Property Testing

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

1 Introduction

In broad terms, property testing is the study of the following class of problems:

> Given the ability to perform (local) queries concerning a particular object (e.g., a function, or a graph), the task is to determine whether the object has a predetermined (global) property (e.g., linearity or bipartiteness), or is far from having the property. The task should be performed by inspecting only a small (possibly randomly selected) part of the whole object, where a small probability of failure is allowed. [1]

In order to define a property testing problem, we need to specify the type of queries that the testing algorithm can perform, and a distance measure between objects. The latter is required in order to define what it means that the object is *far* from having the property. We assume that the algorithm is given a *distance* parameter ϵ. The algorithm should accept with probability at least 2/3 every object that has the property, and should reject with probability at least 2/3 every object that has distance more than ϵ (according to the selected distance measure) from any object having the property. [2]

When the object in question is a function $f : \mathrm{X} \to \mathrm{Y}$ (for finite X and Y), then the natural form of queries is: "What is the value of $f(x)$?" for any choice of $x \in \mathrm{X}$. A natural distance measure between two functions is the fraction of domain elements on which the functions differ. Thus, for example, when testing linearity of functions, the testing algorithm can obtain the value of the tested function f on x's of its choice. If f is a linear function then the algorithm should accept it with probability at least 2/3. However, if the value of f must be modified on more than an ϵ fraction of the domain elements so that it becomes linear, then the algorithm should reject it with probability 2/3. (If f is close to being linear, then the algorithm can either accept or reject.)

When studying graph properties, the form of queries, and in some cases the distance measure, depend on the graph representation. For instance, if graphs are represented by their adjacency matrix, then the queries are of the form: "Is there an edge between vertex v and vertex u"? The distance measure between graphs in this case is the fraction of adjacency-matrix entries on which

[1] A more general definition in which the algorithm cannot necessarily perform queries but rather is given "samples" from the object distributed according to some fixed (possibly unknown) distribution, will be discussed subsequently.

[2] The choice of success probability 2/3 is of course arbitrary, and any constant strictly greater than 1/2 can be used. In order to obtain success probability of $1-\delta$ for any $\delta < 1/3$, the algorithm should be executed $\Theta(\log(1/\delta))$ times, and the majority output taken.

the two graphs differ. Thus, for example, when testing bipartiteness[3] of graphs represented by their adjacency matrix, then the algorithm is only allowed to accept graphs $G = (V, E)$ for which at most $\epsilon|V|^2$ edges should be removed so that they become bipartite.

In this tutorial we mainly focus on testing graph properties, though we shall briefly survey other results as well. For sake of the presentation, some of the following motivational discussion refers to graphs. However, much of it is relevant to testing other types of objects.

1.1 Motivation

The task of testing a certain graph property is a relaxation of the task of deciding *exactly* whether a graph has the property. Namely, an exact decision procedure is required to accept every graph that has the property and reject every graph that *does not have* the property. A testing algorithm is still required to accept every graph that has the property, but is only required to reject every graph that *is far from having* the property. While relaxing the task, we expect the algorithm to observe only a small part of the graph and to run significantly faster than any exact decision procedure. Specifically, we aim at spending time that is *sub-linear in* or even *independent of* the size of the graph.

In fact, as we shall elaborate later, many graph properties have very fast property testing algorithms whose query complexities do not depend at all on the size of the graph. This should be put in contrast to known lower bounds on the complexity of exactly deciding graph properties. Rivest and Vuillemin [RV76] showed that any deterministic procedure for deciding any non-trivial monotone N-vertex graph property must examine $\Omega(N^2)$ entries in the adjacency matrix representing the graph, thus resolving the Aanderaa–Rosenberg Conjecture [Ros73]. The query complexity of *randomized* decision procedures was conjectured by Yao to be also $\Omega(N^2)$. Progress towards proving this conjecture was made by Yao [Yao87], King [Kin91] and Hajnal [Haj91] culminating in an $\Omega(N^{4/3})$ lower bound.

1.1.1 A Tradeoff Between Accuracy and Efficiency

It follows from the above discussion that Property Testing trades *accuracy* for *efficiency*, where accuracy is measured in terms of the distance parameter ϵ. Since we expect the running time of the algorithm to increase as a function of $1/\epsilon$, the algorithm is more accurate as ϵ decreases, but its running time increases.

This paradigm may be useful is several scenarios.

1. A fast property tester can be used to speed up a slow exact decision procedure as follows. Before running the decision procedure, run the tester. If the tester rejects, then we know with high confidence that the property does not hold and it is unnecessary to run the (slower) decision procedure. In fact, it is often the case that when the testing algorithm rejects, it provides a *witness* that the graph does not have the property. On the other hand, if the tester accepts, then an exact decision procedure will determine whether the property is close to holding or actually holds. We thus save time in applications where *typical* graphs are either good (have the property), or very bad (far from having the property).
2. Furthermore, if it is *guaranteed* that graphs are either good or very bad then we may not even need the exact algorithm at all.

[3] A graph is bipartite if its set of vertices can be partitioned into two disjoint subsets such that there are not edges within the subsets.

3. There are circumstances in which knowing that a property nearly holds is good enough and consequently exact decision is unnecessary.
4. In some cases (e.g., connectivity) there are algorithms for "fixing" the graph (that is, modifying it so that it will have the property). If the graph is accepted then we know with high confidence that the number of required modification is not too large, and assuming there is a cost associated with each edge modification, the total cost is not too large.
5. The graph may be too large to fully scan, so one *must* make a decision without observing the whole graph.
6. It may be NP-hard to answer the question exactly, and so (even if scanning the graph is feasible), some form of approximation is inevitable

1.1.2 Relation to Other Notions of Approximation

In the study of approximation algorithms, and in particular approximation algorithms for graph optimization problems, the following is the dominant approach. For each instance (graph) there is a set of feasible solutions (e.g., subsets of vertices that form cliques). With each feasible solution there is an associated cost or utility (e.g., the size of the clique). The goal is to approximate the value of the minimum cost or maximum utility of a feasible solution. In some cases, the goal is to actually *find* a solution whose cost or utility is close to optimal.

Property testing is related to an alternative notion of approximation, namely that of *dual* approximation [HS87, HS88]. Instead of approximating the maximum utility of a feasible solution, dual approximation tries to approximate the distance (in terms of edge-modifications) to having a feasible solution with a certain utility. In the example of the clique for instance, the goal is to approximate the number of edges that must be added in order to obtain a clique with a certain size.

The preferred notion of approximation is naturally dependent on the context in which it is applied. We note that in some cases the two notions coincide. This is true for example in the case of Max-Cut. In the more standard approach, the goal is to approximate the size of a maximum cut (that is, the maximum number of edges crossing any two-way partition). In the dual approach, the goal is to approximate the number of edges that should be added in order to obtain some cut with a given size. In this case, an algorithm for the latter problem can be used to solve the former problem.

1.1.3 Property Testing, Program Testing and PCP

Property testing of functions was first explicitly defined by Rubinfeld and Sudan [RS96] in the context of *program testing*. The goal of a program testing algorithm is to test whether a given program computes a specified function. Here one may choose to test that the program satisfies a certain property (which the function holds) before checking that it computes the specified function itself. This paradigm has been followed both in the theory of program testing [BLR93, RS96, Rub99], and in practice where often programmers first test their programs by verifying that the programs satisfy properties that are known to be satisfied by the function they compute.

Property testing also emerges naturally in the context of probabilistically checkable proofs (PCP). In this context the property being tested is whether the function is a codeword of a specific code. This paradigm, explicitly introduced in [BFLS91], has shifted from testing codes defined by low-degree polynomials [BFL91, BFLS91, FGL$^+$96, AS98, ALM$^+$98] to testing Hadamard codes [ALM$^+$98, BGLR93, BS94, BCH$^+$95, Kiw96, Tre98], and to testing the "long code" [BGS98, Hås96, Hås97, Tre98].

We note that in both the above contexts, the properties tested were algebraic.

1.1.4 Property Testing and Learning

One of the initial motivations for the study of property testing is its relation to Computational Learning Theory. Let the objects we are interested in be functions (in particular, boolean functions), and consider the following variant of our initial description of property testing: Instead of allowing the algorithm to query the tested function f on inputs of its choice, it is provided with a *labeled sample* $\{(x^1, f(x^m)), \ldots, (x^m, f(x^m))\}$, where the x^i's are distributed according to some fixed but unknown distribution D over the domain X. In this case, distance between functions is measured with respect to the distribution D. That is, the distance between functions g and h is $\Pr_{x \sim D}[g(x) \neq h(x)]$.

The above definition of property testing is inspired by the Probably Approximately Correct (PAC) learning model proposed by Valiant [Val84]. In the PAC model, a learning algorithm is given a random sample labeled by an unknown function f as defined above, and is required to output (with high probability) a *hypothesis* h that approximates f well. That is, $\Pr_{x \sim D}[h(x) \neq f(x)] \leq \epsilon$. In the standard PAC model, it is assumed that f belongs to a known class of functions F, and it is either required that h belong to F as well, or to some specified hypothesis class $H \supseteq F$.[4]

Thus, property testing as defined in this subsection can be viewed as a relaxation of PAC learning – instead of requiring a good approximation of the function f, we only ask whether such a good approximation *exists* in a given class (the class of functions having the tested property). Our original definition of property testing which allows queries and defines distance with respect to the uniform distribution can be seen as a relaxation of a variant of the PAC model: Learning with queries under the uniform distribution.

Given the above view, it is reasonable to expect that for some classes of functions (properties), testing can be done much more efficiently than learning (in terms of sample/query complexity and/or running time). This is true for example in the case of linear functions [BLR93], multivariate polynomials [RS96], and monotone functions [GGL$^+$00]. If we have fast testing algorithms that require relatively small samples, we may be able to use such algorithms in the context of learning. Namely, we can use them to choose between alternative hypothesis representations without actually incurring the expense of running the corresponding learning algorithms. For example, suppose that we are considering running C4.5 (a fast algorithm) to find a decision tree hypothesis (a relatively weak representation). But we may also want to consider running backpropagation (a slow algorithm) to find a multilayer neural network (a relatively powerful representation, requiring more data, but with perhaps greater accuracy). Ideally, we would like a fast, low-data test that informs us whether this investment would be worthwhile.

1.2 Testing Graph Properties

The study of *testing graph properties* was initiated by Goldreich Goldwasser and Ron [GGR98]. As noted previously, the precise definition of testing graph properties is dependent on the repre-

[4] In the *agnostic* model [KSS94], nothing is assumed about f, and so the distance between h and f is required to be not much larger than the distance between f and the closest function in H.

sentation of graphs. There are two standard representations of graphs, *adjacency matrices* and *incidence lists*, and we discuss the related testing models below. We restrict our attention to *undirected* graphs.

- *Adjacency-Matrix Model.* Goldreich *et. al.* [GGR98] consider the adjacency-matrix representation of graphs, where the testing algorithm is allowed to probe into the matrix. That is, the algorithm can query whether there is an edge between any two vertices of its choice. In this representation the distance between graphs is the *fraction of entries* in the adjacency matrix on which the two graphs differ. By this definition, for a given distance parameter ϵ, the algorithm should reject every graph that requires more than $\frac{\epsilon}{2} \cdot |\mathrm{V}|^2$ edge modifications in order to acquire the tested property (the factor of $\frac{1}{2}$ is because each edge is represented twice in the matrix). This representation is most appropriate for dense graphs, and the results for testing in this model are most meaningful for such graphs.

- *Incidence-Lists Models.* Goldreich and Ron [GR97] consider the incidence-lists representation of graphs. In the model they consider, graphs are represented by lists of *length* d, where d is a bound on the degree of the graph. Here the testing algorithm can query, for every vertex v and index $i \in \{1, \ldots, d\}$, which vertex is the i'th neighbor of v. If no such neighbor exists then the answer is '0'. Analogously to the adjacency-matrix model, the distance between graphs is defined to be the fraction of entries on which the graphs differ according to this representation. Since the total number of incidence-list entries is $d \cdot |\mathrm{V}|$, a graph should be rejected if the number of edge modifications required in order to obtain the property is greater than $\frac{\epsilon}{2} \cdot d|\mathrm{V}|$. (Once again, the factor of $\frac{1}{2}$ is because each edge (u, v) is represented both as an entry $[u, i]$ and as an entry $[v, j]$).

  A variant of the above model allows the incidence lists to be of varying lengths [PR99a]. In such a case, the distance between graphs is defined with respect to the total number of edges in the graph (or an upper bound on this number). This model is suitable for testing graphs that are not dense but for which there is large variance in the degrees of the graph vertices. Furthermore, some problems are more interesting in this model, in the sense that removing the degree bound makes them less restricted. For example, testing whether a graph has a diameter of at most a bounded size, is less interesting in the bounded degree model, since a bound d on the degree implies a lower bound on the diameter of a graph. Intuitively, testing in this model is at least as hard as testing in the bounded-degree model described above, and in fact in some cases it is strictly harder.

We note that for both the adjacency-matrix model and the bounded-degree incidence-lists model, the representations can be viewed as *functional* representations of graphs. Namely, in the first model it is a function from all $|\mathrm{V}|^2$ vertex-pairs to $\{0, 1\}$, and in the second case it is a function from all $|\mathrm{V}| \cdot d$ pairs of vertex and index, to the set of vertices. Furthermore, the definition of distance between graphs in these models is determined by the representation: it is the symmetric difference between the functions representing the graphs, divided by the size of the domain of the functions. The unbounded-degree incidence-lists model is not a functional representation, and the notion of distance is divorced from the representation.

1.2.1 Techniques

The applicable techniques depend on the choice of representation. A central technique that is used for the adjacency-matrix representation is *random sampling*. Specifically, the algorithm randomly selects a small set U of vertices from the graph G, finds the edges interconnecting the vertices, and determines whether the property holds (or "almost holds") for the small subgraph induced by U. If so, then the algorithm accepts. If not, the algorithm rejects. It is typically straightforward to show that any graph having the property is accepted (always, or with high probability). The crux of the proof is in showing that a graph that is far from having the property is rejected with high probability.

The following general analysis technique is often used for proving the above claim. The sample is viewed as consisting of two disjoint subsamples. The first sample is viewed as implicitly inducing certain *constraints* on all other graph vertices. These constraints are such that if the graph is far from having the property then many vertices (or pairs of vertices) do not obey the constraints. The function of the second part of the sample is to show evidence to the unsatisfied constraints.

The incidence-lists representation requires a different set of techniques, more applicable to sparse graphs. Specifically, because the graphs have few edges, a small random sample of vertices typically has no edges internal to the sample, that is, it is just the empty graph. Thus, algorithms for this setting use additional techniques besides pure random sampling. In particular, some algorithms apply various forms of exhaustive local search [GR97, PR99a] (such as performing a breadth-first-search until a particular number of vertices are observed). Other algorithms use random walks starting from randomly selected vertices [GR99].

