Fast Distributed Algorithms for Testing Graph Properties *

Keren Censor-Hillel† Eldar Fischer† Gregory Schwartzman† Yadu Vasudev†

September 20, 2017

Abstract

We initiate a thorough study of *distributed property testing* – producing algorithms for the approximation problems of property testing in the CONGEST model. In particular, for the so-called *dense* graph testing model we emulate sequential tests for nearly all graph properties having 1-sided tests, while in the general model we obtain faster tests for triangle-freeness and cycle-freeness, and in the sparse model we obtain a faster test for bipartiteness. In addition, we show a logarithmic lower bound for testing bipartiteness and cycle-freeness, which holds even in the stronger LOCAL model.

In most cases, aided by parallelism, the distributed algorithms have a much shorter running time than their counterparts from the sequential querying model of traditional property testing. More importantly, the distributed algorithms we develop for testing graph properties are in many cases much faster than what is known for exactly deciding whether the property holds. The simplest property testing algorithms allow a relatively smooth transitioning to the distributed model. For the more complex tasks we develop new machinery that may be of independent interest.

*A preliminary version of this work appeared in DISC 2016, pages 43-56

†Technion – Israel Institute of Technology, Department of Computer Science. ckeren@cs.technion.ac.il, eldar@cs.technion.ac.il, gregorys@cs.technion.ac.il, yaduvasudev@gmail.com. Supported in part by the Israel Science Foundation (grant 1696/14).

1 Introduction

The performance of many distributed algorithms naturally depends on properties of the underlying network graph. Therefore, a natural goal is to check whether the graph, or some given subgraph, has certain properties. However, in some cases this is known to be hard, such as in the CONGEST model [37]. In this model, computation proceeds in synchronous rounds. In each round, every vertex can send an $O(\log n)$-bit message to each of its neighbors. Lower bounds for the number of rounds of type $\tilde{\Omega}(\sqrt{n} + D)$ are known for *verifying* many *global* graph properties, where n is the number of vertices in the network and D is its diameter (see, e.g. Das-Sarma et al. [42]).[1] For some *local* graph properties faster solutions are known, such as a $\Theta(\sqrt{n})$ round complexity for detecting 4-cycles [14] (see also the excellent survey on local decision problems in [16]). However, for some local graph properties, such as detecting triangles, the round complexity of the problem remains a riddle, with a very recent result showing an $O((n \log n)^{2/3})$-round algorithm for detecting triangles [29], but currently no known non-trivial lower bound.

To overcome such difficulties, we adopt the relaxation used in graph property testing, as first defined in [21, 23], to the distributed setting. That is, rather than aiming for an exact answer to the question of whether the graph G satisfies a certain property P, we settle for distinguishing the case of satisfying P from the case of being ϵ-*far* from it, for an appropriate measure of being far.

Apart from its theoretical interest, this relaxation is motivated by the common scenario of having distributed algorithms for some tasks that perform better given a certain property of the network topology, or given that the graph *almost* satisfies that property. For example, Hirvonen et al. [27] show an algorithm for finding a large cut in triangle-free graphs (with additional constraints), and for finding an $(1 - \epsilon)$-approximation if at most an ϵ fraction of all edges are part of a triangle. Similarly, Pettie and Su [38] provide fast algorithms for coloring triangle-free graphs.

We provide a rigorous study of property testing methods in the realm of distributed computing under the CONGEST model, by constructing fast distributed algorithms for testing various graph properties. An important byproduct of this study is a toolbox that we believe will be useful in other settings as well.

In more detail, we construct 1-*sided error distributed* ϵ-*tests*, in which if the graph satisfies the property then all vertices output `accept`, and if it is ϵ-far from satisfying the property then at least one vertex outputs `reject` with probability at least 2/3. [2] This is aligned with the standard requirements of *distributed decision problems*, where an exact answer is needed, for which all vertices output `accept` if the graph satisfies the property, and at least one vertex outputs `reject`, otherwise (see, e.g., the excellent survey in [16]).

The definition of a graph being ϵ-far from satisfying a property is roughly one of the following (see Section 2 for precise definitions): (1) Changing any ϵn^2 entries in the adjacency matrix does not give a graph that satisfies the property (dense graph model [21]), or (2) changing any $\epsilon \cdot \max\{n, m\}$ entries in the adjacency matrix does not give a graph that satisfies the property, where m is the number of edges (general model [35]). A particular case of (2) is when the degrees are bounded by some constant d, and any resulting graph must comply with this restriction as well (sparse model [23]).

In a *sequential* ϵ-*test*, access to the input is provided by queries, whose type depends on the

[1] Here $\tilde{\Omega}$ hides factors that are polylogarithmic in n.

[2] The probability of rejection can be improved by standard amplification techniques. In particular, the test can be repeated multiple times, and a vertex will reject if it rejects in any of the invocations.

model. In the dense graph model these are *pair queries*, asking whether two vertices v, u are neighbors, and in the general and sparse models these can be either *degree queries*, asking what the degree of a vertex v is, or *neighbor queries*, asking what the i-th neighbor of v is (the ordering of neighbors is arbitrary).[3] While a sequential ϵ-test can touch only a small handful of vertices with its queries, in a distributed test the lack of ability to communicate over large distances is offset by having all n vertices operating in parallel.

1.1 Our contributions

Our first contribution is a general scheme for a near-complete emulation in the distributed context of ϵ-tests originating from the dense graph model (Section 3). This makes use of the fact that in the dense graph model all (sequential) testing algorithms can be made *non-adaptive*, which roughly means that queries do not depend on responses to previous queries (see Section 2 for definition). In fact, such tests can be made to have a very simple structure, allowing the vertices in the distributed model to "band together" for an emulation of the test. There is only one additional technical condition (*non-disjointness*, which we define below), since in the distributed model we cannot handle properties whose counter-examples can be "split" to disjoint graphs. For example, the distributed model cannot hope to handle the property of the graph having no disjoint union of two triangles, a property for which there exists a test in the dense graph model. The reason for this is that if there do exist such triangles, but only such that they are far away from each other, the vertices cannot detect this without communicating over this distance and incurring it as a cost for the number of rounds. The same issue arises with other properties whose violation may span distant vertices or disconnected ones.

Theorem 3.4 *Any ϵ-test in the dense graph model for a non-disjointed property that makes q queries can be converted to a distributed ϵ-test that takes $O(q^2)$ communication rounds.*

We next move away from the dense graph model to the sparse and general models, which are sometimes considered to be more realistic. In the general model, it is impossible to test that a graph is triangle-free using a constant number of queries independent of the size of the graph [2]. In the distributed setting, we can exploit concurrency to break the sequential lower bound of [2], by having all vertices operate concurrently.

Theorem 4.1 *Algorithm 2 (defined in Section 4) is a distributed ϵ-test in the general graph model for the property of triangle-freeness, that requires $O(1/\epsilon^2)$ rounds.*

The sparse and general models inherently require *adaptive* property testing algorithms, since there is no non-adaptive way to follow a path that starts from a certain vertex, or to scan a vertex's neighborhood. Testing triangle freeness sequentially uses adaptivity only to a small degree. However, other problems in the sparse and general models, such as testing if a graph is bipartite, have a high degree of adaptivity built into their sequential algorithms, and we need to take special care for emulating it in the distributed setting.

In the sparse model (degrees bounded by a constant d), we adapt ideas from the bipartiteness testing algorithm of [22], in which we search for odd-length cycles. Here again the performance of

[3] In the literature there are also investigations of a slightly strengthened general model, where pair queries are also allowed; its discussion is out of the scope for this paper.

a distributed algorithm surpasses that of a centralized tester (a number of rounds polylogarithmic in n vs. a number of queries which is $\Omega(\sqrt{n})$ – a lower bound that is given in [23]). The following is proved in Section 5.

Theorem 5.2 *Algorithm 4 (defined in Section 5) is a distributed ϵ-test in the bounded degree graph model for the property of being bipartite, that requires $O(poly(1/\epsilon \cdot \log(n/\epsilon)))$ rounds.*

In the course of proving Theorem 5.2 we develop a method that we consider to be of independent interest[4]. The algorithm works by performing $2n$ random walks concurrently (two starting from each vertex). The parallel execution of random walks despite the congestion restriction is achieved by making sure that the walks have a uniform stationary distribution, and then showing that congestion is "close to average", which for the uniform stationary distribution is constant.

In Section 6 we show a fast test for cycle-freeness. This makes use of a combinatorial lemma that we prove, about cycles that remain in the graph after removing edges independently with probability $\epsilon/2$. The following summarizes our result for testing cycle-freeness.

Theorem 6.3 *Algorithm 6 (defined in Section 6) is a distributed ϵ-test in the general graph model for the property of being cycle-free, that requires $O(\log n/\epsilon)$ rounds.*

We also prove lower bounds for testing bipartiteness and cycle-freeness (matching the upper bound for the latter). Roughly speaking, these are obtained by using the probabilistic method with alterations to construct graphs which are far from being bipartite or cycle-free, but all of their cycles are of length that is at least logarithmic. This technique bears some similarity to the classic result by Erdös [15], which showed the existence of graphs with large girth and large chromatic number. The following are given in Section 7.

Theorem 7.1 *Any distributed 1/100-test for the property of being bipartite requires $\Omega(\log n)$ rounds of communication.*

Theorem 7.3 *Any distributed 1/100-test for the property of being cycle-free requires $\Omega(\log n)$ rounds of communication.*

Table 1 summarizes the known complexities.

Notes about the sequential testing column of Table 1: The best known results for triangle testing in the general model are a superpolynomial lower bound, and an upper bound that is a tower function of $\log(1/\epsilon)$. For general model testing the bounds are usually expressed as functions of n and m (the number of edges), and best power of n for triangle freeness is not known. To our knowledge, cycle freeness outside the dense model was only investigated for directed graphs [7] (in the dense model, undirected graph cycle freeness is almost the same as having $o(n^2)$ edges).

Roadmap: The paper is organized as follows. The remainder of this section consists of related work and historical background on property testing. Section 2 contains formal definitions and some mathematical tools. The emulation of sequential tests for the dense graph model is given in Section 3. In Section 4 we give our distributed test for triangle-freeness. In Section 5 we provide a distributed test for bipartiteness, along with our new method of executing many random walks, and in Section 6 we give our test for cycle-freeness. Section 7 gives our logarithmic lower bounds for testing bipartiteness and cycle-freeness. We conclude with a short discussion in Section 8.

