *may have query complexity ranging from* $O(1/\epsilon)$ *to* $\Omega(k)$, and furthermore such properties are monotone and natural (cf. [19], which builds on [7]). In particular, testing 3-Colorability requires $\Omega(k)$ queries, whereas testing 2-Colorability (i.e., Bipartiteness) requires $\Omega(\sqrt{k})$ queries [21] and can be done using $\widetilde{O}(\sqrt{k})\cdot\text{poly}(1/\epsilon)$ queries [22]. We also mention that many natural properties are testable in query complexity that only depends on the proximity parameter (i.e., ϵ). A partial list includes t-`edge connectivity`, for every fixed t, and `Planarity`. Details follow.

6.1 Testability in $q(\epsilon)$ queries, for any function q

Recall that, with the exception of properties that only depend on the degree distribution, adaptive testers are essential for obtaining query complexity that only depends on ϵ (see Theorem 2, which is due to [38]). Still, as observed in [24], at the cost of an exponentially blow-up in the query complexity, we may assume that the tester's adaptivity is confined to performing searches of *predetermined depth* from several randomly selected vertices. However, the best testing results are typically obtained by testers that perform "more adaptive" searches such as performing searches till a *predetermined number of vertices* is visited. In all these cases, the predetermimed number is a function of the proximity parameter, denoted ϵ, and the degree bound, denoted d.

Testability in $\widetilde{O}(1/\epsilon)$ queries. As shown in Section 2.3, `Graph Connectivity` can be tested in $\widetilde{O}(1/\epsilon)$ time. Essentially, the tester starts a search (e.g., a BFS) from a few randomly selected vertices, but each such search is terminated after a predetermined number of vertices is encountered (rather than after visiting all vertices that are at a predetermined distance from the start vertex). Furthermore, as per Levin's economical work investment strategy (see previous lecture), for every $i \in [\log(1/\epsilon)]$, we select $O(2^i \log(1/\epsilon))$ random start vertices, and conduct searches from each of them, while suspending each search once $O(2^{-i}/d\epsilon)$ vertices are encountered (which guarantees that each of these searches has complexity $O(2^{-i}/\epsilon)$). This tester rejects if and only if it detects a small connected component, and thus it has one-sided error.

[45] The error bound in the first case accounts both for the case that M_ϵ fails to satisfy Condition 4 and for the case that the invoked tester errs. Hence, it is important that Theorem 28 asserts error probability that is strictly smaller than 1/3 and ditto regarding the implementation error in Theorem 34. In the second case (i.e., graphs that are far from the property), we only suffer the error of the invoked tester.

Testability in $O(F(d)/\epsilon)$ queries. The testers of `degree regularity` and `Eulerian` have $O(1/\epsilon)$ query complexity when provided with a degree oracle (see Section 2.2), which can be implemented at the cost of $\log d$ incidence queries. Hence, when using only incidence queries, the complexity grows moderately as a function of d. In contrast, the dependence on d is much larger in the testers of subgraph freeness: Specifically, these testers have query complexity $O(F(d)/\epsilon)$, where F is a polynomial of degree that is linearly related to the radius of the subgraph that determines the property (see Section 2.1).

Testability in $\mathrm{poly}(1/\epsilon)$ queries. As mentioned in Section 2.4, for every fixed $t > 1$, the property t-`edge connectivity` can be tested in $\widetilde{O}(t^3/\epsilon^c)$ time, where $c = \min(t-1, 3)$. For t-`vertex connectivity` the known upper bound is $\widetilde{O}(t/d\epsilon)^t$; see [43].

`Cycle-freeness` can be tested in $O(\epsilon^{-3})$ time, by a tester having two-sided error probability (see Section 2.5). Essentially, the tester compares the number of edges to the number of connected components, while exploring any small connected components that it happens to visit. The two-sided error probability is unavoidable for any `Cycle-freeness` tester that has query complexity $o(\sqrt{k})$ (see Section 3.2).[46]

Testability in more than $\mathrm{poly}(d/\epsilon)$ queries. Viewing cycle-free graphs as graphs that have no K_3-minor, leads us to the following general result of [6], which refers to graph minors (see Section 5). While their original result asserted that any minor-close property can be tested in query complexity that is triple-exponential in $O(d/\epsilon)$, the currently known bound is quasi-polynomial in d/ϵ; see [30], which builds on [26]. (These testers have two-sided error probability.)[47] It is indeed a begging open problem whether the bound can be improved to a polynomial in d/ϵ (see Problem 26).

We mention that proprties in a broader class, which consists of sets of *hyperfinite graphs*, are each testable in complexity $\exp(d^{\mathrm{poly}(1/\epsilon)})$. A graph is called (ϵ, t)-hyperfinite if it is $(2\epsilon/d)$-close to a graph that consists of connected components that are each of size at most t (i.e., removing ϵk edges yields the latter graph). A set of graphs is hyperfinite if there exists a function $T : (0, 1] \to \mathbb{N}$ such that for every $\epsilon > 0$ every graph in the set is $(\epsilon, T(\epsilon))$-hyperfinite. (Minor-closed properties are hyperfinite [2].)[48] The result of Hassidim *et al.* [26] implies that any monotone property of hyperfinite graphs is testable in $\exp(d^{O(T(\mathrm{poly}(1/\epsilon)))})$-time, where T is the foregoing function and a property is called monotone if it is preserved under omission of edges.[49]

6.2 Testability in $\widetilde{O}(k^{1/2}) \cdot \mathrm{poly}(1/\epsilon)$ queries.

The query complexity of testing `Bipartiteness` and `Expansion` is $\widetilde{\Theta}(k^{1/2}) \cdot \mathrm{poly}(1/\epsilon)$, and in both cases the time complexity has the same form. The `Bipartite` tester has one-sided error, and whenever it rejects it may also output a short proof that the graph is not bipartite (i.e., an odd cycle of length $\mathrm{poly}(\epsilon^{-1} \log k)$).