1.3 Testing Other Properties

As mentioned previously, there is a body of work dealing with testing of algebraic properties of functions (see [BLR93, RS96, Rub99, EKK$^+$98]). Testing monotonicity of functions was studied in [GGL$^+$00, DGL$^+$99, EKK$^+$98]. Testing properties defined by regular languages was studied in [AKNS99], and recently Newman [New00] extended this result to bounded-width branching programs. Properties of geometric objects were studied in [EKK$^+$98, CSZ00], and algorithms for testing of clustering were given in [ADPR00]. Metric properties were recently considered in [PR00].

Organization

The rest of the paper consists of three sections. The first (and main) two sections deal with testing graph properties, and the last with other properties. In particular, the first section is dedicated to the adjacency-matrix model. We provide a summary of results in this model, and present in detail the algorithm for testing bipartiteness and its analysis. We also sketch the ideas for testing whether a graph has a clique of a given size, and give some further details for a few other results. The second section is dedicated to testing in the incidence-lists model. Here too we give a summary of the results in this model, and provide more details for two properties: k-connectivity and bipartiteness. The last section gives a summary of other property-testing results that do not deal with graphs.

2 Testing Graph Properties in the Adjacency Matrix Model

2.1 Definitions

We consider undirected, simple graphs (no multiple edges or self-loops). For a graph G, we denote by V(G) its vertex set and by E(G) its edge set (whenever it is clear from the context, we shall simply use V and E). The size of V(G) is denoted by N. We assume, without loss of generality, that

$V(G) = \{1, ..., N\}$. Graphs are represented by their (symmetric) adjacency matrix. Thus, graphs are associated with the (symmetric) boolean function f_G corresponding to this matrix. That is, $f_G(u,v) = 1$ if $(u,v) \in E(G)$, and $f_G(u,v) = 0$ otherwise. For two (not necessarily disjoint) sets of vertices, X_1 and X_2, we let $E(X_1, X_2) \stackrel{\rm def}{=} \{(u,v) \in E(G) : u \in X_1, v \in X_2\}$.

The distance between two N-vertex graphs G_1 and G_2 is defined as the number of unordered pairs $(u,v) \in [N]^2$ such that $f_{G_1}(u,v) \neq f_{G_2}(u,v)$, divided by the total number of pairs,[5] N^2.

Definition 2.1.1 *For any graph property $\mathcal{P}$, and $0 \leq \epsilon \leq 1$, we say that a graph G is ϵ-far from (having) property $\mathcal{P}$, if it has distance greater than ϵ from* every *graph that has the property. Otherwise it is ϵ-close.*

Definition 2.1.2 *A property testing algorithm for property $\mathcal{P}$ working in the adjacency matrix model is given a distance parameter ϵ and can perform queries concerning the existence of edges between any pair of vertices of its choice. If the tested graph has the property, the algorithm should accept with probability at least 2/3, and if it is ϵ-far from having the property then the algorithm should reject with probability at least 2/3.*

2.2 Summary of Results

The following graph properties were studied in [GGR98] and were shown to have testing algorithms with query complexity $\text{poly}(1/\epsilon)$ and time complexity at most $\exp(\text{poly}(1/\epsilon))$. In what follows, N denotes the number of graph vertices.

- Bipartiteness. The algorithm has query complexity and running time $\tilde{O}(\epsilon^{-3})$.[6] Recently, Alon and Krivelevich [AK99] improved the analysis of the algorithm and obtained a bound of $\tilde{O}(\epsilon^{-2})$ on the query complexity and running time.
- k-colorability, $k \geq 3$. The algorithm has query complexity $\tilde{O}\left(k^4/\epsilon^6\right)$ and running time $\exp\left(\tilde{O}\left(k^2/\epsilon^3\right)\right)$. Recently, Alon and Krivelevich [AK99] improved the analysis of the algorithm and obtained a bound of $\tilde{O}\left(k^2/\epsilon^4\right)$ on the query complexity, and $\exp\left(\tilde{O}\left(k/\epsilon^2\right)\right)$ on the running time.
- ρ-Clique. The property is having a clique of size $\rho \cdot N$, where $0 < \rho < 1$ is a constant. The query complexity of the algorithm is $\tilde{O}\left(\rho^2/\epsilon^6\right)$ and the running time is $\exp\left(\tilde{O}\left(\rho/\epsilon^2\right)\right)$.
- ρ-Cut. The property is having a 2-way cut with ρN^2 crossing edges. The query complexity of the algorithm is $\tilde{O}\left(\epsilon^{-7}\right)$ and the running time is $\exp\left(\tilde{O}\left(\epsilon^{-3}\right)\right)$. The algorithm generalizes to k-way cuts, at a multiplicative cost of $O(\log^2(k))$ in the query complexity and in the exponent of the running time. The algorithm can also be modified to test ρ-Bisection. This property is similar to ρ-Cut except that the partition is to equal size subsets. The query complexity is $\tilde{O}\left(\epsilon^{-8}\right)$ and the running time is $\exp\left(\tilde{O}\left(\epsilon^{-3}\right)\right)$.

[5] In [GGR98] (and in the introduction) the distance was defined as the number of such *ordered* pairs (entries in the matrix) divided by N^2. While this seems more appropriate as N^2 is the number of ordered pairs, it implies that each undirected edge in the symmetric difference between the graphs is counted twice, causing slight cumbersomeness in the analysis of the algorithms. Thus, we have chosen here a less natural definition that makes the analysis later simpler.

[6] The $\tilde{O}(\cdot)$ notation, which is used for sake of succinctness, "hides" logarithmic factors (which in all our algorithms are at most quadratic).

NOTES

1. For all the above properties (except bipartiteness) it is very unlikely that there is a testing algorithm having running time $\text{poly}(1/\epsilon)$. If such an algorithm exists, by setting $\epsilon = 1/N$ one would be able to obtain an exact (randomized) decision procedure that runs in polynomial time, and this would imply that $\mathcal{NP} \subseteq \mathcal{BPP}$.

2. The bipartiteness and k-colorability algorithms have one sided error: they always accept graphs that have the property. Furthermore, whenever a graph is rejected, the algorithm supplies *evidence* that it does not have the property. Evidence is in the form of a small subgraph that is not bipartite/k-colorable. All other algorithms have two-sided error and this can be shown to be unavoidable within $o(N)$ query-complexity.

A testing algorithm for k-Colorability whose complexity is independent of N was already implicit in work of Alon *et. al.* [ADL+94]. They build on a constructive version of the Regularity Lemma of Szemerédi [Sze78] which they prove, and the complexity of the resulting testing algorithm is a tower of $\text{poly}(1/\epsilon)$ exponents.

Constructing (Good) Partitions For all the above properties, in case the graph has the desired property, the testing algorithm outputs some auxiliary information which allows to construct, in $\text{poly}(1/\epsilon) \cdot N$ time, a partition that approximately obeys the property. For example, for ρ-Clique, the algorithm will find a subset of vertices of size ρN, such that at most ϵN^2 edges need to be added so that it becomes a clique. In the case of ρ-Cut, the algorithm will construct a partition with at least $(\rho - \epsilon)N^2$ crossing edges. The basic idea is that the partition of the sample that caused the algorithm to accept is used to partition the whole graph. This idea was later also used by Frieze and Kannan [FK99] to obtain polynomial time approximation schemes for various problems, where they apply a relaxed constructive version of Szemerédi's Regularity Lemma [Sze78].

General Graph Partition Properties All the above properties are special cases of a class of graph partition properties. Each property in the class is parameterized by an integer k and by $k+k^2$ pairs of lower and upper bounds in the interval $[0, 1]$. A graph has the property if its vertices can be partitioned into k subsets having relative sizes within the designated upper and lower bounds, and such that the edge densities between the parts are within the required bounds as well. A more precise definition is given in Subsection 2.5.

Not surprisingly, generality has a price, and the testing algorithm for the above class of properties, presented in [GGR98], has query complexity $\left(\tilde{O}(k^2)/\epsilon\right)^{2k+8}$ and running time $\exp\left(\tilde{O}(k^2)/\epsilon\right)^{k+1}$.

First Order Graph Properties Alon, Fischer, Krivelevich and Szegedy [AFKS99], study the class of *first order* graph properties. These are properties that can be formulated by first order expressions about graphs. That is, expressions that contain quantifiers over vertices, boolean connectives, equality of vertices, and adjacency relations. They show that all first order graph expressions containing at most one quantifier, as well as all first order graph expression of the type "$\exists\forall$" (i.e., $\exists x_1, \ldots, x_t \; \forall y_1, \ldots, y_s \; A(x_1, \ldots, x_t, y_1, \ldots, y_s)$ where A is a quantifier-free first order graph expression and t and s are constants), can be tested with query complexity and running time independent of N. Once again generality has a (steep) price: the dependence on the distance parameter ϵ is either a tower of poly$(1/\epsilon)$ exponents or a tower of towers of poly$(1/\epsilon)$ exponents. They also prove that there exist first order graph expressions of the type "$\forall\exists$" that cannot be tested with query complexity and running time independent of N. In particular this is true of a natural property based on graph isomorphism, where the required number of queries is $\Omega(\sqrt{N})$.[7]

We give some more details in Subsection 2.6

Properties of Directed Graphs The adjacency-matrix model can be naturally extended to deal with directed graphs. The algorithm may perform queries of the form: "is there an edge *from* vertex u *to* vertex u", and distance between graphs is defined as the number of *ordered* pairs that are an edge in one graph and not in the other, divided by N^2.

Some properties of undirected graphs have analogies in directed graphs. Furthermore, in some cases the testing algorithms for undirected graphs can be extended to directed graphs. This is true for example in the case of ρ-Cut (see [GGR98, Sec. 10.1]). However, in other cases, the testing problems are quite different. This is true for example for the cycle-freeness property. In order for an undirected graph to be cycle-free it must be very sparse. Namely, it may contain at most $N-1$ edges. Hence, in order to test cycle-freeness of undirected graphs (in the adjacency-matrix model), all that is required is to roughly estimate the number of edges in the graph by sampling. This is not true of directed graphs. Namely, a directed graph may be very dense but still acyclic. Hence, for this property a non-trivial analysis is needed. In [BR00] it is shown that the "natural" algorithm that takes a sample of vertices and accepts or rejects based on the acyclicity of the induced subgraphs, is a testing algorithm. The required sample size is $O(\log(1/\epsilon)/\epsilon)$. The special case in which there is a directed edge (in some direction) between *every* pair of vertices, was treated in [EKK$^+$98] when dealing with total orders. They present an algorithm having complexity poly$(1/\epsilon)$.

2.3 Testing Bipartiteness

Recall that a graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ is *bipartite* if its set of vertices V can be partitioned into two (disjoint) subsets V_1 and V_2 so that there are no edges between vertices that belong to the same subset. If there is no such partition, then the graph is not bipartite. Deciding whether a graph is bipartite or not can be done in time linear in the number of graph edges by performing a Breadth First Search (BFS) starting from an arbitrary vertex.

What does it mean that a graph is ϵ-far from bipartite? For any two-way partition $(\mathrm{V}_1, \mathrm{V}_2)$ of V, we say that an edge $(u, v) \in \mathrm{E}$ is a *violating* edge with respect to $(\mathrm{V}_1, \mathrm{V}_2)$, if either $u, v \in \mathrm{V}_1$ or $u, v \in \mathrm{V}_2$. We say that $(\mathrm{V}_1, \mathrm{V}_2)$ is ϵ-*bad* if the number of violating edges with respect to $(\mathrm{V}_1, \mathrm{V}_2)$, is greater than ϵN^2, otherwise it is ϵ-good. Given the above definition, G is ϵ-far from bipartite if and only if *every* partition $(\mathrm{V}_1, \mathrm{V}_2)$ of V is ϵ-bad.

Suppose we fix a *particular* partition $(\mathrm{V}_1, \mathrm{V}_2)$. If we now take a sample of size $\Theta(1/\epsilon)$ of vertices (or pairs of vertices) and obtain the edges between them, then with high probability we shall see evidence to the badness of the partition in the form of a violating edge. Since in case G is ϵ-far from bipartite, all its partitions are ϵ-bad, the naive use of the above observation is to take a sufficiently large sample of vertices such that with high probability for *every* partition there exists an edge between vertices in the sample that violates the partition. Such a sample induces a subgraph that is necessarily not bipartite, and so the algorithm could simply run a BFS on the sample to detect this. Unfortunately, the number of all two-way partitions of V is exponential in N. Therefore, a

[7]Previous hardness results in [GGR98] only showed the *existence* of such hard-to-test properties (in $\mathcal{NP}$).

straightforward analysis of the above algorithm (which applies a probability union bound) would require that the size of the sample be logarithmic in the number of partitions, that is, linear in N.

Nonetheless, as we shall see below, the naive algorithm that simply takes a uniformly selected sample of vertices and checks whether the induced subgraph is bipartite does work, but requires a slightly more refined analysis.

Test-Bipartite

1. Uniformly and independently select $m = \Theta\left(\frac{\log(1/\epsilon)}{\epsilon^2}\right)$ vertices.
2. For every pair of vertices v and u selected, query whether there is an edge between v and u, thus obtaining the induced subgraph.
3. Perform a (BFS) to determine whether the subgraph induced by the sample is bipartite. If it is bipartite output accept otherwise output reject.

Theorem 1 *The algorithm* Test-Bipartite *is a testing algorithm for bipartiteness. In particular, if the graph is bipartite, then it is always accepted, and if it is ϵ-far from bipartite, then it is rejected with probability at least 2/3. Furthermore, whenever the algorithm rejects a graph, it outputs a* certificate *to the non-bipartiteness of the graph in form of a non-bipartite subgraph having* $O\left(\frac{\log(1/\epsilon)}{\epsilon^2}\right)$ *vertices.*

Proof: The first claim in the theorem, concerning bipartite graphs is obvious: if G is bipartite then *every* subgraph of G is bipartite, and so the graph is accepted for any choice of the sample. The heart of the proof is hence in the second claim, that is, showing that if a graph is ϵ-far from bipartite then with probability at least 2/3 over the choice of the sampled vertices, the resulting induced subgraph is not bipartite. We thus focus on the second part of the theorem, and assume from now on that G is ϵ-far from bipartite. Recall that this implies that for every two-way partition of V there are at least ϵN^2 violating edges. The rough outline of the proof is the following:

1. We shall view the sample of vertices as consisting of two parts, which we refer to as U and S. The set U consists of the first $m_1 = \Theta((\log(1/\epsilon))/\epsilon)$ vertices selected, and the set S of the latter $m_2 = \Theta((\log(1/\epsilon))/\epsilon^2)$ vertices. (Since the vertices are selected independently, repetitions may occur.)
2. We show that with probability at least 5/6 over the choice of U, it can be used to implicitly induce a relatively small number of partitions of the whole graph, that are in a way *consistent with* U (this notion will be clarified later).
3. We then show that with probability at least 5/6 over the choice of the second part of the sample, S, it will contain violating edges with respect to each of the partitions implicitly induced by U.
4. Putting the above two items together we conclude that with probability at least 2/3 over the choice of both parts of the sample, the induced subgraph is not bipartite.

