Electronic Colloquium on Computational Complexity, Revision 2 of Report No. 45 (2018)

The Subgraph Testing Model*

Oded Goldreich† Dana Ron‡

June 21, 2020

Abstract

Following Newman (2010), we initiate a study of testing properties of graphs that are presented as subgraphs of a fixed (or an explicitly given) graph. The tester is given free access to a base graph $G = ([n], E)$, and oracle access to a function $f : E \to \{0, 1\}$ that represents a subgraph of G. The tester is required to distinguish between subgraphs that possess a predetermined property and subgraphs that are far from possessing this property.

We focus on bounded-degree base graphs and on the relation between testing graph properties in the subgraph model and testing the same properties in the bounded-degree graph model. We identify cases in which testing is significantly easier in one model than in the other as well as cases in which testing has approximately the same complexity in both models. Our proofs are based on the design and analysis of efficient testers and on the establishment of query-complexity lower bounds.

Keywords: Property testing, graph algorithms,

*This research was partially supported by the Israel Science Foundation (grants No. 671/13 and 1146/18). A preliminary version was posted on *ECCC* (as TR18-045), and an extended abstract has appeared in the *10th ITCS*.

†Department of Computer Science, Weizmann Institute of Science, Rehovot, Israel. oded.goldreich@weizmann.ac.il

‡School of Electrical Engineering, Tel Aviv University, Tel Aviv, Israel. danaron@tau.ac.il

ISSN 1433-8092

Contents

1 Introduction

Property testing refers to probabilistic algorithms of sub-linear complexity for deciding whether a given object has a predetermined property or is far from any object having this property. Such algorithms, called testers, obtain local views of the object by *performing queries* and their performance guarantees are stated with respect to a distance measure that (combined with a distance parameter) determines which objects are considered far from the property.

In the last couple of decades, the area of property testing has attracted significant attention (see, e.g., [Gol17]). Much of this attention was devoted to testing graph properties in a variety of models ranging from the dense graph model [GGR98], to the bounded-degree graph model [GR02], and to the sparse and general graph models [PR02, KKR04].[1] These models differ in two main parameters: the *types of queries* that potential testers can make, and the *distance measure* against which their performance is measured. (In all cases and throughout this work, unless explicitly stated differently, we consider undirected simple graphs (i.e., no self-loops and parallel edges).)

In all aforementioned models, the input graph is arbitrary, except for its size (and possibly its degree, in the case of the bounded-degree graph model). While some prior works (see, e.g., [BSS10, HKNO09, CSS09, GR10, Ele10, EHNO11, NS13, CMOS19]) restrict the input graph in certain natural ways, the restrictions considered so far were expressed in terms of general ("uniform") graph properties (such as degree bound, hyperfiniteness, planarity, etc). See further discussion in Section 1.5.1.

In contrast, we envision circumstances in which the input is restricted to be a subgraph of some fixed graph that is known beforehand. For example, the fixed graph may represent an existing (or planned) network, and the subgraph represents the links that are actually in operation (or actually constructed). Alternatively, the graph may represent connections between data items that may exist under some known constraints, and the edges of the subgraph represent connections that actually exist. Either way, the input is a subgraph of some fixed graph, and the distance to having the property is measured with respect to subgraphs of the same fixed graph.

1.1 The model

In accordance with the foregoing discussion, in the *subgraph testing model*, we consider a fixed *base graph*, denoted $G = ([n], E)$, and the tester that is given oracle access to a function $f : E \to \{0,1\}$, which represents a subgraph of G in the natural manner (i.e., f represents the subgraph $([n], \{e \in E : f(e) = 1\})$). Alternatively, the base graph G is not fixed, but the tester is given free access to G.

Definition 1.1 (subgraph tester): *Fixing $G = ([n], E)$ and $\Pi_G \subseteq \mathcal{F}_G \stackrel{\text{def}}{=} \{f : E \to \{0,1\}\}$, a* subgraph tester for Π_G *is a probabilistic oracle machine, denoted T, that, on input a* (proximity) *parameter ϵ, and oracle access to a function $f : E \to \{0,1\}$, outputs a binary verdict that satisfies the following two conditions.*

1. T accepts inputs in Π_G: *For every $\epsilon > 0$, and for every $f \in \Pi_G$, it holds that $\Pr[T^f(\epsilon) = 1] \geq 2/3$.*

2. T rejects inputs that are ϵ-far from Π_G: *For every $\epsilon > 0$, and for every $f : E \to \{0,1\}$ that is ϵ-far from Π_G it holds that $\Pr[T^f(\epsilon) = 0] \geq 2/3$, where f is* ϵ-far from Π_G *if for every $h \in \Pi_G$ it holds that $|\{e \in E : f(e) \neq h(e)\}| > \epsilon \cdot |E|$.*

[1] These models are surveyed in Chapters 8, 9, and 10 of the textbook [Gol17].

If the first condition holds with probability 1 (i.e., $\Pr[T^f(\epsilon)=1]=1$ *for* $f \in \Pi_G$*), then we say that* T *has* one-sided error*; otherwise, we say that* T *has* two-sided error.

In the alternative formulation, the subgraph tester is given G as an explicit input (along with ϵ). In this case, the random variable being considered is $T^f(G,\epsilon)$.

Definition 1.1 falls within the framework of massively parameterized property testing (cf. [New10]). The massive parameter is the base graph $G=([n],E)$, and the actual input is a function $f:E\to\{0,1\}$ (which represents a subgraph of G). Actually, Newman explicitly discusses the subgraph testing model in [New10, Sec. 1].[2]

As usual in the area, our primary focus is on the query complexity of such testers, and our secondary focus is on their time complexity. Both complexities are stated as a function of the proximity parameter ϵ and the base graph G. Indeed, the dependency of these complexities on G, or rather on some parameters of G, will be of major interest.

As an illustration, consider the problem of testing whether the subgraph is bipartite. If the base graph is bipartite, then this problem is trivial (since every subgraph is bipartite). If the base graph is $\mathcal{M}$-minor free[3], for any fixed family of graphs $\mathcal{M}$, then testing (with distance parameter ϵ) can be done in $\mathrm{poly}(1/\epsilon)$-time (see Proposition 2.2). Lastly, if the base graph is of bounded-degree, then testing can be done in $\mathrm{poly}(1/\epsilon)\cdot\widetilde{O}(\sqrt{n})$-time (see Theorem 1.2), and this result is optimal in general (i.e., for arbitrary bounded-degree base graphs, see Part 1 of Theorem 1.4).

Our main focus will be on the case that the base graph is sparse (e.g., of bounded-degree). Furthermore, we shall focus on cases in which the subgraph testing model is different from other testing models. Still, let us make a couple of comments regarding cases in which the subgraph testing model coincides with other testing models.

The dense graph model is a special case of subgraph testing. As observed in [New10, Sec. 1], for the base graph $G=K_n$ (i.e., the n-vertex clique), the subgraph testing model coincides with the dense graph model. This is the case since adjacency queries (as in the dense graph model) correspond to edges of the base graph G, and the distance measure used in both models is the same.

General property testing (of Boolean functions) as a special case of subgraph testing. If the base graph G has cn edges for a constant $c\geq 1$, then the subgraph testing model captures property testing (for Boolean functions) at large. This is shown as follows.

Consider in particular the case that $G=([n],E)$ is a cycle with an edge between i and $i+1$, for every $1\leq i\leq n-1$, as well as an edge between n and 1. Then, any function $h:[n]\to\{0,1\}$ can be represented by a subgraph G^h of G that contains the edge between i and $i+1$ (similarly, $(i,1)$ for $i=n$) if and only if $h(i)=1$. In such a case, any query to G^h can be answered by a single query to h. For any property $\mathcal{P}$ of Boolean functions over $[n]$, let $\Pi_G^{\mathcal{P}}=\{G^h:\ h\in\mathcal{P}\}$. Hence, for any $h:[n]\to\{0,1\}$, the distance of h to $\mathcal{P}$ equals the distance of G^h to $\Pi_G^{\mathcal{P}}$. Essentially the same argument can be applied to any base graph $G=([n],E)$, provided that $|E|\geq n$ and $|E|=O(n)$ (the lower bound on E allows to associate each $i\in h^{-1}(1)$ with an edge in G, and the upper bound ensures that distances are maintained up to a constant).

[2] Unfortunately, we forgot this historical fact when writing prior versions of this work, where we only attributed to Newman the conceptualization of the general framework of massively parameterized property testing.

[3] Recall that a graph M is a minor of graph G if M can be obtained from G by vertex deletions, edge deletions and edge contractions; a graph G is $\mathcal{M}$-minor free for a family of graphs $\mathcal{M}$, if no graph in $\mathcal{M}$ is a minor of G.

Testing graph properties in the subgraph model. In general, a property of subgraphs of a base graph G (i.e., a property $\Pi_G \subseteq \mathcal{F}_G$ as in Definition 1.1) is not a graph property (i.e., it is not closed under graph isomorphism).[4] Still, following [New10, Sec. 1], we shall confine ourselves to the case that Π_G corresponds to the set of all subgraphs of G that has some graph property Π. One reason for this restriction is that it allows comparing the task of testing whether a subgraph is in Π_G to the task of testing Π in some other model (e.g., testing Π in the bounded-degree graph model). Abusing notation, in these cases, we may write $\Pi_G = \mathcal{F}_G \cap \Pi$, and sometimes refer to Π_G using the term graph property.

The focus on the case that $\Pi_G = \mathcal{F}_G \cap \Pi$, for some graph property Π, calls for revisiting the claim that any property of Boolean functions can be represented in the subgraph testing model. The issue is that the property Π_G presented above cannot be written as $\mathcal{F}_G \cap \Pi$, for some graph property Π. Still, a variant of it will do.

Figure 1: For $n \geq 6$, the n-vertex path is "oriented" by the additional edge $\{n-3, n-1\}$ (which breaks the symmetry between vertex 1 and vertex n, and more generally between j and $n-j+1$).

Specifically, for $n \geq 6$, consider an n-vertex graph G' consisting of an n-vertex long path augmented with the edge $\{n-3, n-1\}$ (see Figure 1). Observe that the only automorphism of this graph is the identity permutation, and augment G' with self-loops on each of the n vertices, deriving a base graph G with $2n$ edges. (We note that the construction can be modified so that self-loops are avoided, by replacing them with disjoint cycles of length 3.) We can associate any function $h : [n] \to \{0,1\}$ with a subgraph G^h of G that contains G' as well as the self-loop on vertices in $h^{-1}(1)$. Note that, by the asymmetry of G', there is a bijection between the set of Boolean functions over $[n]$ and the subgraphs of G that contain G'. Hence, for each property $\mathcal{P}$ of Boolean functions, there exists a graph property Π such that there is a bijection between $\mathcal{P}$ and the set $\mathcal{F}_G \cap \Pi = \{G^h : h \in \mathcal{P}\}$ (e.g., Π may contain all graphs that are isomorphic to some graph in $\{G^h : h \in \mathcal{P}\}$).

A simplifying assumption. Throughout this paper, we assume that G contains no isolated vertices. This can be assumed without loss of generality, because, for every graph G' that is obtained from the graph $G = ([n], E)$ by adding isolated vertices, it holds that $\mathcal{F}_G = \mathcal{F}_{G'}$, since in both cases the subgraphs are represented by Boolean functions on the same edge-set (i.e., E).

[4] In fact, Π_G is a graph property only in pathological cases that include the case of $\Pi_G = \emptyset$ and the case that G is either the complete graph or the empty graph. Otherwise, the property $\Pi_G \subseteq \mathcal{F}_G$ is not a graph property, since it is not closed under isomorphism (because G is not invariant under all possible relabelings of its vertex set). Note that the n-vertex complete graph and the empty (n-vertex) graph are the only n-vertex graphs that are invariant under all possible relabelings of $[n]$. In contrast, if G is neither empty nor complete, then it contains a vertex w that has degree in $[n-2]$; that is, w's neighbor set, denoted $\Gamma_G(w)$, is neither empty nor contains all other vertices in G. Picking $u \in \Gamma_G(w)$ and $v \notin \Gamma_G(w)$, observe that the permutation π that switches u and v, while keeping all other vertices intact, does not preserve the graph G (i.e., $\pi(G) \neq G$).

1.2 Results

Throughout this paper, the base graph G is viewed as a varying parameter, which may grow when other parameters (e.g., the degree bound d) are fixed. We focus on bounded-degree base graphs and on the relation between testing graph properties in the subgraph model and testing the same properties in the bounded-degree graph (BDG) model.

Recall that in the BDG model [GR02], the tester is explicitly given three parameters: n, d, and ϵ. Its goal is to distinguish with probability at least $2/3$ between the case that a graph $G = ([n], E)$ (of maximum degree bounded by d) has a prespecified property Π, and the case that G is ϵ-far from having the property Π. In this model a graph is said to be ϵ-far from having Π if more than $\epsilon \cdot d \cdot n/2$ edge modifications (additions or removals) are required in order to obtain a graph (of maximum degree bounded by d) that has Π. To this end the tester can perform queries of the form "who is the i^{th} neighbor of vertex v?", for $v \in [n]$ and $i \in [d]$.[5] Unless stated explicitly otherwise, the degree bound d is a constant.

Obviously, the relationship between the subgraph model and the BDG model depends on the property being tested as well as on the base graph used in the subgraph model. We identify cases in which testing is significantly easier in one model than in the other as well as cases in which testing has approximately the same complexity in both models.

More specifically, we distinguish *downward-monotone* graph properties from other graph properties, where a graph property is said to be *downward monotone* if it is preserved under omission of edges (i.e., if $G = ([n], E)$ has the property, then so does $G' = ([n], E')$ for every $E' \subset E$).

1.2.1 Downward-monotone properties

We first observe that, for every bounded-degree graph $G = ([n], E)$ and any downward-monotone graph property Π, testing $\Pi \cap \mathcal{F}_G$ in the subgraph model (w.r.t. the base graph G) reduces to testing Π in the BDG model.

Theorem 1.2 (a general upper bound on the complexity of testing downward-monotone properties (see Section 2.1)): *Let Π be a downward-monotone graph property that is testable with query complexity $Q_d(\cdot,\cdot)$ in the bounded-degree graph model, where $d \geq 2$ denotes the degree bound, and Q_d is a function of the proximity parameter and* (possibly) *the size of the graph. Then, for every base graph $G = ([n], E)$ of maximum degree d, testing whether a subgraph of G satisfies Π* (with proximity parameter ϵ) *can be done with query complexity $d \cdot Q_d(\epsilon', n)$, where $\epsilon' = \epsilon/d$. The same holds with respect to the time complexity. Furthermore, one-sided error is preserved.*

(Note that, for constant d, it holds that $\epsilon' = \Omega(\epsilon)$.) Properties covered by Theorem 1.2 include bipartiteness, cycle-freeness, and all subgraph-freeness and minor-freeness properties. Hence, testers known for these properties in the BDG model (see [Gol17, Chap. 9]) get translated to testers of similar complexity for the subgraph testing model.

While Theorem 1.2 asserts that testing downward-monotone graph properties in the subgraph model is not harder than testing these properties in the BDG model, it gives rise to the question of whether the former task may be easier.

[5] If v has less than i neighbors, then a special symbol is returned. It is sometimes assumed that the algorithm can also query the degree of any vertex of its choice, but this assumption saves at most a multiplicative factor of $\log d$ in the complexity of the algorithm.

Testing in the subgraph model may be trivial. A trivial positive answer holds in case the base graph itself has the property (i.e, $G \in \Pi$). In this case, by the downward-monotonicity of Π, every subgraph of G has the property Π, which means that testing $\Pi \cap \mathcal{F}_G$ in the subgraph model (w.r.t. the base graph G) is trivial.

Testing in the subgraph model may be easier (than in the BDG model). A more interesting case in which testing in the subgraph model may be easier than in the BDG model occurs when the base graph is not in Π, but has some suitable property Π' that is not related to Π. In particular, if the base graph is $\mathcal{M}$-minor free, for some fixed set of graphs $\mathcal{M}$, then, for any downward-monotone graph property Π, testing $\Pi \cap \mathcal{F}_G$ in the subgraph model has complexity that is independent of the size of the tested graph, whereas testing Π in the BDG model may require query complexity that depends on the size of the tested graph. More generally, we consider *hyperfinite* families of graphs [Ele06], where an n-vertex graph G is *t-hyperfinite* for $t : (0,1) \to \mathbb{N}$ if, for every $\epsilon > 0$, removing at most ϵn edges from G results in a graph with no connected component of size exceeding $t(\epsilon)$. We mention that minor-free (bounded-degree) graphs are hyperfinite (with $t(\epsilon) = O(1/\epsilon^2)$).

Theorem 1.3 (on the complexity of testing downward-monotone properties of subgraphs of hyperfinite base graphs (see Section 2.2)): *Let Π be a downward-monotone graph property and $\mathcal{G}$ be a family of t-hyperfinite graphs. Then, for every bounded-degree base graph $G = ([n], E)$ in $\mathcal{G}$, testing whether a subgraph of G satisfies Π can be done in time that depends only on the proximity parameter ϵ. Furthermore, if Π is additive* (i.e., a graph is in Π if and only if all its connected components are in Π)*, then its query complexity is $O(t(\epsilon/4)/\epsilon)$ and the tester has one-sided error.*[6]

Note that by Theorem 1.3, testing bipartiteness of subgraphs of any (bounded-degree) planar graph G has complexity $\mathrm{poly}(1/\epsilon)$, whereas (by [GR02]) testing bipartiteness of n-vertex graphs in the BDG model has complexity $\Omega(\sqrt{n})$.[7]

Testing in the subgraph model may not be easier (than in the BDG model). On the other hand, there are cases in which the testers provided by Theorem 1.2 are essentially the best possible. Indeed, these cases correspond to base graphs that are not hyperfinite.

Theorem 1.4 (testing c-colorability of subgraphs of general bounded-degree base graphs (see Sections 3.1 and 3.2)):

1. *There exist explicit bounded-degree graphs $G = ([n], E)$ such that testing whether a subgraph of G is bipartite, with proximity parameter $1/\mathrm{poly}(\log |E|)$, requires $\widetilde{\Omega}(\sqrt{|E|})$ queries.*
2. *There exist bounded-degree graphs $G = ([n], E)$ such that testing whether a subgraph of G is 3-colorable, with constant proximity parameter, requires $\Omega(|E|)$ queries.*

Item 2 asserts that the complexity of testing 3-Colorability in the subgraph model may be linear, just as in the BDG model. Item 1 should be contrasted with the tester obtained by applying

[6] In general, the tester has two-sided error and the query complexity is at most exponential in $O(t(\epsilon/4)^2)$.

[7] As discussed in Section 1.5.1, weaker results may be obtained by using testers for the BDG model that work under the corresponding promise.

Theorem 1.2 to the known tester for the BDG model [GR99]. The derived tester has complexity $\text{poly}(1/\epsilon) \cdot \widetilde{O}(\sqrt{|E|})$, where ϵ denotes the proximity parameter, whereas Item 1 implies that one cannot obtain better complexity in terms of ϵ and $|E|$; for example, complexity $\text{poly}(1/\epsilon) \cdot Q(|E|)$ is possible only for $Q(n) = \widetilde{\Omega}(\sqrt{n})$. Needless to say, we leave open the question of whether there exist bounded-degree graphs $G = ([n], E)$ such that testing whether a subgraph of G is 2-colorable, *with constant proximity parameter*, requires $\widetilde{\Omega}(\sqrt{n})$ queries (see Problem 3.2).