[46] A one-sided error tester of query complexity $\mathrm{poly}(d/\epsilon) \cdot \widetilde{O}(\sqrt{k})$ for `Cycle-freeness` is presented in Section 4.2.

[47] This is unavoidable in light of the lower bound on the query complexity of one-sided error testers for cycle-freeness (presented in Section 3.2). Furthermore, by [10], for any H that is not cycle-free, one-sided error testing H-minor freeness requires $\Omega(\sqrt{k})$ queries.

[48] See [30, Cor. 2] for details.

[49] We mention that a graph property is called hereditary if it is preserved under omission of vertices (and their incident edges); that is, hereditary graph properties are preserved by induced subgraphs.

In both cases, the algorithm is based on taking many (i.e., $\widetilde{O}(k^{1/2}) \cdot \mathrm{poly}(1/\epsilon)$) *random walks* from a few randomly selected vertices, where each walk has length $\mathrm{poly}(\epsilon^{-1} \log k)$. This algorithmic approach originates in [22], where it was applied to testing `Bipartiteness`; for further details see Section 4.1. (Indeed, this approach is even more natural for testing `Expansion`, but the analysis was blocked by a combinatorial difficulty [23], which was resolved later in [27, 33].)[50]

The $\Omega(k^{1/2})$ lower bounds on the query complexity of testing each of the aforementioned properties were proved in [21]; for details see Section 3. We note that the lower bound for testing Bipartiteness stands in sharp contrast to the situation in the dense graph model, where Bipartite testing is possible in $\mathrm{poly}(1/\epsilon)$-time. This discrepancy is due to the difference between the notions of relative distance employed in the two models.

6.3 Additional issues

Let us highlight some issues that arise from the foregoing exposition.

Adaptive testers versus non-adaptive ones. As stated at the very beginning of this chapter (see Theorem 2), non-adaptive testers are significantly handicapped in the current model. Unless they use $\Omega(\sqrt{k})$ queries, such testers cannot do more than gather statistics regarding the degrees of vertices in a k-vertex graph. In contrast, adaptive testers of constant query complexity can explore local neighborhood in the graph, which allows for deducing numerous global properties as exemplified in Section 2 and 5.

One-sided versus two-sided error probability. The problem of testing cycle-freeness provides a dramatic illustration of the gap between one-sided error and two-sided error. Recall that Theorem 17 asserts that one-sided error testers for cycle-freeness require $\Omega(\sqrt{k})$ queries,[51] whereas two-sided error ϵ-testing is possible within query complexity $\mathrm{poly}(1/\epsilon)$ (see Section 2.5).

Proximity Oblivious Testers. The testers for subgraph freeness and degree regularity (see Sections 2.1 and 2.2, respectively) were obtained by presenting (one-sided error) proximity oblivious testers (of constant-query complexity and detection probability that depends only on the distance of the tested graph from the property). A partial characterization of properties that have such testers appears in [24], where one of the directions relies on a natural combinatorial conjecture.

An application to the study of the dense graph model. As noted several times, the bounded-degree graph model differs fundamentally from the dense graph model. In light of this fact, it is interesting to note that the `Bipartiteness` tester for the *bounded-degree graph model* was used in order to derive an alternative `Bipartiteness` tester for the *dense graph model* [25]. For any $\alpha \geq 0$, assuming that almost all vertices in the k-vertex graph have degree $O(\epsilon^{1-\alpha}k)$, this tester has query complexity $\widetilde{O}(\epsilon^{-(1.5+O(\alpha))})$, which improves over the testers presented in [17, 1]. Essentially, this dense-graph model tester invokes the bounded-degree model tester on the subgraph induced

[50] **Advanced comment:** As stated at the beginning of Section 4.1, it is not *a priori* clear that taking many short random walks (from a random start vertex) is a good strategy towards testing `Bipartiteness`. In contrast, it is apparent that the collision probability of logarithmically long random walks is related to the graph's expansion. Unfortunately, the exact relation between the these measures and the distance of the graph from being an expander is more evasive.

[51] Recall that this lower bound is relatively tight (see Section 4.2).

by a sample S of $\widetilde{O}(\epsilon^{-1})$ random vertices (and emulates neighbor queries regarding a vertex $v \in S$ by making adjacency queries of the form (v, w) for every $w \in S$).

Relations to other areas of algorithmic research. The fact that the bounded-degree graph model is closer (than the dense graph model) to standard algorithmic research offers greater opportunities for interaction at the technical level. Indeed, techniques such as local search and random walks are quite basic in both domains, and the relationship will become even tighter when we move to the general graph model (in the next lecture). A few concrete examples in which such interaction has occurred are stated next.