We start with some definitions.

Definition 2.3.1 *A vertex v is* influential *if its degree in* G *is at least $\frac{\epsilon}{4}N$. Otherwise it is not influential.*

If a vertex v is not influential, then by definition, for every partition $(\mathrm{V}_1, \mathrm{V}_2)$ of V, the number of edges incident to v that are violating with respect to $(\mathrm{V}_1, \mathrm{V}_2)$ is at most $(\epsilon/4)N$. Furthermore, since there are at most N non-influential vertices, the total number of edges incident to non-influential vertices that violate some partition, is at most $(\epsilon/4)N^2$. Intuitively, the means that we shall "not rely" on non-influential vertices for giving us evidence to the fact that G is far from bipartite.

Definition 2.3.2 *For any vertex v and set of vertices* U*, we say that* U covers v *if v has at least one neighbor in* U*.*

We are now ready for our first lemma concerning the first part of the sample U.

Lemma 2.3.1 *With probability at least* $5/6$ *over the choice of the vertices in* U*, all but at most* $(\epsilon/4)N$ *influential vertices are covered by* U*.*

Proof: Consider any fixed influential vertex v. Recall that the vertices in U are selected uniformally and independently. Hence, the probability that U does not cover v, that is, that U contains none of the at least $(\epsilon/4)N$ neighbors of v is at most

$$\left(1-\frac{\epsilon}{4}\right)^{m_1} < \exp\left(\frac{\epsilon}{4}\cdot m_1\right)$$

If we set $m_1 = \frac{4}{\epsilon}\cdot\ln(24/\epsilon)$ then the above probability is at most $(\epsilon/24)$. Since there are at most N influential vertices, this implies that the expected number of influential vertices that are not covered by U is at most $(\epsilon/24)N$. By Markov's inequality, the probability that there are more than $(\epsilon/4)N$ such influential vertices, is at most $1/6$, as required. ■

We assume from now on that U in fact covers all but at most $(\epsilon/4)N$ of the influential vertices. Let C be the set of vertices in V that are covered by U and let R be the remaining vertices. (The sets C and R also contain the vertices of U, but the size of U should be thought of as negligible compared to $|\mathrm{C}\cup\mathrm{R}| = N$.) By our assumption, R contains at most $(\epsilon/4)N$ influential vertices, and possibly all non-influential vertices. Consider a fixed partition $(\mathrm{U}_1, \mathrm{U}_2)$ of U. Then $(\mathrm{U}_1, \mathrm{U}_2)$ can be used to induce a partition $(\mathrm{C}_1, \mathrm{C}_2)$ of C as follows: every vertex in C that has a neighbor in U_1, belongs to C_2, and all other vertices in C (that necessarily have a neighbor in U_2), belong to C_1. Let $(\mathrm{R}_1, \mathrm{R}_2)$ be an arbitrary partition of R (with the only restriction that $\mathrm{U}_1\cap\mathrm{R}\subseteq\mathrm{R}_1$ and $\mathrm{U}_2\cap\mathrm{R}\subseteq\mathrm{R}_2$), and consider the partition $(\mathrm{C}_1\cup\mathrm{R}_1, \mathrm{C}_2\cup\mathrm{R}_1)$ of V.

Since all partitions of V are ϵ-bad, this is in particular true of the partition $(\mathrm{C}_1\cup\mathrm{R}_1, \mathrm{C}_2\cup\mathrm{R}_1)$. Where do the at least ϵN^2 edges reside? Since R contains at most $(\epsilon/4)N$ influential vertices, each incident to at most N edges, and at most N non-influential vertices, each incident to at most $(\epsilon/4)N$ edges, the total number of edges incident to vertices in R is at most $(\epsilon/4)N\cdot N + N\cdot(\epsilon/4)N = (\epsilon/2)N^2$. Thus, the number of violating edges with respect to $(\mathrm{C}_1\cup\mathrm{R}_1, \mathrm{C}_2\cup\mathrm{R}_1)$ that are incident to vertices in R is at most $(\epsilon/2)N^2$, and this is true for *every* possible partition $(\mathrm{R}_1, \mathrm{R}_2)$. This implies that (no matter how R is partitioned) there must be at least $(\epsilon/2)N^2$ violating edges that are incident only to vertices within C_1 or within C_2. As we show in the next lemma, if we now take an additional (sufficiently large) sample (the sample S), then with high probability it will contain a pair of vertices connected by a violating edge (with respect to $(\mathrm{C}_1, \mathrm{C}_2)$). This is then shown to imply that for every partition $(\mathrm{S}_1, \mathrm{S}_2)$ of S, there is some edge between the sample vertices that is violating with respect to $(\mathrm{U}_1\cup\mathrm{S}_1, \mathrm{U}_2\cup\mathrm{S}_2)$. For an illustration of the partition and the violating edges, see Figure 1.

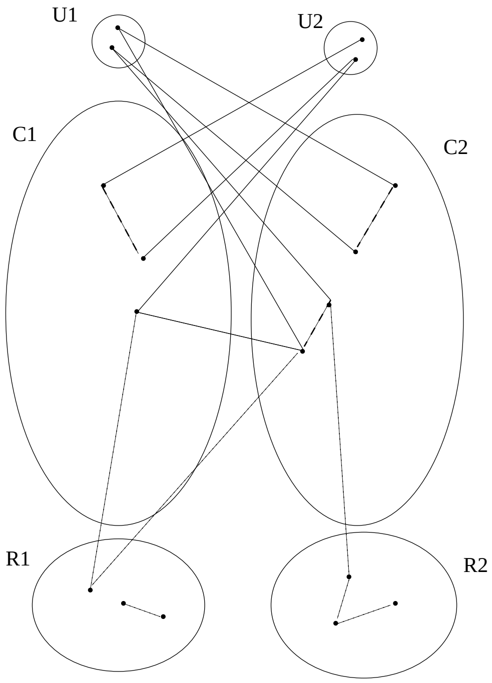


Figure 1: An illustration of the partition induced by $(\mathrm{U}_1, \mathrm{U}_2)$. The vertices in C_2 each have at least one neighbor in U_1, and the vertices in C_1 at least one neighbor in U_2. The vertices in R have no neighbor in either U_1 or U_2, and are partitioned arbitrarily. The edges incident to R (of which there are at most $(\epsilon/2)N^2$, both violating and not violating), are dotted, and the violating edges residing in either C_1 or C_2 (of which there are at least $(\epsilon/2)N^2$), are dashed.

Lemma 2.3.2 *Let* $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ *be a graph that is* ϵ*-far from bipartite,* U *a subset of* V *that covers all but at most* $(\epsilon/4)N$ *of the influential vertices in* G*, and* $(\mathrm{U}_1, \mathrm{U}_2)$ *a fixed partition of* U*. Let* S *be a uniformly and independently selected sample of* $m_2 = \Theta(|\mathrm{U}|/\epsilon)$ *vertices. Then, with probability at least* $1 - 2^{-|\mathrm{U}|}/6$ *over the choice of* S*, for every partition* $(\mathrm{S}_1, \mathrm{S}_2)$ *of* S*, there is some edge between vertices in* $\mathrm{U} \cup \mathrm{S}$ *that is violating with respect to* $(\mathrm{U}_1 \cup \mathrm{S}_1, \mathrm{U}_2 \cup \mathrm{S}_2)$*.*

Proof: It will be convenient to view S as $m_2/2$ pairs of vertices. By the discussion preceding the lemma, for every such pair (v, w), the probability that (v, w) constitutes a violating edge with respect to $(\mathrm{C}_1, \mathrm{C}_2)$ is at least $\epsilon/2$. The probability that among the $m_2/2$ pairs, there is no violating edge, is at most $(1 - (\epsilon/2))^{m_2/2}$ which for $m_2 = (16|\mathrm{U}|/\epsilon)$ is less than $2^{-|\mathrm{U}|}/6$. To complete the claim we need to show that if S contains such a pair, then it is not possible to partition S into $(\mathrm{S}_1, \mathrm{S}_2)$ so that $(\mathrm{U}_1 \cup \mathrm{S}_1, \mathrm{U}_2 \cup \mathrm{S}_2)$ has no violating edges.

Consider an edge (v, w) such that $v, w \in \mathrm{S}$ that violates $(\mathrm{C}_1, \mathrm{C}_2)$, and without loss of generality, assume $v, w \in \mathrm{C}_2$. If we put both vertices either in S_1 or in S_2 then (v, w) is violating with respect to $(\mathrm{S}_1, \mathrm{S}_2)$. However, since v and w belong to C_2, by our definition of the partition $(\mathrm{C}_1, \mathrm{C}_2)$, v has some neighbor $u \in \mathrm{U}_1$ and w has some neighbor $u \in \mathrm{U}_1$. Therefore, if we put $v \in \mathrm{S}_1$ and $w \in \mathrm{S}_2$, then the edge (u, v) will be violating, and if we put $w \in \mathrm{S}_1$ and $v \in \mathrm{S}_2$, then the edge (u', w) will be violating. The lemma follows. ■

Combining Lemma 2.3.2 with the fact that there are $2^{|\mathrm{U}|}$ partitions of U, it follows that with probability at least 5/6 over the choice of S, for *every* partition $(\mathrm{U}_1, \mathrm{U}_2)$ of U, and for every partition $(\mathrm{S}_1, \mathrm{S}_2)$ of S, the sample contains edges that violate $(\mathrm{U}_1 \cup \mathrm{S}_1, \mathrm{U}_2 \cup \mathrm{S}_2)$. In other words, the sample

$\mathrm{U} \cup \mathrm{S}$ cannot be partitioned without violations. Combining this with Lemma 2.3.1, the theorem follows. ■ (Theorem 1)

The Query and Time Complexities As described, the algorithm Test-Bipartite has query and time complexities that are quadratic in the size of the sample. That is, $\Theta(\log^2(1/\epsilon)/\epsilon^4)$. However, given the analysis, we can slightly improve on this bound. Assume the algorithm actually partition the sample into two parts U and S of sizes $m_1 = \Theta(\log(1/\epsilon)/\epsilon)$ and $m_2 = \Theta(\log(1/\epsilon)/\epsilon^2)$, respectively. It views S as consisting of $m_2/2$ pairs of vertices, and queries whether an edge exists only between all pairs of vertices in $\mathrm{U} \times \mathrm{S}$, and between the $m_2/2$ pairs in S. It then checks whether the resulting subgraph is bipartite. Then by the analysis this suffices to obtain the desired success probability while decreasing the complexities to $\Theta(\log^2(1/\epsilon)/\epsilon^3)$.

2.4 Testing ρ-Clique

In this subsection we give the basic underlying ideas of a testing algorithm for the ρ-Clique property. A graph is said to have the property if it contains a clique of size $\rho \cdot N$, for a given constant $0 \leq \rho \leq 1$. Hence, by definition, a graph is ϵ-far from having a ρ clique, if for *every* subset X of the vertices of size ρN, the number of pairs of vertices from X that do not have an edge between them is more than ϵN^2.

While it can be shown that the "natural" algorithm, which simply takes a sample S of vertices and accepts if an only if S contains a clique of size slightly smaller than $\rho|\mathrm{S}|$, will work, there is no direct proof for its correctness. Rather, its correctness follows from that of another somewhat "unnatural" algorithm, which we discuss below. In the case of the natural algorithm, it is very easy to show that a graph having a clique of size ρN will be accepted with high probability, and the difficulty is in proving that a graph which is ϵ-far from having the property is rejected with probability at least 2/3. As we shall see, for the unnatural algorithm it is relatively easy to show that a graph which is ϵ-far from having the property is rejected with probability at least 2/3, but we need to work harder to show that a graph having the property is accepted with probability at least 2/3.

We start by showing how, given a graph that *has a clique* of size ρN it is possible to find a subset of vertices having size ρN that is an *approximate* clique. Namely, the number of pairs of vertices in the set that do not have an edge between them is at most ϵN^2.

An Oracle Aided Procedure Consider first the following mental experiment. Let C be a clique of size ρN in G, and suppose we have access to an oracle that provides us with the number of neighbors that any given vertex $v \in \mathrm{V}$ has in C. We would like to use this information in order to construct a clique C' of size ρN (which may differ from C). Let $d_{\mathrm{C}}(v)$ denote the number of neighbors that vertex v has in C. By definition of a clique, for every $v \in \mathrm{C}$, $d_{\mathrm{C}}(v) = \rho N - 1$. However, there may be other vertices, outside of C, for which the same is true. Consider all vertices v for which $d_{\mathrm{C}}(v) \geq \rho N - 1$, and let their set be denoted by $\mathrm{T(C)}$. Assume we order these vertices according to the number of neighbors they have in $\mathrm{T(C)}$ (i.e., according to the degree they have in the subgraph induced by $\mathrm{T(C)}$), and let C' be the first ρN vertices according to this order (breaking ties arbitrarily). Then we claim that C' is a clique.

To see this, observe that by definition of $\mathrm{T(C)}$, each vertex in C neighbors every vertex in $\mathrm{T(C)}$ (except itself). Thus, each vertex in C has degree $|\mathrm{T(C)}| - 1$ in the subgraph induced by $\mathrm{T(C)}$, which is the maximum possible. Since $|\mathrm{C}| = \rho N$, every vertex in C' must have degree $|\mathrm{T(C)}| - 1$ as well (because the vertices in C are all candidates for the set C$'$ whose size is ρN as well). In other words, every vertex in C$'$ neighbors every (other) vertex in T(C), and in particular it neighbors every other vertex in C$'$, making C$'$ a clique.

Suppose next that instead of having an oracle as described above, we were provided with a uniformly chosen set U$'$ in C of sufficient size (i.e., of size $\Theta(\rho^2 \cdot \log(1/\epsilon))/\epsilon^2))$. Let T(U$'$) be the set of vertices that neighbor every vertex in U$'$. Then, with high probability over the choice of U$'$, almost every vertex in T(U$'$) neighbors almost all vertices in C (where "almost" is all but a fraction of ϵ^2). Similarly to the above, we could order the vertices in T(U$'$) according to their degree in the graph induced by T(U$'$), and take the first ρN vertices, whose set we denote by F(T(U$'$)). By extending the argument given above for the oracle-aided procedure, it can be shown that F(T(U$'$) is close to being a clique.