1.2.2 Other properties (i.e., non downward-monotone properties)

When turning to graph properties that are not downward monotone, the statement of Theorem 1.2 no longer holds. There exist graph properties that are significantly harder to test in the subgraph model than in the BDG model. Specifically:

Theorem 1.5 (testing in the subgraph model may be harder than in the BDG model (see Section 4)): *There exists a property of graphs Π for which the following holds. On the one hand, Π is testable in $O(1/\epsilon)$-time in the bounded-degree graph model. On the other hand, there exist explicit graphs $G = ([n], E)$ of constant degree such that testing whether a subgraph of G satisfies Π requires $\Omega(\log \log n)$ queries. Furthermore, the property Π is upwards monotone, and the family of base graphs is hyperfinite.*[8]

The first part of the furthermore-clause implies that a result analogous to Theorem 1.2 does not hold for upwards monotone (rather than downward-monotone) graph properties, where a graph property is said to be *upwards monotone* if it is preserved under adding edges (i.e., if $G = ([n], E)$ has the property, then so does $G' = ([n], E')$ for every $E' \supset E$). The second part of the furthermore-clause implies that also a result analogous to Theorem 1.3 does not hold for upwards monotone graph properties. While Theorem 1.5 establishes a gap between testing in the subgraph and BDG models, it leave open the question of whether a larger gap can be shown between these two models (see Problem 4.3).

We comment that the property Π used in Theorem 1.5 is related to being Eulerian, and the base graphs are related to a cyclic grid. Hence, it is interesting to note that testing whether subgraphs of a cyclic grid are Eulerian can be done in complexity that only depends on the proximity parameters (see Proposition 4.2).

Turning back to upwards monotone graph properties, we first note the trivial case in which the base graph G does not have the property (which implies that all its subgraphs lack this property as well). A non-trivial case is that of testing minimum degree (see Proposition 2.1). A more interesting case is that of connectivity (which has an $O(1/\epsilon^2)$-query tester in the BDG model [GR02]).

Proposition 1.6 (testing connectivity in the subgraph model – see Section 2.1): *For every base graph $G = ([n], E)$ of maximum degree d, testing whether a subgraph of G is connected can be done in $\text{poly}(d/\epsilon)$-time.*

We mention that even the case of 2-edge connectivity, which has an efficient tester in the BDG model, seems challenging in the subgraph testing model (see Problem 1.9).

[8] See the definition of hyperfinite graphs preceding Theorem 1.3.

A relatively general positive result. We next state a result for a class of properties that are not downward-monotone (and not necessarily upwards monotone either). This result is of the flavor of Theorem 1.2, but introduces an overhead that is logarithmic in the number of vertices. Specifically, we refer to the class of all graph properties that have proximity-oblivious testers of constant query complexity (in the BDG model) [GR11, Sec. 5]. We mention that such properties are "local" in the sense that satisfying them can be expressed as the conjunction of conditions that refer to constant-distance neighborhood in the graph (see Definition 2.4).

Theorem 1.7 (testing local properties in the subgraph model (see Section 2.3)): *Let Π be a local property and suppose that the base graph G is outerplanar*[9] *and of bounded degree. Then, testing whether a subgraph of $G = ([n], E)$ has property Π can be done using $O(\epsilon^{-1} \log n)$ queries.*

The result stated in Theorem 1.7 extends to every graph having constant-size separating sets (the dependence on the size of the separating sets is given explicitly in Theorem 2.5).

Testing in the subgraph model may be easier than in the BDG model. Lastly we note that moving from the BDG model to the subgraph testing model makes the testing task *potentially* easier, since the subgraph tester knows *a priori* the possible locations of edges. This is reflected by the following result, which refers to any (bounded-degree) base graph.

Theorem 1.8 (a property that is extremely easier in the subgraph model): *For every constant d, there exists a graph property Π_d that requires linear query complexity in the bounded-degree model but can be tested using $O(1/\epsilon)$ queries in the subgraph model w.r.t. every base graph of maximum degree d.*

Since the proof of Theorem 1.8 is short and simple, we provide it next.

Proof: Fixing d, let Π_d be a set of d-regular graphs such that testing Π_d in the BDG model (with degree bound d) requires a linear number of queries (e.g., Π_d is the set of 3-colorable d-regular graphs [BOT02]). To establish the upper bound in the subgraph model, observe that for any base graph G that has maximum degree d, the only subgraph of G that may be d-regular is G itself. Therefore, if the base graph G is not in Π_d, then the subgraph-tester can reject without performing any queries. If $G \in \Pi_d$, then the subgraph-tester simply tests whether the subgraph of G is G itself (by performing $O(1/\epsilon)$ queries). ■

The proof of Theorem 1.8 raises the question of whether the upper bound in the theorem holds for downward-monotone properties, and more generally, which properties Π_d can be tested using $O(1/\epsilon)$ queries in the subgraph model *w.r.t. every base graph of maximum degree d*? Alternatively, one may reverse the order of quantifiers and ask whether there exists a graph property Π that satisfies the conclusion of Theorem 1.8 *for any constant d*.

Digest. Assuming that the base graph is of bounded degree, the subgraph testing model and the BDG model differ in two ways. On the one hand, in the subgraph model the tested graph is guaranteed to be a subgraph of a known bounded degree graph, whereas in the BDG model the tested graph is an arbitrary bounded degree graph. This makes testing in the subgraph model

[9] A graph is called outerplanar if it can be drawn in the plane without crossings in such a way that all its vertices belong to the unbounded face of the drawing.

potentially easier, as illustrated by Theorems 1.3 and 1.8. On the other hand, distances in the subgraph model may be significantly larger than in the BDG model, making testing potentially harder as illustrated by Theorem 1.5. The latter phenomenon does not occur when testing a downward-monotone property; this fact is the basis of Theorem 1.2.

1.3 Techniques

Some of the testers presented in this paper (e.g., those referred to in Theorems 1.3 and 1.7) are based on structural properties of the base graph. In some cases (e.g., Theorem 1.3) these structural properties, which are inherited by the subgraphs, make the testing task (in the subgraph model) easier than in the BDG model. The proofs of the lower bounds constitute the more technically challenging part of the paper. Typically, the challenge is emulating lower bounds obtained for other testing models on the subgraph testing model. The brief overviews, especially those referring to the lower bounds, are merely intended to give a flavor of the techniques (and are not supposed to convince the reader of the validity of the claims).

1.3.1 Algorithms

The tester used in proving Theorem 1.2 is a simple emulation of the BDG-model tester by the subgraph tester, and its analysis is based on the observation that the distance between a graph G' and a downward-monotone graph property Π equals the number of edges that should be omitted from G' in order to place the resulting graph in Π. Proposition 1.6 is also proved by a simple emulation of the BDG-model tester, but the analysis of the resulting tester relies on special features of connectivity (and does not extend to 2-connectivity; see Problem 1.9).

The proofs of Theorems 1.3 and 1.7 are more interesting. In both cases we reduce testing subgraphs of the base graph G to testing subgraphs of a fixed subgraph G' of G such that G' is close to G and testing subgraphs of G' is (or seems) relatively easier. Such a reduction is valid since the property that we test is downward monotone, and the subgraph G' is found without making any queries.

In the proof of Theorem 1.3 the fixed subgraph G' consists of small connected components. Hence, in the special case of Theorem 1.3 (i.e., properties that are determined by their connected components), it suffices to test that the subgraphs induced by the connected components of the base graph have the relevant property. In the general case, we follow Newman and Sohler [NS13] in estimating the frequency of the various graphs that are seen in these induced subgraphs. We stress that, unlike in [NS13], we do not use a *partition oracle of the tested graph* (which may be implemented based on standard queries (following Hassidim *et al.* [HKNO09])), but rather determine such a *partition of the base graph* (without making any queries).

Theorem 1.7 is proved by applying a recursive decomposition of the base graph using constant-size separating sets. Essentially, in addition to checking the local neighborhood of random vertices, we also check the local neighborhoods of the vertices in the separating sets that correspond to the path in the recursion tree (i.e., the tree of recursive decomposition) that isolates the chosen vertices. Actually, we check that all these local neighborhoods are consistent with some subgraph that has the property. These additional checks are used in the analysis in order to establish the consistency of the various local neighborhoods (i.e., not only those examined in the same execution).

We highlight the fact that the foregoing testers are non-adaptive (i.e., their queries do not depend on the answers to prior queries). This is remarkable, because the corresponding testers for

the BDG model (which in some cases are actually emulated by our testers) are inherently adaptive. However, these "BDG-model testers" utilize their adaptivity only for conducting local searches in the input graph, whereas in the subgraph testing model the input is a subgraph of a fixed (or *a priori* known) graph, and so the queries that support these local searches can be determined non-adaptively.

1.3.2 Lower bounds

The lower bound on testing 3-colorability of a subgraph (asserted in Part 2 of Theorem 1.4) is proved by observing that the proof of Bogdanov, Obata, and Trevisan [BOT02] asserting that, in the bounded-degree graph model, testing 3-coloring requires linear query complexity can be extended to the subgraph model.[10] This assertion is based on two main observations. The first observation is that, while [BOT02, Thm. 14] asserts a local gap-preserving reduction from 3SAT to 3-Colorability (for bounded degree graphs), *the reduction is actually from a set of 3CNF formulae that have the same clause-structure* (i.e., which variables appear in each of the clauses) *and only differ in the negation-pattern* (i.e., which literal of each variable is used in each of the foregoing occurrences),[11] whose hardness is established in [BOT02, Sec. 6]. The second observation is that, for a fixed clause-structure, the reduction applied in the proof of [BOT02, Thm. 14] can be adapted to produce subgraphs of the same fixed graph. Specifically, the negation-pattern of the given 3CNF determines a sequence of binary choices such that each binary choice determines one edge out of a fixed pair of edges (which is included in the tested subgraph).

The proof of Item 1 of Theorem 1.4 (which refers to testing 2-colorability (bipartiteness) of a subgraph) is more complicated. The basic idea is to emulate the lower bound on bipartite testing established in the BDG model [GR02]. The obvious question is what should be the base graph. It is natural to pick a base graph that allows an embedding of any bounded-degree graph in it such that edges of the embedded graph are mapped to short vertex-disjoint paths. Furthermore, the mapping of edges to paths should be determined in a local manner. We use a base graph that is related to a routing network of logarithmic depth, and employ (randomized) oblivious routing on it. This allows us to map bipartite graphs (of the BDG model) to bipartite subgraphs of the base graph, while mapping graphs that are far from bipartite (in the BDG model) to subgraphs that are far from bipartite (in the subgraph testing model). The actual analysis of this construction is quite complicated (as evident from the length of Section 3.1), because we have to *locally emulate* a subgraph of the base graph (by making few queries to the input graph in the BDG model).

The proof of Theorem 1.5 uses a reduction from testing Eulerian orientations of cyclic grids in the orientation model (defined in Section 1.5.2). As discussed in Section 1.5.2, the orientation model (presented by Halevy *et al.* [HLNT05]) is related to but different from the subgraph testing model. Our reduction maps the (cyclic) grid, used in the lower bound of Fischer *et al.* [FLM+12], to a base graph that looks like such a grid, except that edges are replaced by small gadgets. The orientations of edges in the orientation model are mapped to choices of subgraphs of the corresponding gadgets. In this case, it is easy to locally emulate a subgraph of the base graph by making queries to the orientation oracle, and the claimed $\Omega(\log \log n)$ lower bound follows (from the analogous lower

[10] This replaces a flawed argument, presented in a preliminary version of this work, that supposedly showed a local reduction from the problem of testing whether an input assignment satisfies a fixed 3CNF formula, for which a linear query complexity lower bound was established by Ben-Sasson, Harsha, and Raskhodnikova [BSHR05].

[11] The partition of 3SAT instances to clause-structure versus negation-pattern follows the more general framework of "factor graphs" of CSPs introduced by Feige and Jozeph [FJ12].

bound of [FLM+12]). On the other hand, the property at the image of the reduction is local, and so it is testable within poly$(1/\epsilon)$ queries in the BDG model.

1.4 Some additional open problems

Moving from the BDG model to the subgraph testing model makes the testing task potentially easier, since the subgraph tester knows *a priori* the possible locations of edges. But, when dealing with properties that are not downward monotone, there is an opposite effect that arises from the fact that the distance to the set of subgraphs (of G) that have graph property Π may be much bigger than the distance to the set of (bounded-degree) graphs that have property Π. This may require the subgraph tester to reject the input (since its distance to $\Pi \cap \mathcal{F}_G$ is large), whereas the BDG model tester may be allowed to accept the input (since its distance Π is small). This difficulty is reflected in the following open problems.

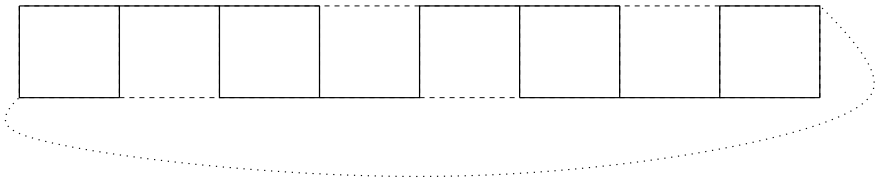

Figure 2: A subgraph of the 2-by-8 grid that misses 4 edges. The subgraph is marked by solid lines, the missing edges by dashed lines, and an external edge that makes this subgraph 2-connected is dotted.

Problem 1.9 (testing 2-connectivity of subgraphs): *Is the query complexity of testing 2-edge-connectivity in the subgraph testing model independent of the size of the graph? What about c-edge-connectivity for any constant $c \in \mathbb{N}$?*

Recall that c-connectivity is testable in the BDG model with complexity that depends only on the proximity parameter [GR02].[12] We note that a straightforward emulation of the BDG-model tester (for 2-connectivity) calls for trying to find a small 2-connected component that has a cut of size at most 1 to the rest of the graph. But this approach fails when considering a base graph that is a 2-by-n grid (since, as illustrated in Figure 2, any subgraph that misses at most one horizontal edge of each vertex (of degree 4) is $O(1/n)$-close to being 2-connected but may be far from any 2-connected subgraph of the 2-by-n grid).

The straightforward emulation of the BDG-model tester also fails for testing whether a subgraph of the n-cycle is a perfect matching (i.e., is 1-regular), but a tester that considers the locations of edges in the subgraph does work (we discuss this shortly at the very end of Section 4). Testers of complexity that does not depend on the graph size do exist for this property when the base graph is a tree (since each tree has at most one perfect matching)[13], but we do not know if they exist when the graph is outerplanar.

[12] The currently best upper-bound is due to [FNY+20].

[13] This perfect matching is determined by a pruning process (started at the leaves), and the tester just checks that the subgraph equals this perfect matching (if it exists). Note that, also in this case, the tester does not emulate the BDG-model tester (which just samples vertices and checks their degree); such an emulation will fail poorly (even when the base graph is a path).

Problem 1.10 (testing whether the subgraph is a perfect matching): *What is the complexity of testing 1-regularity when the base graph is outerplanar? What about the case that the base graph is planar* (e.g., a grid)*? And what about testing degree-regularity?*

Note that c-connectivity, degree-regularity, and Eulerianity are the only properties covered in [Gol17, Chap. 9] that are not downward monotone. Also note that Proposition 4.2 refers to the complexity of testing the Eulerian property for a base graph that is a grid, and it is clear that the underlying ideas apply to base graphs of "similar structure" (as arising in the proof of Proposition 4.2). But what about going beyond that?

Problem 1.11 (testing whether the subgraph is Eulerian): *What is the complexity of testing the Eulerian property in any base graph of bounded degree?*

The foregoing problems are all rooted in the difficulties that are introduced by the fact that distances under the subgraph model may be significantly larger than under the BDG model, which makes the task of the tester potentially harder. On the other hand, the fact that the base graph is known to the tester makes its task potentially easier. Recalling that only the latter effect is relevant in the case of downward monotone properties, gives rise to the following question.

Problem 1.12 (a property that is always easier in the subgraph model): *Does there exist a downward-monotone graph property* Π *such that testing* Π *in the bounded-degree model has higher query complexity than testing* Π *in the subgraph model w.r.t. every base graph of bounded-degree?*

Recall that Theorem 1.8 refers to an upward-monotone property (which depends on the degree bound).

The foregoing problems are aimed at concretizing the abstract challenge of making better use of the knowledge of the base graph that is available to the tester. Although Theorem 1.4 indicates that this extra knowledge is not always helpful, other results point out to cases in which it is helpful. We would like to see more such cases and more substantial use of the knowledge of the base graph.

1.5 Related models

The subgraph testing model is related to two previously studied models: The model of testing graph properties under a promise and the orientation model. These relations are discussed next.

1.5.1 Testing under a promise

As mentioned earlier, testing graph properties under the promise that the tested graph has some (other) property was considered before (see discussion in [Gol17, Sec. 12.2]). In fact, the bounded-degree graph model itself may be viewed as postulating such a promise. More conspicuous cases include the study of testing under the promise that the graph is hyperfinite [NS13] or more specifically planar [BSS10], or with bounded tree-width [EHNO11]. On the other hand, the subgraph testing model is a special case of testing graph properties under a promise, where the promise is that the tested graph is a subgraph of a fixed (base) graph. In continuation to Theorem 1.2, we observe that testing downward-monotone graph properties in the subgraph model can be reduced to testing the same property under a promise that contains the base graph.

Theorem 1.13 (a generalization of Theorem 1.2): *Let $\mathcal{G}$ and Π be downward-monotone graph properties such that $\mathcal{G}$ contains graphs of degree at most d. Suppose that, when promised that the tested graph is in $\mathcal{G}$, the property Π is testable* (in the bounded-degree graph model) *with query complexity $Q_{\mathcal{G}}(\cdot,\cdot)$, where $Q_{\mathcal{G}}$ is a function of the proximity parameter and* (possibly) *the size of the graph. Then, for every base graph $G = ([n], E)$ in $\mathcal{G}$, testing whether a subgraph of G satisfies Π* (with proximity parameter ϵ) *can be done with query complexity $d \cdot Q_{\mathcal{G}}(\epsilon', n)$, where $\epsilon' = \epsilon/d$.*

Hence, results weaker than Theorem 1.3 may be obtained by combining Theorem 1.13 with the tester provided in [NS13] (see discussion in Section 2.2). Indeed, the improved results are due to the fact that in the subgraph model the tester is given the base graph for free. In the current case (of hyperfinite graphs), the tester does not need to query the tested graph in order to obtain a partition oracle of the tested graph; it may just use an adequate partition of the base graph. In general, a main challenge in the study of the subgraph model is in how to utilize the knowledge of the base graph in order to improve the complexity of testing.

1.5.2 The orientation model

The *orientation model*, introduced by Halevy *et al.* [HLNT05], is syntactically related to the subgraph testing model. Similarly to the subgraph model, in the orientation model there is a fixed (undirected) base graph $G = ([n], E)$. However, the goal in the latter model is to test properties of *directed graphs* (digraphs) that are defined by *orientations* of the edges of G. That is, for each edge $\{u, v\} \in E$, either the edge is directed from u to v, or from v to u, and the algorithm may perform queries in order to obtain the orientation of edges of its choice. For a property Π of digraphs, the algorithm should distinguish (with probability at least $2/3$) between the case that the tested orientation $\vec{G}$ has the property Π and the case in which the orientation of more than $\epsilon \cdot |E|$ edges should be flipped in order to obtain the property.

While the subgraph model and the orientation model are syntactically identical, the semantics are very different, as we explain next. Similarly to the subgraph model, an orientation $\vec{G} = ([n], \vec{E})$ of G is defined by a function $f : E \to \{0, 1\}$. Here, $f(e) = 1$ indicates that in $\vec{G}$ the edge e is directed from its smaller endpoint to its larger endpoint (i.e., the edge $\{i, j\}$ is directed from i to j iff $i < j$). Querying the orientation of an edge hence corresponds to querying f, and distance between two functions f and f' (representing two different digraphs) is simply the Hamming distance between the functions.

The fundamental difference in the semantic between the two models is reflected in the fact that natural properties of digraphs in the orientation model do not correspond to natural properties of graphs in the subgraph testing model, and vice versa. For example, the functions f that define Eulerian orientations of an undirected graph $G = ([n], E)$ (as described above) do not necessarily define subgraphs of G (i.e., in which $f(e) = 1$ indicates that e belongs to the subgraph) that are Eulerian. Hence, natural properties in one model do not necessarily translate to natural properties in the other model. Still, it may be possible to emulate or reduce testing properties in one model to testing properties in the other model, as we do in the proof of Theorem 1.5.