- Karger's randomized algorithm for finding minimum-cuts [28] inspired the algorithm for finding small t-connected components (i.e., Algorithm 10) and is used in its analysis (see proof of Theorem 11).
- Distributed network algorithms with few communication rounds were used to obtain property testers, super-fast parameter estimators, and local computation algorithms (see [36] followed by [34]). Implications in the opposite direction were foreseen by Onak [34] and materialized in [11].
- The idea underlying the cycle-freeness tester (presented in Section 2.5) was employed to the design of an algorithm for approximating the weight of a minimum spanning tree in sub-linear time [8].
- The one-sided error testers for cycle-freeness and other minor-free properties yield sub-linear time algorithms for finding natural structures in graphs (see Theorem 24 and the discussion preceding it).

7 Chapter notes

7.1 Historical perspective and credits

The study of property testing in the bounded-degree graph model was initiated by Goldreich and Ron [21], with the aim of allowing the consideration of sparse graphs, which appear in numerous applications. The point was that the dense graph model, introduced before in [17], seems irrelevant to sparse graphs, both because the distance measure that underlies it deems all sparse graphs as close to one another, and because adjacency queries seems unsuitable for sparse graphs. Sticking to the paradigm of representing graphs as functions, where both the distance measure and the type of queries are determined by the representation, the bounded-degree incidence function representation seemed the most natural choice. Indeed, a conscious decision was (and is) made not to capture, at this point (and in this model), sparse graphs that do not have constant (or low) maximum degree.

Most testers presented in Section 2 (which operate via "local searches") are taken from the work of Goldreich and Ron [21]. This includes the (one-sided error) testers for subgraph-freeness and connectivity, and the two-sided error tester for cycle-freeness.[52] (The one-sided error tester for cycle-freeness (presented in Section 4.2) is due to [10], and the tester of degree-regularity (presented

[52] Actually, the two-sided error tester for cycle-freeness is a variant on the tester presented in [21], and the complexity improvement (captured in Theorem 15) has not appeared before.

in Section 2.2) is adapted from the one used for the dense graph model). The tester for t-`Edge Connectivity` is due to [21], and Algorithm 10 was inspired by Karger's work [28].[53] The tester for t-`Vertex Connectivity` is due to Yoshida and Ito [43], while only the case of $t \leq 3$ was handled in (the conference version) of [21].

The $\Omega(\sqrt{k})$ lower bounds on the query complexity of `Bipartitness` (presented in Section 3.1) and its applications to testing cycle-freeness and expansion (presented in Section 3.2) are also due to Goldreich and Ron [21]. The linear lower bound on the query complexity of `3-Colorability` is due to Bogdanov, Obata, and Trevisan [7].

The (random-walk based) tester for `Bipartiteness` was presented by Goldreich and Ron [22], and the one-sided error tester for `Cycle-freeness` was presented in [10]. Using random walks is most natural in the context of testing `Expansion` [23], but the analysis of such testers was successfully completed only in later works [27, 33]. We mention that the reduction of testing uniformity of a distribution to estimating its collision probability, which underlies the expansion tester, has become quite pivotal to the study of testing distributions, which emerged with [4].

Testing minor-free properties was first considered by Benjamini, Schramm, and Shapira [6], who presented testers of query complexity that is triple-exponential in $1/\epsilon$. (These testers as well as all subsequent ones have two-sided error probability.) The bound was improved to a single exponential by Hassidim, Kelner, Nguyen, and Onak [26], who also provided testers for any hyperfinite properties (with double-exponential in $1/\epsilon$ complexity). The quasi-polynomial (in $1/\epsilon$) time-bound for testing minor-free properties (Theorem 25) is due to Levi and Ron [30]. The technique of constructing and using partition oracles was presented explicitly in [26, 34], where two different approaches for constructing such oracles are outlined (see [34, Sec. 2.5.1] and [34, Sec. 2.5.2], respecively).[54] We follow the approach described in Onak [34, Sec. 2.5.2], and its improvement by Levi and Ron [30], which leds to the aforementioned quasi-polynomial bound.

7.2 Directed graphs

Our discussion was confined to undirected graphs. Nevertheless, as noted in the prior lecture, the current model extends naturally to the case of directed graphs. Actually, when considering incidence queries, four natural sub-models emerge:[55]

1. The first two models refers to graphs in which the both the out-degree and the in-degree is bounded by d. In the first sub-model the tester may only query for edges in the forward direction, whereas in the second sub-model both forward and backward directions are allowed:

   (a) In the first sub-model, the directed k-vertex graph $G = ([k], E)$ is represented by a function $g_{\text{out}} : [k] \times [d] \to \{0, 1, ..., k\}$ such that $g_{\text{out}}(v, i) = u$ if the i^{th} out-going edge of v leads to u, and $g_{\text{out}}(v, i) = 0$ if v have less than i out-going edges.

   (b) In the second sub-model, the directed graph $G = ([k], E)$ is represented by two functions, g_{out} and g_{in}, where g_{out} is as in the first sub-model and $g_{\text{in}} : [k] \times [d] \to \{0, 1, ..., k\}$ is defined analogously with respect to in-coming edges (i.e., $g_{\text{in}}(v, i) = u$ if the i^{th} in-coming edge of v arrives from u).

[53] Indeed, the proof of Claim 11.2 is essentially due to [28].

[54] The first approach is applicable to any hyperfinite graph, whereas the second approach is applicable only to minor-free graphs.