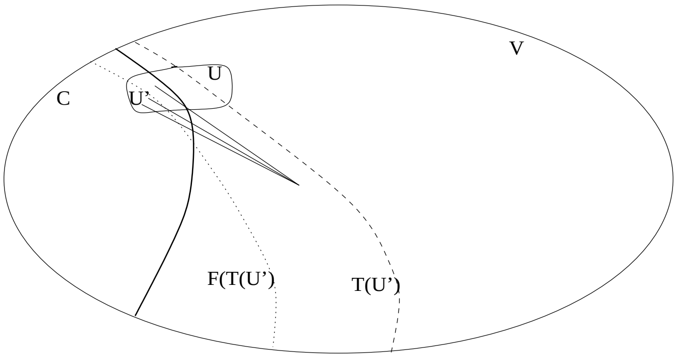
The Actual Approximate-Clique Finding Algorithm Since a uniformly chosen sample in C is not provided to the algorithm, it instead "guesses" such a set. More precisely, it uniformly selects a set U from all graph vertices, and it considers all its subsets U$'$ of size $\frac{\rho}{2}|\mathrm{U}|$. Since with high probability $|\mathrm{U} \cap \mathrm{C}| \geq \frac{\rho}{2}|\mathrm{U}|$, there exists a subset U$'$ contained in C. Since U is selected uniformly, the set U$'$ is uniformly distributed in in C. Hence, with high probability, for this U$'$, almost all vertices in T(U$'$) (the set of vertices that neighbor every vertex in U$'$), neighbor almost every vertex in C. If we now order the vertices in T(U$'$) according to their degree in the subgraph they induce and select the first ρN vertices according to this order, then the resulting set F(T(U$'$)) is close to being a clique (with high probability over the choice of U). For an illustration, see Figure 2.

Thus, the algorithm considers all subsets U$'$ of U having size $\frac{\rho}{2}|\mathrm{U}|$. For each subset U$'$ it finds the vertices in T(U$'$) and among them those in F(T(U$'$)). It selects the set F(T(U$'$)) that is closest to being a clique. By the above discussion, if the graph has a clique of size ρN, then with high probability the selected set will in fact be close to being a clique. On the other hand, if the graph is ϵ-far from having a ρ clique, then, by definition, none of these sets will be close to a clique.

The Testing Algorithm The testing algorithm is based on the above approximate-clique finding algorithm. Similarly to that algorithm, it uniformly selects a set U and considers each of its subsets U$'$ having size $\frac{\rho}{2}|\mathrm{U}|$. The algorithm then: Samples from T(U$'$); Approximates the number of neighbors that each sampled vertex has in T(U$'$); Orders the sampled vertices according to this approximate degree; Takes the first ρ fraction according to this order; And checks how close they are to being a clique.

More precisely, the algorithm performs the following steps:

1. Uniformly and independently select three samples, U, S, and W, where the first two are of size $\Theta(\rho \cdot \log(1/\epsilon))/\epsilon^2))$, and the third of size $\Theta(1/\epsilon^3)$.
2. For each subset U$'$ of U having size $\frac{\rho}{2}|\mathrm{U}|$, do:
   (a) Let $\mathrm{S}(\mathrm{U}') \subseteq \mathrm{S}$ be the set of vertices in S that neighbor all vertices in U$'$. Note that since S is uniformly distributed in V, this set is uniformly distributed in T(U$'$).
   (b) Let $\mathrm{W}(\mathrm{U}') \subseteq \mathrm{W}$ be the set of vertices in W that neighbor all vertices in U$'$. The vertices in W(U$'$) are also uniformly distributed in T(U$'$), and they will serve to approximated the degree of the vertices in S(U$'$).
   (c) For each vertex $v \in \mathrm{S}$, let $\hat{d}(v)$ be the number of neighbors v has in W(U$'$).



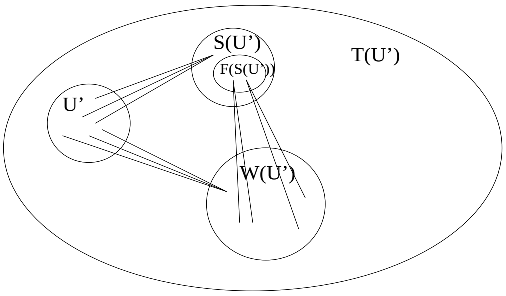


Figure 2: An illustration of the approximate-clique finding algorithm (top figure), and of the testing algorithm (bottom figure). Top figure: the clique C is separated from the rest of the graph vertices by a bold line. The uniformally selected sample U intersects with C and the intersection is denoted by U$'$ (more precisely, U$'$ is a subset of this intersection having size $(\rho/2)|\mathrm{U}|$ but for sake of brevity we assume it coincides with the intersection). The set T(U$'$), denoted by a dashed separating line, is the set of all vertices that neighbor every vertex in U$'$ (it is a superset of C). Finally, F(T(U$'$)) are the first ρN vertices with highest degree in the subgraph induced by T(U$'$). Bottom figure: U$'$, and T(U$'$) are as in the top figure. S(U$'$) and W(U) are the subsets of vertices in the samples S and W, respectively, that intersect T(U$'$). (S and W are not illustrated). Finally, F(S(U$'$)) are the first $\rho|\mathrm{S}|$ vertices in S(U$'$) with the largest number of neighbors in W(U$'$).

(d) Order the vertices in S(U$'$) according to $\hat{d}(\cdot)$. Let F(S(U$'$)) be the first $\rho|\mathrm{S}|$ vertices according to this order. If $\mathrm{S}(\mathrm{U}') < \rho|\mathrm{S}|$, then $\mathrm{F}(\mathrm{S}(\mathrm{U}')) = \mathrm{S}(\mathrm{U}')$.

(e) View the sample S as consisting of pairs $\{s_{2i-1}, s_{2i}\}_{i=1}^{m}$ (where the pairing is predetermined), and let $mis(\mathrm{U}')$ be the number of pairs $(s_{2i-1}, s_{2i}) \in \mathrm{F}(\mathrm{S}(\mathrm{U}')) \times \mathrm{F}(\mathrm{S}(\mathrm{U}'))$ that *do not* have an edge between them.

If for any one of the subsets U$'$, the set F(S(U$'$)) is of size at least $(\rho - \epsilon/80)|\mathrm{S}|$, and $mis(\mathrm{U}') \leq \frac{2\epsilon}{3}m$, then accept, otherwise it reject.

For an illustration, see Figure 2.

If the graph is ϵ-far from having a ρ-clique then for *every* U and $\mathrm{U}' \subset \mathrm{U}$, the set F(T(U$'$)) is far from being a clique. On the other hand, if the graph has a ρ-clique, then as we claimed above, with high probability over the choice of U, for *some* subset U$'$, the set F(T(U$'$)) is is close to being a clique. A probabilistic argument shows that with high probability over the choices of S and W, in the former case, for every U$'$ there are many missing edges between pairs of vertices in F(S(U$'$)), while in the latter case, for some U$'$, there are few missing edges.

The above constitutes a rough sketch of the following theorem.

Theorem 2 *The algorithm described above is a testing algorithm for the ρ-Clique property: If the graph has a clique of size ρN, then it is accepted with probability at least $2/3$, and if it is ϵ-far from having such a clique, then it is rejected with probability at least $2/3$.*

2.5 A General class of Partition Properties

The following framework of a general partition problem captures any graph property which requires the existence of partitions satisfying certain fixed density constraints. These constraints may refer both to the number of vertices in each component of the partition and to the number of edges between each pair of components.

Let $\Phi \stackrel{\text{def}}{=} \left\{\rho_j^{\text{LB}}, \rho_j^{\text{UB}}\right\}_{j=1}^{k} \cup \left\{\varrho_{j,j'}^{\text{LB}}, \varrho_{j,j'}^{\text{UB}}\right\}_{j,j'=1}^{k}$ be a set of non-negative parameters so that $\rho_j^{\text{LB}} \leq \rho_j^{\text{UB}}$ $(\forall j)$ and $\varrho_{j,j'}^{\text{LB}} \leq \varrho_{j,j'}^{\text{UB}}$ $(\forall j, j')$. (LB stands for Lower Bound, and UB stands for Upper Bound.) Let $\mathcal{GP}_\Phi$ be the class of graphs which have a k-way partition $(\mathrm{V}_1, \ldots, \mathrm{V}_k)$ with the following conditions being satisfied.

$$\forall j \qquad \rho_j^{\text{LB}} \cdot N \;\leq\; |\mathrm{V}_j| \;\leq\; \rho_j^{\text{UB}} \cdot N \tag{1}$$

and

$$\forall j, j' \qquad \varrho_{j,j'}^{\text{LB}} \cdot N^2 \;\leq\; |\mathrm{E}(\mathrm{V}_j, \mathrm{V}_{j'})| \;\leq\; \varrho_{j,j'}^{\text{UB}} \cdot N^2 \tag{2}$$

where recall that $\mathrm{E}(\mathrm{V}_j, \mathrm{V}_{j'})$ is the set of edges between vertices in V_j and vertices in $\mathrm{V}_{j'}$. That is, Equation (1) places lower and upper bounds on the relative sizes of the various components of the partition; whereas Equation (2) imposes lower and upper bounds on the density of edges among the various pairs of components.

Below we show how the various properties mentioned in Subsection 2.2 are defined as special cases.

1. Bipartiteness. We set $k = 2$ (as we are interested in a two-way partition), $\rho_1^{\text{LB}} = \rho_2^{\text{LB}} = 0$, and $\rho_1^{\text{UB}} = \rho_2^{\text{UB}} = 1$ (as there are no restrictions on the sizes of the partition subsets), $\varrho_{1,1}^{\text{LB}} = \varrho_{2,2}^{\text{LB}} = \varrho_{1,1}^{\text{UB}} = \varrho_{2,2}^{\text{UB}} = 0$, enforcing the main constraint that there be no edges within the partition subsets, and finally $\varrho_{1,2}^{\text{LB}} = 0$, $\varrho_{1,2}^{\text{UB}} = 1$, since the number of edges between the two subsets is not restricted.

2. k-colorability, $k \geq 3$. This is a generalization of bipartiteness, and the important density bounds are $\varrho_{j,j}^{\text{UB}} = 0$ for every $1 \leq j \leq k$. All lower bounds are 0 and all other upper bounds are 1.

3. ρ-Clique. Here $k = 1$, $\rho_1^{\text{LB}} = \rho_1^{\text{UB}} = \rho$, enforcing the restriction that one subset should be of size ρN, and $\varrho_1^{\text{LB}} = \frac{1}{2}(\rho^2 - \rho/N)$ enforcing the restriction that the subset be a clique.

4. ρ-Cut. Here $k = 2$, and $\varrho_{1,2}^{\text{LB}} = \rho$. For the case of ρ-bisection we add the constraints that $\rho_j^{\text{LB}} = \rho_j^{\text{UB}} = 1/2$ for $j \in \{1, 2\}$.

The testing algorithm for the class of partition properties and its analysis are somewhat complex and we refer the reader interested in further details to [GGR98].

2.6 First Order Graph Properties

Let $A(x_1, \ldots, x_t, y_1, \ldots, y_s)$ be a quantifier free graph expression. That is, it contains equality of vertices, adjacency relations between vertices, and boolean connectives. We say that an expression is of the type "$\exists\forall$" if it has the form $\exists x_1, \ldots, x_t \forall y_1, \ldots, y_s A(x_1, \ldots, x_t, y_1, \ldots, y_s)$. An expression of the type "$\forall\exists$" is defined analogously. Alon, Fischer, Krivelevich and Szegedy [AFKS99] show that it is possible to test any graph property that can be described as an "$\exists\forall$" statement with query complexity and running time independent of the size of the graph. The bound on the number of queries is a tower of towers of $\mathrm{poly}(1/\epsilon, t+s)$ exponents. They also show that there exists a (natural) property concerning isomorphism between (sub)graphs, which can be expressed as a "$\forall\exists$" expression, for which the required query complexity (for constant ϵ) is $\Omega(\sqrt{N})$.

To prove the first (main) result, Alon *et. al.* introduce the following notion of *indistinguishability* between graph properties, which may be useful in general.

Definition 2.6.1 *Two graph properties P and P' are said to be* indistinguishable *if for every $\epsilon > 0$ there exists an integer N_ϵ for which the following holds. For every graph* G *with $N > N_\epsilon$ vertices such that* G *has property P, there exists an N-vertex graph* G$'$ *having property P' such that the distance between* G *and* G$'$ *is at most ϵ. Similarly, for every graph* H$'$ *with $N > N_\epsilon$ vertices such that* H$'$ *has property P', there exists an N-vertex graph* H *having property P such that the distance between* H$'$ *and* H *is at most ϵ.*

For example, the property of being k-colorable is indistinguishable for the first order property that there exist k vertices $v_1, \ldots, v_k$ such that every other vertex is adjacent to exactly one of $v_1, \ldots, v_k$, but no two vertices that have an edge between them are adjacent to the same v_i. To verify this, consider first the case in which the first order expression holds for G. Let G$'$ be the graph in which all edges between the v_i's and the rest of the graph are removed from G. The distance between G and G$'$ is less than $\frac{k}{N}$ which is less than any constant ϵ provided N is sufficiently large. Then we can color each v_i and the vertices it is adjacent to in G with the i'th color, and obtain a k-coloring in G$'$. In the other direction, assume H is k-colorable. Then to obtain H$'$ from H we can pick k vertices $v_1, \ldots, v_k$, each colored by a different color, and add edges between each v_i and the other vertices having the same color. The first order expression holds for H$'$ and the distance between H and H$'$ is smaller than ϵ.

Given the above definition they then show:

Lemma 2.6.1 *If P and P' are indistinguishable graph properties, then P is testable using a number of queries independent of N if and only if P' is testable using a number of queries independent of N. Specifically, if the number of queries used in testing one property is* $\mathrm{Q}(\epsilon)$*, then the other property can be tested using at most $3Q(\epsilon/2)$ queries.*

Next they define the following generalization of the notions of colorability and subgraph-freeness, and show that every "$\exists\forall$" first order property is indistinguishable from such a generalized property.

Definition 2.6.2 *Let $\mathcal{F}$ be a family of graphs (where repetitions are allowed), such that each graph in the family comes with a (not necessarily proper) coloring by at most c colors. A coloring of a graph* G *by c colors (which, again, need not be proper) is called an* $\mathcal{F}$-coloring*, if no member of $\mathcal{F}$ appears as an induced subgraph of* G *with an identical coloring. A graph* G *is called* $\mathcal{F}$-colorable *if there exists an $\mathcal{F}$-coloring of* G*.*

The notion of a (proper) k-coloring discussed above, is a special case of this definition. Another, more complex property for example, is having a coloring with 2 colors without any monochromatic triangle.

Lemma 2.6.2 *For every first order property P of the form:*

$$\exists x_1, \ldots, x_t \forall y_1, \ldots, y_s \ A(x_1, \ldots, x_t, y_1, \ldots, y_s)$$

there exists a family $\mathcal{F}$ of $(2^{t+\binom{t}{2}} + 1)$-colored graphs, each with at most $\max\{2, t+1, s\}$ *vertices such that the property P is indistinguishable from the property of being $\mathcal{F}$-colorable.*

Finally they show:

Theorem 3 *For every constant c and every family $\mathcal{F}$ of c-colorable graphs, the property of being $\mathcal{F}$-colorable can be tested with query complexity and running time independent of the size of the graph.*

In order to prove Theorem 3 they prove their central technical lemma which is a variant of Szemerédi's Regularity Lemma [Sze78].