1.6 Organization

Following the structure of Section 1.3, we distinguish the presentation of algorithmic results from the presentation of results that have a dominant lower-bound aspect. The former are presented in

Section 2, whereas the latter appear in Sections 3 and 4. Specifically, Section 2 contains the proofs of Theorems 1.2, 1.3, and 1.7 as well as Proposition 1.6. The proof of the two parts of Theorem 1.4 is provided in Section 3, which can be read independently of one another, since they use unrelated techniques. The proof of Theorem 1.5 appears in Section 4.

2 Algorithms

In this section we prove Theorems 1.2, 1.3, and 1.7, as well as Proposition 1.6. These results refer to different types of base graphs and different classes of properties. We have organized them according to the type of the base graph. Recall that G is assumed to have no isolated vertices, so that $|E| \geq n/2$.

2.1 General bounded-degree base graphs

In this section $d \geq 2$ is a fixed constant, and the base graph G is an arbitrary graph in which each vertex has degree at most d (and at least 1).

2.1.1 Testing downward-monotone properties

We first consider any graph property Π that is preserved under edge omission. Such properties are said to be *downward monotone*. We prove Theorem 1.2, which asserts that *for every graph $G = ([n], E)$ of degree at most d and any downward-monotone graph property Π, testing $\Pi \cap \mathcal{F}_G$ in the subgraph model* (w.r.t. the base graph G) *is not harder than testing Π in the bounded-degree graph* (BDG) *model*.

Proof of Theorem 1.2. Given oracle access to $f : E \to \{0,1\}$, the subgraph tester invokes the tester of the BDG model, and emulates an incidence oracle for the subgraph of G represented by f in the natural manner. That is, the query $(v, i) \in [n] \times [d]$ is answered with the i^{th} vertex in the set $\Gamma_f(v) = \{u : \{u, v\} \in E \ \&\ f(u, v) = 1\}$, where vertices are ordered arbitrarily (e.g., by lexicographic order), and the answer is $\perp$ if $|\Gamma_f(v)| < i$. This means that each query (v, i) of the BDG model tester, denoted T, is answered by first retrieving $\Gamma_f(v)$, which in turn amounts to making at most d queries to f (i.e., querying all edges incident to v in G). Hence, the subgraph tester emulates the execution of T on the graph $G^f = ([n], \{e \in E : f(e) = 1\})$.

In the analysis, downward monotonicity is used to associate distance from Π in each of the two models with the number of edges that should be omitted from the subgraph (in order to yield a graph in Π). Specifically, in both cases, the distance from the property reflects the number of edges that should be omitted in order to make the graph satisfy the property (because adding edges never decreases that distance).[14] Specifically, if $f \in \Pi \cap \mathcal{F}_G$, then $G^f \in \Pi$, and T accepts (with probability at least 2/3 in general, and with probability 1 if T has one-sided error). On the other hand, if $f : E \to \{0,1\}$ is ϵ-far from $\Pi \cap \mathcal{F}_G$, then (by downward monotonicity of Π) any graph in Π that is closest to G^f must be a subgraph of G^f (i.e., is in $\Pi \cap \mathcal{F}_G$ and so differs from G^f on more than $\epsilon \cdot |E|$ edges). It follows that G^f is ϵ'-far from Π for $\epsilon' = \frac{\epsilon \cdot |E|}{dn/2} \geq \frac{\epsilon}{d}$. ∎

[14] That is, letting $\Delta(G', G'')$ denote the symmetric difference between the edge-sets of G' and G'', we have $\Delta(G^f, \Pi) = \Delta(G^f, \Pi \cap \mathcal{F}_G)$, where $\Delta(G', \Pi') = \min_{G'' \in \Pi'}\{\Delta(G', G'')\}$.

Proof of Theorem 1.13. The proof is identical to the proof of Theorem 1.2, except that here we rely on the hypothesis that $\mathcal{G}$ is downward monotone in order to infer that any subgraph of G is in $\mathcal{G}$. ■

Tolerant version of Theorem 1.13. Loosely speaking, tolerant testing, introduced in [PRR06], calls for distinguishing objects close to the property from objects that are far from the property. That is, given proximity parameters $\gamma < \epsilon$, one is required to accept (w.h.p.) any object that is γ-close to the property and reject (w.h.p.) any object that is ϵ-far from the property. Observing that distances in the subgraph and BDG models are related by a factor of $\rho \stackrel{\text{def}}{=} \frac{|E|}{dn/2}$, we obtain a reduction of γ-tolerant ϵ-testing whether subgraphs of G are in Π to $\rho\gamma$-tolerant $\rho\epsilon$-testing of Π under the promise $\mathcal{G}$.

2.1.2 Testing properties that are not downward-monotone

Theorem 1.2 does not apply to properties that are not downward-monotone. Still, several such properties are quite easy to test in the subgraph testing model. One simple example is the property of having a specified minimal degree.

Proposition 2.1 (testing minimal degree in the subgraph model): *For $d' \geq 1$, testing whether all vertices in the subgraph have degree at least d' can be done in time $O(d/\epsilon)$.*

Proof: If d' is bigger than the minimum degree of the base graph $G = ([n], E)$, then the tester rejects without performing any queries. Otherwise, the tester selects $\Theta(1/\epsilon)$ vertices, uniformly at random, and computes their degrees in the tested subgraph G^f, by querying all their incident edges in G. The tester accepts if and only if all sampled vertices have degree at least d'.

Hence, the tester makes $O(d/\epsilon)$ queries, and always accepts subgraphs that have the property. To prove that it rejects subgraphs that are ϵ-far from having the property with probability at least $2/3$, we establish the contrapositive statement. Consider a graph G^f that is accepted with probability at least $1/3$. This implies that the number of vertices in G^f whose degree is smaller than d' is at most $(\epsilon/2) \cdot n$. Since in G every vertex has degree at least d', it is possible to add edges to G^f in order to obtain a subgraph that has the property, whereas the number of required added edges is at most $(\epsilon n/2) \cdot d' \leq \epsilon \cdot |E|$. ■

Proof of Proposition 1.6. We now turn to the proof of Proposition 1.6, which asserts the existence of *a* poly(d/ϵ)*-time tester for connectivity in the subgraph model.* If the base graph $G = ([n], E)$ is not connected, then testing is trivial (since all subgraphs of G are disconnected). Otherwise (i.e., the base graph G is connected), connectivity of the input $f \in \mathcal{F}_G$ can be tested by emulating the tester used for the BDG model [GR02]. This tester samples vertices and explores their local neighborhood in search of small connected components.

The analysis is even simpler than the original (bounded-degree) one since we can add edges without worrying about the degree bound (similarly to the analysis of testing connectivity in the sparse (unbounded-degree) model [PR02]). Specifically, we use the fact that if f represents a subgraph with t connected components, then by modifying f at one entry we can obtain a function that represents a subgraph with $t-1$ connected components. (This relies on the fact that G must contain edges between the connected components of G^f.) ■

As noted in the introduction (see Section 1.4), the argument does not extend to 2-connectivity. The reason is that in that case the known tester for the BDG model [GR02] does not search for arbitrary 2-connected components but rather for 2-connected components that are connected to the rest of the graph by at most one edge. The problem with that tester is that its analysis requires the ability to add edges between any given pair of such 2-connected components, whereas we can only add edges that exist in the base graph.

2.2 Hyperfinite base graphs

A graph $G = ([n], E)$ is said to have an (ϵ, t)-*partition* if its vertex set can be partitioned into connected components of size at most t such that the number of edges between these components is at most ϵn.

Recall that a graph M is called a *minor* of a graph G if M is isomorphic to a graph that can be obtained by (zero or more) edge contractions on a subgraph of G. A graph G is M-*minor free* if M is not a minor of G. If G has degree at most d and is minor-free (i.e., G is M-minor free for some fixed subgraph M), then it has an $(\epsilon, O((d/\epsilon)^2))$-partition, for every $\epsilon > 0$ (the size of M is "hidden" in the $O(\cdot)$ notation – see [AST90, HKNO09]).

More generally, Theorem 1.3 refers to any family of hyperfinite graphs, where a family of graph $\mathcal{G}$ is *hyperfinite* if there exists a function $t : (0,1) \to \mathbb{N}$ such that, for every $\epsilon > 0$, every graph in the family has an $(\epsilon, t(\epsilon))$-partition. We shall first prove the second clause in the theorem, which refers to downward-monotone properties that are additive (i.e., determined by the connected components of the graph).

2.2.1 A special case of interest

We say that a graph property Π is *additive* if it holds that a graph is in Π if and only if all its connected components are in Π. We note that if a property is downward monotone and additive, then it is closed under the removal of edges and vertices, but the converse is not necessarily true. In particular this means that not every downward-monotone graph property is additive. For example, consider the graph property Π that consists of all graphs that either constitute of a single (Hamiltonian) cycle or consist of a collection of isolated paths and vertices. Note that Π is closed under omission of edges and vertices, but a graph that consists of several isolated cycles is not in Π (i.e., Π is not additive).[15]

Proposition 2.2 (testing downward-monotone properties that are additive): *Let $\Pi \neq \emptyset$ be a downward-monotone graph property that is additive. Let $G = ([n], E)$ be a graph of maximum degree d, and $t : (0,1) \to \mathbb{N}$ be such that, for every $\epsilon > 0$, the graph G has an $(\epsilon, t(\epsilon))$-partition. Then, testing whether a subgraph of G is in Π can be done by performing $O(d^2 \cdot t(\epsilon/4)/\epsilon)$ queries. Furthermore, the tester is non-adaptive and has one-sided error.*

In particular, Proposition 2.2 implies that, for every fixed graph M, *testing bipartiteness of a subgraph of G, when G is M-minor free, can be done in* $\mathrm{poly}(d/\epsilon)$*-time, when given an $(\epsilon/4, O((d/\epsilon)^2))$-partition of G.*[16] This is much more efficient than testing bipartiteness in the bounded-degree model, for which the query complexity is $\Omega(\sqrt{n})$ [GR02]. It is also more efficient than testing bipartiteness of bounded-degree graphs under the promise that the graph is minor-free, let alone under

[15] Indeed, if a graph is in (this) Π, then all its connected components are in Π, but the converse does not hold.

[16] Such a partition can be found in polynomial-time [AST90].

the weaker promise that the graph is t-hyperfinite. Indeed, under these promises, the tester may implement an $(\epsilon/4, t(\epsilon/4))$-partition oracle of the tested subgraph, but such an implementation requires more than $\text{poly}(t(\epsilon/4))$ queries. Specifically, in the special case of minor-free graphs the best implementation known uses $O(d/\epsilon)^{O(\log(1/\epsilon))}$ queries [LR15], whereas in the general (t-hyperfinite) case the best implementation known uses $\exp(d^{O(t(\text{poly}(1/\epsilon)))})$ queries [HKNO09],

Proof: Let $(C_1, \ldots, C_r)$ be an $(\epsilon/4, t(\epsilon/4))$-partition of G. Given query access to $f : E \to \{0,1\}$, which represents the subgraph $G^f = ([n], \{e \in E : f(e) = 1\})$, we select at random $\Theta(d/\epsilon)$ vertices, and for each selected vertex v we inspect all edges in the subgraph of $G = ([n], E)$ induced by the part C_i that contains v (i.e., we query all pairs $(u, w) \in E \cap (C_i \times C_i)$). We accept if and only if all the observed subgraphs are in Π; that is, we accept if and only if for each inspected C_i it holds that the subgraph of G^f induced by C_i is in Π.

In what follows we use the premise of the proposition that Π is downward monotone and additive, so that it is preserved under the omission of edges and connected components (and hence under the omission of edges and vertices). Observe that if G^f is in Π, then so are the subgraphs of G^f induced by the C_i's. Hence, our tester accepts $G^f \in \Pi$ with probability 1. On the other hand, if G^f is ϵ-far from Π, then the subgraph of G^f obtained by omitting all edges between the C_i's is $(\epsilon/2)$-far from Π (since $(\epsilon/4)n \leq (\epsilon/2)|E|$). Denoting the latter subgraph by $\hat{G}^f$, we claim that at least $(\epsilon/4)n/d$ of its vertices reside in connected components that are not in Π. Conditioned on this claim, and since each connected component of $\hat{G}^f$ corresponds to a subgraph of G^f that is induced by a part C_i, it follows that the tester rejects G^f with probability at least $2/3$ (for an appropriate constant in the $\Theta(\cdot)$ notation for the size of the vertex sample).

Assume, contrary to the claim, that less than $(\epsilon/4)n/d$ vertices reside in connected components of $\hat{G}^f$ that are not in Π. Recall that Π is preserved under the omission of edges and vertices (so that in particular Π contains the graph consisting of a single vertex). Therefore, by omitting at most $\frac{d \cdot (\epsilon/4)n/d}{2} < (\epsilon/2)|E|$ edges (i.e., all edges that belong to connected components that are not in Π), we obtain a graph in which all connected components belong to Π. By additivity of Π, the resulting graph belongs to Π. But this contradicts the hypothesis that $\hat{G}^f$ is $(\epsilon/2)$-far from Π. ■

2.2.2 Greater generality at larger cost

A more general result refers to graph properties that are downward monotone but not necessarily additive, i.e., that are only preserved under edge omissions (and to hyperfinite base graphs G). The cost of this generalization is an increase in the query complexity of the tester, as asserted next.

Proposition 2.3 (testing general downward-monotone properties): *Suppose that Π is a downward-monotone graph property and that, for some $t : [0,1] \to \mathbb{N}$ and every $\epsilon > 0$, the graph $G = ([n], E)$ has an $(\epsilon, t(\epsilon))$-partition. Then, we can test whether a subgraph of G is in Π with query complexity $O(d^2 \cdot \exp(t(\epsilon/4)^2)/\epsilon^2)$.*

We mention that the exponential dependence on t of the query complexity of the foregoing tester is unavoidable (in the general case of downward-monotone graph properties). Consider, for example, the following base graph and downward-monotone property.

- The base graph is an $\sqrt{n}$-by-$\sqrt{n}$ grid augmented by diagonal edges in each grid cell.

  (Denoting the grid vertices by pairs $(i,j) \in [\sqrt{n}] \times [\sqrt{n}]$, the diagonal edges are of the form $\{(i,j), (i+1, j+1)\}$.)

- The downward-monotone property Π is defined as follows. A graph is in Π if there exists a k such that the graph consists of connected components that are each a k-by-k grid augmented by some of the foregoing diagonal edges such that at most half of the possible patterns (created by the diagonal edges) occur in these small grids.

  (A pattern that occurs in such a connected component corresponds to an k^2-bit long string, representing the existence of the corresponding diagonal edges.)[17]

Now, on proximity parameter $\epsilon > 0$, consider the task of distinguishing between the case that the subgraph consists of $0.1/\epsilon$-by-$0.1/\epsilon$ grids in which half of the $N \stackrel{\text{def}}{=} \Theta(2^{(0.1/\epsilon)^2})$ possible patterns occur and the case in which all N patterns occur. A lower bound of $\Omega(\sqrt{N})$ follows by a birthday paradox argument, whereas $N = \exp(\Omega(t(\epsilon)^2)$.[18]

Proof: By the premise of the proposition, for every $\epsilon > 0$, the base graph G has an $(\epsilon/4, t(\epsilon/4))$-partition. Let $g \in \mathcal{F}_G$ denote the all-ones function (i.e., $G^g = G$), and let g' be $\epsilon/2$-close to g such that g' describes a subgraph $G^{g'}$ of G in which each connected component has size at most $t(\epsilon/4)$. Hence, $G^{g'}$ is a subgraph of G that is obtained from G by removing the at most $(\epsilon/4)n \leq (\epsilon/2)|E|$ edges between parts in the $(\epsilon/4, t(\epsilon/4))$-partition of G.

By the closure of Π to edge omissions, each function $f \in \mathcal{F}_G \cap \Pi$ is $(\epsilon/2)$-close to the function $f' \in \mathcal{F}_G \cap \Pi$ such that $f'(e) = f(e) \wedge g'(e)$. Let Π'_G denote the set of graphs obtained in this way; that is, $\Pi'_G = \{f \wedge g' : f \in \mathcal{F}_G \cap \Pi\}$. Since Π is a graph property, it follows that $\Pi'_G = \mathcal{F}_G \cap \Pi'$, where Π' is the set of all graphs that are isomorphic to graphs that belong to Π'_G. Hence, the set Π'_G is closed under all automorphisms of the graph G.

Recalling that Π'_G and likewise Π' contain only graphs that consist of connected components of size at most $t = t(\epsilon/4)$, it follows that Π' is characterized by the frequencies in which the various graphs of size at most t appear as connected components. Hence, $f \in \mathcal{F}_G$ describes a graph in Π if and only if $f' = f \wedge g'$ is in $\mathcal{F}_G \cap \Pi'$, where Π' is characterized in terms of the number of connected components that are isomorphic to each of the graphs with at most $t(\epsilon/4)$ vertices (and contain no smaller connected components). It follows that testing with proximity parameter ϵ whether subgraphs of G satisfy Π can be performed by estimating these numbers in the subgraph described by $f \wedge g'$, where f is the tested function.

Lastly, we note that estimating the frequencies in which the various $t(\epsilon/4)$-vertex graphs appear as connected components can be done using $O(d^2 \cdot \exp(t(\epsilon/4)^2)/\epsilon^2)$ queries, where the term $\exp(t(\epsilon/4)^2)$ accounts for the number of $t(\epsilon/4)$-vertex graphs. (This is analogous to learning a distribution that has support size $\exp(t(\epsilon/4)^2)$.) ■

2.3 Local properties and base graphs with small separators

Loosely speaking, a graph property is called *local* if satisfying it can be expressed as the conjunction of local conditions, where each local condition refers to a constant-distance neighborhood of one of the graph's vertices. (For example, for every fixed graph H, being H-free is a local property.) A precise definition is given next.

[17] Here we ignore the $O(1)$ automorphisms of a k-by-k grid. Taking these automorphisms into account, we only have $\Omega(2^{k^2})$ patterns.

[18] Note that by using [VV17, Val12] we can obtain a lower bound of $\widetilde{\Omega}(N)$, but this improvement is insignificant here since we merely aim at a lower bound of the form $\exp(\Omega(1/\epsilon^2))$.

Definition 2.4 (local property): *For a constant $\ell \in \mathbb{N}$, the ℓ-*neighborhood *of a vertex v in a graph G is the subgraph of G induced by all vertices that are at distance at most ℓ from v. A property Π of n-vertex graphs is called ℓ-*local *if there exists a graph property Π' such that G is in Π if and only if the ℓ-neighborhood of each vertex in G is in Π'.* (Actually, Π' is a set of rooted graphs, where the root corresponds to the "center" of the ℓ-neighborhood.)[19] *A graph property $\Pi = \bigcup_n \Pi_n$ is* local *if there exists a constant ℓ such that Π_n is an ℓ-local property of n-vertex graphs.*

We mention that this definition coincides with [GR11, Def. 5.2], and that (in the bounded degree graph model) every graph property that has a proximity-oblivious tester of constant query complexity is local [GR11, Sec. 5].

For $s : \mathbb{N} \to \mathbb{N}$ we say that a graph $G = ([n], E)$ has *separating sets of size s* if for every set of vertices $U \subseteq [n]$ there exists a subset $S \subseteq U$ of at most $s(|U|)$ vertices such that the subgraph of G induced by $U \setminus S$ has no connected component of size greater than $\frac{2}{3} \cdot |U|$. For example, every tree has separating sets of size 1, every outerplanar graph has separating sets of size 2 [Hea87, Lem. 3], and n-vertex planar graphs have separating sets of size $O(\sqrt{n})$ [LT79].