[55] Actually, two additional models can be presented by considering in-coming edges only. These models are analogous to the two sub-models that consider out-going edges only. We believe that the forward direction is more natural.

These models were introduced and studied in [5].

2. The other two models refers to graphs in which the out-degree is bounded by d, but there is no bound on the in-degree. Again, in the first sub-model the tester may only query for edges in the forward direction, whereas in the second sub-model both forward and backward directions are allowed. That is, in the first sub-model the graph is represented by a function $g_{\text{out}} : [k] \times [d] \to \{0, 1, ..., k\}$ as in Model 1a, whereas in the second sub-model the tester is also provided with oracle access to a function $g_{\text{in}} : [k] \times [k] \to \{0, 1, ..., k\}$ that represents the in-coming edges (as in Model 1b).

   To the best of our knowledge, these models were not considered so far.

These four different models can be justified by different settings, and they vastly differ in their power. Needless to say, graph of bounded out-degree and unbounded in-degree are not captured by the first couple of models. The gap between the two query models (in the case that both the out-degree and the in-degree are bounded) was demonstrated by Bender and Ron, who initiated the study of testing properties of directed graphs [5]. In particular, they showed that strong connectivity can be tested (with one-sided error) by $\widetilde{O}(1/\epsilon)$ forward and backward queries [5, Sec. 5.1], but when only forward queries are allowed the query complexity of testing strong connectivity is $\Omega(\sqrt{k})$ (even when allowing two-sided error [5, Sec. 5.2]).[56] A recent study of the gap between these two models shows that if a property can be tested with a constant number of queries in the bi-directional query model, then it can be tested in a sublinear number of queries in the uni-directional query model [9].[57]

Another task studied in [5] is testing whether a given directed graph is acyclic (i.e., has no directed cycles). They presented an `Acyclicity` tester of poly$(1/\epsilon)$ complexity in the adjacency predicate model, and showed that in the incidence list model no `Acyclicity` tester can work with $o(k^{1/3})$ queries (even when both forward and backward queries are allowed). The question of whether `Acyclicity` can be tested with $o(k)$ queries (in the bounded-degree digraph model) remains open. In general, it seems that the study of these models deserves more attention than it has received so far.

7.3 Exercises

Exercise 1 (determining the degree of a vertex): *Given a k-vertex graph G of maximal degree d, represented by $g : [k] \times [d] \to \{0, 1, ..., k\}$, prove the following claims.*

1. *The degree of a given vertex can be determined using $\lceil \log(d+1) \rceil$ incidence queries.*
2. *Determining the degree of a given vertex requires at least $\lceil \log(d+1) \rceil$ incidence queries.*
3. *Show that a randomized algorithm* (which errs with probability at most 1/3)*, may use one query less, and this is optimal upto one query.*

[56] The lower bound can be strengthened to $\Omega(k)$ when considering only one-sided error testers. In the case of two-sided error, sublinear complexity is possible also in the uni-directional model (i.e., for every constant $\epsilon > 0$, strong connectivity is ϵ-testable by $k^{1-\Omega(1)}$ forward queries) [15, Apdx. A.3].

[57] The transformation does not preserve one-sided error probability, and this is inherent, since there are properties that have a constant-query one-sided error tester in the bi-directional model but no sublinear-query one-sided error tester in the uni-directional model.

Indeed, the first two claims refer to deterministic algorithms.

Guideline: Regarding Claim 1, given vertex v, reduce the problem of determining its degree to the problem of determining $i \in [d+1]$ such that $f(i) = 0$ and $f(i-1) = 1$, where $f : \{0,1,...,d+1\} \to \{0,1\}$ is defined such that $f(j) = 1$ if and only if either $g(v,j) \in [k]$ or $j = 0$.

Turning to Claim 2 (and assuming $d < k$), for each $i \in \{0,1,...,d\}$, consider the function g_i such that $g_i(k,j) = j$ and $g_i(j,1) = k$ for every $j \in [i]$, whereas $g(v,j) = 0$ for all other values. (The point is that $g_i(v,j) \in \{0, f(v,j)\}$ for all i, v, j, so each query has at most two possible answers.) Observe that any q-query algorithm that determines i (which equals the degree of vertex k) when given access to an unknown function g_i yields a depth q binary decision tree that has at least $d+1$ leaves (which are labeled by the different possible outputs).

For the positive part of Claim 3, consider an algorithm that discard of a random $\lfloor d/3 \rfloor$-subset of $[d]$ when applying the reduction of Claim 1. (That is, for a selected subset I of size $\lfloor d/3 \rfloor$, we consider the function $f' : (\{0,1,...,d+1\} \setminus I) \to \{0,1\}$ rather than the function f.) For the negative part, consider a choice of internal coin tosses for which the residual deterministic algorithm errs on at most $\lfloor (d+1)/3 \rfloor$ of the possible i's (considered in Claim 2).

Exercise 2 (non-adaptive tester of triangle-freeness):[58] *Show that triangle-freeness has a non-adaptive tester of query complexity $O(d\sqrt{k/\epsilon})$.*

Guideline: First observe that a graph that is ϵ-far from being triangle-free must have more than $\epsilon dk/2$ different triangles, since each triangle can be removed by omitting a single edge. Hence, at least $\epsilon k/2$ different vertex pairs participate in some triangle, which means that a random vertex-pair participates in a triangle with probability at least $\Omega(\epsilon/k)$. The non-adaptive tester selects a random sample S of $O(\sqrt{k/\epsilon})$ vertices, and queries the graph at all the vertex-index pairs $S \times [d]$. The claim is proved by considering the vertex-pairs $S \times S$.