[4]This technique was recently independently and concurrently devised in [20] for a different use.

Property	Exact (rounds)	Testing [This paper] (rounds)	Sequential Testing (queries)
Non-disjointed property	—	$O(q^2)$ if there is a q-query test	q
Triangle-Freeness	$O((n \log n)^{2/3})$ [29]	$O(\epsilon^{-2})$	Dense: $f(\epsilon)$ [Various] General: power of n [2]
Bipartiteness		$O(\text{poly}(\epsilon^{-1} \log(n\epsilon^{-1})))$ $\Omega(\log n)$	Dense: $\tilde{O}(\epsilon^{-2})$ [3] General: $\tilde{\Theta}(\sqrt{n})/\text{poly}(\epsilon)$, see [31]
Cycle-freeness	$\Theta(D)$ (graph's diameter)	$O(\log n/\epsilon)$ $\Omega(\log n)$	Dense: Property is trivial

Table 1: Summary of known complexities for exact distributed decision, distributed testing, and sequential testing. Our upper bounds are for the CONGEST model, while our lower bounds hold also for the LOCAL model.

1.2 Related work

The only previous work that directly relates to our distributed setting is due to Brakerski and Patt-Shamir [9]. They show a *tolerant* property testing algorithm for finding large (linear in size) *near-cliques* in the graph. An ϵ-near clique is a set of vertices for which all but an ϵ-fraction of the pairs of vertices have an edge between them. The algorithm is tolerant, in the sense that it finds a linear near-clique if there exists a linear ϵ^3-near clique. That is, the testing algorithm considers two thresholds of being close to having the property (in this case – containing a linear size clique). We are unaware of any other work on property testing in this distributed setting. We also emphasize that [9] has already successfully implemented the approach of sampling vertices of the graph and using ideas from a centralized tester in order to obtain their distributed algorithm. A similar general line is used by our emulation in Section 3 for obtaining our extended statement.

Testing in a different distributed setting was considered in Arfaoui et al. [5]. They study testing for cycle-freeness, in a setting where each vertex may collect information of its entire neighborhood up to some distance, and send a short string of bits to a central authority who then has to decide whether the graph is cycle-free or not.

Testing allows us fast solutions for problems for which exactly detecting whether the graph satisfies a property or not is an expensive task. The latter are called *distributed decision problems*, and were rigorously studied in the distributed computing literature (see, e.g., [16] for a recent thorough survey).

Distributed algorithms for (exactly) deciding if a graph satisfies a given property are also studied in the context of proof labeling schemes, introduced by Korman et al. [32] (for extensions see, e.g., Baruch et al. [6]). In this setting, each vertex is given some external label, and by exchanging labels the vertices need to decide whether a given property of the graph holds. This is different from our setting, in which no information other than vertex IDs is available. Another setting that is related to proof-labelling schemes, but differs from our model, is the prover-verifier model of Foerster et al. [18].

Sequential property testing has the goal of computing without processing the entire input. The

wider family of *local computation algorithms* (LCA) is known to have connections with distributed computing, as shown by Parnas and Ron [36]. A recent study by Göös et al. [26] proves that under some conditions, the fact that a centralized algorithm can query distant vertices does not help with speeding up computation. However, they consider the LOCAL model, and their results apply to certain properties that are not influenced by distances.

Finding induced subgraphs is a crucial task and has been studied in several different distributed models (see, e.g., [10, 12, 13, 30]). Notice that for *finding* subgraphs, having *many* instances of the desired subgraph can help speedup the computation, as in [12]. This is in contrast to algorithms that perform faster if there are *no* or only *few* instances, as explained above, which is why we test for, e.g., the property of being *triangle-free*, rather for the property of *containing* triangles. (Notice that these are not the same, and in fact every graph with $3/\epsilon$ or more vertices is ϵ-close to having a triangle.)

Parallelizing many random walks was addressed in [1], where the question of graph covering via random walks is discussed. It is shown there that for certain families of graphs there is a substantial speedup in the time it takes for k walks starting from the same vertex to cover the graph, as compared to a single walk. No edge congestion constraints are taken into account. In [43], it is shown how to perform, under congestion, a single random walk of length L in $\tilde{O}(\sqrt{LD})$ rounds, and k random walks in $\tilde{O}(\sqrt{kLD} + k)$ rounds, where D is the diameter of the graph. Our method has no dependence on the diameter, allowing us to perform a multitude of *short walks* much faster.

1.3 Historical overview

The first papers to consider the question of property testing were [8] and [41]. The original motivations for defining property testing were its connection to some Computerized Learning models, and the ability to leverage some properties to construct Probabilistically Checkable Proofs (PCPs – this is related to property testing through the areas of Locally Testable Codes and Locally Decodable Codes, LTCs and LDCs). Other motivations since then have entered the fray, and foremost among them are sublinear-time algorithms, and other big-data considerations. Since virtually no property can be decidable without reading the entire input, property testing introduces a notion of the allowable approximation to the original problem. In general, the algorithm has to distinguish inputs satisfying the property, from inputs that are *ϵ-far* from it. For more information on the general scheme of "classical" property testing, consult the surveys [17, 24, 39].

The dense graph model was defined in the seminal work of Goldreich, Goldwasser and Ron [21]. The dense graph model has historically started combinatorial property testing in earnest, but it has some drawbacks. The main one being the distance function, which is suitable for graphs having many edges (hence the name "dense graph model") – any graph with $o(n^2)$ edges is indistinguishable in this model from an empty graph.

The stricter and, at times more plausible, distance function is one which is relative to the actual number of edges, rather than the maximum $\binom{n}{2}$. The general model was defined in [35], while the sparse model was defined already in [23]. The main difference between the sparse and the general graph models is that in the former there is also a guaranteed upper bound d on the degrees of the vertices, which is given to the algorithm in advance (the query complexity may then depend on d, either explicitly, or more commonly implicitly by considering d to be a constant).

2 Preliminaries

2.1 Additional background on property testing

While the introduction provided rough descriptions of the different property testing models, here we provide more formal definitions. The dense graph model for property testing is defined as follows.

Definition 2.1 (dense graph model [21])**.** *The dense graph model considers as objects graphs that are given by their adjacency matrix. Hence it is defined by the following features.*

- **Distance:** *Two graphs with n vertices each are considered to be* ϵ-close *if one can be obtained from the other by deleting and inserting at most ϵn^2 edges (this is, up to a constant factor, the same as the normalized Hamming distance).*
- **Querying scheme:** *A single query of the algorithm consists of asking whether two vertices $u, v \in V$ form a graph edge in E or not.*
- **Allowable properties:** *All properties have to be invariant under permutations of the input that pertain to graph isomorphisms (a prerequisite for them being graph properties).*

The number of vertices n is given to the algorithm in advance.

As discussed earlier, the sparse and general graph models for property testing relate the distance function to the actual number of edges in the graph. They are formally defined as follows.

Definition 2.2 (sparse [23] and general [2] graph models)**.** *These two models consider as objects graphs given by their adjacency lists. They are defined by the following features.*

- **Distance:** *Two graphs with n vertices and m_1 and m_2 edges are considered to be* ϵ-close *if one can be obtained from the other by deleting and inserting at most $\epsilon \max\{n, m_1, m_2\}$ edges*[5]*.*
- **Querying scheme:** *A single query consists of either asking what is the degree of a vertex v, or asking what is the i'th neighbor of v (the ordering of neighbors is arbitrary).*
- **Allowable properties:** *All properties have to be invariant under graph isomorphisms (which here translate to a relabeling that affects both the vertex order and the neighbor ids obtained in neighbor queries), and reordering of the individual neighbor lists (as these orderings are considered arbitrary).*

In this paper, we mainly refer to the distance functions of these models, and less so to the querying scheme, since the latter will be replaced by the processing scheme provided by the distributed computation model. Note that most property testing models get one bit in response to a query, e.g., "yes/no" in response to "is uv an edge" in the dense graph model. However, the sparse and general models may receive $\log n$ bits of information for one query, e.g., an id of a neighbor of a vertex. Also, the degree of a vertex, which can be given as an answer to a query in the general model, takes $\log n$ bits. Since the distributed CONGEST model allows passing a vertex id or a vertex degree along an edge in $O(1)$ rounds, we can relate all three graph models.

Another important point is the difference between 1-sided and 2-sided testing algorithms, and the difference between non-adaptive and adaptive algorithms.

[5] Sometimes in the sparse graph model the allowed number of changes is ϵdn, as relates to the maximum possible number of edges; when d is held constant the difference is not essential.

Definition 2.3 (types of algorithms)**.** *A property testing algorithm is said to have* 1-sided error *if there is no possibility of error on accepting satisfying inputs. That is, an input that satisfies the property will be accepted with probability* 1*, while an input* ϵ*-far from the property will be rejected with a probability that is high enough (traditionally this means a probability of at least* 2/3*). A* 2-sided error *algorithm is also allowed to reject satisfying inputs, as long as the probability for a correct answer is high enough (traditionally at least* 2/3*).*

A property testing algorithm is said to be non-adaptive *if it decides all its queries in advance (i.e. based only on its internal coin tosses and before receiving the results of any query), while only its accept/reject output may depend on the actual input. An* adaptive *algorithm may make each query in turn based on the results of its previous queries (and, as before, possible internal coin tosses).*

In the following we address both adaptive and non-adaptive algorithms. However, we restrict ourselves to 1-sided error algorithms, since the notion of 2-sided error is not a good match for our distributed computation model. 2-sided error is difficult to deal with in the distributed decision, since if even a single node rejects, then the entire graph is rejected.

2.2 Mathematical background

The Multiplicative Chernoff Bound (see, e.g., [34]) plays an important role in our analysis. We state it here for completeness.