Theorem 2.5 (Theorem 1.7, generalized): *Let Π be an ℓ-local property and let $s : \mathbb{N} \to \mathbb{N}$ be such that $s(n) < n/\log_2^2 n$. Suppose that the base graph G is of bounded degree d and has separators of size s. Then, testing whether a subgraph of $G = ([n], E)$ has property Π can be done by performing $O(\epsilon^{-1} s(n) \log n \cdot d^{\ell+1})$ queries. Furthermore, the tester is non-adaptive and has one-sided error.*

Proof: We consider a recursive decomposition of the graph G, obtained by applying the guaranteed separators, and a tree that corresponds to these applications. Specifically, the root of the tree corresponds to the separating set, denoted S_λ, that disconnects the graph $G_\lambda \stackrel{\text{def}}{=} G$. Collecting the resulting connected components into two subgraphs, each containing at most two-thirds of G's vertices, we proceed to obtain separating sets, denoted S_0 and S_1, for each of these two subgraphs, denoted G_0 and G_1, respectively. In general, an internal node in the tree is labeled by a string α and corresponds to the subgraph G_α as well as to a separating set S_α for G_α. The children of this node correspond to subgraphs $G_{\alpha 0}$ and $G_{\alpha 1}$ that result from removing S_α from G_α (where the number of vertices in each of these subgraphs is at most two-thirds of the number of vertices in G_α). When the subgraph reaches some constant size, the process stops. Hence, the leaves of the tree correspond to subgraphs of constant size. For a leaf labeled by α, we let S_α be the set of vertices of the subgraph G_α.

For the sake of clarity, we reserve the term 'node' for nodes in the tree (describing the recursive decomposition), and the term 'vertex' for the vertices of G. We shall never talk of edges of the (rooted) tree, but only of the descendance and ancestry relations induced by it. Recall that each node in the tree is associated with a set of vertices of G, and note that these sets form a partition of the vertex set of G. We say that vertex v *resides* in a node labeled by α if $v \in S_\alpha$. Observe that edges of the graph G can connect vertices that reside in the same node and vertices that reside in nodes that are in an ancestry relation, but *cannot* connect vertices that reside in nodes that are not in an ancestry relation (equiv., reside in nodes $\alpha' 0 \alpha''$ and $\alpha' 1 \alpha'''$ for any $\alpha', \alpha'', \alpha'' \in \{0,1\}^*$).

We are now ready to describe the tester for Π, which is an ℓ-local property for some constant $\ell \in \mathbb{N}$. Given a fixed based graph $G = ([n], E)$ and oracle access to a subgraph represented by $f : E \to \{0,1\}$, the tester repeats the following procedure $\Theta(d/\epsilon)$ times, where if no invocation of the procedure causes rejection, then it accepts.

[19] Marking the root is important only in case that the center of the graph of radius ℓ cannot be uniquely determined.

1. Uniformly select a vertex that resides in one of the leaves of the decomposition tree. Recalling that $s(n) < n/\log^2 n$, it follows that a constant fraction of the vertices of G resides in leaves of the tree.[20]

2. For each vertex v of G that resides in a node on the path from the selected leaf to the root (including both the leaf and the root), explore the ℓ-neighborhood of v in G (i.e., query f on each of the edges in that neighborhood).

3. If the subgraph discovered in the previous step is not consistent with any n-vertex subgraph of G that has property Π, then reject.

   Note that the aforementioned discovered subgraph includes not only the explored edges but also indication that certain edges do not exist in the subgraph (i.e., the latter include all non-edges of G as well as some edges of G that were queried by the procedure and answered by the value 0).

The query complexity of this procedure is $O(s(n)\log n \cdot d^\ell)$, where d is the degree-bound of G. Clearly, the tester always accepts subgraphs of G that have the property Π. It remains to show that if the subgraph is ϵ-far from Π, then the probability that a single invocation of the procedure causes rejection is $\Omega(\epsilon/d)$.

We establish the contrapositive statement. Suppose that the foregoing procedure rejects with probability $\rho < 1$. We show that it suffices to modify an $O(\rho \cdot d)$ fraction of the edges in G in order to obtain a graph that satisfies Π. We say that a leaf of the tree is *good* if the procedure does not reject when it selects a vertex that resides in this leaf. We say that an internal node of the tree is *good* if it appears on the path from some good leaf to the root. Note that $\rho < 1$ implies that there exist good leaves, and hence the root of the tree is good. More generally, if a node is good, then all its ancestors are good. Also note that each vertex that resides in a good node has an ℓ-neighborhood in G^f that satisfies the local condition (i.e., the ℓ-neighborhood is in Π'), where recall that G^f denotes the subgraph of G defined by f.

Hence, we only need to modify the neighborhoods of vertices residing in bad nodes, and we should do so without harming the neighborhoods of vertices that reside in good nodes. But before explaining how this is done, we note that the number of vertices that reside in internal nodes belonging to the subtree rooted in node α is only a constant factor larger than the number of vertices that reside in the leaves of this subtree. On the other hand, considering the set of bad nodes that have good parents, we note that ρ equals the fraction of vertices that reside in leaves of the subtrees rooted at these bad nodes.

Consider an arbitrary bad node, denoted $\alpha\sigma$, that has a good parent, denoted α. Then, the ℓ-neighborhoods of the vertices residing in node α satisfy the local condition (in the subgraph G^f). We claim that the ℓ-neighborhoods of vertices in $G_{\alpha\sigma}$ can be modified so that they satisfy the local conditions as well without modifying the ℓ-neighborhoods of any vertex that resides in a good node. To verify this claim observe the intersection of the ℓ-neighborhoods of vertices in $G_{\alpha\sigma}$ and the ℓ-neighborhoods of vertices that reside in good nodes is contained in the intersection of the ℓ-neighborhoods of vertices in $G_{\alpha\sigma}$ and the ℓ-neighborhoods of vertices that reside either in node α or in one of its ancestors. The reasoning is that if vertex v in $G_{\alpha\sigma}$ is adjacent in G to a vertex u,

[20] Denoting the vertex set of G_α by V_α, we use the fact that S_α contains at most an $s(|V_\alpha|)/|V_\alpha| < \log_2^{-2}|V_\alpha|$ fraction of V_α, whereas $|V_{\alpha\sigma}| \leq \frac{3}{2} \cdot |V_\alpha|$ (for every $\sigma \in \{0,1\}$) and $\prod_{i>O(1)}\left(1-\frac{1}{i^2}\right) > 0.99$.

then either u is in $G_{\alpha\sigma}$ or u is in $S_{\alpha'}$ such that α' is a (not necessarily proper) prefix of α (in other words, u cannot be in G_β, where β has a prefix that is not a prefix of α).[21]

Recall that by Item 3 of the procedure (based on which the notion of good node is defined) the fact that node α is good, implies that the ℓ-neighborhoods of vertices in $G_{\alpha\sigma}$ can be modified to satisfy Π' in a manner that is consistent with the ℓ-neighborhoods of all vertices that reside in node α and its ancestors, and so with the ℓ-neighborhoods of all vertices that reside in good nodes. It follows that by modifying f on $G_{\alpha\sigma}$, while maintaining the ℓ-neighborhoods of vertices in S_α (as well as $S_{\alpha'}$ for each α' that is an ancestor of α) intact, we can "fix" the ℓ-local neighborhood of all vertices in $G_{\alpha\sigma}$.

The foregoing process modifies f into a function that describes a subgraph of G that is in Π, while modifying $O(\rho \cdot d \cdot n) = O(\rho \cdot d \cdot |E|)$ edges. The theorem follows. ■

3 Testing in the subgraph model may not be easier than in the BDG model

As observed in Theorem 1.2, testing downward-monotone graph properties in the subgraph model (w.r.t. any bounded-degree base graph) can be reduced to testing the same property in the BDG model. Here we show that there exist base graphs for which the result obtained by the reduction cannot be significantly improved. Specifically, we prove Theorem 1.4, which refers to the complexity of testing c-colorability (of n-vertex graphs) in the subgraph model, for $c = 2$ and $c = 3$.

The case of $c = 2$ is proved in Section 3.1, and the case of $c = 3$ is proved in Section 3.2. The proof presented in Section 3.1 is far more complex than the one in Section 3.2. The results proved are incomparable: The lower bound presented in Section 3.1 (for the case of $c = 2$) refers to a testing problem of intermediate complexity (i.e., $\widetilde{\Theta}(\sqrt{n})$) in the bounded-degree model, and it refers to an explicit base graph, but only to proximity parameter value of $1/\mathrm{poly}(\log n)$). In contrast, the lower bound presented in Section 3.2 (for the case of $c = 3$) refers to a testing problem of extreme complexity (i.e., $\Omega(n)$) in the bounded-degree model, and it refers to a constant value of the proximity parameter, but to a non-explicit base graph.

3.1 Testing bipartiteness

In this section we prove Part 1 of Theorem 1.4, which is restated next.

Theorem 3.1 (testing bipartiteness in the subgraph model): *There exist explicit graphs $G = ([n], E)$ of constant degree such that testing with proximity parameter $1/\mathrm{poly}(\log n)$ whether a subgraph of G is bipartite requires $\widetilde{\Omega}(\sqrt{n})$ queries.*

Recalling that testing bipartiteness of n-vertex graphs in the BDG model with distance parameter ϵ can be done in $\mathrm{poly}(1/\epsilon) \cdot \widetilde{O}(\sqrt{n})$-time [GR99], it follows that the bipartite tester for the subgraph model obtained by invoking Theorem 1.2 is the best possible (for the case that the proximity parameter equals $1/\mathrm{poly}(\log n)$).

We establish Theorem 3.1 in several stages, as detailed in the following subsections. For the ease of readability, some of the stages present simpler, preliminary constructions and arguments, which are then corrected/refined in later stages.

[21] See the end of the second paragraph of the proof.

The underlying strategy is to *reduce* the problem of testing bipartiteness in the bounded-degree model to testing the same property in the subgraph model, and to apply the lower bound established in [GR02]. Due to some technical difficulties, it is simpler to *emulate*, in the subgraph model, the lower bound of [GR02] (or rather of [KKR04]), which refers to testing bipartiteness in the bounded-degree model.[22] The key idea is to embed an arbitrary bounded-degree graph as a subgraph of a routing network such that edges of the original graph are represented by vertex-disjoint paths in the routing network.

3.1.1 A lower bound construction for testing generalized 2-colorability

We start by recalling the lower bound of [GR02, KKR04]; actually, we shall present a small variation on the original argument. Specifically, we consider a *generalized notion of 2-colorability*, where the edges of the graph are labeled by $\neq$ and $=$, and the 2-coloring has to satisfy the corresponding constraint. That is, if the edge is labeled $\neq$ (resp., $=$), then its endpoints should be assigned opposite (resp., equal) colors. An algorithm for testing this property (of edge-labeled graphs), receives the label of each queried edge (in addition to the unknown endpoint). (We mention that testing this generalized property can be locally reduced to testing bipartiteness by replacing $=$-labeled edges with paths of length two; hence, the testing problem is no harder than testing bipartiteness.) From this point on we refer to this generalized notion of 2-colorability simply as 2-colorability.[23]

We consider two distributions over d-regular n-vertex graphs with such labeling, where we may use any constant $d \geq 3$. In both distributions a graph is selected by combining d random perfect matchings (while allowing parallel edges). The two distributions differ by the edge labels.

1. In the first distribution the edge labels are selected uniformly at random.

2. In the second distribution the edge labels are determined by selecting a random 2-partitioning of the n vertices and setting the edge-labels so that this 2-partition is a valid 2-coloring; that is, if both endpoints are assigned to the same part (resp., to different parts), then the edge is labeled $=$ (resp., $\neq$).

By the definition of the second distribution, its support consists of edge-labeled graphs that are all 2-colorable. The arguments in [GR02, KKR04] can be readily adapted to show that, with overwhelmingly high probability, the edge-labeled graphs of the first distribution are $\Omega(1)$-far from being 2-colorable, Furthermore, similarly to what is shown in [GR02, KKR04], an algorithm that makes q queries to a graph drawn from one of the two distributions can distinguish between the two cases with probability $O(q^2/n)$. The reasoning is that the answers to these queries are identically distributed as long as no cycle is observed (and the latter event occurs with probability $O(q^2/n)$).

3.1.2 The subgraph-model construction: using a routing network

Turning to the emulation in the subgraph model, we first describe the base graph G that we use. To be precise, we present an initial construction that will later be refined. Let $g : [n] \times [d] \to [n]$ be the incidence function of the graph that we wish to emulate (i.e., a graph drawn according to the

[22] That is, we do not reduce testing bipartiteness in the BDG model to testing bipartiteness in the subgraph model, but rather reduce solving a specific distributional problem in the BDG model to solving a (related) problem in the subgraph model.

[23] When considering the distance to generalized 2-coloring, it does not matter if one allows (and counts) only edge omission, or allows (and counts) also/only edge-label modifications.

foregoing distribution over graphs) and let $L : [n] \times [d] \to \{\neq, =\}$ denote the labeling of its edges (i.e., as chosen according to one of the foregoing distributions over edge labels). Actually, it will be more convenient to use $\{1, 2\}$ instead of $\{\neq, =\}$, where 1 corresponds to $\neq$; this allows treating an edge labeled $\sigma \in \{1, 2\}$ as a σ-long path. We assume, w.l.o.g., that $g(g(v, i), i) = v$; that is, we use the i^{th} "port" of each vertex for connecting the edge of the i^{th} matching (i.e., if an edge between v and u is selected in the i^{th} random matching (for $i \in [d]$), then the edge connects port i of v to port i of u (such that $g(v, i) = u$ and $g(u, i) = v$)).

The base graph. The base graph G that we use is *related* to a Beneš routing network [Ben65] with $n \cdot d$ sources and $n \cdot d$ sinks. The main idea is to represent the $n \cdot d$ edges of g by paths in such a routing network. The network supports "randomized oblivious routing" (see below) from the first layer (of sources) to the last layer (of sinks). In addition, there is a special "zero layer" of size n, corresponding to the vertices of G. The vertices in this layer are connected both to the first layer of the routing network and to the last layer (so as to allow edges in the emulated graph to correspond to paths in the routing network). Details follow.

Let $\ell = \lceil \log_2(nd) \rceil$, and consider a fixed injective mapping $\texttt{bin}$ from $[n] \times [d]$ to $\{0, 1\}^\ell$. The network, denoted R_ℓ, has $2\ell + 1$ layers. For each layer $j \in [2\ell + 1]$, and $\alpha \in \{0, 1\}^\ell$, there is an associated vertex (j, α) (belonging to layer j). The edges between the layers are defined as follows. For each $j \in [2\ell]$ and $\alpha \in \{0, 1\}^\ell$, there is an edge between (j, α) and $(j + 1, \alpha)$. In addition, for $j \in [\ell]$ there is also an edge between (j, α) and $(j + 1, \alpha \oplus e_j)$ where $e_j = 0^{j-1}10^{\ell-j}$, and for $j \in [\ell + 1, 2\ell]$ there is an edge between (j, α) and $(j + 1, \alpha \oplus e'_j)$, where $e'_j = 0^{j-\ell-1}10^{2\ell-j}$. These edges are called *routing edges*.

In addition, the base graph G has a *zero layer*, which consists of n vertices. Each vertex $v \in [n]$ in this layer is connected to d distinct vertices in layer 1, and to d distinct vertices in layer $2\ell + 1$. Specifically, each $v \in [n]$ in the zero layer is connected by paths of length two (with distinct intermediate vertices) to the vertices $(1, \texttt{bin}(v, 1)), \ldots, (1, \texttt{bin}(v, d))$ in layer 1, and is connected by *edge-label gadgets* to the vertices $(2\ell + 1, \texttt{bin}(v, 1)), \ldots, (2\ell + 1, \texttt{bin}(v, d))$ in layer $2\ell + 1$. Each edge-label gadget consists of a direct edge and a path of length two. For an illustration of the construction – see Figure 3.

In what follows, we shall use the shorthand 2-*path* for a length-2 path. We stress that all the 2-paths, both for connecting layer zero to layer 1 and for connecting layer $2\ell + 1$ to layer zero, use distinct intermediate vertices.

Randomized routing. For each $v \in [n]$ and $i \in [d]$, we shall route $(1, \texttt{bin}(v, i))$ to $(2\ell + 1, \texttt{bin}(g(v, i), i))$, by selecting $r \in \{0, 1\}^\ell$ uniformly at random, and using the unique path that leads from $(1, \texttt{bin}(v, i))$ to $(\ell + 1, r)$ and from $(\ell + 1, r)$ to $(2\ell + 1, \texttt{bin}(g(v, i), i))$. This path is defined as follows. For $j \in [\ell]$ (resp., $j \in [\ell + 1, 2\ell]$) in the j^{th} step, if the current vertex in the path is (j, α), then we take the edge to $(j + 1, \alpha')$, where $\alpha' = \alpha \oplus 0^{j-1}r_j0^{\ell-j}$ (resp., $\alpha' = \alpha \oplus 0^{\ell-j-1}r_j0^{2\ell-j}$). Hence, the edge $\{v, g(v, i)\}$ is mapped to a 2ℓ-long path that leads from $(1, \texttt{bin}(v, i))$ to $(2\ell + 1, \texttt{bin}(g(v, i), i))$ (via $(\ell + 1, r)$), called a *routing path*.

We augment this routing path by a 2-path leading from vertex v in the zero layer to $(1, \texttt{bin}(v, i))$ and by the adequate part of the edge-label gadget that connects $(2\ell + 1, \texttt{bin}(g(v, i), i))$ to $g(v, i)$; that is, if $L(v, i) = 1$, then we use the corresponding edge, and otherwise we use the 2-path, which means that we always use an $L(v, i)$-long path. Combining the routing path with these edges, we obtain a $(2 + 2\ell + L(v, i))$-long path from v to $g(v, i)$ (both residing in layer 0), called an *augmented*

routing path. For an illustration of such an augmented routing path, see Figure 3.

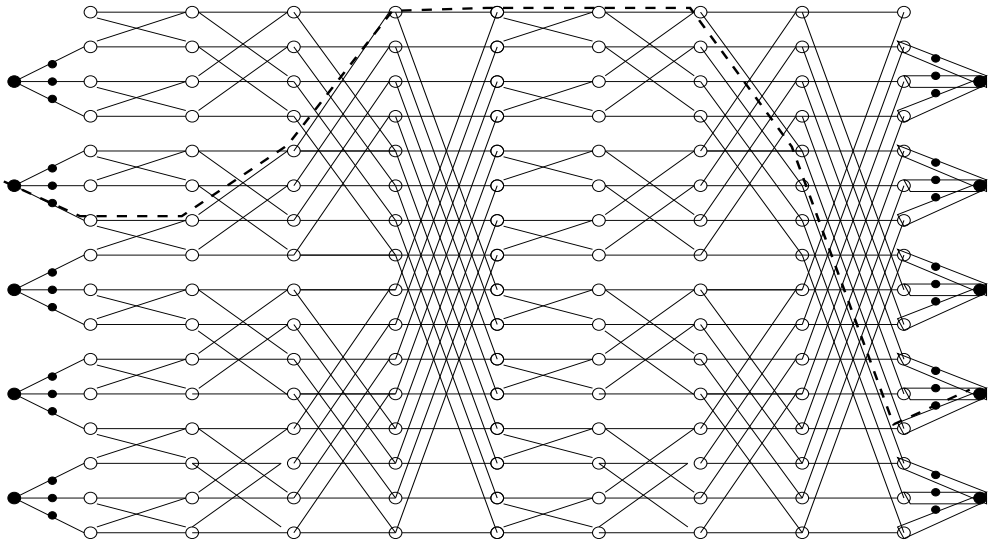

Figure 3: The routing network for $\ell = 4$ and its augmentation for $n = 5$ and $d = 3$ (i.e., $g : [5] \times [3] \to [5]$). The zero layer is drawn twice, once on the left and once on the right. The vertices in this layer are connected by 2-paths to vertices in layer one, and by edge-label gadgets (a direct edge and a 2-path) to layer $2\ell + 1 = 9$. The wide dashed line depicts an augmented routing-path from vertex 2 to vertex $4 = g(2, 3)$ (via the top vertex of the middle layer).

Now, suppose that we select such a random routing path for each $(v, i) \in [n] \times [d]$, and condition on the event that these routing paths are vertex-disjoint (which is highly unlikely to be the case).[24] Relying on the vertex-disjointness of the routing-paths, it follows that if (g, L) is 2-colorable, then the subgraph that consists of all the augmented routing paths is bipartite. This relies on the fact that a legal 2-coloring of an augmented routing path from v to $g(v, i)$, which has length $2 + 2\ell + L(v, i)$, assigns these two vertices colors that satisfy the (generalized) 2-coloring condition. On the other hand, if (g, L) is not 2-colorable, then the subgraph consisting of all the augmented routing paths is not bipartite. Furthermore, relying on the edge-disjointness of the routing paths, any 2-coloring of the (vertices of the) latter subgraph that has t monochromatic edges, yields a 2-coloring of g with at most t violating edges (i.e., edges that violate the constraints of L). This is the case because a legal 2-coloring of the augmented routing path from v to $g(v, i)$, which has length $2 + 2\ell + L(v, i)$, yields a legal 2-coloring of the edge $\{v, g(v, i)\}$ with respect to L.