Exercise 3 (testing subgraph freeness for an unconnected subgraph): *Let H be a fixed graph that consists of the connected components $H_1, ..., H_m$, having radii $r_1, ..., r_m$, respectively. Let $r = \max_{i \in [m]}\{r_i\}$. Show that H-freeness has a* (one-sided error) *proximity-oblivious tester of query complexity $O(d^{r+1})$ and polynomial detection probability. Alternatively, H-freeness has a* (one-sided error) *tester of query complexity $\widetilde{O}(m) \cdot d^{r+1}/\epsilon$.*

Guideline: A vertex $v \in [k]$ is called i-detecting if it is a center of a copy of H_i that resides in G. Prove that if G is at distance δ from H-free, then, for every $i \in [m]$, the graph G must have at least $\delta k/2$ vertices that are i-detecting. The POT selects m random vertices, performs an r-deep BFS from each, and rejects if and only if (vertex disjoint) copies of all the H_i's were found in these m searches. The ϵ-tester performs selects $O(m \log m)/\epsilon$ random vertices performs an r-deep BFS from each, and rejects if and only if (vertex disjoint) copies of all the H_i's were found in these searches.

Exercise 4 (on testing whether a graph is Eulerian): *Suppose that $G = ([k], E)$ has maximum degree d, and k' of its vertices have odd degree. Prove that there exists a k-vertex Eulerian graph $G' = ([k], E')$ of maximum degree d such that the symmetric difference between E and E' is at most $3k'/2$. Present a* (one-sided error) *proximity oblivious tester based on this observation, and determine its query complexity and detection probability.*

[58] Based on a result in [15, Apdx. A.2].

Guideline: If d is even, then vertices of odd degree have degree smaller than d. In this case, we just proceed in iteration, such that at each iteration we choose a pair of vertices of (current) odd degree and change their adjacency relation (i.e., if they were adjacent, then we omit the edge between them, and otherwise we insert an edge between them). Hence, we are done in $k'/2$ iterations, while modifying a single edge in each iteration. If d is odd, then we first omit a single edge from each vertex that has degree d, while observing that this does not increase the number of vertices of odd degree.[59] Once this stage is completed, we proceed as before, while observing that (from this point on) vertices of odd degree have degree at most $d-2$.

Exercise 5 (on testing whether a graph is connected and Eulerian): *We stress that all graphs here are of maximal degree d, and distances between them are as in Definition 1.*

1. *Prove that if $G=([k],E)$ is ϵ_1-close to being connected and ϵ_2-close to being Eulerian, then it is $O(\epsilon_1+\epsilon_2)$-close to a graph that is both connected and Eulerian.*
2. *Using Item 1, present and analyze a tester for the property of being a connected Eulerian graph.*

Guideline: Let $G'=([k],E')$ be an Eulerian graph that is ϵ_1-close to G, and note that G' is $(\epsilon_1+\epsilon_2)$-close to being connected. Assuming that G' has k' connected components, turn it into a connected graph while preserving the degrees of all vertices and making $2k'$ modifications.[60]

Exercise 6 (finding small 2-edge connected components):[61] *Given a vertex v that resides in a set S of size at most s such that the subgraph of $G=([k],E)$ induced by S is 2-edge-connected and the cut $(S,[k]\setminus S)$ contains at most one edge, prove that the following algorithm finds S in time $O(ds)$.*

1. *Invoke a DFS at the vertex v and suspend its execution as soon as more than s vertices are encountered.*
2. *If the DFS detected a connected component of size at most s, then output it.*

   Otherwise, consider a directed *graph, denoted $\vec{G}$, that is obtained from G as follows. If $\{u,v\}$ is an edge of the DFS-tree that was first traversed* (during the DFS) *from u to v, then only the edge directed from v to u is placed in $\vec{G}$* (i.e., we do not include edges in the direction used by the DFS in discovering a new vertex). *Any edge $\{u,v\}$ that is not an edge of the DFS-tree is replaced by a pair of anti-parallel edges* (i.e., we include both (u,v) and (v,u), where (x,y) denotes the edge directed from x to y).
3. *Invoke a directed search, starting at vertex v, on the directed graph $\vec{G}$, and output the set of vertices visited in this search. The directed search* (e.g., a directed BFS or DFS) *only traverses directed edges in the forward direction.*

Guideline: Suppose that $(S,[k]\setminus S)$ contains a single edge. Then, in Step 2, this edge must be traversed by the DFS in the direction from S to $[k]\setminus S$, and it follows that Step 3 outputs a subset of S. Prove that the output cannot be a strict subset of S, by relying on the hypothesis that S is 2-edge-connected.

[59] Note that the number of vertices of degree d (in the original graph) is at most k'.

[60] Using the fact that each of the connected components is Eulerian, omit a single edge $\{u_i,v_i\}$ from the i^{th} connected component, and add the edges $\{v_i,u_{i+1}\}$ for $i=1,...,k'$, where the index $k'+1$ is views as 1.

[61] Proved in [21].