Fact 2.4. *Suppose that* $X_1, ..., X_n$ *are independent random variables taking values in* $\{0, 1\}$*. Let* X *denote their sum and let* $\mu = E[X]$ *denote its expected value. Then, for any* $\delta > 0$*,*

$$Pr[X < (1-\delta)\mu] < (\frac{e^{-\delta}}{(1-\delta)^{(1-\delta)}})^{\mu},$$
$$Pr[X > (1+\delta)\mu] < (\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}})^{\mu}.$$

Some convenient variations of the bounds above are:

$$Pr[X \geq (1+\delta)\mu] < e^{-\delta\mu/3}, \quad \delta \geq 1$$
$$Pr[X \geq (1+\delta)\mu] < e^{-\delta^2\mu/3}, \quad \delta \in (0,1)$$
$$Pr[X \leq (1-\delta)\mu] < e^{-\delta^2\mu/2}, \quad \delta \in (0,1).$$

We also use the following very simple observation.

Fact 2.5. *If* $X_1, \ldots, X_n$ *are independent random indicator* 0/1 *variables, and* $X = \sum_{i=1}^{n} X_i$ *satisfies* $E[X] \geq 4$*, then* $Pr[X < 1] < \frac{1}{4}$*.*

Proof. This follows from a straightforward calculation:

$$Pr[X < 1] = Pr[X_1 = \cdots = X_n = 0] = \prod_{i=1}^{r}(1 - Pr[X_i = 0]) < e^{-\sum_{i=1}^{r} Pr[X_i=0]} = e^{-E[X]} < \frac{1}{4}$$

□

3 Distributed emulation of sequential tests in the dense graph model

We begin by showing that under a certain assumption of being *non-disjointed*, which we define below, a property P that has a sequential test in the dense graph model that requires q queries can be tested in the distributed setting within $O(q^2)$ rounds. We prove this by constructing an emulation that translates sequential tests to distributed ones. For this we first introduce a definition of a *witness* graph and then adapt [25, Theorem 2.2], restricted to 1-sided error tests, to our terminology.

Definition 3.1. *Let P be a property of graphs with n vertices. Let G' be a graph with $k < n$ vertices. We say that G' is a* witness against *P, if it is not an induced subgraph of any graph that satisfies P.*

Notice that if G' has an induced subgraph H that is a witness against P, then by the above definition G' is also a witness against P.

The work of [25] transforms tests of graphs in the dense graph model to a canonical form where the query scheme is based on vertex selection. This is useful in particular for the distributed model, where the computational work is essentially based in the vertices. We require the following special case for 1-sided error tests.

Lemma 3.2 (**[25, Theorem 2.2]**)**.** *Let P be a property of graphs with n vertices. If there exists a 1-sided error ϵ-test for P with query complexity $q(n, \epsilon)$, then there exists a 1-sided error ϵ-test for P that uniformly selects a set of $q' = 2q(n, \epsilon)$ vertices, and accepts if and only if the induced subgraph is not a witness against P.*

Our emulation leverages Lemma 3.2 under an assumption on the property P, which we define as follows.

Definition 3.3. *We say that P is a* non-disjointed *property if for every graph G that does not satisfy P and an induced subgraph G' of G such that G' is a witness against P, G' has some connected component which is also a witness against P. We call such components* witness components*.*

We are now ready to formally state our main theorem for this section.

Theorem 3.4. *Any ϵ-test in the dense graph model for a non-disjointed property that makes q queries can be converted to a distributed ϵ-test that takes $O(q^2)$ communication rounds.*

The following lemma essentially says that non-disjointed properties can be tested by examining connected subgraphs, which is exactly what we need to forbid in a distributed setting.

Lemma 3.5. *A property P is non-disjointed if and only if all minimal witnesses against P are connected.*

Here *minimal* refers to the standard terminology, which means that no proper induced subgraph is a witness against P.

Proof. First, if P is non-disjointed and G does not satisfy P, then for every subgraph G' of G that is a witness against P, G' has a witness component. If G' is minimal then it must be connected,

since otherwise it contains a connected component which is a witness against P, which contradicts the minimality of G.

For the other direction, if all the minimal witnesses that are induced subgraphs of G are connected, then every induced subgraph G' that is a witness against P is either minimal, in which case it is connected, or is not minimal, in which case there is a subgraph H of G' which is connected and a minimal witness against P. The connected component C of G' which contains H is a witness against P (otherwise H is not a witness against P), and hence it follows that P is non-disjointed. □

Next, we give the distributed test (Algorithm 1). The test has an outer loop in which each vertex selects itself with probability $5q/n$, collects its neighborhood of a certain size of edges between *selected* vertices in an inner loop, and rejects if it identifies a witness against P. The outer loop repeats two times because not only does the sequential test have an error probability, but also with some small probability we may randomly select too many or not enough vertices needed to emulate it. Repeating the main loop twice reduces the error probability back to below 1/3. In the inner loop, each vertex collects its neighborhood of selected vertices and checks if its connected component is a witness against P. To limit communications this is done only for components of selected vertices that are sufficiently small: if a vertex detects that it is part of a component with too many edges then it accepts and does not participate until the next iteration of the outer loop.

Algorithm 1: Emulation algorithm with input q for property P

Variables: U_v edges known to v, U'_v edges to update and send (temporary variables)

perform 2 **times**
  reset the state for all vertices
  for each *vertex* v **simultaneously**
    Vertex v *selects* itself with probability $5q/n$
    if v *is selected* **then**
      Notify all neighbors that v is selected
      Set $U'_v = \{(v,u) \in E|$ u is selected$\}$ and $U_v = \emptyset$
      perform $10q$ **times**
        `# At each iteration` U_v `is a subgraph of` v`'s connected component`
        $U'_v = U'_v \backslash U_v$ `# only need recently discovered edges`
        $U_v = U_v \cup U'_v$ `# add them to` U_v
        if $|U_v| \leq 100q^2$ **then** `# don't operate if there are too many edges`
          Send U'_v to all selected neighbours of v `# propagate known edges`
        Wait until the time bound for all other vertices to finish this iteration
        Set U'_v to the union of edge sets received from neighbors
      if $U_v \cup U'_v$ *is a witness against* P **then**
        Vertex v outputs `reject` (ending all operations)
    else
      Wait until the time bound for all other vertices to finish this iteration of the outermost loop
Every vertex v that did not `reject` outputs `accept`

To analyze the algorithm, we begin by proving that there is a constant probability for the number of selected vertices to be sufficient and not too large.

Lemma 3.6. *The probability that the number of vertices selected by the algorithm is between q and $10q$ is more than $2/3$.*

Proof. For every $v \in V$, we denote by X_v the indicator variable for the event that vertex v is selected. Note that these are all independent random variables. Using the notation $X = \sum_{v \in V} X_v$ gives that $E[X] = 5q$, because each vertex is selected with probability $5q/n$. Using the Chernoff Bound from Fact 2.4 with $\delta = 4/5$ and $\mu = 5q$, we can bound the probability of having too few selected vertices:

$$Pr[X < q] = Pr[X < (1-\delta)\mu] < (\frac{e^{-4/5}}{(1-(4/5))^{(1-(4/5))}})^{5q} = (\frac{5}{e^4})^q < \frac{1}{10}.$$

For bounding the probability that there are too many selected vertices, we use the other direction of the Chernoff Bound with $\delta = 1$ and $\mu = 5q$, giving:

$$Pr[X > 10q] = Pr[X > (1+\delta)\mu] < (\frac{e}{2^2})^{5q} = (\frac{e^5}{2^{10}})^q < \frac{2}{10}.$$

Thus, with probability at least $2/3$ it holds that $q \leq X \leq 10q$. □

Now, we can use the guarantees of the sequential test to obtain the guarantees of our algorithm.

Lemma 3.7. *Let P be a non-disjointed graph property. If G satisfies P then all vertices output* `accept` *in Algorithm 1. If G is ϵ-far from satisfying P, then with probability at least $2/3$ there exists a vertex that outputs* `reject`*.*

Proof. First, assume that G satisfies P. Vertex v outputs `reject` only if it is part of a witness against P, which is, by definition, a component that cannot be extended to some H that satisfies P. However, every component is an induced subgraph of G itself, which does satisfy P, and thus every component can be extended to G. This implies that no vertex v outputs `reject`.

Now, assume that G is ϵ-far from satisfying P. Since the sequential test rejects with probability at least $2/3$, the probability that a sample of at least q vertices induces a graph that cannot be extended to a graph that satisfies P is at least $2/3$. Since P is non-disjointed, the induced subgraph must have a connected witness against P. We note that a sample of more than q vertices does not reduce the rejection probability. Hence, if we denote by A the event that the subgraph induced by the selected vertices has a connected witness against P, then $Pr[A] \geq 2/3$, conditioned on that at least q vertices were selected.

However, a sample that is too large may cause a vertex to output `accept` because it cannot collect its neighborhood. We denote by B the event that the number of vertices sampled is between q and $10q$, and by Lemma 3.6 its probability is at least $2/3$. We bound $Pr[A \cap B]$ using Bayes' Theorem, obtaining $Pr[A \cap B] = Pr[A|B]Pr[B] \geq (2/3)^2$. Since the outer loop consists of 2 independent iterations, this gives a probability of at least $1 - (1 - 4/9)^2 \geq 2/3$ for having a vertex that outputs `reject`. □

We now address the round complexity. Each vertex only sends and receives information from its q-neighborhood about edges between the chosen vertices. If too many vertices are chosen we

detect this and accept. Otherwise we only communicate the chosen vertices and their edges, which requires $O(q^2)$ communication rounds using standard *pipelining*.[6] Together with Lemma 3.7, this proves Theorem 3.4.

3.1 Applications: k-colorability and perfect graphs

Next, we provide some examples of usage of Theorem 3.4. A result by Alon and Shapira [4] states that all graph properties closed under induced subgraphs are testable in a number of queries that depends only on $1/\epsilon$. We note that, except for certain specific properties for which there are ad-hoc proofs, the dependence is usually a tower function in $1/\epsilon$ or worse (asymptotically larger).

From this, together with Lemma 3.2 and Theorem 3.4, we deduce that if P is a non-disjointed property closed under induced subgraphs, then it is testable, for every fixed ϵ, in a constant number of communication rounds.