3 Testing Graph Properties in the Incidence-Lists Models

3.1 Definitions

As in the adjacency-matrix model, we use the notation G, V, E, and N to denote the graph, its vertex set, its edge set, and the number of graph vertices, respectively.

The Bounded-Degree Incidence-Lists Model In this model, a graph G whose vertices have degree at most d is represented by a two-dimensional array of size $N \times d$ (which can be viewed as N *lists*), where for each vertex v and integer $i \in \{1, \ldots, d\}$ the value of the corresponding entry is the i^{th} neighbor of v. If v has less than d neighbors then this value may be 0 (where $0 \notin$ V). Distance between graphs is defined as the number of pairs of vertices that have an edge between them in one graph but not in the other, divided by $d \cdot N$.[8] The notion of ϵ-far (ϵ-close) to having a property is defined analogously to the definition in the adjacency matrix model (Definition 2.1.1).

The Unbounded-Degree Incidence-Lists Model In this model a graph G is represented by N incidence lists of possibly varying lengths, where the length of each list (i.e., the degree of the vertex the list corresponds to) is provided at the head of the list. We denote by $d(v)$ the degree of vertex v. Distance between graphs is defined with respect to a given upper bound M on the number of graph edges. Namely, the distance between graph G_1 and G_2 with respect to the bound M is the number of (unordered) pairs of vertices that are an edge in one graph but not in the other, divided by M. When using this notion of distance to define the distance of a graph G to having a property, we assume G has at most M edges, but do not necessarily assume that the closest graph having the property has at most this many edges. Thus the unbounded-degree model is more general than the bounded degree model except in this technical aspect (as in the bounded-degree model the closest graph having the property must have at most dN edges and degree bound d as well).

[8]Once again, for sake of simplicity, this definition slightly differs from that discussed in the introduction and in [GR97]. There, distance is defined as the fraction of entries in the $N \times d$ matrix representation on which the two graphs differ. According to that definition, each (undirected) edge (v, u) in the symmetric difference between the graphs is counted twice - once as an entry $[v, i]$ and once as an entry $[u, j]$, while here we count each edge only once. (Distance as defined here is also insensitive to the ordering of the edges incident to a vertex, though this issue looses its relevance once we talk about the distance to having the property.)

In both models the testing algorithm can query, for each vertex v and every i (where $1 \leq i \leq d$ in the first model and $1 \leq i \leq d(v)$ in the second), what vertex is the i'th neighbor of v. The acceptance and rejection requirements of a testing algorithm are as in the adjacency-matrix model (Definition 2.1.2), with the appropriate notions of distance.

3.2 Summary of Results

As noted previously in the introduction, intuitively, testing in the bounded-degree model is easier (not harder) than testing in the unbounded-degree model. In fact, the bound on the degree of every vertex can make testing strictly easier (as we see below in the case of cycle-freeness) and there is no known property for which it is actually harder. However, as mentioned in Subsection 3.1, this bound also has a price: the definition of distance between a graph and the property (class of graphs having the property) is more restricted. This sometimes raises the following difficulty when testing in the bounded-degree model. Suppose the property is such that if a graph does not have the property then edges should be added so that it have the property (e.g. connectivity). Assume we would like to show that if a graph is accepted with probability greater than $1/3$ then it is close to having the property (which is equivalent to showing that if it is far from having the property then it is rejected with probability at least $2/3$). Then we would like to show that by adding a relatively small number of edges, the graph can be made to have the property. The difficulty is that in doing so we must maintain the degree bound of the graph, and this may sometimes be possible only by *removing* other edges. Hence, this technical difficulty sometimes makes the analysis simpler in the unbounded degree model (as is the case for connectivity).

The following is a list of results concerning testing properties in the incidence-lists model.

3.2.1 Bounded-Degree Model

- Connectivity. The algorithm has query complexity and running time $\tilde{O}(\epsilon^{-1})$ [GR97].
- k-Edge-Connectivity. The algorithm has query complexity and running time $\tilde{O}(k^3 \cdot \epsilon^{-3+\frac{2}{k}})$ [GR97]. For $k = 2, 3$ there are improved algorithms whose running-times are $\tilde{O}(\epsilon^{-1})$ and $\tilde{O}(\epsilon^{-2})$, respectively.
- 2 and 3 -Vertex-Connectivity. The algorithms have query complexity and running time $\tilde{O}(\epsilon^{-2})$ and $\tilde{O}(\epsilon^{-3})$, respectively [GR97].
- Eulerian. The algorithm has query complexity and running time $\tilde{O}(\epsilon^{-1})$ [GR97].
- Cycle-Freeness. The algorithm has query complexity and running time $O\left(\frac{1}{\epsilon^3} + \frac{d}{\epsilon^2}\right)$ [GR97].

  As we shortly discuss in Subsection 3.5, in the case of acyclicity of *directed* graphs, there is a lower bound of $\Omega(N^{1/3})$ (for constant ϵ and d) on the query complexity of any testing algorithm for the property [BR00].
- Bipartiteness. There is a lower bound of $\Omega(\sqrt{N})$ on the query complexity of testing this property (for constant ϵ and d) [GR97]. There is an algorithm having query complexity and running time $\sqrt{N} \cdot \text{poly}(\log N/\epsilon)$ [GR99].
- Expansion. There is a lower bound of $\Omega(\sqrt{N})$ on the query complexity of testing whether a graph has a given constant expansion (for constant ϵ and d) [GR97].

3.2.2 Unbounded-Degree Model

- Connectivity, k-edge-connectivity, 2 and 3-vertex-connectivity, Eulerian. The algorithms for these properties that work in the bounded-degree model can be adapted to work in the unbounded-degree model at a multiplicative cost of $O(\epsilon^{-1})$ in the query complexity and running time (while actually simplifying the analysis).
- Cycle-Freeness. Any algorithm for testing this property (in the unbounded-degree model) has query complexity $\Omega(\sqrt{N})$ (for constant ϵ and d). This can easily be verified by observing that the following two graphs (families of graphs) cannot be distinguished using $o(\sqrt{N})$ queries: One graph is the empty graph, which is cycle free, and the other graph contains only a single clique of size $\sqrt{N}$, and is very far from being cycle-free [PR99a].
- Diameter. There is a family of algorithms that test whether the diameter of a graph is bounded by a given parameter D, or is ϵ-far from any graph with diameter at most $\beta(D)$ [PR99a]. The function $\beta(D)$ ranges between $D+4$ and $4D+2$, depending on the algorithm. All algorithms have query complexity and running time $\tilde{O}(\epsilon^{-3})$, but differ in the ranges ϵ values for which they are applicable.

3.3 Testing k-Edge-Connectivity

In this subsection we describe and analyze the algorithm for testing connectivity (i.e., $k = 1$). We then sketch the ideas required for testing k-Edge-Connectivity for $k \geq 2$. We present the algorithms in the unbounded-degree model. Let the tested graph G have at most M edges (where $\mathrm{M} \geq N-1$ or otherwise the algorithm could immediately reject), and in what follows, whenever we say ϵ-far (or ϵ-close), we mean with respect to the bound M.

3.3.1 Testing Connectivity

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

Lemma 3.3.1 *If* G *is ϵ-far from being connected than it has more than ϵM connected components.*

Proof: Assume contrary to the claim that G has at most ϵM connected components, and denote these components by $\mathrm{C}_1, \ldots, \mathrm{C}_k$, $k \leq \epsilon M$. Then by adding an edge between some vertex in C_i and some vertex in C_{i+1}, for every $1 \leq i < k$, the graph can be made connected. As the total number of edges added is $k - 1 < \epsilon M$, we obtain a contradiction to the premise of the lemma that G is ϵ-far from being connected. ■

The following notation will be used throughout this subsection: $\tilde{\epsilon} \stackrel{\text{def}}{=} \frac{M}{N} \cdot \epsilon$. Thus, $\tilde{\epsilon}$ is ϵ times the average degree of G. As an immediate corollary of Lemma 3.3.1 we get:

Corollary 3.3.2 *If* G *is ϵ-far from being connected, then* G *has at least $\frac{\epsilon M}{2}$ connected components each containing less than $2/\tilde{\epsilon}$ vertices.*

Proof: By Lemma 3.3.1, G has more than ϵM connected components. The number of connected components containing at least $2/\tilde{\epsilon}$ vertices is at most $\frac{N}{2/\tilde{\epsilon}} = \frac{\epsilon M}{2}$. So the remaining ones are at least $\epsilon M - \frac{\epsilon M}{2} = \frac{\epsilon M}{2}$ in number, and each contains less than $2/\tilde{\epsilon}$ vertices. ■

An implicit implication of Lemma 3.3.1 is that for $\epsilon \geq \frac{N}{M}$, *every* graph is ϵ-close to being connected (as otherwise the lemma would imply the existence of an N-vertex graph with more than N connected components). Thus we may assume that $\epsilon < \frac{N}{M}$. By using the fact that each connected component contains at least one vertex we conclude that if G is ϵ-far from being connected then the probability that a uniformly selected vertex belongs to a connected component which contains less than $2/\tilde{\epsilon}$ vertices, is at least $\frac{\epsilon M/2}{N} = \tilde{\epsilon}/2$. Therefore, if G is ϵ-far from being connected and we uniformly select $m = 4/\tilde{\epsilon}$ vertices, then the probability that no selected vertex belongs to a component of size less than $2/\tilde{\epsilon}$ is bounded above by

$$\left(1 - \frac{\tilde{\epsilon}}{2}\right)^m < \exp\left(-\frac{\tilde{\epsilon}}{2} \cdot m\right) = \exp(-2) < \frac{1}{3}$$

Once we select such a vertex, we may detect that it belongs to a small connected component by performing a BFS until no new vertices are reached. This gives rise to the following testing algorithm, where we assume that $N \geq 2/\tilde{\epsilon}$ (since otherwise a connected graph having less than $2/\tilde{\epsilon}$ vertices would be rejected in Step (2)). If $N < 2/\tilde{\epsilon}$, we can determine if the graph is connected by simply inspecting the whole graph (which takes time $O(M) = O(\epsilon^{-1})$).

Connectivity Testing Algorithm

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

Since a connected graph consists of a single component, the algorithm never rejects a connected graph. By the discussion preceding the algorithm and Corollary 3.3.2, if a graph is ϵ-far from connected then it is rejected with probability at least 2/3. Since the number of edges traversed in each BFS is at most the number of vertices visited, squared,[9] the query complexity and running time of the algorithm are $m \cdot (2/\tilde{\epsilon})^2 = O(\tilde{\epsilon}^{-3})$. We note that the choice to perform a BFS is quite arbitrary, and that any other linear-time (in the number of edges) search method (e.g., DFS) will do.

The complexity of the Connectivity Tester can be improved by applying Corollary 3.3.2 more carefully. Above, when analyzing the probability that the algorithm selects a vertex in a small component, we considered the extreme case in which the component consists of a single vertex. On the other hand, when analyzing the complexity of scanning the component, we considered the extreme case in which the component consists of $\Theta(1/\tilde{\epsilon})$ vertices. Instead, suppose that all components in the conclusion of Corollary 3.3.2 were of the same size, denoted s. Then the probability that a vertex in such a component is selected is at least $s \cdot \frac{\epsilon M/2}{N} = \frac{s \cdot \tilde{\epsilon}}{2}$. This implies that it suffices to set $m = \Theta(1/(s\tilde{\epsilon}))$ in Step (1) of the algorithm above, and that in Step (2) it suffices to stop the search after $s+1$ vertices (at a cost of $O(s^2)$). Thus, the overall complexity would be $O(s/\tilde{\epsilon})$, provided that such s exists and is given to the algorithm.

[9]Here is where the algorithm in the bounded-degree model saves, as in that model the complexity of each BFS is only $O(d/(\epsilon d)) = O(\epsilon^{-1})$.

Since the latter assumption does not hold, we use a relaxed generalization of the above idea: That is, suppose that G has at least $L \stackrel{\rm def}{=} \frac{\tilde{\epsilon}}{2} \cdot \mathrm{M}$ connected components each of size less than $2/\tilde{\epsilon}$. Then, there exists an $i \leq \ell \stackrel{\rm def}{=} \log(2/\tilde{\epsilon})$ so that G has at least $\frac{L}{\ell}$ connected components of size ranging between 2^{i-1} and $2^i - 1$ (see the proof of Lemma 4 for details). We do not know this i, but we may try them all. This suggests the following improved algorithm.

Connectivity Testing Algorithm – Improved Version:

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

Once again, if the graph is connected, then it is always accepted. It thus remains to show:

Theorem 4 *If* G *is ϵ-far from being connected, then the improved connectivity testing algorithm will reject it with probability at least $\frac{2}{3}$. The query complexity and running time of the algorithm are $O\left(\frac{\log^2(1/(\tilde{\epsilon}))}{\tilde{\epsilon}^2}\right)$.*

Proof: Let B_i be the set of connected components in G which contain at most $2^i - 1$ vertices and at least 2^{i-1} vertices. Let $\ell \stackrel{\rm def}{=} \lceil \log(2/\tilde{\epsilon}) \rceil$. By Corollary 3.3.2 we know that $\sum_{i=1}^{\ell} |\mathrm{B}_i| \geq \frac{\epsilon \cdot M}{2} = \frac{\tilde{\epsilon}}{2} \cdot N$. Hence, there exists an index $j \in \{1, 2, ..., \ell\}$ so that $|\mathrm{B}_j| \geq \frac{\tilde{\epsilon} \cdot N}{2 \cdot \ell}$. Thus, the number of vertices residing in components belonging to B_j is at least $2^{j-1} \cdot |\mathrm{B}_j|$. It follows that the probability that a uniformly selected vertex resides in one of these components is at least

$$\frac{2^{j-1} \cdot |\mathrm{B}_j|}{N} \geq \frac{\tilde{\epsilon} \cdot 2^j}{4 \cdot \ell} = \frac{2}{m_j}$$

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

3.3.2 Testing k-Connectivity for $k > 1$

A subset of vertices $\mathrm{S} \subseteq \mathrm{V}$ is said to be k–edge-connected if there are k edge-disjoint paths between each pair of vertices in S. A graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ is k-(edge-)connected if V is k-edge-connected. The structure of the testing algorithm for k-Connectivity where $k > 1$ is similar to the structure of the Connectivity Tester (i.e., the case $k = 1$): The algorithm uniformly select a set of vertices and for each vertex selected, it tests if it belongs to a small component of the graph that is separated from the rest of the graph by an edge-cut of size less than k. Similarly to the $k = 1$ case, it can be shown that if a graph is ϵ-far from being k–connected then it has many such components. This can be shown by defining an auxiliary graph [DW98] whose nodes are components of the graph and that is based on the *cactus* structure of [DKL76]. In addition, there are efficient procedures for recognizing such a component given a vertex that resides in it. In what follows we sketch these procedures, for the different values of k. For simplicity we assume the tested graph is $k-1$-connected (where this assumption can be removed).