Exercise 7 (proof of Claim 11.1):[62] *Prove Claim 11.1 by considering the cut $C' = (S', [k] \setminus S')$ at any stage during the execution of Algorithm 10.*

Guideline: If $C' \neq C$ (equiv., $S' \subset S$), then the algorithm does not halt in Step 1 (since G_S is t-connected). But then C' must contain an edge of the lightest spanning tree of G_S, whereas all edges of this tree are lighter than any edge of C. Hence, in Step 3, the algorithm chooses an edge of $C' \setminus C$. It follows that Algorithm 10 keeps extending the current set S' without ever choosing an edge that belongs to C (equiv., the extension is by a vertex in $S \setminus S'$).

Exercise 8 (obtaining a rough estimation for the number of edges): *Show that by using $O(1/\epsilon)$ queries, one can distinguish the case that a k-vertex graph of maximum degree d has more than $m \stackrel{\text{def}}{=} \max(2k, \epsilon dk/2)$ edges from the case that it has less than $m/2$ edges.*

Guideline: Show that such an approximation can be obtained by taking a sample of $O(dk/m)$ pairs, and note that $O(dk/\max(2k, \epsilon dk/2)) = O(\min(d, \epsilon^{-1})) = O(1/\epsilon)$.

Exercise 9 (graphs that are far from being bipartite may lack odd cycles of sub-logarithmic length): *Show that $O(1)$-regular k-vertex graphs that are $\Omega(1)$-far from being bipartite may have no odd-cycle of length $o(\log k)$.*

Guideline: This follows from Theorem 16 (by considering a potential tester that selects a random start vertex and explores all vertices at distance $D = o(\log k)$ from it).[63] Alternatively, note that constructions of such graphs are known, while noting that expander graphs are far from being bipartite).

Exercise 10 (graphs that are far from being bipartite have odd cycles of logarithmic length):[64] *Prove that if a* (bounded degree) *k-vertex graph is ϵ-far from being bipartite, then it has an odd cycle of length at most $L = O(\epsilon^{-1} \log k)$.*

Guideline: For starters, note that if all vertices of the graph are at distance at most $(L-1)/2$ from some vertex v, then the claim follows by considering a BFS that starts at v. (Note that some layer of this BFS must contain an edge, which yields an odd cycle of length at most $1 + (L-1)$, since otherwise the graph is bipartite.) In the general case, we apply an iterative process. In each iteration, we pick an arbitrary vertex in the residual graph and perform a truncated BFS that is suspended when the next layer of the BFS grows by less than an $\epsilon/2$ factor; that is, denoting the number of vertices visited by the current BFS so far is k', we stop (at the current layer) if the next layer has less than $\epsilon k'/2$ vertices. Hence, the BFS is suspended after at most $\log_{1+0.5\epsilon} k$ iterations, and if some layer of this BFS contains an edge, then we are done (as in the simple case). Otherwise, we omit the explored k'-vertex portion of the graph as well as all edges that are incident at it, which means that we omitted less than $\epsilon k' d/2$ edges. Note that this iterative process must find an edge in some layer of some BFS, because otherwise the graph can be made bipartite by omitting $\epsilon dk/2$ edges.

[62] Proved in [21].

[63] Note that if the graph G has $o(k)$ vertices that resides on odd-cycles of of length at most $2D-1$, then G is $o(1)$-close to being bipartite.

[64] Proved in [22], analogously to the proof of the widely known upper bound on the girth of graphs with certain edge density, where the girth of a graph is the length of its shortest simple cycle. Specifically, the claim asserts that if a k-vertex graph has at least $(1+\epsilon) \cdot k$ edges (which means that, in the context of the bounded-degree model, it is viewed as $\Omega(\epsilon)$-far from being cycle-free), then it has a cycle of length at most $O(\epsilon^{-1} \log k)$.

Exercise 11 (a detail for the proof of Claim 21.2): *Let $p_1(u)$ and $p'_1(u)$ be as in Claim 21.2 (and in its proof sketch). Prove that $p'_1(u) \in [0.9 \cdot p_1(u), 1.1 \cdot p_1(u)]$.*

Guideline: It is instructive to view the ℓ-step walks (starting at vertex s) as ℓ-long sequences over $[2d]$ such that the i^{th} symbol in the sequence $(\alpha_1, ..., \alpha_\ell)$ is interpreted as moving to the α_i^{th} neighbor of the current vertex if that vertex has at least α_i neighbors, and staying in place otherwise. Hence, symbols in $[d+1, 2d]$ always represent staying in place, since the degree of the current vertex is always at most d. Now, define $S_{\ell,\ell'}$ as the set of ℓ-long sequences with exactly ℓ' symbols in $[d+1, 2d]$ that correspond to walks of odd path-parity that end at u. Lastly, for every $\gamma \in [d]^{\ell-\ell'}$, let

$$S^{\gamma}_{\ell,\ell'} \stackrel{\text{def}}{=} \{\alpha \in S_{\ell,\ell'} : \exists I \text{ s.t. } |I| = \ell' \wedge \alpha_I \in [d+1, 2d] \wedge \alpha_{[\ell]\setminus I} = \gamma\}$$

The sets $S_{\ell-1,\ell'-1}$ and $S^{\gamma}_{\ell-1,\ell'-1}$ are define analogously. Prove the following claims:

1. $\mathbf{Pr}_{\alpha\in[2d]^\ell}[\alpha \notin \cup_{\ell'\in[0.49\ell,0.51\ell]} S_{\ell,\ell'}] = \exp(-\Omega(\ell)) < 0.01/k$.
2. For every $\ell' \in [\ell]$ and $\gamma \in [d]^{\ell-\ell'}$, it holds that $\ell' \cdot |S^{\gamma}_{\ell,\ell'}| = d \cdot \ell \cdot |S^{\gamma}_{\ell-1,\ell'-1}|$.