3.1.3 Refining the construction and removing the congestion

The problem with the foregoing description is that the randomized routing (suggested by Valiant [Val82, VB81]) is unlikely to be congestion-free; that is, the random routing paths are unlikely to be vertex-disjoint (as long as intermediate layers have $o(n^2)$ vertices). Nevertheless, with very high probability (e.g., with probability at least $1 - 2^{-10\ell}$), randomized routing has congestion $O(\ell)$, where the *congestion* is the maximum number of routing-paths that use a single routing-vertex.

[24] We shall deal with the collisions later on, while capitalizing on the fact that their number can be bounded.

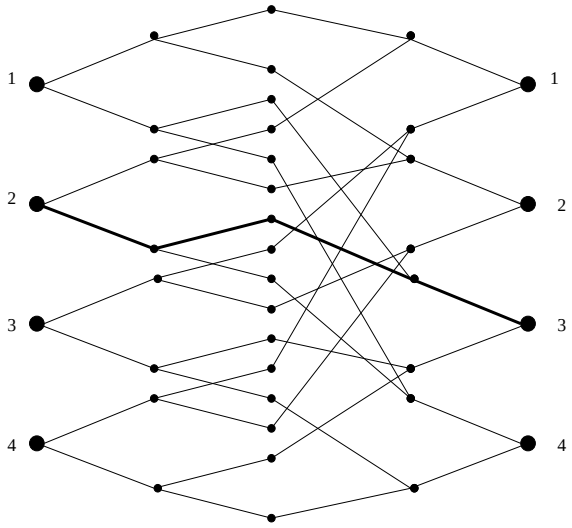

Figure 4: The quadratic routing network for $\ell' = 4$. The wide line depicts a routing-path from source 2 (on the left) to sink 3 (on the right). Recall that this gadget replaces a routing-edge (and connects the corresponding clouds).

To get rid of this congestion, we replace each vertex of the routing network R_ℓ by a *cloud* (independent set) of $\ell' = \Omega(\ell)$ vertices; we shall set $\ell' = \mathrm{poly}(\ell)$, where ℓ' is a power of 2, in order to facilitate the analysis. (We stress that we replace the vertices at layers $1, \ldots, 2\ell + 1$, but the vertices of layer zero remain intact.)[25]

It would have been simplest to replace each routing-edge of the network R_ℓ by a complete bipartite graphs between the corresponding clouds (of size ℓ' each), but we aim at having a constant-degree graph. Hence, we connect each pair of clouds by a routing network of size quadratic in ℓ' such that this (sub-)network supports deterministic oblivious routing between any ℓ' sources and ℓ' sinks. Specifically, we construct such a network by using $2\ell'$ balanced binary trees, each with ℓ' leaves. We use a tree rooted at each source, and a tree rooted at each sink such that the trees are disjoint except that, for every source $i \in [\ell']$ and sink $j \in [\ell']$, the j^{th} leaf of the former tree is identified with the i^{th} leaf of the latter tree. For an illustration, see Figure 4.

The vertices of the zero layer of the base graph remain intact, and we connect them to each of the d corresponding clouds in the first and last layer by using balanced binary trees as above. Specifically, when connecting vertex $v \in [n]$ in the zero layer to the cloud that replaces $(1, \mathtt{bin}(v, i))$ (for $i \in [d]$), we use a binary tree rooted at v whose leaves are the vertices in the cloud corresponding to $(1, \mathtt{bin}(v, i))$. Likewise, when connecting the cloud $(2\ell+1, \mathtt{bin}(u, i))$ to vertex u in the zero layer we use a binary tree with leaves in $(2\ell + 1, \mathtt{bin}(u, i))$, and connect the root of this tree to u by the edge-label gadget described above. (We stress that the latter binary tree is not rooted at u but rather at a distinct auxiliary vertex, and that this auxiliary vertex is connected to u by an edge-label gadget, which consist of a direct edge and a two-edge path running in parallel to it.)[26] This results in our final base graph, denoted G.

[25] In an alternative construction, the vertices of layer 1 and layer $2\ell + 1$ also remain intact (and only the vertices of layers 2 through 2ℓ are replaced by clouds of size ℓ' (as above)).

[26] Hence, letting $h = \log_2 \ell'$, the total length of the (shortest) path from v to the auxiliary vertex connected to $g(v, i)$ by an edge-label gadget is $h + 2\ell \cdot 2h + h$.

Note that G has $O(\ell \cdot dn \cdot (\ell')^2) = \widetilde{O}(n)$ vertices, since the routing network has $2\ell \cdot dn \cdot 2$ routing edges and each routing edge is replace by a small routing (sub-)network having $O((\ell')^2)$ vertices. Each vertex in G has degree $O(d)$, where vertices of layer zero have degree $d \cdot 2 + d \cdot 2$ and all other vertices have degree at most 4.

A central observation is that any routing with congestion at most ℓ' (on R_ℓ) can be mapped to a set of vertex-disjoint paths in G. Actually, the mapping can be selected at random (but not obliviously). Specifically, given a routing on R_ℓ, we assign to the different routing-paths that pass through each vertex in R_ℓ, distinct vertices in the corresponding cloud. The specific assignments are selected at random (i.e., uniformly conditioned on distinctness).[27] Hence, the routing-paths are assigned disjoint sequences of vertices (in the various clouds), and the actual paths in the graph G are uniquely defined by using the corresponding paths on the corresponding trees.

3.1.4 A precise definition of the distribution over subgraphs

Let us spell out the distribution over subgraphs (of the base graph G) that is associated with the graph represented by g and L. First, we select a random routing of the edges of g (i.e., routing $(1, \mathtt{bin}(v,i))$ to $(2\ell+1, \mathtt{bin}(g(v,i),i))$ for each $(v,i) \in [n] \times [d]$), obtaining $n \cdot d \leq 2^\ell$ routing-paths that go from the first layer to the last layer. We assume that this set of routing-paths has congestion at most ℓ' (otherwise the process is aborted). Next, we select a random sequence of vertices for each of these routing-paths such that a single vertex is chosen in each cloud of each routing-path and each vertex is chosen for at most one routing-path.

Specifically, let $h = \lceil \log_2 \ell' \rceil$ denote the height of the binary trees connecting the clouds. For a routing-path $(1, \alpha^{(1)}), \ldots, (2\ell+1, \alpha^{(2\ell+1)})$, a selection of vertices $(w^{(1)}, \ldots, w^{(2\ell+1)})$ in the corresponding clouds yields a path of length $2\ell \cdot 2h$. This path consists of the unique paths through the corresponding pairs of binary trees that connect $w^{(i)}$ to $w^{(i+1)}$, for every $i \in [2\ell]$. This defines a set of $n \cdot d$ vertex-disjoint paths from the first layer to the last layer such that, for every $(v,i) \in [n] \times [d]$, a distinct vertex of the cloud $(1, \mathtt{bin}(v,i))$ is connected by a concatenation of paths (through binary trees) to a distinct vertex of the cloud $(2\ell+1, \mathtt{bin}(g(v,i),i))$. We call these paths *actual routing paths*.

Next, we augment these paths by using the relevant edges that connect them to the zero layer. Specifically, for every $(v,i) \in [n] \times [d]$, we pick the tree-path connecting v (in layer zero) to the vertex in layer 1 that is used to route (v,i), and the tree-path connecting the relevant vertex in layer $2\ell+1$ to the vertex $g(v,i)$ in the zero layer. In the latter tree-path (or rather when moving from the root of this tree to $g(v,i)$), we pick the adequate part of the corresponding edge-label gadget; that is, we pick the $L(v,i)$-path of this gadget. We call these edges the *L-selected gadget edges*, and refer to the path going from v to $g(v,i)$ (via a vertex of cloud $(1, \mathtt{bin}(v,i))$ and a vertex of cloud $(2\ell+1, \mathtt{bin}(g(v,i),i))$) as a *full routing path*. That is, a full routing path consists of an actual routing path between clouds 1 and $2\ell+1$ and paths that connect its endpoint to corresponding vertices in layer zero. The random subgraph contains a collection of full routing paths that corresponds to the randomized routing of g, where these paths are vertex-disjoint except for their endpoints (which are all at layer zero).

[27] Indeed, the routing on G is not oblivious, since the vertices used in each cloud are selected in a dependent manner. However, the various routing-paths in R_ℓ are independent of one another. Ditto for the choices of the vertex-sets used for routing in the different clouds. These facts will be used in our analysis.

Distance from being bipartite. Note that if the graph defined by g is 2-colorable (in a generalized sense) with respect to the labeling L, then the subgraph defined by the foregoing full routing paths is bipartite. This is the case since the foregoing paths are vertex-disjoint (except for their endpoints which are in layer zero), and the full routing path from v to $g(v,i)$ has length $h + 4\ell \cdot h + h + L(v,i) = (4\ell+2) \cdot h + L(v,i)$, Recalling that the edge $\{v, g(v,i)\}$ satisfies the corresponding L-constraint (i.e., v and $g(v,i)$ are assigned the same color if and only if $L(v,i) = 2$), we can use the generalized 2-coloring of g to obtain a 2-coloring of our subgraph. On the other hand, if g is ϵ-far from being 2-colorable with respect to L in the bounded-degree model, then the latter subgraph is $(\epsilon/\mathrm{poly}(\ell))$-far from being bipartite. This relies on the foregoing observation by which the number of L-violating edges in a 2-coloring of g is upper-bounded by the number of monochromatic edges in the best 2-coloring of the subgraph of G that corresponds to g and L, while recalling that the number of edges in G is $\mathrm{poly}(\ell)$ times larger than in g.

The final distributions. So far we have described the distribution of subgraphs associated with each instance (g, L) of the 2-coloring problem. Combining this distribution with the two distributions defined in Section 3.1.1, we obtain two distributions of subgraphs that differ only in the distribution of edges chosen in the edge-label gadgets. Note that the subgraphs of the second distribution are bipartite, whereas (with overwhelmingly high probability) the subgraphs of the first distribution are $(1/\mathrm{poly}(\ell))$-far from being bipartite.

3.1.5 A process for answering queries according to the distribution on subgraphs

Our goal is to show that any algorithm that makes less than $\sqrt{n}/c$ queries (for a sufficiently large constant c) to a random subgraph selected from one of the two distributions cannot distinguish (with sufficiently high constant probability) between the case that the subgraph is drawn from the first distribution and the case that it is drawn from the second distribution. Following [GR02, KKR04], we observe that as long as the algorithm observes no cycle in the subgraph, the two distributions look identical, since the subgraphs differ only in the part of the edge-label gadgets used and these parts reveal no information about the identity of the distribution used unless a cycle is formed.[28] Hence, we will focus on upper-bounding the probability that such a cycle is observed.

Intuitively, a cycle in the subgraph corresponds to a sequence of full routing-paths, which in turn correspond to edges of the underlying graph defined by g (i.e., the graph consisting of d random perfect matching).[29] Our plan is to construct, on-the-fly and in response to queries of the algorithm, a random subgraph (according to each of the two distributions) and show that $\Omega(\sqrt{n})$ queries are required in order to observe a cycle in this subgraph. A first attempt proceeds as follows.

We start by selecting upfront a single vertex in each cloud of layer 1 and of layer $2\ell+1$, and connecting them to the corresponding vertex of the zero layer, while revealing all corresponding tree-paths (not including the choices for the part of the edge-label gadgets in use). Hence, we may assume that the algorithm never queries these tree-paths (which do not include the edge-label gadgets). Queries to edges of the edge-label gadgets are handled by selecting uniformly at random which part of the queried gadget to use, and answering accordingly. (Recall that as long as no cycle

[28] In this case, determining the selected edge-label gadgets according to a random 2-coloring yields the same distribution as determining these edge-label gadgets uniformly at random. As in [GR02, KKR04], the distributions in question are fixed, but we describe processes that generate them on-the-fly in response to querioes of the algorithm.

[29] Indeed, a cycle in the subgraph yields a cycle in the underlying graph defined by g, but the converse does not necessarily hold since the corresponding edge-gadget were not necessarily queried.

is seen, this choice is consistent with both distributions.) Hence, our focus is on the queries of the algorithm that refer to edges in the trees that correspond to routing-edges.

Consider such a generic query to an edge e in some tree that corresponds to a routing-edge, denoted $\{(j,\alpha),(j+1,\alpha')\}$, where $j \in [2\ell]$. If we have already determined whether or not e is in the subgraph, then we answer accordingly. Otherwise, we proceed as follows, where we assume that $j \in [\ell]$, while treating the case of $j \in [\ell+1, 2\ell]$ analogously (as detailed later).

1. Conditioned on the random choices made so far, we decide at random whether or not e is in the emulated subgraph. We stress that, here and in the sequel, such conditional random choices refer to the marginal distribution of the choice in question.

2. We answer the query according to the value determined in Step 1, and continues to the following steps if and only if the answer is 1 (i.e., the edge e is in the subgraph).

3. Conditioned on the random choices made so far, we select at random a routing-path that uses the routing-edge $\{(j,\alpha),(j+1,\alpha')\}$. This is done as follows.

   (a) We select at random an unused cloud in layer 1 (equiv., a pair $(v,i) \in [n] \times [d]$ such that the value of g at (v,i) is still undetermined) and a random (not necessarily unused) cloud in layer $\ell+1$, denoted $(\ell+1,r)$, such that the routing-path from $(1,\mathtt{bin}(v,i))$ to $(\ell+1,r)$ passes through the routing-edge $\{(j,\alpha),(j+1,\alpha')\}$.

   (b) We select at random a vertex $u \in [n]$ such that the value of g at (u,i) is still undetermined, and set $g(v,i)=u$ and $g(u,i)=v$.

   The selected routing-path is the unique routing-path that goes from $(1,\mathtt{bin}(v,i))$ to $(2\ell+1,\mathtt{bin}(u,i))$, while passing through $(\ell+1,r)$.

4. Conditioned on the random choices made so far, we select a random path of actual edges (in G) that is consistent with the selected routing-path and uses the edge e. That is, we select, for each cloud on the routing-path, a random vertex that was not used by previous routing-paths such that the path determined by the choice for the j^{th} and $j+1^{\text{st}}$ clouds passes through the edge e. Specifically:

   (a) For every $p \in [2,2\ell] \setminus \{j, j+1\}$, we select $w^{(p)}$ uniformly among all unused vertices that reside in the p^{th} cloud determined in Step 3.

   (b) We set $w^{(j)}$ in the j^{th} cloud and $w^{(j+1)}$ in the $j+1^{\text{st}}$ cloud such that the edge e resides on the unique $2h$-long path that leads from $w^{(j)}$ to $w^{(j+1)}$; that is, if e is on the path that leads to the τ^{th} leaf of the σ^{th} tree, then we use the σ^{th} vertex in the j^{th} cloud and the τ^{th} vertex in the $j+1^{\text{st}}$ cloud. (Note that in case $j=1$, the choice of $w^{(j)}$ is consistent with the choice of vertices for the clouds of layer 1 (i.e., by the very fact that we reached the current step).)

   The choice of these vertices determines an actual path from layer 1 to layer $2\ell+1$, and this path avoids the vertices used in prior paths.

Although the random choices made in Steps 3 and 4 are not revealed to the actual algorithm, we consider them as if they were revealed since we condition on them later. (Indeed, the reader may consider the case that the actual paths that correspond to the selected routing-path are revealed to

the algorithm for free.) The case of $j \in [\ell+1, 2\ell]$ is handled analogously, where the routing-path is selected by first selecting a path between the middle layer and the last layer, and determining a cloud in the first layer later.

Consider the process of responding to the first query, which is an edge that corresponds to a routing-edge between two neighboring clouds. Observe that a positive answer to this query results in selecting uniformly a routing-path that goes through this routing-edge. It is tempting to think that an analogous statement holds also with respect to subsequent queries, except that routing-paths that connect used endpoints (in layers 1 and $2\ell+1$) are now avoided. Unfortunately, this is not accurate, since the fact that a routing-edge was used by previous routing-paths conditions its use in subsequent routing-paths (because the number of routing-paths that go through a cloud is bounded). Likewise and actually more acutely, negative answers to previous queries (i.e., determining that certain edges are not in the subgraph) also conditions the subsequent choices of routing-paths. At the extreme, if the algorithm queries all edges that correspond to a specific routing-edge, then this routing-edge cannot be used by any subsequent routing-path (regardless of the answers provided). Lastly, the foregoing conditioning of the choice of the routing-paths does condition the choice of the ("routed") edges of g. Consequently, wishing to treat the edges of g as if they are selected uniformly requires bounding the effect of the aforementioned conditioning.

3.1.6 Revising the process for answering queries

In light of the above, we revise the process of constructing the subgraph on-the-fly. The key observation is that determining the use of an actual edge (both in case it is not in the subgraph and in case it is used for some determined routing-path) eliminates its *a priori* potential use by other routing-paths. Specifically, the determinion of an edge that corresponds to a routing-edge from layer $j \in [\ell]$ to layer $j+1$ restricts each of the $2^{\ell-j}$ clouds of the middle layer that could have used it for routing to any of 2^{j-1} clouds of the first layer. (Ditto for $j \in [\ell+1, 2\ell]$ and restrictions on clouds of the last layer.) This restriction becomes significant if many actual edges that correspond to the same routing-edge were determined, but the restriction is not so significant otherwise (i.e., when only few edges that correspond to this routing-edge were determined).

Fixing a threshold $t = \Theta(\ell^3)$, we say that a routing-edge is *problematic* if more than t of its actual edges were determined. Note that actual edges are determined not only when they are queried explicitly but also when they are chosen for a routing-path (see Step 4 in Section 3.1.5). Hence, each query may increase the number of determined edges by $\widetilde{O}(\ell)$ units (since each of the 2ℓ routing-edge uses $2\log_2 \ell' = O(\log \ell)$ actual edges for the actual routing of a path through the two corresponding binary trees).

Each problematic edge contributes $2^{\ell-1}$ *restrictions*, which correspond to the $2^{\ell-1}$ possible routing-paths that go through this routing-edge in the half of R_ℓ to which it belongs. These restrictions are "charged" to the middle layer so that each cloud in that layer is charged with all routing-paths that reach it after passing through a problematic routing-edge. Specifically, a problematic edge from layer $j \in [\ell]$ to layer $j+1$ contributes 2^{j-1} restrictions to each of the $2^{\ell-j}$ clouds of the middle layer that have a routing-path that passes through this routing-edge; these 2^{j-1} restrictions correspond to the clouds in the first layer that are reachable via such routing-paths. Analogous restrictions arise from problematic edges in the second part of R_ℓ (i.e., connecting layer j and layer $j+1$ for $j \in [\ell+1, 2\ell]$). Hence, the sum of restrictions that apply to a cloud in the middle layer is due to both the first and the last layers. Next, we define the *restriction level* of a cloud in the middle layer as the sum of all restrictions it accumulates from problematic edges, and

consider such a level to be *high* if it exceeds $\eta \cdot 2^\ell$, where $\eta > 0$ is a sufficiently small constant (e.g., $\eta = 0.001$).

We *augment the process of constructing the subgraph on-the-fly* (described in Section 3.1.5) so that, after answering each query, we determine routing-paths for all clouds that reached a high restriction level. Specifically, conditioned on the random choices made so far, for each cloud that reached a high level, we determine the number of routing-paths that go through this cloud, the routing-paths themselves, and the actual paths (as in Step 4) that correspond to them. (Effectively, this determines also all the edges of the tree that correspond to the routing-edges incident to this cloud.)