2-Connectivity and 3-Connectivity In the case of 2-connectivity, the procedure is given a vertex v, and is required to output found in case v belongs to a 2-connected subset of vertices C of size at most $n < N$, that is separated from the rest of the graph by a cut of size 1. (If the graph is 2-connected, the procedure should output not-found). The procedure performs the following steps.

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

The running time of the procedure is $O(n^2)$, and clearly, if the graph is 2-connected, then it outputs not-found. Thus assume v belongs to a 2-connected subset of vertices C of size at most n, that is separated from the rest of the graph by a cut of size 1. Since the first DFS terminates after seeing $n+1$ vertices, it must visit at least one vertex outside of C. This is possible only by traversing the single edge (u, v) from $u \in \mathrm{C}$ to $v \notin \mathrm{C}$. Thus, (u, v) must be a edge in the DFS tree T (with u being the parent). This ensures that the second search will never exit C. In other words, $\mathrm{S}_2 \subseteq \mathrm{C}$. Using the fact that C is 2-connected, it can be shown that the second search will reach *every* vertex in C (that is, $\mathrm{S}_2 = \mathrm{C}$), and hence will detect the cut.

The procedure for 3-connectivity is given a vertex v that belongs to a subset of vertices C that is 3-connected. If the cut $(\mathrm{C}, \mathrm{V} \setminus \mathrm{C})$ has size 2 and $|\mathrm{C}| \leq n$ $(n < N)$, then the algorithm should output found. The procedure first performs a DFS until $n+1$ vertices are discovered. Next, for each edge e in this DFS-tree (which contains n edges), it "omits" e from the graph and invokes the 2-connectivity procedure on the residual graph. The procedure has running time $O(n^3)$, and its correctness is argued as follows.

Clearly the initial DFS must cross an edge of the cut $(\mathrm{C}, \mathrm{V} \setminus \mathrm{C})$ and so its DFS-tree has at least one cut edge. Let the graph resulting from omitting this cut edge from G be denoted G'. In G' the cut $(\mathrm{C}, \mathrm{V} \setminus \mathrm{C})$ contains a single edge in the resulting graph, denoted G'. While the removal of this edge might decrease the connectivity of C (which was 3 in G), it is at least 2–connected in G'. By the correctness of the procedure for 2-connectivity, we are done.

k-Connectivity, $k \geq 2$ The following applies to any $k \geq 2$, but for $k = 2, 3$ we have described more efficient procedures above. The procedure for detecting whether a vertex v belongs to a k-connected subset C of size at most n such that the cut $(\mathrm{C}, \mathrm{V} \setminus \mathrm{C})$ has size $k-1$, is based on Karger's Contraction Algorithm [Kar93] which is a randomized algorithm for finding a minimum cut in a graph.

Given a vertex v and a size bound n, the following randomized search process is performed $\Theta(n^{2-\frac{2}{k}})$ times, or until a cut $(\mathrm{S}, \mathrm{V} \setminus \mathrm{S})$ of size less than k is found:

Random search process: Starting from the singleton set $\{v\}$, the procedure maintains the set, denoted S, of vertices it has visited. In each step, as long as $|\mathrm{S}| < n$ and the cut $(\mathrm{S}, \mathrm{V} \setminus \mathrm{S})$ has size at least k, the procedure selects at random (as specified below) an edge to traverse among the cut edges in $(\mathrm{S}, \mathrm{V} \setminus \mathrm{S})$ and adds the new vertex reached to S. In case the cut $(\mathrm{S}, \mathrm{V} \setminus \mathrm{S})$ has size less than k, output found. If $|\mathrm{S}| = n$ then the current search is completed. Otherwise, proceed to the next step (i.e., select a new random edge from the cut $(\mathrm{S}, \mathrm{V} \setminus \mathrm{S})$).

In case none of the $\Theta(n^{2-\frac{2}{k}})$ invocations of the above process has detected a small cut, output not-found.

The random selection of edges to traverse is done as follows. We think of uniformly and independently assigning each edge in the graph a cost in $[0, 1]$. Then, at each step of the procedure, we select the edge with lowest cost in the current cut $(\mathrm{S}, \mathrm{V} \setminus \mathrm{S})$. This is implemented as follows: Whenever a new vertex is added to S, its incident edges that were not yet assigned costs are each assigned a random cost uniformly in $[0, 1]$. Thus, whenever we need to select an edge from the current cut $(\mathrm{S}, \mathrm{V} \setminus \mathrm{S})$, all edges in the cut have costs, and we select the edge with lowest cost (just as in the mental experiment in which all graph edges are assigned uniform costs at the beginning).

The correctness of the procedure follows from a probabilistic analysis that bounds the probability that an edge in the cut $(\mathrm{C}, \mathrm{V} \setminus \mathrm{C})$ is selected before $\mathrm{S} = \mathrm{C}$. For further details see [GR97].

3.4 Testing Bipartiteness

As defined in Subsection 2.3, a graph is said to be *bipartite* if its set of vertices can be partitioned into two disjoint sets having no violating edges. An equivalent characterization of bipartite graphs, which we use in this subsection, is that they contain no odd-length cycles. In what follows we sketch both the lower bound on testing bipartiteness in the bounded-degree incidence-lists model, and the (almost matching) upper bound.

3.4.1 The Lower Bound

The following theorem is proved in [GR97].

Theorem 5 *Testing Bipartiteness (in the bounded-degree incidence-lists model) with distance parameter 0.01 requires* $\frac{1}{4} \cdot \sqrt{N}$ *queries.*

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

The two families are defined as follows. Let N be an even integer.[10]

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

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

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

Clearly, all graphs in $\mathcal{G}_2^N$ are bipartite as all cycles in the graph are of even length. Consider a graph uniformly selected in $\mathcal{G}_1^N$. We would like to show that with high probability it is far from bipartite. Such a graph can be selected by selecting a random permutation of the vertices $1, \ldots, N$ on the cycle, and then uniformly selected a matching between the vertices. For any fixed permutation of the vertices on the cycle, consider any fixed two-way partition of the vertices. If there are many violating cycle edges with respect to this partition (give the ordering of the vertices on the cycle), then we are done. Thus, assume there aren't many. Then it can be shown that with high probability over the choice of the matching edges, many of these edges will violate the partition. By bounding the number of such partitions, and using a probability union bound, the claim follows.

The remainder of the proof is focused on showing that a testing algorithm that performs less than $\frac{1}{4}\sqrt{N}$ queries is not able to distinguish (with sufficient probability) between a graph chosen randomly from $\mathcal{G}_2^N$ (which is always bipartite) and a graph chosen randomly from $\mathcal{G}_1^N$ (which with high probability is far from bipartite). This is done by defining two processes, one for each class of graphs. Each process answers the queries of the testing algorithm while randomly constructing a graph in the respective class (where the distribution on the resulting graphs can be shown to be uniform in the class). The crux of the proof is that for any testing algorithm, the two distributions on the query-answer sequences induced by the two processes, are statistically indistinguishable as long as the sequence is of length less than $\alpha\sqrt{N}$. This essentially follows from the fact that in sequences of such length, if we consider the subgraph induced by the query-answer sequence, then it does not contain a cycle (either even or odd).

3.4.2 The Algorithm

Since the algorithm can make queries of the form: "who is the i'th neighbor of vertex v", it can perform *walks* on G. Namely, starting from any vertex s, it can obtain the sequence of vertices lying on any path $i_1, i_2, \ldots, i_t$ (where each i_j is an edge label) that originates from s by querying: who is the i_1^{th} neighbor of s, who is the i_2^{th} neighbor of the vertex returned, and so on. In particular, the algorithm described below performs *random* walks on G: At each step, if the degree of the current vertex v is $d' \leq d$, then the walk *remains* at v with probability $1 - \frac{d'}{2d} \geq \frac{1}{2}$, and for each $u \in \Gamma(v)$, the walk *traverses* to u with probability $\frac{1}{2d}$. Thus, the stationary distribution over the vertices is uniform.

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

For every walk (or, more generally, any sequence of steps), there corresponds a *path* in the graph. The path is determined by those steps in which an edge is traversed (while ignoring all steps in which the walks stays in the same vertex). Such a path is not necessarily simple, but does not contain self loops. Note that when referring to the length of a walk, we mean the total number of steps taken, including steps in which the walk remains at the current vertex, while the length of the corresponding path does not include these steps.

Test-Bipartite (Incidence-Lists model)

- Repeat $T = \Theta\left(\frac{1}{\epsilon}\right)$ times:
  1. Uniformly select s in V.
  2. (a) Let $K = \text{poly}((\log N)/\epsilon) \cdot \sqrt{N}$, and $L = \text{poly}((\log N)/\epsilon)$;
     (b) Perform K random walks starting from s, each of length L;
     (c) If some vertex v is reached (from s) both on a prefix of a random walk corresponding to an even-length path and on a prefix of a walk corresponding to an odd-length path then reject.
- In case the algorithm did not reject in any one of the above iterations, it accepts.

Theorem 6 *The algorithm* Test-Bipartite *is a testing algorithm for bipartiteness. In particular, if the graph is bipartite it is always accepted, and if it is ϵ-far from bipartite it is rejected with probability at least* 2/3*. Furthermore, whenever the algorithm rejects a graph it outputs a* certificate *to the non-bipartiteness of the graph in form of an odd-length cycle of length* $\text{poly}(\epsilon^{-1} \log N)$.

Clearly, a bipartite graph is always accepted as it contains no odd-length cycles. Hence the heart of the proof is showing that if a graph is ϵ-far from bipartite it is rejected with probability at least 2/3. This is shown by proving the contrapositive statement: If a graph is accepted with probability greater than 1/3, then it is ϵ-close to bipartite. Namely, by removing at most $\epsilon \cdot dN$ edges, it can be made bipartite. The proof of this statement is somewhat complex, and here we only provide the underlying ideas.

The Rapidly–Mixing Case To gain intuition, consider first the following "ideal" case: From each starting vertex s in G, and for every $v \in$ V, the probability that a random walk of length $L = \text{poly}((\log N)/\epsilon)$ ends at v is at least $\frac{1}{2N}$ and at most $\frac{2}{N}$ – i.e., approximately the probability assigned by the stationary distribution. (Note that this ideal case occurs when G is an expander). Let us fix a particular starting vertex s. For each vertex v, let p_v^0 be the probability that a random walk (of length L) starting from s, ends at v and corresponds to an even-length path. Define p_v^1 analogously for odd-length paths. Then, by our assumption on G, for every v, $p_v^0 + p_v^1 \geq \frac{1}{2N}$.

We consider two cases regarding the sum $\sum_{v \in \text{V}} p_v^0 \cdot p_v^1$ — In case the sum is (relatively) "small", we show that there exists a partition (V_0, V_1) of V that is ϵ-good, and so G is ϵ-close to being bipartite. Otherwise (i.e., when the sum is not "small"), we show that the probability that the algorithm finds an odd cycle when performing the random walks starting from s, is constant. This implies that in case G is accepted with probability greater than $\frac{1}{3}$, then G is ϵ-close to being bipartite. In what follows we give some intuition concerning the two cases.

Consider first the case in which $\sum_{v \in \text{V}} p_v^0 \cdot p_v^1$ is smaller than $c \cdot \frac{\epsilon}{N}$ for some suitable constant $c < 1$. Let the partition (V_0, V_1) be defined as follows: $\text{V}_0 = \{v : p_v^0 \geq p_v^1\}$ and $\text{V}_1 = \{v : p_v^1 > p_v^0\}$.

Consider a particular vertex $v \in \mathrm{V}_0$. By definition of V_0 and our rapid-mixing assumption, $p_v^0 \geq \frac{1}{4N}$. Assume v has neighbors in V_0. Then for each such neighbor u, $p_u^0 \geq \frac{1}{4N}$ as well. However, since there is a probability of $\frac{1}{2d}$ of taking a transition from u to v in walks on G, we can infer that each neighbor u contributes $\Omega(\frac{1}{2d} \cdot \frac{1}{4N})$ to the probability p_v^1. (This inference is not completely straightforward since both p_u^0 and p_v^1 correspond to walks of length exactly L, but this slight difficulty can be overcome.) Thus, if there are many (more than ϵdN) violating edges with respect to $(\mathrm{V}_0, \mathrm{V}_1)$, then the sum $\sum_{v \in \mathrm{V}} p_v^0 \cdot p_v^1$ is large (greater than $\epsilon dN \cdot \frac{1}{4N} \cdot \frac{1}{8dn} \geq c \cdot \frac{\epsilon}{N}$), contradicting our case hypothesis.

We now turn to the second case ($\sum_{v \in \mathrm{V}} p_v^0 \cdot p_v^1 \geq c \cdot \frac{\epsilon}{N}$). For every fixed pair $i, j \in \{1, \ldots, K\}$, (recall that $K = \Omega(\sqrt{N})$ is the number of walks taken from s), consider the 0/1 random variable that is 1 if and only if both the i^{th} and the j^{th} walk end at the same vertex v but correspond to paths with different parity. Then the expected value of each random variable is $\sum_{v \in \mathrm{V}} 2 \cdot p_v^0 \cdot p_v^1$. Since there are $K^2 = \Omega(N)$ such variables, the expected value of their sum is greater than 1. These random variables are not pairwise independent, nonetheless we can obtain a constant bound on the probability that the sum is 0 using Chebyshev's inequality (cf., [AS92, Sec. 4.3]).

The General Case Unfortunately, we may not assume in general that for every (or even some) starting vertex, all (or even almost all) vertices are reached with probability $\Theta(1/N)$. Instead, for each vertex s, we may consider the set of vertices that are reached from s with relatively high probability on walks of length $L = \text{poly}((\log N)/\epsilon)$. As was done above, we could try and partition these vertices according to the probability that they are reached on random walks corresponding to even-length and odd-length paths, respectively. The difficulty that arises is how to combine the different partitions induces by the different starting vertices, and how to argue that there are few violating edges between vertices partitioned according to one starting vertex and vertices partitioned according to another.

To overcome this difficulty, we proceed in a slightly different manner. Let us call a vertex s *good*, if the probability that the algorithm finds an odd cycle when performing random walks starting from s, is at most 0.1. Then, assuming G is accepted with probability greater than $\frac{1}{3}$, all but at most $\frac{\epsilon}{16}$ of the vertices are *good*. We define a partition in stages as follows. In the first stage we pick any *good* vertex s. What we can show is that not only is there a set of vertices S that are reached from s with high probability and can be partitioned without many violations (due to the goodness of s), but also that there is a small cut between S and the rest of the graph. Thus, no matter how we partition the rest of the vertices, there cannot be many violating edges between S and $\mathrm{V} \setminus \mathrm{S}$. We therefore partition S (as above), and continue with the rest of the vertices in G.