   Hence, $\mathbf{Pr}_{\alpha\in[2d]^\ell}[\alpha \in S^{\gamma}_{\ell,\ell'}] = \frac{d\ell}{2d\cdot\ell'} \cdot \mathbf{Pr}_{\beta\in[2d]^{\ell-1}}[\beta \in S^{\gamma}_{\ell-1,\ell'-1}]$.

Finally, combine the two claims.

Exercise 12 (a detail for the proof of Lemma 22.1): *Recall that the proof provided in the text assumed that G is connected. Extend the proof to the general case.*

Guideline: Focus on the case of $\Delta > k/2$, while observing that the reduction of the general case to the case of $\Delta > k/2$ made no reference to connectivity. Considering a graph with m connected components, having sizes $k_1, ..., k_m$, use the fact that with probability at least $\max(0.5, 1-\exp(-\Omega(k_i)))$, the i^{th} connected component is $\Omega(1)$-far from being bipartite.

Exercise 13 (a detail for the proof of Lemma 30): *Suppose that $G' = ([k'], E')$ is an H-minor free graph with weights on its edges and let $h(v)$ denote the other endpoint of the heaviest edge incident at vertex v (of G'). Show that the set of directed edges $\{(v, h(v)) : v \in [k']\}$ has total weight that is at least a $1/d_H$ fraction of twice the total weight of the edges of G', where d_H is a constant such that every H-minor free graph has a vertex of degree at most d_H.* (Consequently, the set of undirected edges $\{\{v, h(v)\} : v \in [k']\}$ has total weight that is at least a $1/d_H$ fraction of the total weight of the edges of G'.)[65]

Guideline: Consider an iterative process of selecting a vertex of smallest degree in G', and omitting it and its edges from G'. Note that each of the resulting graphs is H-minor free, and thus has a vertex of degree at most d_H. It follows that the heaviest edge of this vertex, which may not be present at the current graph, has weight that is at least a $1/d_H$ fraction of the weight of the edges omitted at this step. That is, if at the current iteration we omitted the vertex v and all edges incident at it, then the weight of $h(v)$ is at least a $1/d_H$ fraction of the total weight of the edges that are incident to v at the beginning of the current iteration.

[65] Indeed, an edge $\{u, v\}$ of E' may be the heaviest edge of both u and v; that is, it is possible that $h(v) = u$ and $h(u) = v$. The factor of two in the main claim corresponds to replacing each edge of E' by a pair of anti-parallel directed edges each of having the weight of the original undirected edge.

Exercise 14 (graph properties are not random self-reducible): *Show that, except for a few trivial cases, graph properties of k-vertex graphs in the incidence function representation are not random self-reducible by $o(k)$ queries.*[66]

Guideline: See exercise in the lecture notes on the dense graph model.

References

[1] N. Alon and M. Krivelevich. Testing k-Colorability. *SIAM Journal on Disc. Math.*, Vol. 15 (2), pages 211-227, 2002.

[2] N. Alon, P.D. Seymour, and R. Thomas. A separator theorem for graphs with an excluded minor and its applications. In *22nd ACM Symposium on the Theory of Computing*, pages 293–299, 1990.

[3] N. Alon and J.H. Spencer, *The Probabilistic Method*, John Wiley & Sons, Inc., 1992. Third edition, 2008.

[4] T. Batu, L. Fortnow, R. Rubinfeld, W.D. Smith and P. White. Testing that Distributions are Close. In *41st IEEE Symposium on Foundations of Computer Science*, pages 259–269, 2000.

[5] M. Bender and D. Ron. Testing acyclicity of directed graphs in sublinear time. *Random Structures and Algorithms*, pages 184–205, 2002.

[6] I. Benjamini, O. Schramm, and A. Shapira. Every Minor-Closed Property of Sparse Graphs is Testable. In *40th STOC*, pages 393–402, 2008.

[7] A. Bogdanov, K. Obata, and L. Trevisan. A lower bound for testing 3-colorability in bounded-degree graphs. In *43rd FOCS*, pages 93–102, 2002.

[8] B. Chazelle, R. Rubinfeld, and L. Trevisan. Approximating the minimum spanning tree weight in sublinear time. In *19th ICALP*, pages 190–200, 2001.

[9] A. Czumaj, P. Peng, and C. Sohler. Properties Directed Networks. Talk at the *Sublinear Algorithms Workshop*, JHU, 2016.

[10] A. Czumaj, O. Goldreich, D. Ron, C. Seshadhri, A. Shapira, and C. Sohler. Finding cycles and trees in sublinear time. *Random Structures and Algorithms*, Vol. 45(2), pages 139–184, 2014.

[11] G. Even, M. Medina, and D. Ron. Best of Two Local Models: Local Centralized and Local Distributed Algorithms. CoRR abs/1402.3796, 2014.