We stress that we treat the clouds that reach a high restriction level iteratively, and that this treatment may cause additional clouds to reach a high restriction level. The number of clouds that may reach a high level due to the response to a single query may be very large (and we upper-bound it in Section 3.1.7). Yet, with overwhelmingly high probability, throughout the entire iterative process (of answering a query), the number of actual edges that are determined in each relevant routing-edge (i.e., a routing-edge that is not incident to a cloud that reaches a high level) is $O(\ell)$. This is the case because, with overwhelmingly high probability, the number of routing-paths that pass through any routing edge is $O(\ell)$. It follows that routing-edges that became problematic during the interactive process are not "really problematic" since the number of actual edges determined in each of them only exceeds the threshold $t = \Theta(\ell^3)$ by $O(\ell)$. The same holds for the routing-edges that became problematic due to the last query; actually, the number of actual edges that are determined in each of these problematic routing-edges is exactly $t + 1$ (since the move from a non-problematic state to problematic state is due to a single path). Consequently, when we select a random routing-path, at least one of its endpoints (i.e., the endpoint that is on the side opposite to the currently handled cloud) is almost uniformly distributed in $[n]$ (just as would be the case if there would be no restrictions at all). This is the case because (as shown next) the number of clouds that reach a high restriction level is smaller that the number of queries, and since routing-edges that are not really problematic behave almost as if none of their actual edges was determined. Details follow.

3.1.7 Completing the lower bound

Turning to the analysis of the revised construction, we first note that when answering q queries we directly determined $q \cdot \widetilde{O}(\ell)$ actual edges, but in addition we might have determined additional edges due to the selection of routing-paths for clouds that exceeded the restriction level of $\eta \cdot 2^\ell$, where each such cloud may determine $O(\ell)$ routing-paths (which determine $\widetilde{O}(\ell)$ edges each). Denoting the set of clouds that reached a high level by X, we claim that

$$|X| \;<\; \frac{q \cdot \widetilde{O}(\ell) + |X| \cdot \widetilde{O}(\ell^2)}{t} \cdot \frac{2^{\ell-1}}{\eta 2^\ell} \;. \tag{1}$$

The first factor represents an upper bound on the number of problematic routing-edges (to be established next), $2^{\ell-1}$ upper-bounds the number of restrictions introduced by each problematic edge, and $\eta \cdot 2^\ell$ lower-bounds the restriction level that causes a cloud to be included in X. As for the justification of the upper bound on the number of problematic routing-edges, the numerator accounts for the $\widetilde{O}(\ell)$ edges that are determined directly by each query as well as the number of edges determined by handling clouds in X. (Recall that the latter handling amounts to determining $O(\ell)$ routing-paths, and each determines $\widetilde{O}(\ell)$ actual edges.) Indeed, the edges that are determined

not to be in the subgraph when dealing with a cloud of high restriction level are not counted, since the corresponding cloud was already counted in X. Turning back to Equation (1) and using $t \gg \widetilde{O}(\ell^2)/\eta$, we get $|X| \ll q$.

We now turn to the analysis of the distribution of routing-paths used when answering individual queries and handling the clouds that reach a high restriction level. Note that the number of such routing-paths is at most $q+|X|\cdot O(\ell)$, since each query yields at most a single routing-path, whereas a cloud that reaches a high level yields $O(\ell)$ routing-paths. Hence, at most $q + |X| \cdot O(\ell)$ clouds in the first (resp., last) layer are ruled out (by previous routing-paths), and the other clouds have almost all their routing-paths intact. This is the case because a cloud (in the middle layer) that has restriction level below $\eta \cdot 2^\ell$ can reach at least $1 - \eta$ fraction of the first (and last) layer clouds by routing-paths that have no problematic routing-edges. (Recall that clouds (in the middle level) that reach a high level during the process of answering a single query may have additional problematic routing-edges, but these edges are not really problematic towards the analysis that follows.) Hence, conditioned on the choice of the first endpoint (for the routing-path), the second endpoint is selected with probability that is $(\eta' + (1-\eta')\eta'')$-close to uniform, where $\eta' = \frac{q+|X|\cdot O(\ell)}{n} + \eta$ represents the fraction of discarded clouds, and $\eta'' = 1 - \left(1 - \frac{t+O(\ell)}{\ell'}\right)^\ell$ represents the deviation caused by routing-edges that are not really problematic (i.e., have less than $t+O(\ell)$ determined edges). Lastly, observe that $\eta' = o(1) + \eta$ and $\eta'' = 1 - (1 - o(1/\ell))^\ell = o(1)$.

To summarize, we answered q queries by revealing to the querying algorithm at most $q + |X| \cdot O(\ell) = O(q\ell)$ routing-paths, where each route reveals an edge of the underlying random graph g. Each revealed edge is determined by first determining a cloud in the middle layer, and then determining a random cloud in the last layer (assuming that the edge in the first half of R_ℓ; otherwise a random cloud in the first layer is revealed). As shown above, the random choice of the latter cloud is almost uniform, which means that the corresponding value of g is selected almost uniformly. It follows that, when answering q queries, a cycle is formed with probability $O(q^2/n)$. This completes the proof of Theorem 3.1.

3.1.8 An open problem

Recalling that Theorem 3.1 only lower-bounds the query complexity of testing whether a subgraph is bipartite in the case that the proximity parameter is the reciprocal of a polylogarithmic function (in the size of the base graph). We believe that the same lower bound holds also for constant values of the proximity parameter, but the current strategy used in proving Theorem 3.1 fails to prove it. For starters, we propose the following challenge.

Problem 3.2 (more on testing bipartiteness in the subgraph model): *Prove that there exist explicit graphs $G = ([n], E)$ of constant degree such that testing,* with proximity parameter $O(1)$*, whether a subgraph of G is bipartite requires $n^{\Omega(1)}$ queries.*

3.2 Testing 3-Colorability

In this section we prove Part 2 of Theorem 1.4, which is restated next.

Theorem 3.3 (testing 3-Colorability in the subgraph model): *There exist graphs $G = ([n], E)$ of constant degree such that testing whether a subgraph of G is 3-colorable for a constant ϵ requires $\Omega(n)$ queries.*

Theorem 3.3 is proved by observing that the proof of Bogdanov, Obata, and Trevisan [BOT02] asserting that, in the bounded-degree graph model, testing 3-coloring requires linear query complexity can be extended to the subgraph model.[30] This assertion is based on two main observations.

1. The first observation is that, while [BOT02, Thm. 14] asserts a local gap-preserving reduction from 3SAT to 3-Colorability (for bounded degree graphs), *the reduction is actually from a set of 3CNF formulae that have the same clause-structure* (i.e., which variables appear in each of the clauses) *and only differ in the negation-pattern* (i.e., which literal of each variable is used in each of the foregoing occurrences),[31] whose hardness is established in [BOT02, Sec. 6].

2. The second observation is that, for a fixed clause-structure, the reduction applied in the proof of [BOT02, Thm. 14] can be adapted to produce subgraphs of the same fixed graph. Specifically, the negation-pattern of the given 3CNF determines a sequence of binary choices such that each binary choice determine one edge out of a fixed pair of edges (which is included in the tested subgraph).

We mention that the reduction used in the proof of [BOT02, Thm. 14] is a variant of a rather standard approximation-preserving reduction (of Petrank [Pet94]), and we will present yet another variant of it.

In continuation to the foregoing discussion, we define a massively parameterized problem that refers to testing the satisfiability of a 3CNF formula that is represented by a negation-pattern to be applied to a fixed clause-structure. That is, the massive parameter, which is fixed per each length, is a function of the form $f : [m] \times [3] \to [n]$, and the input is a negation parameter of the form $p : [m] \times [3] \to \{0,1\}$ such that the *3CNF formula specified by f and p* consists of m clauses such that for every $j \in [m]$ and $k \in [3]$ the variable $f(j,k)$ appears unnegated (resp., negated) as the k^{th} literal of clause j if $p(j,k) = 0$ (resp., $p(j,k) = 1$). Following [BOT02], we focus on the case that $m = \Theta(n)$, and furthermore on the case that each variable appears in a constant number of clauses (i.e., $|\{(j,k) : f(j,k) = i\}| = O(1)$ for each $i \in [n]$).

Claim **3.3.1** (implicit in [BOT02, Sec. 6]): *For some universal constant c the following holds. For every $n \in \mathbb{N}$, there exists a function $f : [m] \times [3] \to [n]$ such that*

1. *For each $i \in [n]$, it holds that $|\{(j,k) \in [m] \times [3] : f(j,k) = i\}| \leq c$.*

2. *Given query access to $p : [m] \times [3] \to \{0,1\}$, distinguishing* (with success probability 2/3) *between the case that the formula specified by f and p is satisfiable and the case that any assignment satisfies less that 90% of the clauses, requires $\Omega(n)$ queries to p.*

[30] This replaces a flawed argument, presented in a preliminary version of this work, that supposedly showed a local reduction from the problem of testing whether an input assignment satisfies a fixed 3CNF formula, for which a linear query complexity lower bound was established by Ben-Sasson, Harsha, and Raskhodnikova [BSHR05].

[31] The partition of 3SAT instances to clause-structure versus negation-pattern follows the more general framework of "factor graphs" of CSPs introduced by Feige and Jozeph [FJ12]. Specifically, the factor graph of a CSP instance determines which variables appear in each of the constraints, and [FJ12] consider fixing such a factor graph for each input length, where the input itself only determines which predicate (in a fixed family) is applied in each constraint. In case of 3SAT, the factor graph corresponds to the clause-structure and the actual input only determines the negation-pattern. We mention that Feige and Jozeph [FJ12] showed that approximating Max3SAT (to within a certain constant factor), for some fixed factor graphs, is NP-Hard.

For the sake of completeness, we next give the high level idea of the proof of Claim 3.3.1. The main step is establishing an analogous result for 3LIN, which refers to linear equations over GF(2) with three variables in each equation and each variable occurring in a constant number of equations. Here we consider a fixed m-by-n matrix A over GF(2), with three 1-entries per row and a constant number of 1-entries per each column. For a fixed matrix A, given query access to a vector $b \in \mathrm{GF}(2)^m$, the testing question refers to whether $Ax = b$ has a solution. By [BOT02, Lem. 19], there exist (explicit) matrices A such that distinguishing the case that $Ax = b$ is solvable and the case that any assignment satisfies at most 51% of the equations requires making $\Omega(n)$ queries. As noted in [BOT02], using the standard gadget reduction of 3LIN to 3SAT, the claim follows.[32]

Proof of Theorem 3.3: In order to prove the theorem, we take a closer look at the rather standard approximation-preserving reduction of 3SAT to 3-Colorability, when applied to the problem referred to in Claim 3.3.1. This reduction is merely a small variant of the standard reduction of 3SAT to 3-Colorability (cf. [Gol08, Prop. 2.27]), which uses gadgets for each clause and each variable. The point (which is fully formalized below), is that these gadgets are fixed, and only the connections between them depend on the input 3CNF formula. Furthermore, if the k^{th} literal in clause j contains an occurrence of variable $f(j,k)$, then the j^{th} clause-gadget is connected to the $f(j,k)^{\text{th}}$ variable-gadget, independently of the value of the negation pattern p. The value of $p(j,k)$ determines only to which of the two vertices in the variable-gadget we connect the relevant edge (i.e., the k^{th} outgoing edge of the clause-gadget). Hence, we place all (actually both) possibilities in the base graph (i.e., connect to both vertices), and make the actual choice in the subgraph of the base graph. Details follow.

The base graph G_f and its subgraphs. For a fixed clause-structure $f : [m] \times [3] \to [n]$, we consider the base graph G_f that is obtained by a slight variant of the standard reduction of 3SAT to 3-Colorability, which will contain a subgraph (denoted G_f^p) that corresponds to each negation-pattern $p : [m] \times [3] \to \{0,1\}$. We shall then show that if the 3CNF formula specified by f and p is satisfiable, then G_f^p is 3-colorable, whereas if every assignment to this formula fails to satisfy a certain constant fraction of its clauses, then G_f^p is far from being 3-colorable. The base graph G_f consists of a tri-partite graph G' with $s = \max(n,m)$ vertices on each of the three sides, a gadget per each of the n variables of the formula, a gadget per each of the m clauses of this formula, and edges connecting some of these components, as described in detail next.

- We call the three sets in the tri-partite graph G', `ground`, `true` and `false`, and denote the vertices in them by $\{\mathtt{g}_\ell\}_{\ell=1}^s$, $\{\mathtt{t}_\ell\}_{\ell=1}^s$ and $\{\mathtt{f}_\ell\}_{\ell=1}^s$, respectively. Each pair of these sets is connected by a regular bipartite expander graph of constant degree. Specifically, these bipartite expander graphs satisfy the *mixing property* with (constant) error ϵ_0 which means that the fraction of edges that connect any two subsets equals the product of the densities of these subsets up-to a deviation of ϵ_0. (Namely, letting $V_1 =$ `ground`, $V_2 =$ `true` and $V_3 =$ `false`, for any two subsets $S \subset V_k$ and $T \subset V_{k'}$ (where $k \neq k' \in [3]$), we have that

[32] Specifically, each equation is replaced by four out of the eight possible clauses that refer to the corresponding variables, where the choice of negation pattern is determined by the corresponding bit in b (i.e., if the j^{th} equation is $x_{a(j,1)} + x_{a(j,2)} + x_{a(j,3)} = b_j$, where $a(j,k)$ is the k^{th} 1-entry in row j of A, then we set $f(4j+d,k) = a(j,k)$ for each $d \in [4]$ and determine $p(4j+d,k)$ according to b_j only). (For example, if $b_j = 1$, then we use the clauses $x_{a(j,1)} \vee x_{a(j,2)} \vee x_{a(j,3)}$, $\neg x_{a(j,1)} \vee \neg x_{a(j,2)} \vee x_{a(j,3)}$, $x_{a(j,1)} \vee \neg x_{a(j,2)} \vee \neg x_{a(j,3)}$, and $\neg x_{a(j,1)} \vee x_{a(j,2)} \vee \neg x_{a(j,3)}$, and otherwise we use the four remaining clauses.) Hence, for a fixed A and varying b, we obtain a fixed clause-structure f and varying negation-pattern p. Lastly, note that if an assignment violates at least 49% of the equations, then it does not satisfy at least $0.49/4 > 0.1$ of the clauses in the corresponding 3CNF.

$\left|\frac{|E(S,T)|}{|E(V_k,V_{k'})|} - \frac{|S|}{|V_k|}\cdot\frac{|T|}{|V_{k'}|}\right| \leq \epsilon_0$, where $E(X,Y)$ denotes the set of edges between a pair of subsets X and Y.) The constant ϵ_0 will be picked to be sufficiently small, and this will mean that the expander will be $\mathrm{poly}(1/\epsilon_0)$-regular. Furthermore, at least half of the edges of G_f reside in these three expanders.

All subgraphs of G_f that we shall consider contain all the foregoing (expander) edges. This implies that in any 3-coloring of the vertices of G_f^p that has few monochromatic edges, a large majority of the vertices in each of these three sets will be assigned a distinct color, which may be thought of as having the name of this set. We note that the reason for introducing these sets is that the graphs we construct must have a bounded (constant) degree. If we didn't have this constraint, then each of these sets could be replaced by a single vertex (with degree (at most) s).

- The gadget associated with i^{th} variable (i.e., x_i) consists of a pair of vertices, associated with the two literals of this variable (i.e., x_i and $\neg x_i$) and an edge that connects these pair of vertices. We also refer to the corresponding vertices as x_i and $\neg x_i$, respectively. In addition, each of these two vertices is connected to a corresponding vertex in `ground`; that is, for each $i \in [n]$, both x_i and $\neg x_i$ are connected to $\mathtt{g}_i$.

  All subgraphs of G_f that we shall consider contain all the foregoing edges. This implies that for any legal 3-coloring of G_f^p in which $\mathtt{g}_i$ is colored `ground`, if x_i (resp., $\neg x_i$) is colored `true`, then its neighbor $\neg x_i$ (resp., x_i) is colored `false`.

- The gadget associated with the j^{th} clause contains six designated vertices such that one of these vertices, called the *head* vertex and denoted by $\mathtt{h}_j$, is connected by edges to $\mathtt{g}_j$ and $\mathtt{f}_j$. (Indeed, vertex $\mathtt{g}_\ell$ in the `ground` set is connected to both the ℓ^{th} variable-gadget (if such exists) and the ℓ^{th} clause-gadget (if such exists).) In addition, the j^{th} clause-gadget is connected by edges to the six vertices, called its *terminals*. These vertices belong to the variable-gadgets that are associated with the literals that may appear in the j^{th} clause (i.e., the two literals of each of the variables $x_{f(j,1)}$, $x_{f(j,2)}$ and $x_{f(j,3)}$). Recall that the clause-structure f only determines the variables that appear in the clause, whereas the actual corresponding literals are determined by the negation-pattern p.

  All subgraphs that we shall consider contain all the edges of the gadget and three of the edges going to its terminals. Specifically, the subgraph G_f^p will contain edges going from the j^{th} clause-gadget to the three literals indicated by the three pairs $(f(j,1),p(j,1))$, $(f(j,2),p(j,2))$ and $(f(j,3),p(j,3))$. For an illustration, see Figure 5.

  The clause-gadget has the following property, when it appears as part of a subgraph G_f^p. In each legal 3-coloring of the vertices of the j^{th} clause-gadget in which $\mathtt{h}_j$ is colored `true`, at least one of the three terminals to which it is connected in G_f^p is colored `true`. This is the case since, as detailed in the caption of Figure 5, a legal 3-coloring of the gadget in which these three terminals are colored `false` forces the head vertex to be colored `false`. On the other hand (also as detailed in the caption of Figure 5), for any coloring of these three terminals in which at least one of them is colored `true`, there exists a legal 3-coloring of the vertices of the clause-gadget in which $\mathtt{h}_j$ is colored `true`.

Using the fact that each variable in the formula described by f and any p occurs in a constant number of clauses, it follows that G_f has $O(s) = O(n+m)$ vertices and constant degree.

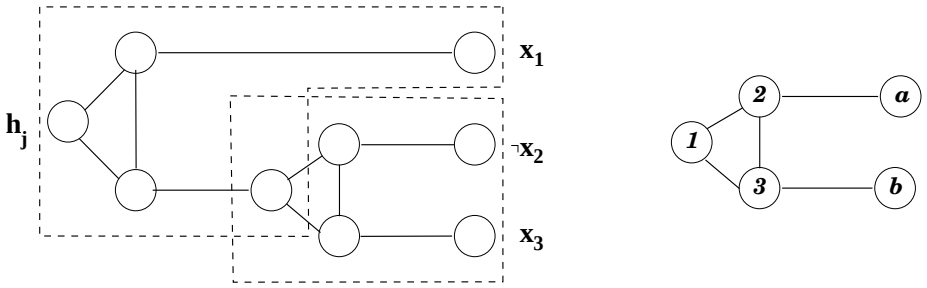

Figure 5: The clause-gadget and its connections to terminal vertices. The l.h.s depicts a clause-gadget that is connected to the vertices (terminals) $x_1, \neg x_2$ and x_3 (that belong to the vertex gadgets of variables 1, 2 and 3, respectively). Note that in G_f there are connections to all six vertices: $x_1, \neg x_1, x_2, \neg x_2, x_3, \neg x_3$, and here we depict only the edges in G_f^p. The head vertex $\mathtt{h}_j$ is indicated, and the gadget itself (together with its terminals) combines two "sub-gadgets" that share one vertex. The sub-gadget and a generic legal 3-coloring of it are depicted on the r.h.s, where this generic coloring uses the colors $1, 2, 3$ and generic $a, b \in \{1, 2, 3\}$. Note that if $a = b$, then $a = b = 1$ must hold. This implies that for any legal 3-coloring of the clause-gadget and its terminals, it holds that if the three terminals of the gadget are assigned the same color, c, then the head vertex is also assigned the color c. On the other hand, it is always possible to set $a = 1$ (by setting $b \in \{2, 3\}$), and so for every $k \in [3]$ there exists a legal 3-coloring of the clause-gadget that assigns x_k the same color as the head vertex.