Example – k-colorability: The property of being k-colorable is testable in a distributed manner by our algorithm. All minimal graphs that are witnesses against P (not k-colorable) are connected, and therefore according to Lemma 3.5 it is a non-disjointed property. It is closed under induced subgraphs, and by [3] there exists a 1-sided error ϵ-test for k-colorability that uniformly selects $O(k\log(k)/\epsilon^2)$ vertices, and its number of queries is the square of this expression (note that the polynomial dependency was already known by [21]). Our emulation implies a distributed 1-sided error ϵ-test for k-colorability that requires $O(\text{poly}(k/\epsilon))$ rounds.

Example – perfect graphs: A graph G is said to be *perfect* if for every induced subgraph G' of G, the chromatic number of G' equals the size of the largest clique in G'. Another characterization of a perfect graph is via *forbidden subgraphs*: a graph is perfect if and only if it does not have odd holes (induced cycles of odd length at least 5) or odd anti-holes (the complement graph of an odd hole) [11]. Both odd holes and odd anti-holes are connected graphs. Since these are all the minimal witnesses against the property, according to Lemma 3.5 it is a non-disjointed property. Using the result of Alon-Shapira [4] we know that the property of a graph being perfect is testable. Our emulation implies a distributed 1-sided error ϵ-test for being a perfect graph that requires a number of rounds that depends only on ϵ.

4 Distributed test for triangle-freeness

In this section we show a distributed ϵ-test for triangle-freeness. Notice that since triangle-freeness is a non-disjointed property, Theorem 3.4 gives a distributed ϵ-test for triangle-freeness under the dense graph model with a number of rounds that is $O(q^2)$, where q is the number of queries required for a sequential ϵ-test for triangle-freeness. However, for triangle-freeness, the known number of queries is a tower function in $\log(1/\epsilon)$ [19].

Here we leverage the inherent parallelism that we can obtain when checking the neighbors of a vertex, and show a test for triangle-freeness that requires only $O(1/\epsilon^2)$ rounds (Algorithm 2). Importantly our algorithm works for the *general* graph model (where distances are relative to the actual number of edges), which subsumes the dense graph model analyzed in the previous section.

[6] Pipelining means that each vertex has a buffer for each edge, which holds the information (edges between chosen vertices, in our case) it needs to send over that edge. The vertex sends the pieces of information one after the other.

In the sequential setting, a test for triangle-freeness in the general model requires $n^{\Omega(1)}$ queries by [2]. Our proof actually follows the groundwork laid in [2] for the general graph model — their algorithm selects a vertex and checks two of its neighbors for being connected, while we perform the check for all vertices in parallel.

Algorithm 2: Triangle freeness test

```
for each vertex v simultaneously
    perform 32/ε² times
        Select w_1, w_2 ∈ N(v), w_1 ≠ w_2 uniformly at random
        Send w_2 to w_1 # Ask w_1 if it is a neighbor of w_2
        foreach w_u sent by u ∈ N(v) do # Asked by u if v is a neighbor of w
            if w_u ∈ N(v) then
                Send "yes" to u
            else
                Send "no" to u
        if received "yes" from w_1 then
            reject (ending all operations)
accept (for vertices that did not reject)
```

Theorem 4.1. *Algorithm 2 is a distributed ϵ-test in the general graph model for the property of triangle-freeness, that requires $O(1/\epsilon^2)$ rounds.*

Our proof follows the argument described in [2], by distinguishing edges that connect two high-degree vertices from those that do not. Formally, let $b = 2\sqrt{m/\epsilon}$, where m is the number of edges in the graph, and denote $B = \{v \in V \mid deg(v) \geq b\}$. We say that an edge $e = (u, v)$ is *light* if $v \notin B$ or $u \notin B$, and otherwise, we say that it is *heavy*. That is, the set of heavy edges is $H = \{(u, v) \in E \mid u \in B, v \in B\}$.

In a nutshell, our analysis considers light edges, and argues that if G is ϵ-far from being triangle-free then there are at least $\epsilon m/2$ light edges in triangles, denoted by T. For an endpoint of such an edge which is not in B, the probability of detecting a triangle is at least $1/b^2$, giving that the expected number of detected triangles in this case is at least $\epsilon^2/8$. Summing over all $O(1/\epsilon^2)$ iterations and applying a Chernoff bound then gives the claimed result.

Formally, we begin with the following simple claim about the number of heavy edges.

Claim 4.2. *The number of heavy edges, $|H|$, is at most $\epsilon m/2$.*

Proof. The number of heavy edges is $|H| \leq |B|(|B| - 1)/2 < |B|^2/2$. Since $|B|b \leq 2m$, we get that $|B| \leq \frac{2m}{b} = \frac{2m}{2\sqrt{m/\epsilon}} = \sqrt{\epsilon m}$. This gives that $|H| \leq \frac{1}{2}|B|^2 \leq \epsilon m/2$. □

Next, we fix an iteration i of the algorithm. Every vertex v chooses two neighbors w_1, w_2. Let $A = \{(v, w_1) \in E \mid v \in V \setminus B\}$, where w_1 is the first of the two vertices chosen by the low-degree vertex v. Let $T = \{e \in E \mid e \text{ is a light edge in a triangle}\}$, and let $A_T = T \cap A$. We say that an edge $(v, w_1) \in A_T$ is *matched* if (v, w_2) is in the same triangle as (v, w_1). If $(v, w_1) \in A_T$ is matched then $\{v, w_1, w_2\}$ is a triangle that is detected by v.

We begin with the following lemma that states that if G is ϵ-far from being triangle-free, then in any iteration i we can bound the expected number of matched edges from below by $\epsilon^2/8$. Let Y be the number of matched edges.

Lemma 4.3. *If G is ϵ-far from being triangle-free, then the expected number of matched edges in a single iteration of the algorithm satisfies $E[Y] \geq \epsilon^2/8$.*

Proof. For every $e \in A_T$, let Y_e be a random variable indicating whether e is matched. Then $Y = \sum_{e \in A_T} Y_e$, giving the following bound:

$$E[Y|A_T] = E[\sum_{e \in A_T} Y_e | A_T] = \sum_{e \in A_T} Pr[\text{e is matched}] \geq |A_T|/b, \tag{1}$$

where the last inequality follows because a light edge in A_T is chosen by a vertex with degree at most b, hence the third triangle vertex gets selected with probability at least $1/b$.

Next, we argue that $E[|A_T|] \geq |T|/b$. To see why, for every edge e, let X_e be a random variable indicating whether $e \in A$. Let $X = \sum_{e \in T} X_e = |A_T|$. Then,

$$E[|A_T|] = E[X] = E[\sum_{e \in T} X_e] = \sum_{e \in T} E[X_e] = \sum_{e \in T} Pr[e \in A] \geq |T|/b, \tag{2}$$

where the last inequality follows because a light edge has at least one endpoint with degree at most b. Hence, this edge gets selected by it with probability at least $1/b$.

It remains to bound $|T|$ from below. We claim that $|T| \geq \epsilon m/2$. To prove this, first notice that, since G is ϵ-far from being triangle free, it has at least ϵm triangle edges, since otherwise we can just remove all of them and make the graph triangle free with less than ϵm edge changes. By Claim 4.2, the number of heavy edges satisfies $|H| \leq \epsilon m/2$. Subtracting this from the number of triangle edges gives that at least $\epsilon m/2$ edges are light triangle edges, i.e.,

$$|T| \geq \epsilon m/2. \tag{3}$$

Finally, by Inequalities (1), (2) and (3), using iterated expectation we get:

$$E[Y] = E_{A_T}[E[Y|A_T]] \geq E[\frac{|A_T|}{b}] \geq \frac{|T|}{b^2} \geq \frac{\epsilon m}{2} \frac{1}{4m/\epsilon} = \epsilon^2/8.$$

□

We can now prove the correctness of our algorithm, as follows.

Lemma 4.4. *If G is triangle-free then all vertices output* `accept` *in Algorithm 2. If G is ϵ-far from being triangle-free, then with probability at least 2/3 there exists a vertex that outputs* `reject`*.*

Proof. If G is triangle free then in each iteration v receives "no" from w_1 and after all iterations it returns `accept`.

Assume that G is ϵ-far from being triangle-free. Let $Z_{i,v}$ be an indicator variable for the event that vertex v detects a triangle at iteration i. First, we note that the indicators are independent, since a vertex detecting a triangle does not affect the chance of another vertex detecting a triangle (note that the graph is fixed), and the iterations are done independently. Now, let

$Z = \sum_{i=1}^{32/\epsilon^2} \sum_{v \in V} Z_{i,v}$, and notice that Z is the total number of detections over all iterations. Lemma 4.3 implies that for a fixed i, it holds that $E[\sum_{v \in V} Z_{i,v}] = E[Y] \geq \epsilon^2/8$, so Z sums to:

$$E[Z] = E\left[\sum_{i=1}^{32/\epsilon^2} \sum_{v \in V} Z_{i,v}\right] = \sum_{i=1}^{32/\epsilon^2} E\left[\sum_{v} Z_{i,v}\right] \geq \sum_{i=1}^{32/\epsilon^2} \epsilon^2/8 = 4.$$

By Fact 2.5 this means that with probability more than 2/3, at least one triangle is detected and the associated vertex outputs `reject`, which completes the proof. □

In every iteration, an edge needs to carry only two messages of size $O(\log n)$ bits, one sent from each vertex v to w_1 and one sent back by w_1. Since there are $O(1/\epsilon^2)$ iterations, this implies that the number of rounds is $O(1/\epsilon^2)$ as well. This, together with Lemma 4.4, completes the proof of Theorem 4.1.

5 Distributed bipartiteness test for bounded degree graphs

In this section we show a distributed ϵ-test for being bipartite for graphs with degrees bounded by d. Our test builds upon the sequential test of [22] and, as in the case of triangle freeness, takes advantage of the ability to parallelize queries. While the number of queries of the sequential test is $\Omega(\sqrt{n})$ [23], the number of rounds in the distributed test is only *polylogarithmic* in n and polynomial in $1/\epsilon$. As in [22], we assume that d is a constant, and omit it from our expressions. In particular, it is implicit in the O notation for L, which is the length of the random walks that we execute, as we elaborate below.