In the next stage, and those that follow, we consider the subgraph H induces by the yet "unpartitioned" vertices. If $|\mathrm{H}| < \frac{\epsilon}{4}N$ then we can partition H arbitrarily and stop since the total number of edges adjacent to vertices in H is less than $\frac{\epsilon}{4} \cdot dN$. If $|\mathrm{H}| \geq \frac{\epsilon}{4}N$ then it can be shown that any *good* vertex s in H that has a certain additional property (which at least half of the vertices in H have), determines a set S (whose vertices are reached with high probability from s) with the following properties: S can be partitioned without having many violating edges among vertices in S; and there is a small cut between S and the rest of H. Thus, each such set S accounts for the violating edges between pairs of vertices that both belong to S as well as edges between pairs of vertices such that one vertex belongs to S and one to $\mathrm{V(H)} \setminus \mathrm{S}$. Adding it all together, the total number of violating edges with respect to the final partition is at most $\epsilon \cdot dN$.

The set S. To prove the existence of such sets S, consider first the initial stage in the partition process (i.e., here H = G). Recall that in this stage we are looking for a subset of vertices $S \subseteq V$, all reached with relatively high probability from some good vertex s, that are separated from the rest of G by relatively few edges. From the previous discussion we know that if for all (or almost all) vertices v in G, a random walk of length $\mathrm{poly}((\log N)/\epsilon)$ starting from s ends at v with probability $\Theta(1/N)$ then we can define a good partition of all of G and be done. Thus assume we are not in this case. Namely, there is a significant fraction of vertices that are reached from s with probability that differs significantly from $1/N$. In other words, the distribution on the ending vertices (when starting from s) is far from stationary. What can be shown (using techniques of Mihail [Mih89]) is that this implies the existence of a small cut between some set of vertices S that are each reached from s with probability that is roughly $1/\sqrt{|S| \cdot N}$ and the rest of G. Furthermore, it can be shown that S has an additional property that combined with the fact that s is good implies that it can be partitioned without having many violating edges.

In the next stages of the partition process, we would have liked to apply the same techniques to determine small cuts (with other desired properties) in subgraphs H of G. If we could at each stage "cut-away" the subgraph H from the rest of G and perform walks only inside H then we would have proceeded as in the first stage. However, these subgraphs H are only determined by the analysis while the algorithm, oblivious to the analysis, always performs random walks on all of G. Therefore we would like to have a way to map walks in G to walks in H so that probabilities of events occurring in imaginary walks on H can be related to events occurring in the real walks on G. This is done by defining a special Markov chain given H and relating walks on the Markov chain to walks on G. For further details see [GR99].

3.5 Directed Graphs

As noted at the end of Subsection 2.2, for some properties of undirected graphs that have analogous properties in directed graphs, algorithms that work on undirected graphs can be transformed to work on directed graphs. An example in the incidence-lists model is connectivity. A directed graph is (strongly) connected if there is a directed path in the graph from any vertex to any other vertex. The algorithm of [GR97] presented in Subsection 3.3 can be extended to the directed case *if the algorithm can also perform queries about the incoming edges to each vertex* [PR99b]. Otherwise, (the algorithm can only perform queries about outgoing edges), a simple lower bound of $\sqrt{N}$ on the number of queries can be obtained. In the case of testing acyclicity of directed graphs the situation is more complex. Here the cycle-freeness test for undirected graphs [GR97] can not be transformed to work on directed graphs, as it is (partly) based on the observation that that an undirected graph contains no cycles only if it has at most $N-1$ edges. Furthermore, in the case of directed graphs, even if there is access to both incoming and outgoing edges, every algorithm for testing acyclicity must use $\Omega(N^{\frac{1}{3}})$ queries [BR00].

4 Testing Other Properties

In this section we provide a brief summary of results on testing properties of objects other than graphs. In particular, most results concern functions.

Definition 4.0.1 *For a given function $f : \mathrm{X} \to \mathrm{Y}$, and a property $\mathcal{P}$ (defined over functions with domain X and range Y), we say that f is ϵ-far from having property $\mathcal{P}$, if for every $g : \mathrm{X} \to \mathrm{Y}$, $\Pr_{x\sim \mathrm{U}}[f(x) \neq g(x)] > \epsilon$, (where U denotes the uniform distribution), otherwise it is ϵ-close to $\mathcal{P}$.*

4.1 Testing Algebraic Properties

4.1.1 Testing Linearity

In this subsection we present a test due to Blum, Luby and Rubinfeld [BLR93], with a slightly modified analysis due to Sudan [Sud99].

Definition 4.1.1 (linearity of a function) *Let* F *be a finite field. A function* $f : \mathrm{F}^m \to \mathrm{F}$ *is called* linear *(or more precisely,* multi-linear*) if there exist constants* $a_1, \ldots, a_m \in \mathrm{F}$ *s.t. for all* $x = (x_1, \ldots, x_m) \in \mathrm{F}^m$ *it holds that* $f(x) = \sum_{i=1}^{m} a_i x_i$.

It is not hard to verify the following fact, which provides an alternative definition of linearity.

Fact 4.1.1 (alternative definition of linearity) *A function* $f : \mathrm{F}^m \to \mathrm{F}$ *is* linear *if and only if for every* $x, y \in \mathrm{F}^m$ $f(x) + f(y) = f(x + y)$.

The following test uniformly selects pairs of elements in the field, and checks whether linearity (according to the second definition) is violated.

Linearity Test

1. Uniformly and independently select $m = \Theta(\epsilon^{-1})$ pairs of elements $x, y \in \mathrm{F}$.
2. For every pair of elements selected, check whether $f(x) + f(y) = f(x + y)$.
3. If for any of the selected pairs linearity is violated (that is $f(x) + f(y) \neq f(x + y)$), then reject, otherwise, accept.

By Fact 4.1.1, if f is linear then it is always accepted. It thus remains to prove:

Theorem 7 *If f is ϵ-far from linear then with probability at least* $2/3$*, Linearity Test rejects it.*

Here we shall give a simple proof of the theorem for the case in which f is "not too far" from linear. Namely, that its distance from some linear function is bounded away from $\frac{1}{2}$ (that is, the distance is at most $\frac{1}{2} - \gamma$ for some constant γ).

Proof: We say that a pair of elements x, y are a *violating* pair, if $f(x) + f(y) \neq f(x + y)$. Let δ denote the (exact) distance of f from linearity (so that in particular, $\delta > \epsilon$). We shall show that the probability that a single uniformly selected pair of elements is a violating pair, is at least $3\delta(1 - 2\delta)$. For δ bounded away from $\frac{1}{2}$, this probability is $\Omega(\delta)$. Since the test selects $\Theta(1/\epsilon) = \Omega(1/\delta)$ pairs, the probability that no violating pair is selected is at most $1/3$ (for the appropriate choice of constant in the $\Omega(\cdot)$ notation).

Let g be a linear function at distance δ from f. Let $\mathrm{G} \stackrel{\mathrm{def}}{=} \{x : f(x) = g(x)\}$ be the set of *good* elements in F on which f and g agree. For any pair x, y, if among the three elements, x, y, and $(x + y)$ two of them belong to G while the third doesn't, then x, y are a violating pair. Hence,

$$\begin{aligned} \Pr[x,\, y \text{ are a violating pair }] \quad & \geq \quad \Pr[x \notin \mathrm{G},\, y \in \mathrm{G},\, (x+y) \in \mathrm{G}] \\ & \quad + \; \Pr[x \in \mathrm{G},\, y \notin \mathrm{G},\, (x+y) \in \mathrm{G}] \\ & \quad + \; \Pr[x \in \mathrm{G},\, y \in \mathrm{G},\, (x+y) \notin \mathrm{G}] \qquad (3) \end{aligned}$$

Consider the first probability in the above sum (the treatment of other two is analogous, as the important property of any triplet is that every two of the elements are pairwise independent).

$$\begin{aligned}\Pr[x \notin \mathrm{G},\, y \in \mathrm{G},\, (x+y) \in \mathrm{G}] &= \Pr[x \notin \mathrm{G}] \cdot \Pr[y \in \mathrm{G},\, (x+y) \in \mathrm{G} \mid x \notin \mathrm{G}] \\ &= \delta \cdot (1 - \Pr[y \notin \mathrm{G} \text{ or } (x+y) \notin \mathrm{G} \mid x \notin \mathrm{G}]) \qquad (4)\end{aligned}$$

By using a probability union bound, and the fact that both y and $(x+y)$ are uniformly distributed,

$$1 - \Pr[y \notin \mathrm{G} \text{ or } (x+y) \notin \mathrm{G} \mid x \notin \mathrm{G}] \geq 1 - 2\Pr[y \notin \mathrm{G} \mid x \notin \mathrm{G}] \qquad (5)$$

Since x and y are chosen independently,

$$\Pr[y \notin \mathrm{G} \mid x \notin \mathrm{G}] = \Pr[y \notin \mathrm{G}] = \delta \qquad (6)$$

and so by combining Equations (3)– (6), the probability of selecting a violating pair is at least $3 \cdot \delta \cdot (1 - 2\delta)$. ■

4.1.2 Testing (Low-Degree) Polynomials

We present a test for univariate polynomials that is based on a basic property of polynomials, where we follow the presentation of Sudan [Sudan-PhD]. There are also tests for multivariate polynomials but their analysis is more complex (see for example [GLR+91, RS96, RS97, AS97]).

Definition 4.1.2 *Let* F *be a finite field. A function* $f : \mathrm{F} \to \mathrm{F}$ *is a (univariate)* polynomial of degree d, *if there exist coefficients* $c_0, \ldots, c_d \in \mathrm{F}$, *such that* $f(x) = \sum_{i=0}^{d} c_i \cdot x^i$.

Recall that given any $d+1$ pairs $\{(x_i, y_i)\}_{i=0}^{d}$, where $x_i, y_i \in \mathrm{F}$, there exists a *unique* degree d polynomial h such that $h(x_i) = y_i$ for every $i \in \{0, \ldots, d\}$, and h can be found by interpolation.

Low-Degree Test

1. Repeat the following $m = 2/\epsilon$ times:
   (a) Uniformly and independently select $d+2$ distinct points $x_0, \ldots, x_{d+1} \in \mathrm{F}$.
   (b) Check (by interpolating) whether there exists a degree d polynomial q such that $q(x_i) = f(x_i)$ for every $i \in \{0, \ldots, d+1\}$.
2. If in any of the iterations evidence was found that f is not a degree d polynomial, then reject, otherwise, accept.

Clearly, if f is a degree d polynomial, then it is always accepted. It thus remains to prove:

Theorem 8 *If f is ϵ-far from any degree d polynomial then with probability at least $2/3$, Low-Degree Test rejects it.*

Proof: Let δ be the distance between f and a closest degree d polynomial (so that $\delta > \epsilon$). We show that in each iteration of the algorithm, the probability that the check in Step (1b) fails, is at least δ. Since $m = 2/\epsilon > 2/\delta$ iterations are performed, the probability that all checks succeed is $(1-\delta)^{2/\epsilon} < \exp(-2) < 1/3$.

Let g be a degree d polynomial closest to f (so that the distance between f and g is δ). We fix $z_0, \ldots, z_d$, and let h be the unique degree d polynomial such that $h(z_i) = f(z_i)$ for every $i \in \{0, \ldots, d\}$. By definition of δ, we have that the probability over a uniformly chosen point z_{d+1} in F that $h(z_{d+1}) = f(z_{d+1})$, is at most $1 - \delta$. Now,

$$\begin{aligned}
&\Pr_{x_0,\ldots,x_{d+1}} [\exists \text{ degree } d \text{ polynomial } q \text{ s.t. } \forall i \in \{0, \ldots, d\},\ q(x_i) = f(x_i)] && (7)\\
&\leq \max_{x_0,\ldots,x_d} \Pr_{x_{d+1}} [\text{ a degree } d \text{ poly that agrees with } f \text{ on } x_0, \ldots, x_d, \text{ agrees on } x_{d+1}\ ] && (8)\\
&\leq 1 - \delta && (9)
\end{aligned}$$

and so the probability that such a polynomial q does *not* exist is at least δ as claimed. ■

4.1.3 Testing Other Algebraic Properties

Functional Equations Rubinfeld [Rub99] studies properties of functions $f : \mathrm{X} \to \mathrm{Y}$ that can be characterized by *(quantified) functional equations* of the form: $\forall x, y \in \mathrm{X},\ F[f(x), f(y), f(x+y), f(x-y)] = 0$. For example, linearity falls into this framework since it can be characterized by $\forall x, y \in \mathrm{X},\ f(x+y) - f(x) - f(y) = 0$.

Such a characterization is said to be *robust* [RS96] if whenever the functional equation holds for f *for most* x, y, there exists a function g that has the property and is *close* to f. This implies that the property can be tested by verifying that the functional equation holds on a sample of uniformly selected pairs x, y.

Rubinfeld shows several sufficient conditions for robustness. In particular, functional equations of the (additive) form $\forall x, y,\ f(x+y) = G[f(x), f(y)]$ are robust, where many trigonometric functions can be characterized by such equations. d'Alembert's equation: $\forall x, y,\ f(x+y) + f(x-y) = 2f(x)f(y)$ is also robust, and so are several variants of it. Rubinfeld also provides necessary conditions for robustness (that have a combinatorial form).

Group Operations Let G be a finite set. Let $\circ$ be an operation on pairs of elements in G, so that for every $x, y \in \mathrm{G}$, $x \circ y \in \mathrm{G}$. The operation $\circ$ is *associative*, if for every $x, y, z \in \mathrm{G}$, $x \circ (y \circ z) = (x \circ y) \circ z$. An *identity* element with respect to $\circ$, is an element $e \in \mathrm{G}$ such that for every $x \in \mathrm{G}$, $x \circ e = x$. An *inverse* of an element $x \in \mathrm{G}$, is an element $x' \in \mathrm{G}$ such that $x \circ x' = e$.

The operation $\circ$ is a *group* operation if it is associative, has a unique identity element with respect to $\circ$, and every element has an inverse under $\circ$. Ergun et. al. [EKK+98] describe an algorithm having complexity $\tilde{O}(|\mathrm{G}|/\epsilon)$ for testing whether $\circ$ is a group operation, given access to the value of $x \circ y$ on pairs of elements of its choice under the assumption that the operation $\circ$ is cancelative.[11]. This assumption can be removed at a further multiplicative cost of $O(\sqrt{\mathrm{G}})$.

4.2 Testing Regular Languages

In this subsection we sketch the result of Alon, Krivelevich, Newman, and Szegedy [AKNS99], showing that for every regular language $\mathrm{L} \subseteq \{0,1\}^*$, there exists a testing algorithm for L. Namely, the algorithm accepts every word $w \in \mathrm{L}$, and rejects with probability at least 2/3 every word w that differs on more than $\epsilon \cdot |w|$ bits from any $w' \in \mathrm{L}$. The running time of the algorithm is $\tilde{O}(\epsilon^{-1})$, that is, independent of the length n of w. (The running time is dependent on the size of

[11] An operation $\circ$ is *cancelative* if $a \circ c = b \circ c$ implies $a = b$, and $a \circ b = a \circ c$ implies $b = c$

the (smallest) finite automaton accepting M, but this size is a fixed constant with respect to n). Recently, Newman [New00] extended this result and gave an algorithm having query complexity poly$(1/\epsilon)$ for testing whether a word w is accepted by a given constant-width branching program. We note that Alon *et. al.* also show that a very simple context free language (of all strings of the form $vv^R u$, where v^R denotes the reversal of v), cannot be tested using $o(\sqrt{N})$ queries.