The local reduction. For a fixed clause-structure f, we spell out the mapping of the negation-pattern p to a subgraph of the base graph G_f. The subgraph, denoted G_f^p, contains all the edges of the large bipartite expanders as well as all edges of all the gadgets. In addition, it contains both edges connecting each variable-gadget to the corresponding vertex in `ground`, and three of the edges connecting each clause-gadget to its terminals. Specifically, for each $j \in [m]$ and $k \in [3]$, the subgraph G_f^p contains the edge going from the j^{th} clause-gadget to the literal indicated by the pair $(f(j,k), p(j,k))$ (i.e., it is connected to $x_{f(j,k)}$ if $p(j,k) = 0$, and to $\neg x_{f(j,k)}$ otherwise).

This mapping is local in the sense that each query to the subgraph G_f^p can be answered by making at most one query to p. Hence, if the mapping is gap-preserving (as shown next), then a tester for 3-colorability of subgraphs of G_f yields a tester of similar complexity for satisfiability of formulae that are described by the fixed clause-structure f and the varying negation-pattern (which serves as input). Recall that a mapping is *gap-preserving* if it maps yes-instances to yes-instance while mapping "far away" instances to far-away instances. We establish both features next.

Satisfying formulae are mapped to 3-colorable subgraphs. Let ϕ be the formula described by f and p, and suppose that ϕ is satisfiable by the assignment $\tau : [n] \to \{\mathtt{true}, \mathtt{false}\}$. To show that G_f^p is 3-colorable, we introduce the following legal 3-coloring.

1. Each vertex in the tri-partite graph is given the color corresponding to its set (i.e., for each $\ell \in [s]$, the vertex $\mathtt{t}_\ell$ is colored `true`, $\mathtt{f}_\ell$ is colored `false`, and $\mathtt{g}_\ell$ is colored `ground`).

   Hence, there are no monochromatic edges between these vertices.

2. For each $i \in [n]$, the vertex x_i (which belongs to the i^{th} variable-gadget) is colored $\tau(i)$ and the vertex $\neg x_i$ is colored $\neg\tau(i)$.

Hence, the edge $\{x_i, \neg x_i\}$ is not monochromatic, and neither are the edges $\{x_i, \mathtt{g}_i\}$ and $\{\neg x_i, \mathtt{g}_i\}$.

3. For each $j \in [m]$, the head vertex $\mathtt{h}_j$ is colored `true`. Hence, the edges $\{\mathtt{h}_j, \mathtt{f}_j\}$ and $\{\mathtt{h}_j, \mathtt{g}_j\}$ are not monochromatic. Since the assignment τ is a satisfying assignment, for each clause-gadget there is at least one terminal vertex that is colored `true`. Therefore, by the aforementioned property of the clause-gadgets, there exists a 3-coloring of the other gadget vertices that does not introduce any monochromatic edges.

Formulae that are far from being satisfiable are mapped to subgraphs that are far from being 3-colorable. We say that a formula ϕ is ϵ'-far from being satisfiable if every assignment to the variables of ϕ satisfies less than $(1-\epsilon')$-of its clauses (otherwise it is ϵ'-close to being satisfiable). Again, let ϕ be the formula described by f and p, and suppose that ϕ is far from being satisfiable. We shall show that the subgraph G_f^p is far from being 3-colorable, by showing that if G_f^p is ϵ-close to being 3-colorable, then ϕ is $O(\epsilon+\epsilon_0)$-close to being satisfiable, where ϵ_0 is the mixing error of the expander. Suppose that $\chi : V_f \to \{\mathtt{ground}, \mathtt{true}, \mathtt{false}\}$ is a 3-partition of the vertices of G_f^p that has at most $\mu = \epsilon \cdot |E_f|$ monochromatic edges, where $(V_f, E_f) = G_f$.

Without loss of generality, assume that the plurality of the χ-values within each set of the tri-partite graph G' equals the name of this set. Since at most μ of the edges between the parts of the tri-partite graph $G' = (V', E')$ are monochromatic (and $|E'| \geq |E_f|/2$), it follows that all but at most $B = 9 \cdot (6\epsilon + \epsilon_0) \cdot |V'| < 18\mu + 3\epsilon_0 \cdot |E_f|$ of the vertices in the three sets are assigned the plurality (i.e., majority) color of their set, where the inequality uses $|V'| < |E_f|/3$. This is shown by using the mixing property of the expanders that connect the three parts. Specifically, observe that if one of these parts contains $\epsilon' \cdot |V'|/3$ vertices that are not assigned the plurality color, then there must be at least $(\epsilon' \cdot (1/3) - \epsilon_0) \cdot |E'|/3 \geq ((\epsilon'/3) - \epsilon_0) \cdot |E_f|/6$ monochromatic edges between these vertices and the plurality vertices of some other part. Hence, $((\epsilon'/3) - \epsilon_0)/6 \leq \epsilon$, which implies $\epsilon' \leq 3 \cdot (6\epsilon + \epsilon_0)$.

Now consider the vertices $\{x_i\}_{i=1}^k$ and $\{\neg x_i\}_{i=1}^k$ of the variable-gadgets. For each $i \in [n]$, we say that i is a *consistent index* if $\chi(\mathtt{g}_i) = \mathtt{ground}$ and there are no monochromatic edges among the edges $\{x_i, \neg x_i\}$, $\{x_i, \mathtt{g}_i\}$ and $\{\neg x_i, \mathtt{g}_i\}$ (so that either $\chi(x_i) = \mathtt{true}$ and $\chi(\neg x_i) = \mathtt{false}$ or $\chi(x_i) = \mathtt{false}$ and $\chi(\neg x_i) = \mathtt{true}$). Since at most μ of the edges within and incident to the variable-gadgets are monochromatic, all but at most $B' = B + \mu < 19\mu + 3\epsilon_0 \cdot |E_f|$ of the indices $i \in [n]$ are consistent indices.

Based on the coloring χ we define a truth assignment $\sigma : [n] \to \{\mathtt{true}, \mathtt{false}\}$ as follows. If i is a consistent index, then $\sigma(i) = \chi(x_i)$, and otherwise we set $\sigma(i)$ (arbitrarily) to `true`. We claim that σ violates $O(\mu)$ of the clauses of ϕ. To verify this, suppose that $j \in [m]$ is such that (1) $\chi(\mathtt{g}_j) = \mathtt{ground}$ and $\chi(\mathtt{f}_j) = \mathtt{false}$, (2) there are no monochromatic edges incident to the vertices of the j^{th} clause-gadget, and (3) its terminals correspond to consistent indices. By combining (1) and (2), it holds that $\chi(\mathtt{h}_j) = \mathtt{true}$, and by the aforementioned property of the clause-gadgets, at least one of the terminal vertices connected to this clause-gadget must be colored `true` as well. This implies that σ satisfies the j^{th} clause in ϕ.

Letting c denote the constant that upper-bounds the number of occurrences of a variable in the formula, we upper-bound the number of indices $j \in [m]$ for which one of the foregoing requirements does not hold by $B + \mu + c \cdot B'$, where the first term is due to $\chi(\mathtt{g}_j) \neq \mathtt{ground}$ or $\chi(\mathtt{f}_j) \neq \mathtt{false}$, the second term is due to monochromatic edges incident to vertices of the clause-gadget, and the third terms is due to inconsistent indices (since each inconsistent index affects only the clauses in which

the corresponding variable appears). Recalling that $B' = B + \mu < 19\mu + 3\epsilon_0 \cdot |E_f|$ and $\mu = \epsilon \cdot |E_f|$, we get $B + \mu + c \cdot B' = (c+1) \cdot B' < (c+1) \cdot (19\epsilon + 3\epsilon_0) \cdot |E_f|)$, and the claim follows.

Conclusion. Lastly, we show how a tester T for 3-colorability of subgraphs of G_f can be used to obtain an algorithm ("distinguisher") D for the following task. Given query access to a negation-pattern p, the algorithm D distinguishes (with probability at least 2/3) between the case that a 3CNF formula ϕ described by (the fixed clause-structure) f and p is satisfiable and the case in which it is ϵ-far from being satisfiable. The distinguisher D invokes T and answers its edge-queries in the natural manner; that is, all queries are answered 1, except for the queries that correspond to the edges between the clause-gadgets and their terminals. That is, for edge-queries that correspond to an edge between the j^{th} clause-gadget and one of its terminals, denoted y, the distinguisher D queries p, and responds accordingly. More specifically, if y is an unnegated (resp., negated) form of the k^{th} variable occurring in the clause, then D answers 1 if and only if $p(j,k) = 0$ (resp., $p(j,k) = 1$). Hence, on input p such that f and p describe a satisfiable formula (resp., a formula that is ϵ-far from being satisfiable), D invokes T while providing it with oracle access to a 3-colorable subgraph of G_f (resp., a subgraph of G_f that is $\Omega(\epsilon - \epsilon_0)$-far from being 3-colorable). Recalling that Claim 3.3.1 applies to $\epsilon = 0.1$ and letting $\epsilon_0 = \epsilon/2$, Theorem 3.3 follows. ∎

4 Testing in the subgraph model may be harder than in the BDG model

An indication that testing in the subgraph model may be harder than testing in the BDG model is given by analogy to the orientation model of Halevy *et al.* [HLNT05]. Specifically, Fischer *et al.* [FLM+12] proved that testing whether the orientation of an k-by-k cyclic grid is Eulerian (i.e., the indegree of each vertex equals its outdegree, where the digraph is not necessarily connected) requires $\Omega(\log\log k)$ queries.[33] In contrast, in the bounded-degree (directed) graph model, testing whether a directed graph is Eulerian can be done by sampling $\Theta(1/\epsilon)$ vertices and comparing their in-degree to their out-degree.[34] Actually, this analogy can be transformed into a proof of the following result, which is essentially a restatement of Theorem 1.5

Theorem 4.1 (testing in the subgraph model may be harder than in the BDG model): *There exists a property of graphs Π for which the following holds. On the one hand, Π is testable in $O(1/\epsilon)$-time in the bounded-degree graph model. On the other hand, there exist explicit graphs $G = ([n], E)$ of constant degree such that testing whether a subgraph of G satisfies Π requires $\Omega(\log\log n)$ queries. Furthermore, the property Π is upwards monotone, and the base graph G has $(\epsilon, O(1/\epsilon^2))$-partitions for every $\epsilon > 0$.*

[33] The cited bound is for two-sided error adaptive testers. The lower bounds for restricted testers are higher. In fact, Fischer *et al.* [FLM+12, Sec. 9] proved that a two-sided (resp., one-sided) error non-adaptive tester must make $\widetilde{\Omega}(\sqrt{\log k})$ (resp., $\Omega(\sqrt{k})$) queries.

[34] This claim is proved by showing that if the fraction of violating vertices is ρ, then the digraph is $O(\rho)$-close to being Eulerian. First observe that if the number of edges in the graph is at most $2d^2$ (where d is the constant degree bound), then we can get an Eulerian graph by removing all (constant number of) edges. Otherwise, we can apply the following iterative process. First, we pick a pair (u,v) such that u has a deficit of in-coming edges and v has a deficit of out-going edges. Next, we pick a directed edge $x \to y$ such that $x \not\to u$ and $v \not\to y$, and $x, y \notin \{u, v\}$ (where such an edge must exist given the lower bound on the number of edges). Then, we omit the edge $x \to y$ from the digraph and insert the edges $x \to u$ and $v \to y$. Note that the number of edges can only increase in each iteration (so that it is always greater than $2d^2$) and the total (incoming and outgoing) deficit degreases in each iteration, where initially it is at most $d\rho n$.

The hardness result is presented by a reduction that preserves non-adaptivity and one-sided error, and consequently stronger lower bounds hold for classes of restricted testers for Π in the subgraph model. Specifically, any non-adaptive tester must make $\widetilde{\Omega}(\sqrt{\log n})$ queries, and a lower bound of $\Omega(n^{1/4})$ queries holds for non-adaptive testers that have one-sided error.

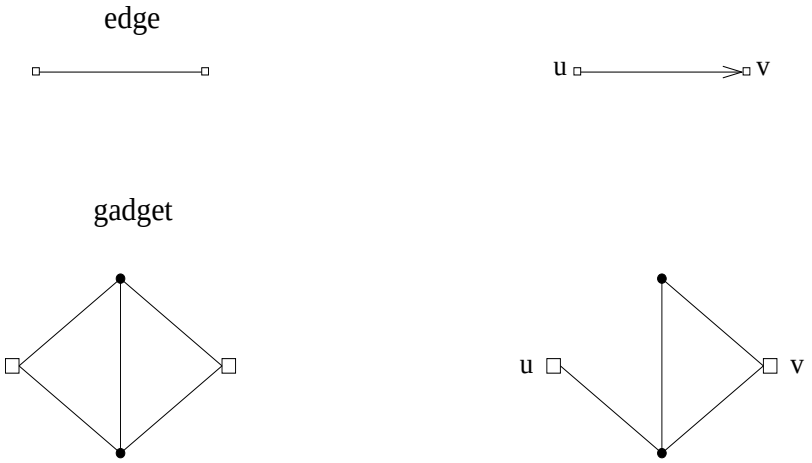

Figure 6: The (edge) gadget and the representation of the orientation of the edge $\{u, v\}$. The main vertices are depicted as squares.

4.1 Proving the main claim of Theorem 4.1

We first establish the main claim of the theorem, while using a property Π that is neither upwards nor downward monotone. The furthermore claim is established in Section 4.2 by considering an upwards monotone closure of a variant of Π.

We start with establishing the lower bound for the subgraph testing model, by reducing the testing problem considered in [FLM+12, Sec. 9] to the testing problem considered here. Fischer *et al.* [FLM+12] proved that testing whether the orientation of a k-by-k cyclic grid is Eulerian requires $\Omega(\log \log k)$ queries. We shall replace each edge of this cyclic grid G_k by a *gadget* consisting of two parallel paths of length two, each using a distinct auxiliary vertex, and an edge connecting these two auxiliary vertices (see Figure 6). The resulting graph, denoted G, will serve as our base graph. Note that G has k^2 vertices of degree eight, called its *main vertices*, and $4k^2$ (auxiliary) vertices of degree three. Furthermore, the set of main vertices is an independent set in G. Also observe that G has $(\epsilon, O(1/\epsilon^2))$-partitions for every $\epsilon > 0$. In particular, each part corresponds to a $(c/\epsilon) \times (c/\epsilon)$ sub-grid (for a constant $c > 1$).

Fischer *et al.* [FLM+12] viewed the orientation of vertical (resp., horizontal) edges in the cyclic grid as either `up` or `down` (resp., `right` or `left`). An edge directed up (resp., down) is outgoing (resp., in-coming) at its lower endpoint and in-coming (resp., outgoing) at its higher endpoint, and ditto for the horizontal edges. Such an orientation is Eulerian if each vertex has two in-coming edges and two out-going edges. We represent an orientation of an edge from u to v in the cyclic grid by assigning the value 1 to all but one of the edges of the corresponding gadget such that the missing edge is (one of the two edges) incident to u (see Figure 6).

Hence, there is a one-to-two mapping between orientations of G_k and subgraphs of G in which each (edge) gadget has exactly one missing edge. It is tempting to let Π simply be the subset of subgraphs of G that are obtained when the orientation of G_k is Eulerian. As we shall see in the proof of Claim 4.1.1, this definition of Π allows to prove that testing Π in the subgraph model with base graph G requires $\Omega(\log \log n)$ queries. However, we also need to show that testing Π in the BDG model can be done by performing $O(1/\epsilon)$ queries. To facilitate this upper bound we define Π differently (in a slightly more cumbersome way) so that it includes a larger subset of graphs. However, this definition still satisfies that the subgraphs of G in $\Pi \cap \mathcal{F}_G$ are exactly those that correspond to Eulerian orientations of G_k.

The basic observation is that each Eulerian orientation of G_k corresponds to a subgraph of G in which each main vertex has degree six (since the corresponding vertex in G_k has indegree two and outdegree two), and in each connected pair of auxiliary vertices, one of them has degree two and one has degree three. This suggests defining Π as the set of graphs that satisfy the following conditions:

1. Each vertex in the graph has degree six, three, or two.
2. Each vertex of degree two is connected to one vertex of degree three and to one vertex of degree six. Furthermore, these two neighbors are connected.
3. Each vertex of degree three is connected to one vertex of degree two and to two vertices of degree six.
4. Each vertex of degree six is connected to four vertices of degree three and to two vertices of degree two.

In other words, a graph in Π consists of vertices of degree six that are connected between them by subgraphs that contain (in addition to these two vertices) one vertex of degree two and one vertex of degree three (where the latter two vertices are connected by an edge). These subgraphs can be viewed as having an orientation, which is determined by the missing edge (i.e., by which of the two degree 6 vertices misses an edge in the subgraph connecting them), and each vertex of degree six participates in two subgraphs of each of the two orientations.

Hence, if a subgraph $\tilde{G}$ of G belongs to Π, then it corresponds to a orientation of G_k in which each vertex has indegree two and outdegree 2 (so that this orientation is Eulerian). The converse is also true: if a subgraph $\tilde{G}$ of G is such that the orientation of G_k corresponding to $\tilde{G}$ is Eulerian, then $\tilde{G}$ belongs to Π. Observe though that not every graph in Π necessarily corresponds to an orientation of G_k (for some k), but it corresponds to some Eulerian directed graph.

Claim **4.1.1** (reducing testing Eulerianity to testing Π in the subgraph model): *Testing whether the orientation of G_k is Eulerian* (with proximity parameter ϵ) *is reducible to testing whether the subgraph of the base graph G satisfies property* Π (with proximity parameter $0.4 \cdot \epsilon$). *The reduction preserves the number of queries.*

Proof: In accordance with the foregoing discussion, we represent (or emulate) an orientation of G_k by a subgraph of G as follows. Each edge $\{u, v\}$ of G_k that is directed from u to v is represented by a subgraph of the corresponding gadget in which an edge incident to vertex u is missing (since there are two such possible edges, the choice of which edge is missing, may be arbitrary).

Let T be a tester for Π in the subgraph model. We derive a tester T' for Eulerianity in the orientation model as follows. Given oracle access to an orientation of G_k, the tester T' invokes T and answers its (i.e., T's) queries regarding edges in G by making queries to the corresponding directed edges of G_k. Specifically, when T queries an edge in the gadget that corresponds to the edge $\{u,v\}$ of G_k, tester T' queries the orientation of $\{u,v\}$ and answers accordingly. (Actually, since only two of the edges of the gadget are used to represent the orientation, queries to the other three edges can be answered (by 1) without making any query to the base graph.)

By the construction of the subgraph of G, if the orientation of G_k is Eulerian, then T' answers in a manner that is consistent with a subgraph that satisfies Π. On the other hand, we claim that if the orientation H of G_k is ϵ-far from being Eulerian, then T' answers in a manner that is consistent with a subgraph G^H of G that is 0.4ϵ-far from Π.

To verify this, let $\tilde{G}$ be a subgraph of G that belongs to Π and is closest to G^H. Let $\tilde{\epsilon}$ denote the distance between G^H and $\tilde{G}$. Since $\tilde{G}$ belongs to Π, it defines an orientation $\tilde{H}$ of G_k that is Eulerian. Furthermore, the number of edges that are oriented differently in $\tilde{H}$ as compared to H is exactly one-half the symmetric difference between the edge set of G^H and the edge set of $\tilde{G}$. To verify the latter statement, consider any pair of main vertices u,v in G that correspond to neighboring vertices in G_k and the pair of auxiliary vertices x,y by which they are connected in G. Observe that both in G^H and in $\tilde{G}$ one of these vertices has degree two and one has degree three, and one of them is connected to both u and v, and the other to only one of them. If the subgraph induced by $\{u,v,x,y\}$ differs between G^H and $\tilde{G}$, then it differs in exactly two edges, implying that the orientation of the edge between u and v differs between H and $\tilde{H}$. Finally, as the number of edges in G_k is smaller by a factor of five than the number of edges in G (i.e., $2k^2$ versus $10k^2$), the distance between the orientations H and $\tilde{H}$ is $5\tilde{\epsilon}/2$, which must be greater than ϵ, implying that $\tilde{\epsilon} > 0.4\epsilon$. The claim follows. ■

Hence, the $\Omega(\log\log k)$ lower bound on testing Eulerianity in the orientation model yields a corresponding lower bound for testing Π the subgraph model, with respect to an explicit 8-regular $O(k^2)$-vertex graph. It is left to show that, in the BDG model, testing Π is easy. In fact, we show that (in the BDG model) Π has a proximity oblivious tester, a notion we recall after the claim.