[12] S. Even. *Graph Algorithms*. Computer Science Press, 1979. Second edition (edited by G. Even), Cambridge University Press, 2011.

[13] O. Goldreich. *Foundation of Cryptography – Basic Tools*. Cambridge University Press, 2001.

[66] See definition in the lecture notes on testing juntas.

[14] O. Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008.

[15] O. Goldreich. Introduction to Testing Graph Properties. In [16].

[16] O. Goldreich (ed.). *Property Testing: Current Research and Surveys*. Springer, LNCS, Vol. 6390, 2010.

[17] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, pages 653–750, July 1998. Extended abstract in *37th FOCS*, 1996.

[18] O. Goldreich and T. Kaufman. Proximity Oblivious Testing and the Role of Invariances. In *15th RANDOM*, pages 579–592, 2011.

[19] O. Goldreich, M. Krivelevich, I. Newman, and E. Rozenberg. Hierarchy Theorems for Property Testing. *ECCC*, TR08-097, 2008. Extended abstract in the proceedings of *RANDOM'09*.

[20] O. Goldreich and L.A. Levin. A hard-core predicate for all one-way functions. In the proceedings of *21st ACM Symposium on the Theory of Computing*, pages 25–32, 1989.

[21] O. Goldreich and D. Ron. Property testing in bounded degree graphs. *Algorithmica*, pages 302–343, 2002. Extended abstract in *29th STOC*, 1997.

[22] O. Goldreich and D. Ron. A sublinear bipartite tester for bounded degree graphs. *Combinatorica*, Vol. 19 (3), pages 335–373, 1999. Extended abstract in *30th STOC*, 1998.

[23] O. Goldreich and D. Ron. On Testing Expansion in Bounded-Degree Graphs. *ECCC*, TR00-020, March 2000.

[24] O. Goldreich and D. Ron. On Proximity Oblivious Testing. *SIAM Journal on Computing*, Vol. 40, No. 2, pages 534–566, 2011. Extended abstract in *41st STOC*, 2009.

[25] M. Gonen and D. Ron. On the Benefit of Adaptivity in Property Testing of Dense Graphs. In *Proc. of RANDOM'07*, LNCS Vol. 4627, pages 525–539, 2007. *Algorithmica* (special issue for RANDOM and APPROX 2007), Vol. 58 (4), pages 811–830, 2010.

[26] A. Hassidim, J. Kelner, H. Nguyen, and K. Onak. Local Graph Partitions for Approximation and Testing. In *50th FOCS*, pages 22–31, 2009.

[27] S. Kale and C. Seshadhri. Testing expansion in bounded degree graphs. In *35th ICALP*, pages 527–538, 2008. (Preliminary version appeared as TR07-076, *ECCC*, 2007.)

[28] D. Karger. Global min-cuts in $\mathcal{RNC}$ and other ramifications of a simple mincut algorithm. *SODA*, pages 21–30, 1993.

[29] T. Kaufman, M. Krivelevich, and D. Ron. Tight Bounds for Testing Bipartiteness in General Graphs. *SIAM Journal on Computing*, Vol. 33 (6), pages 1441–1483, 2004.

[30] R. Levi and D. Ron. A Quasi-Polynomial Time Partition Oracle for Graphs with an Excluded Minor. *ACM Transactions on Algorithms*, Vol. 11 (3), pages 24:1-24:13, 2015.

[31] L.A. Levin. One-way functions and pseudorandom generators. In proc. of the *17th ACM Symposium on the Theory of Computing*, pages 363–365, 1985.

[32] M. Mihail. Conductance and convergence of Markov chains– A combinatorial treatment of expanders. In *30th FOCS*, pages 526–531, 1989.

[33] A. Nachmias and A. Shapira. Testing the expansion of a graph. TR07-118, *ECCC*, 2007.

[34] K. Onak. *New sublinear methods in the struggle against classical problems*. PhD thesis, Massachusetts Institute of Technology, 2010.

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

[36] M. Parnas and D. Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theoretical Computer Science*, Vol. 381 (1-3), pages 183–196, 2007.

[37] M. Parnas, D. Ron, and R. Rubinfeld. Tolerant Property Testing and Distance Approximation. *Journal of Computer and System Sciences*, Vol. 72 (6), pages 1012–1042, 2006.

[38] S. Raskhodnikova and A. Smith. A note on adaptivity in testing properties of bounded-degree graphs. *ECCC*, TR06-089, 2006.

[39] N. Robertson and P.D. Seymour. Graph minors. I. Excluding a Forest. *Journal of Combinatorial Theory, Series B*, Vol. 35 (1), pages 39–61, 1983.

[40] N. Robertson and P.D. Seymour. Graph minors. XIII. The Disjoint Paths Problem. *Journal of Combinatorial Theory, Series B*, Vol. 63 (1), pages 65–110, 1995.

[41] N. Robertson and P. D. Seymour. Graph Minors. XX. Wagner's Conjecture. *Journal of Combinatorial Theory, Series B*, Vol. 92 (1), pages 325–357, 2004.

[42] D. Ron. Algorithmic and Analysis Techniques in Property Testing. *Foundations and Trends in TCS*, Vol. 5 (2), pages 73–205, 2010.

[43] Y. Yoshida and H. Ito. Property Testing on k-Vertex-Connectivity of Graphs. *Algorithmica*, Vol. 62 (3), pages 701–712, 2012.