Let us first outline the algorithm of [22], since our distributed test borrows from its framework and our analysis is in part derived from it. The sequential test basically tries to detect odd cycles. It consists of T iterations, in each of which a vertex s is selected uniformly at random and K random walks of length L are performed starting from the source s. If, in any iteration with a chosen source s, there is a vertex v which is reached by an even prefix of a random walk and an odd prefix of a random walk (possibly the same walk), then the algorithm rejects, as this indicates the existence of an odd cycle. Otherwise, the algorithm accepts. To obtain an ϵ-test the parameters are chosen to be $T = O(1/\epsilon)$, $K = O(\epsilon^{-4}\sqrt{n}\log^{1/2}(n/\epsilon))$, and $L = O(\epsilon^{-8}\log^6 n)$.

The main approach of our distributed test is similar, except that we perform fewer walks from every vertex, namely $O(\text{poly}(1/\epsilon \cdot \log n/\epsilon))$. This is because we can run random walks in parallel originating from all vertices at once. However, a crucial challenge that we need to address is that several random walks may collide on an edge, violating its congestion bound. The reason for this congestion is that each random walk must store the ID of its origin and the number of steps made so far, in order to detect a violation. Thus, a single step of a single random walk requires sending $\Omega(\log n + \log \ell)$ bits across an edge. Therefore, simulating a single step of many random walks may require multiple communication rounds, as only $O(\log n)$ bits may be communicated over an edge in each round.

To address this issue, our central observation is that *lazy* random walks (chosen to have a uniform stationary distribution) provide for a very low probability of having too many of these collisions at once. The main part of the analysis is in showing that with high probability there will never be too many walks concurrently in the same vertex, so we can comply with the congestion bound. We begin by formally defining the lazy random walks that we use (to recall the concept of random walks and stationary distributions, we refer the reader to standard textbooks, e.g., [34]).

Definition 5.1. *A* lazy *random walk over a graph G with degree bound d is a random walk, that is, a (memory-less) sequence of random variables $Y_1, Y_2, \ldots$ taking values from the vertex set V, where the transition probability $Pr[Y_k = v | Y_{k-1} = u]$ is $1/2d$ if uv is an edge of G, $1 - deg(u)/2d$ if $u = v$, and 0 in all other cases.*

The stationary distribution for the lazy random walk of Definition 5.1 is uniform [40, Section 8]. Next, we describe a procedure to handle one step of moving the random walks (Algorithm 3), followed by our distributed test for bipartiteness using lazy random walks from every vertex concurrently (Algorithm 4).

Algorithm 3: Move random walks once with input ξ

```
Variables: W_v walks residing in v (multiset), H_v history of walks through v
Input: ξ, the maximum congestion per vertex allowed
# each walk is characterized by (i, u) where i is the number of actual moves
  and u is the origin vertex
for each vertex v simultaneously
    if |W_v| ≤ ξ then # give up if exceeded the maximum allowed
        for every (i, u) in W_v do
            draw next destination w (according to the lazy walk scheme)
            if w ≠ v then # walk exits v
                send (i + 1, u) to w
                remove (i, u) from W_v
    wait until the maximum time for all other vertices to process up to ξ walks
    add the walks received by v to W_v and H_v # walks entering v
```

The maximum congestion that is allowed per vertex is denoted by ξ. It is quite immediate that Algorithm 3 takes $O(\xi)$ communication rounds.

Algorithm 4: Distributed bipartiteness test

```
Variables: W_v walks residing in v (multiset), H_v history of walks through v
perform η = O(ε^{-9} log(n/ε)) times
    for each vertex v simultaneously
        initialize H_v and W_v with two copies of the walk (0, v)
    perform L = O(ε^{-8} log^6 n) times
        move walks using Algorithm 3 with input ξ = γ + 2 = 3(2 ln n + ln L) + 2
    for each vertex v simultaneously
        if H_v contains (i, u) and (j, u) for some u, even i and odd j then
            reject (ending all operations) # odd cycle found
accept (for vertices that did not reject)
```

Our main result here is that Algorithm 4 is indeed a distributed ϵ-test for bipartiteness.

Theorem 5.2. *Algorithm 4 is a distributed ϵ-test in the bounded degree graph model for the property of being bipartite, that requires $O(poly(1/\epsilon \cdot \log(n/\epsilon)))$ rounds.*

The number of communication rounds is immediate from the algorithm – it is dominated by the L calls to Algorithm 3, making a total of $O(\xi L)$ rounds, which is indeed $O(\text{poly}(1/\epsilon \cdot \log(n/\epsilon)))$. To prove the rest of Theorem 5.2 we need some notation, and a lemma from [22] that bounds from below the probability of detecting odd cycles if G is ϵ-far from being bipartite.

Given a source s, if there is a vertex v which is reached by an even prefix of a random walk w_i from s and an odd prefix of a random walk w_j from s, we say that walks w_i and w_j *detect a violation*. Let $p_s(k, \ell)$ be the probability that, out of k random walks of length ℓ starting from s, there are two that detect a violation. Using this notation, $p_s(K, L)$ is the probability that the sequential algorithm outlined in the beginning rejects in an iteration in which s is chosen. Since we are only interested in walks of length L, we denote $p_s(k) = p_s(k, L)$. A good vertex is a vertex for which this probability is bounded as follows.

Definition 5.3. *A vertex s is called* good *if $p_s(K) \geq 1/10$.*

In [22] it was proved that being far from bipartite implies having many good vertices.

Lemma 5.4 ([22])**.** *If G is ϵ-far from being bipartite then at least an $\epsilon/16$-fraction of the vertices are good.*

In contrast to [22], we do not perform K random walks from every vertex in each iteration, but rather only 2. Hence, what we need for our analysis is a bound on $p_s(2)$. To this end, we use K as a parameter, and express $p_s(2)$ in terms of K and $p_s(K)$.

Lemma 5.5. *For every vertex s, $p_s(2) \geq 2p_s(K)/K(K-1)$.*

Proof. Fix a source vertex s. For every $i, j \in [K]$, let $q_{i,j}$ be the probability of walks w_i, w_j from s detecting a violation. Because different walks are independent, we conclude that for every $i \neq j$ it holds that $q_{i,j} = p_s(2)$. Let $A_{i,j}$ be the event of walks w_i, w_j detecting a violation. We have

$$p_s(K) = Pr[\bigcup_{i,j} A_{i,j}] \leq \sum_{i,j} Pr[A_{i,j}] = p_s(2)K(K-1)/2,$$

which implies that $p_s(2) \geq 2p_s(K)/K(K-1)$. □

Using this relationship between $p_s(2)$ and K and $p_s(K)$, we prove that our algorithm is an ϵ-test. First we prove this for the random walks themselves, ignoring the possibility that Algorithm 3 will skip moving random walks due to its condition in Line 2.

Lemma 5.6. *If G is ϵ-far from being bipartite, and we perform $\eta = O(\epsilon^{-9} \log(n/\epsilon))$ iterations of starting 2 random walks of length L from every vertex, then the probability that no violation is detected is bounded by $1/4$.*

Proof. Assume that G is ϵ-far from being bipartite. By Lemma 5.4, at least $n\epsilon/16$ vertices are good, which means that for each of these vertices s, $p_s(K) \geq 1/10$. This implies that $\sum_{s \in V} p_s(K) \geq n\epsilon/160$. Now, let $X_{i,s}$ be a random variable indicating whether there are two random walks starting

at s that detect a violation in the i^{th} iteration of the algorithm. Let $X = \sum_{i=0}^{\eta} \sum_{s \in V} X_{i,s}$. We prove that $Pr[X < 1] < 1/4$. First, we bound $E[\sum_{s \in V} X_{i,s}]$ for some fixed i:

$$\begin{aligned} E[X] &= E\left[\sum_{i=0}^{\eta} \sum_{s \in V} X_{i,s}\right] = \sum_{i=0}^{\eta} \sum_{s \in V} E[X_{i,s}] \\ &= \sum_{i=0}^{\eta} \sum_{s \in V} p_s(2) \geq \sum_{i=0}^{\eta} \sum_{s \in V} \frac{2p_s(K)}{K(K-1)} \\ &= \frac{2}{K(K-1)} \sum_{i=0}^{\eta} \sum_{s \in V} p_s(K) \geq \frac{2}{K(K-1)} \sum_{i=0}^{\eta} \frac{n\epsilon}{160} \\ &= \frac{\eta n \epsilon}{80K(K-1)} \geq \frac{\eta n \epsilon}{80K^2}. \end{aligned}$$

For $\eta = 320K^2/n\epsilon = O(\epsilon^{-9} \log(n/\epsilon))$ it holds that $E[X] \geq 4$. Fact 2.5 then implies $Pr[X < 1] < \frac{1}{4}$, which completes the proof. □

In the following discussion, we use the term *iteration* for a round of the inner loop of Step 4 inside Algorithm 4. We analyze the status of the random walks after each time they are moved by an iteration of this inner loop.

As explained earlier, the main hurdle on the road to prove Theorem 5.2 is in proving that the allowed congestion will not be exceeded. We prove the following general claim about the probability for k lazy random walks of length ℓ from each vertex to exceed a maximum *congestion factor* of ξ walks allowed in each vertex at the beginning of each iteration.

Lemma 5.7. *Let k and ℓ be parameters, and suppose $\gamma = 3(2 \ln n + \ln \ell)$ satisfies $\gamma > k$. Then with probability at least $1 - 1/n$, running k lazy random walks of length ℓ originating from every vertex will not exceed the maximum* congestion factor *of $\xi = \gamma + k = 3(2 \ln n + \ln \ell) + k$ walks allowed in each vertex at the beginning of each iteration, if $\gamma > k$.*

We show below that substituting $k = 2$, $\ell = L$ and $\gamma = 3(2 \ln n + \ln L)$ in Lemma 5.7, together with Lemma 5.6, gives the correctness of Algorithm 4.

To prove Lemma 5.7, we argue that it is unlikely for any vertex to have more than $k + \gamma$ walks in any iteration. Given that this is indeed the case in every iteration, the lemma follows by a union bound. We denote by $X_{v,i}$ the random variable whose value is the number of random walks at vertex v at the beginning of the i-th iteration. That is, it is equal to the size of the set W_v in the description of the algorithm.

Lemma 5.8. *For every vertex $v \in V$ and every iteration i it holds that $E[X_{v,i}] = k$.*

Proof. Let us first define random variables for our walks. Enumerating our kn walks (k from each of the n vertices) arbitrarily, let $Y_1^r, Y_2^r, \ldots$ denote the sequence corresponding to the r'th walk, that is, Y_i^r is the vertex where the r'th walk is stationed at the beginning of the i'th iteration. In particular, $X_{v,i} = |\{r : Y_i^r = v\}|$.