Claim **4.1.2** (testing Π in the BDG model): *Property Π can be tested in the bounded-degree graph model, with distance parameter ϵ, using $O(1/\epsilon)$ queries. Actually, Π has a one-sided error proximity oblivious tester that makes a constant number of queries and has a linear detection probability function.*

Recall that a (one-sided error) *proximity oblivious tester* (see [GR11]) is a tester-like algorithm that does not get a proximity parameter, but rejects objects that do not have the property with probability that is lower-bounded by a function of their distance from the property. This function is called the detection probability function, and is denoted ρ. Note that a constant-query proximity oblivious tester with detection probability function ρ implies a standard tester of query complexity $O(1/\rho(\epsilon))$.

Proof: Note that the number of vertices in graphs in Π must be a multiple of five, since there are two vertices of degree two (resp., three) per each vertex of degree six. Hence, the tester rejects if the number of vertices in the input graph, which is given as explicit input, is not a multiple of five.[35]

[35] Alternatively, one may redefine Π such that, for every $n \in \mathbb{N}$, it contains all n-vertex graphs that consist of a $5 \cdot \lfloor n/5 \rfloor$-vertex graph and $n \bmod 5$ isolated vertices.

Otherwise, the tester selects uniformly a vertex s in the input graph, and explores its depth-three neighborhood.

1. If s has degree six and its depth-three neighborhood is consistent with a graph in Π, then the tester accepts.

   To spell out the consistency condition, it means that s has four neighbors of degree three and two neighbors of degree two, and that these neighbors belong to four subgraphs that connect s to four distinct degree-six vertices such that each subgraph contains two degree-six vertices, one degree-two vertex and one degree-three vertex (and all edges of this 4-vertex subgraph are incident to the latter two vertices).

2. If s has degree two such that one of its neighbors has degree three and the other has degree six, then the tester accepts.

3. If s degree three such that one of its neighbors has degree two and the other two neighbors have degree six, then the tester accepts.

4. Otherwise, the tester rejects.

By the definition of Π, this tester always accepts graphs in Π. It remains to show that if a graph is ϵ-far from Π, then it is rejected with probability at least $2/3$. We establish the contrapositive statement. Namely, consider any graph that is rejected by the tester with probability ρ. We shall prove that the graph is at distance $O(\rho)$ from Π (by showing that it is possible to add/remove $O(\rho n)$ edges and obtain a graph in Π).

Denoting the input graph by $([n], E)$, let D_i denote the set of vertices of degree i, and $A \subseteq D_6 \cup D_2 \cup D_3$ denote the set of initial choices (of the vertex s) under which the tester accepts (i.e., $|A| = (1-\rho) \cdot n$). Let $E' \subseteq E$ be the subset of edges that are incident to vertices in A, and note that $|E \setminus E'| \leq d \cdot \rho n$, where d denotes the degree bound (with respect to which testing is defined). Letting $A_i = D_i \cap A$, we note that each of the edges in E' is incident to $A_2 \cup A_3$ (since each edge with one endpoint in A_6 must have its other endpoint in $A_2 \cup A_3$). Furthermore, the edges in E' can be partitioned among edge-disjoint oriented gadgets. Specifically, each *oriented gadget* consists of two vertices of D_6, one vertex of A_2 and one vertex of A_3 such that the latter vertex is connected to all other vertices in the subgraph and the degree-two vertex is connected to one of the vertices of D_6. Let D'_6 denote the subset of vertices in D_6 that participate in oriented gadgets, and note that $D'_6 \supseteq A_6$. Also note that the number of such gadgets, denoted m, satisfies $m = |A_2| = |A_3|$ and $3m = 6 \cdot |D'_6|$. It follows that $2.5 \cdot m = |A_2| + |A_3| + |D'_6| \geq |A|$, so that $m \geq |A|/2.5$, and hence $|D'_6| = 0.5m \geq 0.2 \cdot |A| \geq 0.2 \cdot (1-\rho)n$.

We now define an auxiliary digraph G' over the vertex set D'_6 such that there is a directed edge from u to v in G' if these two vertices are connected (in E') by an oriented gadget that misses an edge incident to u. We observe that, in G', each vertex in A_6 has two incoming edges and two outgoing edges, and so the set of vertices that violate this condition equals $D'_6 \setminus A_6 = D'_6 \setminus A \subseteq [n] \setminus A$, which means that their number is at most ρn.

If $|D'_6| < 0.2 \cdot n$, then we augment G' with $0.2 \cdot n - |D'_6| \leq 0.2\rho n$ vertices (which initially have no incident edges), and denote this set of vertices by D''_6. If $|D'_6| > 0.2 \cdot n$, then we remove $|D'_6| - 0.2n$ vertices that belong to $D'_6 \setminus A_6$ from G', and denote this set of vertices by D'''_6. In either case, let the resulting digraph be denoted by G''. Since $|D''_6| \leq 0.2\rho n$ and $|D'''_6| \leq \rho n$, in either case, the total number of vertices in G'' that either do not have two incoming edges or do not have two

outgoing edges is at most ρn. This implies that G'' is $O(\rho)$-close to an Eulerian digraph (over $0.2n$ vertices) in which all vertices have in-degree two (see Footnote 34).

Finally, we perform the corresponding modifications on the original undirected graph $G = ([n], E)$ to obtain a graph in Π. To be precise, recall that we let E' denote the edges incident to vertices in A (so we have already removed all $O(\rho n)$ edges that are not incident to any vertex in A). If $D''_6 \neq \emptyset$, then we can associate each vertex in D''_6 with some vertex in $[n] \setminus A$ (which currently has no incident edges), and if $D'''_6 \neq \emptyset$, then we remove all $(O(\rho n))$ edges incident to D'''_6 as well as to their neighbors in $A_2 \cup A_3$ (so that we have removed all edges in the gadgets that they participated in). Now, each removal of a directed edge from the digraph G' in the process of obtaining the Eulerian digraph G'' corresponds to the removal of all edges from the corresponding gadget in the undirected graph we are modifying. Similarly, each addition of a directed edge corresponds to the addition of a corresponding gadget (since the number of vertices in G' and G'' is exactly $0.2n$, we do not lack vertices that can serve as auxiliary vertices in the gadgets). Hence, the total number of modifications performed in the process of obtaining a graph in Π from G is $O(\rho n)$, as claimed. ■

We have thus completed the proof of the main part of Theorem 4.1. Recall that since the base graph we used is planar (and has maximum degree 8), it has $(\epsilon, O(1/\epsilon^2))$-partitions for every $\epsilon > 0$. This establishes the second part of the furthermore claim of the theorem.

4.2 Proving the secondary ("furthermore") claim of Theorem 4.1

We establish the first part of the furthermore claim (i.e., upwards monotonicity of the property) by using the same base graph G and a variant of the property Π, whereas Π itself is neither upwards nor downward monotone.

We first introduce a graph property Π' that contains both Π and the set of all graphs having an edge that connects two vertices of degree at least four. Note that the base graph G contains no subgraph in $\Pi' \setminus \Pi$, since the only vertices of degree at least four in G are not connected in G. Hence, the move from Π to Π' has no effect on the subgraph testing model, but it makes testing in the BDG model almost trivial (since each n-vertex graph is $O(1/n)$-close to Π').[36]

Next, we consider the upwards monotone closure of Π', denoted Π''; that is, an n-vertex graph is in Π'' if and only if it contains an n-vertex subgraph in Π'. Needless to say, the triviality of testing Π' in the BDG model extends to $\Pi'' \supseteq \Pi'$ (since distances to Π'' are not larger than distances to Π'). Hence, we focus on verifying that Claim 4.1.1 extends to Π''. Towards this end, we use the same mapping (of directed graphs to subgraphs of G) that was presented in the original proof. We show that this mapping constitutes a reduction of testing whether an orientation of G_k is Eulerian to testing whether a subgraph of G is in Π''.

First note that Eulerian orientations of G_k are mapped to subgraphs of G that are in $\Pi \subseteq \Pi''$. Next note that, in each subgraph of G that is in Π (or rather in $\Pi \cap \mathcal{F}_G$), each gadget misses a single edge (out of four designated ones), whereas in each subgraph of G that is in Π'' (or rather in $\Pi'' \cap \mathcal{F}_G$) each gadget misses *at most* one edge (out of four designated ones). However, the subgraphs that are at the image of the reduction always miss a single edge (out of four designated ones) in each gadget. Hence, if a missing edge in the subgraph in the image of the reduction

[36] Formally, on input n and ϵ, and oracle access to a tested graph, the tester accepts if $\epsilon > 10/n$, while making no queries. Note that in this case, the tested graph is ϵ-close to Π', since it suffices to add at most nine edges (and omit at most one). Otherwise (i.e., $\epsilon \leq 10/n$), the tester explores the entire tested graphs and decided accordingly, when in this case its query complexity is $O(1/\epsilon)$.

indicates an orientation that should be changed, then this edge must be added to the subgraph. (Indeed, unlike in Π, correcting the wrong indication does not mandate omitting a different edge from the same gadget; but the former addition suffices towards proving the claim, and we lose only a factor of two in the number of edge modifications as compared to Claim 4.1.1.)

4.3 Testing whether subgraphs of the grid are Eulerian

Needless to say, Theorem 4.1 does not refer to the Eulerian property (of undirected graphs) but rather to a property that results from emulating directed Eulerian graphs by certain gadgets. Actually, it is easy to test whether a subgraph of the (plain or cyclic) grid is Eulerian.

Proposition 4.2 (testing whether a subgraph of a grid is Eulerian): *For any $k < n$, let $G = ([n], E)$ be either the k-by-n/k grid or the k-by-n/k cyclic grid. Then, testing whether a subgraph of G is Eulerian with distance parameter ϵ can be done in time* $\mathrm{poly}(1/\epsilon)$.

Proof: Consider a partition of the grid into sub-grides of side-lengths $\Theta(1/\epsilon)$, which we refer to as *squares*, with a $\Theta(1)$-unit wide *intermediate* grid between them (e.g., a 3-unit wide intermediate grid will do). The construction is illustrated in Figure 7. The tester selects $\Theta(1/\epsilon)$ such squares uniformly at random, and accepts if and only if all vertices that reside in the sampled squares have an even degree (where edges with one endpoint in the square are counted too). This tester has query complexity $O(\epsilon^{-3})$, and it always accepts Eulerian subgraphs.

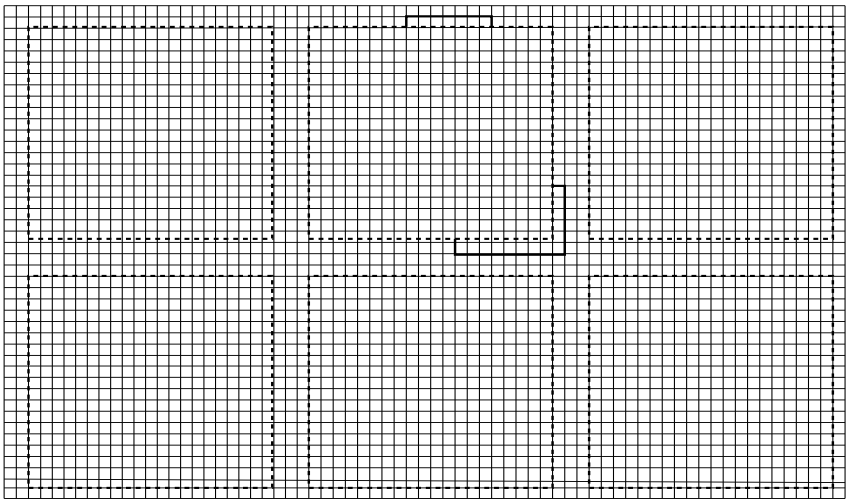

Figure 7: A grid with six squares depicted as dashed boxes and the 3-unit wide intermediate grid. Two connections are shown in solid lines.

To complete the analysis of this tester, suppose that the subgraph is ϵ-far from being Eulerian, and let ρ denote the fraction of the squares that contain a vertex that has an odd degree in the subgraph. Our goal is to show that $\rho = \Omega(\epsilon)$. This is established by showing that the subgraph is $(\rho + 0.5\epsilon)$-close to being Eulerian. Specifically, to modify it into an Eulerian graph we first omit all edges that are incident to vertices that reside in bad squares (i.e., squares that contain vertices of odd degree) as well as all edges that are incident to the intermediate grid. (The fraction of edges internal to bad squares is at most ρ, whereas the fraction of edges incident to the intermediate grid is $\frac{\Theta(1)}{\Theta(1/\epsilon)} \leq \epsilon/4$.)

Next, we use the intermediate grid in order to connect vertices that lie on the boundary of a good square and have an odd number of neighbors in the square (and had a single neighbor in the intermediate grid, before we omitted all edges incident to the intermediate grid). Such connections can be made by vertex-disjoint paths that go along the sides of the square (and at distance 1 from it), since the connected vertices all lie on the boundary of the same square (see Figure 7). The total fraction of edges used for these connections is $\epsilon/4$, and so the claim follows. ■

4.4 Open problems

Theorem 4.1 shows that a property that, for some constant ϵ, is testable in a constant number of queries in the BDG model but requires a double-logarithmic number of queries in the subgraph model. We wonder whether a larger gap can be established.

Problem 4.3 (a larger gap between the subgraph and the BDG models): *For a function $q : \mathbb{N} \to \mathbb{N}$ such that $q(n) = \omega(\log \log n)$, does there exist a graph property Π such that Π is testable in $\mathrm{poly}(1/\epsilon)$-time in the bounded-degree model, although there exist graphs $G = ([n], E)$ of constant degree such that testing whether a subgraph of G satisfies Π requires $\Omega(q(n))$ queries.*

Recall that such results are know for restricted testers; specifically, for non-adaptive testers we can establish the claim for $q(n) = \widetilde{\Omega}(\log n)$, and $q(n) = \Omega(n^{1/4})$ holds for one-sided error non-adaptive testers.

On the other hand, we wonder about the complexity of testing degree regularity in the subgraph model, while recalling that this property is testable with $O(1/\epsilon)$ queries in the BDG model. Note that testing 1-regularity of a subgraph of the cycle does not reduce to checking the degrees of random vertices, and one needs to take into account the location of edges. Details follow.

Consider a $2n$-vertex cycle and a random subgraph of it that consists of $n-1$ edges (i.e., having exactly $2n-2$ vertices of degree 1). Then, with high probability the subgraph is $\Omega(1)$-far from being 1-regular, but one cannot distinguish this subgraph from a 1-regular subgraph by making $o(n)$ *degree* queries. (On the other hand, making two random edge queries, and taking into account the locations of these edges, yields a POT with linear detection probability.)

Acknowledgements

We are grateful to the reviewers of TOCT for their extremely useful comments.

References

[AST90] N. Alon, P.D. Seymour, and R. Thomas. A separator theorem for graphs with an excluded minor and its applications. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing (STOC)*, pages 293–299, 1990.

[Ben65] V. E. Beneš. Mathematic theory of connecting networks and telephone traffic. *Mathematics in Science and Engineering*, 17, 1965.

[BOT02] A. Bogdanov, K. Obata, and L. Trevisan. A lower bound for testing 3-colorability in bounded-degree graphs. In *Proceedings of the Forty-Third Annual Symposium on Foundations of Computer Science (FOCS)*, pages 93–102, 2002.

[BSHR05] E. Ben-Sasson, P. Harsha, and S. Raskhodnikova. 3CNF properties are hard to test. *SIAM Journal on Computing*, 35(1):1–21, 2005.

[BSS10] I. Benjamini, O. Schramm, and A. Shapira. Every minor-closed property of sparse graphs is testable. *Advances in mathematics*, 223:2200–2218, 2010.

[CMOS19] A. Czumaj, M. Monemizadeh, K. Onak, and C. Sohler. Planar graphs: Random walks and bipartiteness testing. *Random Structures and Algorithms*, 55(1):104–124, 2019.

[CSS09] A. Czumaj, A. Shapira, and C. Sohler. Testing hereditary properties of nonexpanding bounded-degree graphs. *SIAM Journal on Computing*, 38(6):2499–2510, 2009.

[EHNO11] A. Edelman, A. Hassidim, H. N. Nguyen, and K. Onak. An efficient partitioning oracle for bounded-treewidth graphs. In *Proceedings of the Fifteenth International Workshop on Randomization and Computation (RANDOM)*, pages 530–541, 2011.

[Ele06] G. Elek. The combinatorial cost. *ArXiv Mathematics e-prints*, 2006.

[Ele10] G. Elek. Parameter testing with bounded degree graphs of subexponential growth. *Random Structures and Algorithms*, 37:248–270, 2010.

[FJ12] U. Feige and S. Jozeph. Universal factor graphs. In *Automata, Languages and Programming: Thirty-Ninth International Colloquium (ICALP)*, pages 339–350, 2012.

[FLM+12] E. Fischer, O. Lachish, A. Matsliah, I. Newman, and O. Yahalom. On the query complexity of testing orientations for being eulerian. *ACM Transactions on Algorithms*, 8(2):15:1 – 15:41, 2012.

[FNY+20] S. Forster, D. Nanongkai, L. Yang, T. Saranurak, and S. Yingchareonthawornchai. Computing and testing small connectivity in near-linear time and queries via fast local cut algorithms. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2046–2065. SIAM, 2020.

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

[Gol08] O. Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008.

[Gol17] O. Goldreich. *Introduction to Property Testing*. Cambridge University Press, 2017.

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

[GR02] O. Goldreich and D. Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002.

[GR10] M. Gonen and D. Ron. On the benefit of adaptivity in property testing of dense graphs. *Algorithmica*, 58(4):811–830, 2010.

[GR11] O. Goldreich and D. Ron. On proximity oblivious testing. *SIAM Journal on Computing*, 40(2):534–566, 2011.

[Hea87] L.S. Heath. Embedding outerplanar graphs in small books. *SIAM Journal on Algebraic and Discrete Methods*, 8(2):198–218, 1987.

[HKNO09] A. Hassidim, J. Kelner, H. Nguyen, and K. Onak. Local graph partitions for approximation and testing. In *Proceedings of the Fiftieth Annual Symposium on Foundations of Computer Science (FOCS)*, pages 22–31, 2009.

[HLNT05] S. Halevy, O. Lachish, I. Newman, and D. Tsur. Testing orientation properties. *Electronic Colloquium on Computational Complexity (ECCC)*, 12(153), 2005.

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

[LR15] R. Levi and D. Ron. A quasi-polynomial time partition oracle for graphs with an excluded minor. *ACM Transactions on Algorithms*, 11(3):24:1–24:13, 2015.

[LT79] R.J. Lipton and R.E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Discrete Math*, 36(2):177–189, 1979.

[New10] I. Newman. Property testing of massively parametrized problems – a survey. In O. Goldreich, editor, *Property Testing: Current Research and Surveys*, pages 142–157. Springer, 2010. LNCS 6390.

[NS13] I. Newman and C. Sohler. Every property of hyperfinite graphs is testable. *SIAM Journal on Computing*, 42(3):1095–1112, 2013.

[Pet94] E. Petrank. The hardness of approximation: Gap location. *Computational Complexity*, 4:133–157, 1994.

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

[PRR06] M. Parnas, D. Ron, and R. Rubinfeld. Tolerant property testing and distance approximation. *Journal of Computer and System Sciences*, 72(6):1012–1042, 2006.

[Val82] L.G. Valiant. A scheme for fast parallel communication. *SIAM Journal on Computing*, 11(2):350–361, 1982.

[Val12] G. Valiant. *Algorithmic Approaches to Statistical Questions*. PhD thesis, University of California at Berkeley, 2012.

[VB81] L.G. Valiant and G.J. Brebner. Universal schemes for parallel communication. In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing (STOC)*, pages 263–277, 1981.

[VV17] G. Valiant and P. Valiant. Estimating the unseen: Improved estimators for entropy and other properties. *Journal of the ACM*, 64(6):37:1–37:41, 2017.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il