Now let us define new random variables Z_i^t in the following manner: First, we choose uniformly at random a permutation $\sigma : [kn] \to [kn]$. Then we set $Z_i^t = Y_i^{\sigma(t)}$ for all $1 \leq i \leq \ell$ and $1 \leq t \leq kn$. The main thing to note is that for any fixed t, $Z_1^t, Z_2^t, \ldots$ is a random walk (as it is equal to one of the random walks $Y_1^r, Y_2^r, \ldots$). But also, for every t, Z_1^t is uniformly distributed over the vertex set

of G, because we started with exactly k random walks from every vertex. Additionally, since the uniform distribution is stationary for our lazy walks, this means that the unconditional distribution of each Z_i^t is also uniform.

Now, since σ is a permutation, it holds that $X_{v,i} = |\{r : Y_i^r = v\}| = |\{t : Y_i^{\sigma(t)} = v\}| = |\{t : Z_i^t = v\}|$. The expectation (by linearity of expectation) is thus $E[X_{v,i}] = \sum_{t=1}^{kn} Pr[Z_i^t = v] = k$. □

We can now prove Lemma 5.7.

Proof of Lemma 5.7. We first claim that for every iteration $i \in [\ell]$ and every vertex $v \in V$, with probability at least $1 - 1/\ell n$ it holds that $X_{v,i} \leq k + \gamma$. To show this, first fix some $v \in V$. Let $Z_{j,i}$ be the indicator variable for the event of the j'th walk residing at vertex v at the beginning of iteration i, where $j \in [kn]$. Then $X_{v,i} = \sum_{j=1}^{kn} Z_{j,i}$, and the variables $Z_{j,i}$, where $j \in [kn]$, are all independent. We use the Chernoff Bound of Fact 2.4 with $\delta = \gamma/k \geq 1$ and $\mu = k$ as proven in Lemma 5.8, obtaining:

$$Pr[X_{v,i} > k + \gamma] = Pr[X_{v,i} > (\gamma/k + 1)k] < e^{-\delta\mu/3} = e^{-\gamma/3} = e^{-(2\ln n + \ln \ell)} = 1/\ell n^2.$$

Applying the union bound over all vertices $v \in V$ and all iterations $i \in [\ell]$, we obtain that with probability at least $1 - 1/n$ it holds that $X_{v,i} \leq k + \gamma$ for all v and i. □

Lemma 5.9. *If G is bipartite then all vertices output* `accept` *in Algorithm 4. If G is ϵ-far from being bipartite, then with probability at least $2/3$ there exists a vertex that outputs* `reject`*.*

Proof. If G is bipartite then all vertices output `accept` in Algorithm 4, because there are no odd cycles and thus no violation detecting walks.

If G is ϵ-far from bipartite, we use Lemma 5.6, in conjunction with Lemma 5.7 with parameters $k = 2$, $\ell = L$ and $\gamma = 3(2\ln n + \ln L)$ as used by Algorithm 4. By a union bound the probability to accept G will be bounded by $1/4 + 1/n < 1/3$ (assuming $n > 12$), providing for the required bound on the rejection probability. □

Lemma 5.9, with the communication complexity analysis of Algorithm 4, gives Theorem 5.2.

6 Distributed test for cycle-freeness

In this section, we give a distributed algorithm to test if a graph G with m edges is cycle-free or if at least ϵm edges have to be removed to make it so. Intuitively, in order to search for cycles, one can run a breadth-first search (BFS) and have a vertex output `reject` if two different paths[7] reach it. The downside of this exact solution is that its running time depends on the diameter of the graph. To overcome this, a basic approach would be to run a BFS from each vertex of the graph, but for shorter distances. However, running multiple BFSs simultaneously is expensive, due to the congestion on the edges. Instead, we use a prioritization rule that drops BFS constructions with lower priority, which makes sure that one BFS remains alive, but for this to work we need to carefully choose our prioritization.[8]

[7] Note that the paths we address in this section are not necessarily simple.

[8] A more involved analysis of multiple prioritized BFS executions was used in [28], allowing all BFS executions to fully finish in a short time without too much delay due to congestion. Since we require a much weaker guarantee, we can avoid the strong full-fledged prioritization algorithm of [28] and settle for a simple rule that keeps one BFS tree alive. Also, the multiple BFS construction of [33] does not fit our demands as it may not reach all desired vertices within the required distance, in case there are many vertices that are closer.

Our technique consists of three parts. First, we make the graph G sparser, by removing each of its edges independently with probability $\epsilon/2$. We denote the sampled graph by G' and prove that if G is far from being cycle-free then so is G', and in particular, G' contains a cycle.

Then, we run a partial BFS over G' from each vertex, while prioritizing by ids: each vertex keeps only the BFS that originates in the vertex with the largest id and drops the rest of the BFSs. The length of this procedure is according to a threshold $T = 20\log(n)/\epsilon$. The intuition for choosing this threshold is that if G were acyclic, then with high probability G' would not have any component with a larger diameter. We prove here a sort of converse, that finding a longer path in G' provides a guide for finding a cycle in G.

The above BFS gives detection of a cycle that is contained in a component of G' with a low diameter of up to T, if such a cycle exists, since a surviving BFS covers the component. Such a cycle is also a cycle in G. If no such cycle exists in G', then G' has a some component with diameter larger than T. For large components, we take each surviving BFS that reached some vertex v at a certain distance ℓ, and from v we run a new partial BFS in the *original* graph G. These BFSs are again prioritized, this time according to the distance ℓ. Our main tool here is a claim that says that with high probability, if there is a shortest path in G' of length $T/2$ between two vertices, then there is a cycle in G between them of length at most T. This allows our BFSs on G to find such a cycle.

We start with the following combinatorial lemma that shows the above claim.

Lemma 6.1. *Given a graph G, let G' be obtained by deleting each edge in G with probability $\epsilon/2$, independently of other edges. Then, with probability at least $1 - 1/n^3$, every vertex $v \in G'$ that has a vertex $w \in G'$ at a distance at least $10\log(n)/\epsilon$, has a closed path passing through it in G, that contains a simple cycle, of length at most $20\log(n)/\epsilon$.*

Proof. Note first that if v has a vertex w in G' of distance at least $10\log(n)/\epsilon$, then it has a vertex w' (on the shortest path from v to w) whose distance is exactly $\ell = 10\log(n)/\epsilon$. From now on we assume that the distance between v and w is exactly ℓ, by moving from w to w'.

We now show that for every pair u, v of vertices in G that are at a distance of ℓ, one of the shortest paths between u and v is removed in the graph G' with high probability. For a pair of vertices u and v at a distance ℓ in G, the probability that a shortest path is not removed is $(1-\epsilon/2)^\ell$, which is at most $1/n^5$. Therefore, by a union bound over all pairs of vertices, with probability at least $1 - 1/n^3$, at least one edge is removed from at least one shortest path between every pair of vertices that are at a distance of $10\log(n)/\epsilon$. Conditioned on this, we prove the lemma.

Now, suppose that v and w are two vertices in G' at a distance of $10\log(n)/\epsilon$. Let P' be this shortest path in G'. Suppose that P is the shortest path between v and w in G. If $|P| < 10\log(n)/\epsilon$, then this path is no longer present in G' (and thus is distinct from P') and $P \cup P'$ is a closed path in G, passing through v that has a simple cycle of length at most $20\log(n)/\epsilon$. If $|P| = 10\log(n)/\epsilon$, then there are at least two shortest paths between v and w in G of length $10\log(n)/\epsilon$, the one in G' and one that was removed, which we choose for P. Therefore, $P \cup P'$ is a closed path passing through v of length at most $20\log(n)/\epsilon$, and hence contains a simple cycle of length at most $20\log(n)/\epsilon$ in it. □

Next, we prove that indeed there is a high probability that G' contains a cycle if G is far from being cycle-free.

Claim 6.2. *If G is ϵ-far from being cycle-free, then with probability at least $1 - e^{-\epsilon^2 m/32}$, G' is $\epsilon/4$-far from being cycle-free.*

Proof. The graph G' is obtained from G by deleting each edge with probability $\epsilon/2$ independently of other edges. The expected number of edges that are deleted is $\epsilon m/2$. Therefore, by the Chernoff Bound from Fact 2.4, the probability that at least $3\epsilon m/4$ edges are deleted is at most $\exp(-\epsilon^2 m/32)$, and the claim follows. □

We now describe a multiple-BFS algorithm that takes as input a length t and a priority condition $\mathcal{P}$ over vertices, and starts performing a BFS from each vertex of the graph. This is done for t steps, in each of which a vertex keeps only the BFS with the highest priority while dropping the rest. Each vertex also maintains a list L_v of BFSs that have passed through it. The list L_v is a list of 3-tuples $(\mathrm{id}_u, \ell, \mathrm{id}_p)$, where id_u is the id of the root of the BFS, ℓ is the depth of v in this BFS tree and id_p is the id of the parent of v in the BFS tree. Initially, each vertex v sets L_v to include a BFS starting from itself, and then continues this BFS by sending the tuple $(\mathrm{id}_v, 1, \mathrm{id}_v)$ to all its neighbors, where id_v is the identifier of the vertex v. In an intermediate step, each vertex v may receive a BFS tuple from each of its neighbors. The vertex v then adds these BFS tuples to the list L_v and chooses one among L_v according to the priority condition $\mathcal{P}$, proceeding with the respective BFS and discontinuing the rest. Even when a BFS is discontinued, the information that the BFS reached v is stored in the list L_v.

Algorithm 5 gives a formal description of the breadth-first search that we use in the testing algorithm for cycle-freeness.

Algorithm 5: BFS with a priority condition

Input: Length L, Priority condition $\mathcal{P}$
Variables: L_v list of BFS tuples passing through v

```
for each vertex v simultaneously
    Initialize L_v to {(id_v, 0, id_v)}.
    Send (id_v, 1, id_v) to all neighbors of v.
perform L times times
    for each vertex v simultaneously
        if v receives (id_{u_1}, ℓ_1, id_{p_1}), ..., (id_{u_r}, ℓ_r, id_{p_r}) from its neighbors then
            Add (id_{u_1}, ℓ_1, id_{p_1}), ..., (id_{u_r}, ℓ_r, id_{p_r}) to L_v.
            Select (id_{u_j}, ℓ_j, id_{p_j}) from L_v according to P over id_{u_i}
            Send (id_{u_j}, ℓ_j + 1, id_v) to all neighbors of v except p_j.
```

We now give more informal details of the test for cycle-freeness. By Lemma 6.1, with high probability G' is such, that if there is a vertex v in G' from which there is a shortest path to some w of length at least $T/2 = 10\log(n)/\epsilon$, then there is a closed path in G starting from v that contains a cycle of length $20\log(n)/\epsilon$. In the first phase of the algorithm, each vertex gets its name as its vertex id, and performs a BFS on the graph G' in the hope of finding a cycle. The BFS is performed using Algorithm 5, where the priority condition in the intermediate steps is selecting the BFS with the lowest origin id. If the cycle is present in a component of diameter at most $20\log(n)/\epsilon$ in G', then it is discovered during this BFS. To check if there is a cycle, each vertex v checks for the presence of tuples $(\mathrm{id}_u, \ell, \mathrm{id}_p)$ and $(\mathrm{id}_u, \ell', \mathrm{id}_{p'})$ in L_v.

If no cycle is discovered in this step, then we change the ids of the vertices to reflect the new priorities that will be used in Algorithm 5 for the second BFS phase. The new ids are set in the

following way: The id of each vertex v is now a tuple (ℓ, v), where ℓ is the largest depth at which v occurs in a BFS tree among all the breadth-first searches that reached v at the end of the first BFS phase.

We perform a BFS in G using Algorithm 5, where the priority condition is to select the BFS whose root has the lexicographically highest id. If there is some vertex with $\ell \geq 10\log(n)/\epsilon$, then the highest priority vertex is such a vertex, and by Lemma 6.1, the BFS starting from that vertex will detect a cycle in G.

Algorithm 6 gives a formal description of the tester for cycle-freeness.

Algorithm 6: Cycle-freeness test

Variables: L_v list of BFS tuples passing through v, vertex identifier id_v

`# Construct` G' `by deleting edges with probability` $\epsilon/2$.

for each *vertex* v **simultaneously**
- For each neighbor $u < v$, mark the edge $e = (u, v) \in G$ with probability $\epsilon/2$ for deletion.
- Send each marked edge $e = (u, v)$ to its corresponding u.
- Set $\mathrm{id}_v = v$.

for each *vertex* v **simultaneously**
- Delete all edges incident on v that have been marked for deletion.

`# Search for cycles in small diameter components.`

use *Algorithm 5* **to**
- perform BFS on G' for $20\log(n)/\epsilon$ steps, with the priority condition being choosing the BFS with the lowest root id.

for each *vertex* v **simultaneously**
- If L_v contains two tuples $(\mathrm{id}_u, \ell, \mathrm{id}_p)$ and $(\mathrm{id}_u, \ell', \mathrm{id}_{p'})$, output `reject`.
- Set $\mathrm{id}_v = (\ell_j, v)$ where ℓ_j is the highest among all tuples $(\mathrm{id}_{u_i}, \ell_i, \mathrm{id}_{p_i})$ in L_v.

use *Algorithm 5* **to**
- perform BFS on G for $10\log(n)/\epsilon$ steps, with the priority condition being choosing the BFS with the lexicographically highest root id.

for each *vertex* $v \in G$ **simultaneously**
- If L_v contains two tuples $(\mathrm{id}_u, \ell_j, \mathrm{id}_p)$ and $(\mathrm{id}_u, \ell', \mathrm{id}_{p'})$, output `reject`.
- output `accept`, if v did not output `reject` earlier.

We now prove the correctness of the algorithm.

Theorem 6.3. *Algorithm 6 is a distributed ϵ-test in the general graph model for the property of being cycle-free, that requires $O(\log(n)/\epsilon)$ rounds.*

Proof. Notice that a vertex in Algorithm 6 outputs `reject` only when it detects a cycle. Therefore, if G is cycle-free, then every vertex outputs `accept` with probability 1.

Suppose that G is ϵ-far from being cycle-free. Notice that, with probability at least $1-1/n^3$, the assertion of Lemma 6.1 holds. Furthermore, from Claim 6.2, we know that G' is $\epsilon/4$-far from being cycle-free, with probability $1 - e^{-\epsilon^2 m/32}$, and hence contains at least one cycle. This cycle could be in a component of diameter less than $20\log(n)/\epsilon$, or it could be in a component of diameter at least $20\log(n)/\epsilon$ in G'. We analyse the two cases separately.

Suppose there is a cycle in a component C of G' of diameter at most $20\log(n)/\epsilon$. Let u be the vertex with the smallest id in C. In Algorithm 6, the BFS starting at u is always propagated at any intermediate vertex due to the priority condition. Furthermore, since the diameter of C is at most $20\log(n)/\epsilon$, this BFS reaches all vertices of C. Hence, this BFS detects the cycle and at least one vertex in C outputs `reject`.

On the other hand, if the cycle of G is present in a component in G' of diameter at least $20\log(n)/\epsilon$, then after Step 11 of the algorithm, each vertex v gets the length of the longest path from the origin, among all the BFSs that reached v, as the first component of its id. The vertex v that gets the lexicographically highest id in the component has a vertex w that is at least $10\log(n)/\epsilon$ away in G', since the radius of the component is at least half the diameter. Therefore, by Lemma 6.1, it is part of a cycle of length at most $20\log(n)/\epsilon$ in G. Hence, the vertex with the highest priority in the BFS on G is a vertex u that has a vertex at a distance of at least $10\log(n)/\epsilon$ in G', and there is a closed path through u that contain a simple cycle of length at most $20\log(n)/\epsilon$. At least one vertex on this simple cycle will output `reject` when Algorithm 6 is run on G.

The number of rounds is $O(\log(n)/\epsilon)$, since Algorithm 6 performs two breadth-first searches in the graph with this number of rounds. □

7 Lower bounds for testing bipartiteness and cycle-freeness

In this section, we prove that any distributed algorithm for ϵ-testing bipartiteness or cycle-freeness in bounded-degree graphs requires $\Omega(\log n)$ rounds of communication[9]. We construct bounded-degree graphs that are ϵ-far from being bipartite, such that all cycles are of length $\Omega(\log n)$. We argue that any distributed algorithm that runs in $O(\log n)$ rounds does not detect a witness for non-bipartiteness. We also show that the same construction proves that every distributed algorithm for ϵ-testing cycle-freeness requires $\Omega(\log n)$ rounds of communication. Formally, we prove the following theorem.

Theorem 7.1. *Any distributed* $1/100$*-test for the property of being bipartite requires* $\Omega(\log n)$ *rounds of communication.*

To prove Theorem 7.1, we show the existence of a graph G' that is far from being bipartite, but all of its cycles are of at least logarithmic length. Since in T rounds of a distributed algorithm, the output of every vertex cannot depend on vertices that are at distance greater than T from it, no vertex can detect a cycle in G' in less than $O(\log n)$ rounds, which proves Theorem 7.1. To prove the existence of G' we use the probabilistic method with alterations, and prove the following.

Lemma 7.2. *Let G be a random graph on n vertices where each edge is present with probability $1000/n$. Let G' be obtained by removing all edges incident with vertices of degree greater than 2000, and one edge from each cycle of length at most $\log n/\log 1000$. Then, with probability at least $1/2 - e^{-100} - e^{-n}$, G' is $1/100$-far from being bipartite.*

Since a graph that is ϵ-far from being bipartite is also ϵ-far from being cycle-free, we immediately obtain the same lower bound for testing cycle-freeness, as follows.

Theorem 7.3. *Any distributed* $1/100$*-test for the property of being cycle-free requires* $\Omega(\log n)$ *rounds of communication.*

[9]Our lower bound applies even to the less restricted LOCAL model of communication, which does not limit the size of the messages.

The rest of this section is devoted to proving Lemma 7.2. We need to show three properties of G': (a) that it is far from being bipartite, (b) that it does not have small cycles, and (c) that its maximum degree is bounded. We begin with the following definition, which is similar in spirit to being far from satisfying a property and which will assist us in our proof.

Definition 7.4. *A graph G is* k-removed from being bipartite *if at least k edges have to be removed from G to make it bipartite.*

Note that a graph G with maximum degree d is ε-far from being bipartite if it is εdn-removed from being bipartite.

Let G be a random graph on n vertices where for each pair of vertices, an edge is present with probability $1000/n$, independently of other pairs. The expected number of edges in the graph is $500(n-1)$. Since the edges are sampled independently with probability $1000/n$, by the Chernoff Bound from Fact 2.4, with probability at least $1 - e^{-10n}$ the graph has at least $400n$ edges. We now show that G is far from being bipartite, with high probability.

Lemma 7.5 (**far from being bipartite**). *With probability at least $1 - e^{-199n}$, G is $20n$-far from being bipartite.*

Proof. Fix a bipartition (L, R) of the vertex set of G such that $|L| \geq n/2$. For each pair of vertices $u, v \in L$, let $X_{u,v}$ be a random variable which is 1 if the edge (u, v) is present in G and 0 otherwise. Its expected value is $E[X_{u,v}] = 1000/n$. The random variable $X = \sum_{u,v \in L} X_{u,v}$ counts the number of edges within L. By the linearity of expectation, $E[X] \geq \binom{n/2}{2} 1000/n \geq 30n$. Since the random variables $X_{u,v}$ are independent, by the Chernoff Bound from Fact 2.4, we have that $\Pr[X < 20n] \leq \exp(-200n)$. Therefore, with probability at least $1 - \exp(-200n)$, there are at least $20n$ edges within L. The total number of such bipartitions of G is at most 2^{n-1}. Taking a union bound over all such bipartitions, the probability that at least one of the bipartitions contains less than $20n$ edges within its L side is at most $\exp(-199n)$, and the lemma follows. □

The expected degree of a vertex v in G is $1000(1-1/n)$. Therefore, by the Chernoff Bound from Fact 2.4, the probability that the degree of v is greater than 2000 is at most $\exp(-300(1-1/n))$. We now show that, with sufficiently high probability, the number of edges that are incident with high degree vertices is small. We can remove all such edges to obtain a bounded-degree graph that is still far from being bipartite.

Lemma 7.6 (**mostly bounded degrees**). *With probability at least $1 - e^{-100}$, there are at most n edges that are incident with vertices of degree greater than* 2000 *in G.*

Proof. For a pair u, v of vertices, the probability that there is an edge between them and that one of u or v is of degree greater than 2000 is $\Pr[(u, v) \in E] \cdot \Pr[u \text{ or } v \text{ has degree } \geq 2000 | (u, v) \in E]$. This is at most $(1000/n) \cdot 2 \cdot \exp(-300(1-1/n))$. Therefore, the expected number of edges that are incident with a vertex of degree greater than 2000 is at most $1000n \cdot \exp(-300(1-1/n))$. By Markov's inequality, the probability that there are at least n edges that are incident with vertices of degree greater than 2000 is at most $1000 \cdot \exp(-300(1-1/n))$. This completes the proof of the lemma. □

We now bound the number of cycles of length at most $O(\log n)$ in the graph G.

Lemma 7.7 (**few small cycles**). *With probability at least* $1/2$*, there are at most* $2n$ *cycles of length at most* $\log n/\log 1000$ *in* G*.*

Proof. For any k fixed vertices, the probability that there is a cycle among the k vertices is at most $k!(1000/n)^k$. Therefore the expected number of cycles in G of length at most k is at most 1000^k. For $k = \log n/\log 1000$, this means that the expected number of cycles in G of length at most $\log n/\log 1000$ is n. Therefore, with probability at least $1/2$ there are at most $2n$ cycles of length at most $\log n/\log 1000$ in G. □

We are now ready to prove Lemma 7.2, completing our lower bounds. Intuitively, since G does not contain many high degree vertices and many small cycles, removing them to obtain G' only changes the distance from being bipartite by a small term.

Proof. With probability $1 - e^{-n}$, there are at least $400n$ edges in G and by Lemma 7.5 G is $20n$-removed from being bipartite. By Lemma 7.6, with probability at least $1 - e^{-100}$, there are at most n edges incident with vertices of degree greater than 2000 and by Lemma 7.7 with probability at least $1/2$ there are at most $2n$ cycles of length at most $\log n/\log 1000$. Hence, with probability at least $1/2 - e^{-100} - e^{-n}$, G' is a graph with degree at most 2000, that is $17n$-removed from being bipartite. Therefore, G' is $1/100$-far from being bipartite. □

8 Discussion

This paper initiates a thorough study of distributed property testing. It provides an emulation technique for the dense graph model and constructs fast distributed algorithms for testing triangle-freeness, cycle-freeness and bipartiteness. We also present lower bounds for both bipartiteness and triangle freeness.

This work raises many important open questions, the immediate of which is to devise fast distributed testing algorithms for additional problems. One example is testing freeness of other small subgraphs. More ambitious goals are to handle dynamic graphs, and to find more general connections between testability in the sequential model and the distributed model. Finally, there is fertile ground for obtaining additional lower bounds in this setting, in order to fully understand the complexity of distributed property testing.

References

[1] Noga Alon, Chen Avin, Michal Koucký, Gady Kozma, Zvi Lotker, and Mark R. Tuttle. Many random walks are faster than one. *Combinatorics, Probability & Computing*, 20(4):481–502, 2011.

[2] Noga Alon, Tali Kaufman, Michael Krivelevich, and Dana Ron. Testing triangle-freeness in general graphs. *SIAM J. Discrete Math.*, 22(2):786–819, 2008.

[3] Noga Alon and Michael Krivelevich. Testing k-colorability. *SIAM J. Discrete Math.*, 15(2):211–227, 2002.

[4] Noga Alon and Asaf Shapira. A characterization of the (natural) graph properties testable with one-sided error. *SIAM J. Comput.*, 37(6):1703–1727, 2008.

[5] Heger Arfaoui, Pierre Fraigniaud, David Ilcinkas, and Fabien Mathieu. Distributedly testing cycle-freeness. In *Graph-Theoretic Concepts in Computer Science - 40th International Workshop, WG 2014, Nouan-le-Fuzelier, France, June 25-27, 2014. Revised Selected Papers*, pages 15–28, 2014.

[6] Mor Baruch, Pierre Fraigniaud, and Boaz Patt-Shamir. Randomized proof-labeling schemes. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 315–324, 2015.

[7] Michael A. Bender and Dana Ron. Testing properties of directed graphs: acyclicity and connectivity. *Random Struct. Algorithms*, 20(2):184–205, 2002.

[8] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. *J. Comput. Syst. Sci.*, 47(3):549–595, 1993.

[9] Zvika Brakerski and Boaz Patt-Shamir. Distributed discovery of large near-cliques. *Distributed Computing*, 24(2):79–89, 2011.

[10] Keren Censor-Hillel, Petteri Kaski, Janne H. Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. Algebraic methods in the congested clique. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 143–152, 2015.

[11] Maria Chudnovsky, Neil Robertson, Paul Seymour, and Robin Thomas. The strong perfect graph theorem. *ANNALS OF MATHEMATICS*, 164:51–229, 2006.

[12] Danny Dolev, Christoph Lenzen, and Shir Peled. "tri, tri again": Finding triangles and small subgraphs in a distributed setting - (extended abstract). In *Distributed Computing - 26th International Symposium, DISC 2012, Salvador, Brazil, October 16-18, 2012. Proceedings*, pages 195–209, 2012.

[13] Andrew Drucker, Fabian Kuhn, and Rotem Oshman. The communication complexity of distributed task allocation. In *ACM Symposium on Principles of Distributed Computing, PODC '12, Funchal, Madeira, Portugal, July 16-18, 2012*, pages 67–76, 2012.

[14] Andrew Drucker, Fabian Kuhn, and Rotem Oshman. On the power of the congested clique model. In *ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014*, pages 367–376, 2014.

[15] Paul Erdös. Graph theory and probability. *canad. J. Math*, 11:34G38, 1959.

[16] Laurent Feuilloley and Pierre Fraigniaud. Survey of distributed decision. *Bulletin of the EATCS*, 119, 2016.

[17] Eldar Fischer. The art of uninformed decisions: A primer to property testing. *Current Trends in Theoretical Computer Science: The Challenge of the New Century*, I:229–264, 2004.

[18] Klaus-Tycho Foerster, Thomas Luedi, Jochen Seidel, and Roger Wattenhofer. Local checkability, no strings attached. In *Proceedings of the 17th International Conference on Distributed Computing and Networking, ICDCN 2016, Singapore, January 4-7, 2016.*, 2016.

[19] Jacob Fox. A new proof of the graph removal lemma. *CoRR*, abs/1006.1300, 2010.

[20] Mohsen Ghaffari, Fabian Kuhn, and Hsin-Hao Su. Manuscript. 2016.

[21] Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45(4):653–750, 1998.

[22] Oded Goldreich and Dana Ron. A sublinear bipartiteness tester for bounded degree graphs. *Combinatorica*, 19(3):335–373, 1999.

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

[24] Oded Goldreich and Dana Ron. Algorithmic aspects of property testing in the dense graphs model. In *Property Testing - Current Research and Surveys [outgrow of a workshop at the Institute for Computer Science (ITCS) at Tsinghua University, January 2010]*, pages 295–305, 2010.

[25] Oded Goldreich and Luca Trevisan. Three theorems regarding testing graph properties. *Random Struct. Algorithms*, 23(1):23–57, 2003.

[26] Mika Göös, Juho Hirvonen, Reut Levi, Moti Medina, and Jukka Suomela. Non-local probes do not help with graph problems. *CoRR*, abs/1512.05411, 2015.

[27] Juho Hirvonen, Joel Rybicki, Stefan Schmid, and Jukka Suomela. Large cuts with local algorithms on triangle-free graphs. *CoRR*, abs/1402.2543, 2014.

[28] Stephan Holzer and Roger Wattenhofer. Optimal distributed all pairs shortest paths and applications. In *Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing*, PODC '12, pages 355–364, New York, NY, USA, 2012. ACM.

[29] Taisuke Izumi and François Le Gall. Triangle finding and listing in CONGEST networks. In *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 381–389, 2017.

[30] Jarkko Kari, Martín Matamala, Ivan Rapaport, and Ville Salo. Solving the induced subgraph problem in the randomized multiparty simultaneous messages model. In *Structural Information and Communication Complexity - 22nd International Colloquium, SIROCCO 2015, Montserrat, Spain, July 14-16, 2015, Post-Proceedings*, pages 370–384, 2015.

[31] Tali Kaufman, Michael Krivelevich, and Dana Ron. Tight bounds for testing bipartiteness in general graphs. *SIAM J. Comput.*, 33(6):1441–1483, 2004.

[32] Amos Korman, Shay Kutten, and David Peleg. Proof labeling schemes. *Distributed Computing*, 22(4):215–233, 2010.

[33] Christoph Lenzen and David Peleg. Efficient distributed source detection with limited bandwidth. In *ACM Symposium on Principles of Distributed Computing, PODC '13, Montreal, QC, Canada, July 22-24, 2013*, pages 375–382, 2013.

[34] Michael Mitzenmacher and Eli Upfal. *Probability and computing - randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

[35] Michal Parnas and Dana Ron. Testing the diameter of graphs. *Random Struct. Algorithms*, 20(2):165–183, 2002.

[36] Michal Parnas and Dana Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theor. Comput. Sci.*, 381(1-3):183–196, 2007.

[37] David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics, 2000.

[38] Seth Pettie and Hsin-Hao Su. Distributed coloring algorithms for triangle-free graphs. *Inf. Comput.*, 243:263–280, 2015.

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

[40] Dana Ron. Algorithmic and analysis techniques in property testing. *Foundations and Trends in Theoretical Computer Science*, 5(2):73–205, 2009.

[41] Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM J. Comput.*, 25(2):252–271, 1996.

[42] Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of distributed approximation. *SIAM J. Comput.*, 41(5):1235–1265, 2012.

[43] Atish Das Sarma, Danupon Nanongkai, Gopal Pandurangan, and Prasad Tetali. Distributed random walks. *J. ACM*, 60(1):2, 2013.