We start by recalling some definitions.

Definition 4.2.1 *A* deterministic finite automaton (DFA) M *over the alphabet* $\{0,1\}$ *is defined by a set of* states $\mathrm{Q} = \{q_0, \ldots, q_{m-1}\}$*, a subset* $\mathrm{F} \subseteq \mathrm{Q}$ *of* accepting *states, and a* transition *function* $\delta : \mathrm{Q} \times \{0,1\} \mapsto \mathrm{Q}$*. The state* q_0 *is called the* initial *state. The transition function* δ *is extended to be defined on* $\{0,1\}^*$ *in the following recursive manner: For every* $q \in \mathrm{Q}$*,* $u \in \{0,1\}^*$*, and* $\sigma \in \{0,1\}$*,* $\delta(q, u\sigma) = \delta(\delta(q,u),\sigma)$*, where* $\delta(q,\lambda) = q$ *(*λ *denoting the empty string).*
We say that M accepts *a word* $w \in \{0,1\}^*$*, if* $\delta(q_0, w) \in \mathrm{F}$ *(otherwise it* rejects *it).*

We shall use the definition of regular languages that is based on DFA.

Definition 4.2.2 *A language* $\mathrm{L} \subseteq \{0,1\}^*$ *is said to be* regular *if there exists a DFA* M *such that* M *accepts all words* $w \in \mathrm{L}$*, and no other words.*

We thus assume that a regular language L is given by providing the (smallest) DFA M that accepts it. Given a DFA M, it induces a directed graph $\mathrm{G(M)} = (\mathrm{V}, \mathrm{E})$ in a straightforward manner: $\mathrm{V} = \mathrm{Q}$, and $\mathrm{E} = \{(q_i, q_j) : \exists \sigma \in \{0,1\}, \delta(q_i, \sigma) = q_j\}$. We shall refer to the vertices of G(M) as *states*.

Given a word $w \in \{0,1\}^n$, we first assume that the language L contains words of length n (or otherwise w can be directly rejected). Let u be a sub-word of w that starts at position i. That is $w = u'uu''$ where $|u'| = i$. We say that u is *feasible with respect to the DFA* M *starting from position* i if there exists a state q such that q can be reached in G(M) from q_0 in exactly $i-1$ steps and there is some path in G(M) from $q' = \delta(q, w)$ to an accepting state. When the index i is clear from the context we just say that u feasible. It is possible to verify whether u is feasible, in time that depends only on the size of M. Clearly, if a word w contains a sub-word u that is *not* feasible, then $w \notin \mathrm{L}$. The algorithm tries to find evidence to w not belonging to L in the form of infeasible sub-words.

Following Alon *et. al.* [AKNS99], we describe a special case of regular languages and show that for these languages, every word that is ϵ-far from belonging to the language, contains many short infeasible sub-words. Hence, an algorithm that simply samples such sub-words and checks whether they are feasible, will, with high probability, detect that a word w is ϵ-far from belonging to the language. The analysis of Alon *et. al.* reduces the general case to this special case. (For further details on the general case see [AKNS99].)

We make the following assumptions concerning the DFA M that accepts the language L. First, M contains a single accepting state, denoted q_{acc}. Second, the set of states Q of M can be partitioned into to subsets C and D such that:

1. The subset C contains both q_0 and q_{acc}.
2. The subgraph of G(M) induced by C is strongly connected. We denote this subgraph by G(C).

3. There are no edges in G(M) going from states in D to states in C (though there may be edges going in the other direction).

We further assume for simplicity that the greatest common divisor (GCD) of cycle lengths in G(C) is 1. This implies that there exists a constant $r = r(\mathrm{G(C)})$, referred to as the *reachability* constant of G(C), such that for every two states x, y in G(M), and for every $n \geq r$, there exists a directed path from x to y in G(C) of length n. The size of r is at most quadratic in $|\mathrm{C}|$. (If the GCD if cycle lengths is not 1 then a slightly different notion of reachability constant is required).

Lemma 4.2.1 *Let* M *be constrained as described above, and let* w *be a word of length* n *that is* ϵ*-far from the language* L *accepted by* M *(where* M *accepts some words of length* n*). Then the number of infeasible sub-words of* w *having length at most* $\frac{4r}{\epsilon}$ *is at least* $\frac{\epsilon n}{4r}$.

Proof: We shall construct a sequence of disjoint minimal-length infeasible sub-words of w. That is, each sub-word is infeasible, but each of its prefixes (and in particular its longest prefix) is feasible. Let the starting position of the j'th sub-word, u_j, be s_j, then we shall select the sub-words so that for very j, $s_j \geq r+1$, and $s_j + |u_j| \leq n - r$.

We constructs these sub-words in a *greedy* manner. The first sub-word, u_1, starts at position $s_1 = r+1$, and is the shortest sub-word of w, starting at position s_1, that is infeasible. The next sub-word, u_2, starts at position $s_2 = s_1 + |u_1|$, and is the shortest infeasible sub-word that starts at position s_2. In general, the j'th sub-word starts at position $s_j = s_{j-1} + |u_{j-1}|$ and is the shortest infeasible sub-word that starts at this position. The procedure terminates when position $n - r$ is reached, and the last sub-word is "cut-off" at this position. Hence, the last sub-word may actually be feasible. Note that for every position $i \leq n - r$, the empty sub-word is always feasible, and so each infeasible sub-word has length at least 1. Let the number of sub-words obtained be h. Then,

$$|w| \;=\; n \;=\; 2 \cdot r + \sum_{j=1}^{h} |u_j| \tag{10}$$

For each $1 \leq j \leq h$, let $\bar{u}_j$ be the prefix of u_j of length $|u_j| - 1$ (so that $\bar{u}_j$ may be empty) Recall that by definition of u_j, $\bar{u}_j$ is feasible. For every $1 \leq j \leq h$ we fix a state $q_{i_j} \in \mathrm{C}$ so that q_{i_j} is reachable from q_0 in $s_j - 1$ steps, and so that $\delta(q_{i_j}, \bar{u}_j) \in \mathrm{C}$. Note that such a state in C must exist because $\bar{u}_j$ is feasible, and by our assumption that there are no edges going from D to C. Not also that because u_j is infeasible, $\delta(q_{i_j}, u_j) \in \mathrm{D}$ (so that the last bit in u_j "forces" a transition to D from which there is no way to reach the accepting state in C).

For a given word $w' \in \{0,1\}^n$, we denote by $\mathrm{dist}(w, w')$ the number of bits on which w and w' differ. We shall show that there exists a word $w^* \in \mathrm{L}$, having length n, such that $\mathrm{dist}(w, w^*) \leq (2 \cdot h \cdot r)/n$. This will give us a lower bound on h. The construction is done inductively, where in the j'th step we obtain a word w_j of length $s_j - 1$ that is feasible from position 1. Based on our assumptions on M this in particular means that the sequence of states traversed given w_j are all in C. The basic idea is to modify w so that each bit (at the end of an infeasible sub-word u_j) that causes a transition from a state in C to a state in D, is replaced by a bit that causes a transition to another state in C (from which the accepting state can be reached). In order to "glue" these modifications together, a little more work is needed. Details follow.

The initial word, w^0, is some word of length r that is feasible from position 1. In general, we construct w^j based on w^{j-1} in the following manner. Let $\tilde{w}^{j-1}$ be the prefix of length $s_j - 1 - r$ of w^{j-1}, and let $p_j = \delta(q_0, \tilde{w}^{j-1})$. Since there exists some path of length r from p_j to q_{i_j} (where q_{i_j}

was defined above), by modifying (some of) the last r bits of w^{j-1} we can obtain a word z_j such that $\delta(q_0, z_j) = q_{i_j}$. If $j < h$ we let w^j be the concatenation of z_j, $\bar{u}_j$, and some bit b_j such that $\delta(q_0, w^{j-1}\bar{u}_j b_j) \in \mathrm{C}$ (since G(C) is strongly connected, such a bit must exist). In case $j = h$ we let $w^* = w^h$ be the concatenation of z_h, $\bar{u}_h$, and some word v of length r so that $\delta(q_0, w^{h-1}\bar{u}_j v) = q_{\mathrm{acc}}$, implying that $w^* \in \mathrm{L}$. By this construction,

$$\mathrm{dist}(w, w^*) \leq \frac{1}{n}\left((h-1)\cdot r + 2r\right) = \frac{hr + r}{n} \leq \frac{2hr}{n}$$

By our assumption on w, $\mathrm{dist}(w, w^*) \geq \epsilon$, and hence $h \geq \frac{\epsilon n}{2r}$.

Since all the infeasible sub-words, $u_1, \ldots, u_{h-1}$, are disjoint, the number of infeasible sub-words having length greater than $4r/\epsilon$ is less than $n/(4r/\epsilon) = (\epsilon n)/(4r)$. Since the total number of infeasible sub-words is at least $\frac{\epsilon n}{2r} - 1$, the lemma follows. ■

4.3 Testing Monotonicity

A function $f : \{0,1\}^n \to \{0,1\}$ is said to be monotone if $f(x) \leq f(y)$ for every $x \prec y$, where $\prec$ denotes the natural partial order among strings (i.e., $x_1 \cdots x_n \prec y_1 \cdots y_n$ if $x_i \leq y_i$ for every i and $x_i < y_i$ for some i).

The algorithm for testing monotonicity presented in [GGL+00] whose query complexity and running time are linear in n and $1/\epsilon$ performs a simple local test: It verifies whether monotonicity is maintained for randomly chosen pairs of strings that differ exactly on a single bit. More precisely,

Monotonicity Testing Algorithm

On input n, ϵ and oracle access to $f : \{0,1\}^n \to \{0,1\}$, repeat the following steps up to n/ϵ times

1. Uniformly select $x = x_1 \cdots x_n \in \{0,1\}^n$ and $i \in \{1, \ldots, n\}$.
2. Obtain the values of $f(x)$ and $f(y)$, where y results from x by flipping the i^{th} bit (that is, $y = x_1 \cdots x_{i-1}\, \bar{x}_i\, x_{i+1} \cdots x_n$).
3. If $x, y, f(x), f(y)$ demonstrate that f is not monotone then reject.
   That is, if either $(x \prec y) \wedge (f(x) > f(y))$ or $(y \prec x) \wedge (f(y) > f(x))$ then reject.

If all iterations are completed without rejecting then accept.

Thus the algorithm has a similar structure to the linearity testing algorithm. In the analysis of the algorithm, the probability of observing a local violation of monotonicity is related to the global measure relevant to testing – the minimum distance of the function to any monotone function. For further details see [GGL+00].

The definition of monotonicity can be extended in a straightforward manner to monotonicity of functions $f : \Sigma^n \to \{0,1\}$ where there is a total order over Σ. The algorithm can be modified so as to yield a testing algorithm having query complexity and running time $O\left(\frac{n \cdot \log|\Sigma|}{\epsilon}\right)$ [GGL+00]. The notion of monotonicity can be further extended to functions mapping to totally ordered ranges and the testing algorithm adapted in a corresponding manner. The dependence of the query complexity and running time of the modified algorithm are logarithmic in $|\Xi|$ [DGL+99]. The *spot-checker for sorting* presented in [EKK+98, Sec. 2.1] implies a tester for monotonicity with respect to functions from any fully ordered domain to any fully ordered range, having query and time complexities that are logarithmic in the size of the domain. This corresponds to the special case $n = 1$ (with general Σ and Ξ).

Another extension of testing monotonicity of boolean functions is testing *unateness* of functions. A function $f : \{0,1\}^n \to \{0,1\}$ is said to be unate if for every $i \in \{1,\ldots,n\}$ exactly one of the following holds: whenever the i^{th} bit is flipped from 0 to 1 then the value of f does not decrease; *or* whenever the i^{th} bit is flipped from 1 to 0 then the value of f does not decrease. Thus, unateness is a more general notion than monotonicity. The algorithm for testing monotonicity of boolean functions over $\{0,1\}^n$ can be extended to test whether a function is unate or far from any unate function at an additional cost of a (multiplicative) factor of $\sqrt{n}$ [GGL+00].

4.4 Testing using Random Examples

As noted in Subsection 1.1.4, one of the initial motivations for the study of property testing is its relation to Computational Learning Theory. In particular, while in some learning models queries are allowed, it is usually preferable that the learning algorithm be only provided with random examples. In analogy, here we shall assume that the testing algorithm is given a labeled sample $\{(x^1, f(x^1)), \ldots, (x^m, f(x^m))\}$, where the x^i's are distributed according to some fixed (possibly unknown) distribution D over the domain X. Distance between functions is defined as the weight, according to D of the symmetric distance between the functions. Namely, $\text{dist}(f,g) = \Pr_{x\sim D}[f(x) \neq g(x)]$.

In particular we consider the case in which the underlying distribution D is uniform (and so the distance measure between functions is as in the case where queries are allowed). In some cases, such as testing monotonicity, allowing only random examples makes the problem essentially intractable [GGL+00]. In other cases, while testing with queries is more efficient, there are still efficient testing algorithms that use only random examples.

This is in particular true of testing for decision trees over $[0,1]^d$ (for constant d) [KR98]. That is, the property is belonging to the class of decisions trees over $[0,1]^d$ having at most s nodes. This class of functions is defined as follows. Given an input $\vec{x} = (x_1, \ldots, x_d)$, the (binary) decision at each node of the tree is whether $x_i \geq a$ for some $i \in \{1,\ldots,d\}$ and $a \in [0,1]$. The labels of the leaves of the decision tree are in $\{0,1\}$. We define the *size* of such a tree to be the number of leaves, and we let DT_s^d denote the class of decision trees of size at most s over $[0,1]^d$. Thus, every tree in DT_s^d determines a partition of the domain $[0,1]^d$ into at most s axis aligned rectangles, each of dimension d (the leaves of the tree), where all points belonging to the same rectangle have the same label.

The testing algorithm for decision trees decides whether to accept or reject a function f by pairing "nearby" points in the sample, and checking that such pairs have common labels. More precisely, it will consider a certain collection of d-dimensional grids that partition the domain into *cells*. For each grid the algorithm computes the fraction of pairs of points that fall into the same grid cell and have the same label. If for some grid this fraction is above a certain threshold then it accepts, otherwise it rejects. The heart of the analysis is a combinatorial argument, which shows that there exists a (not too large) set of (relatively coarse) d-dimensional grids $G_1, \ldots, G_k$ for which the following holds: for every function $f \in \text{DT}_s^d$, there exists a grid G_i such that a "significant" fraction of the cells in G_i "fit inside" the leaves of f — that is, there are not too many cells of G_i that intersect a decision boundary of f.

The following theorem is proved in [KR98]. Note that it uses a more relaxed notion of testing: First the algorithm needs to distinguish between functions in DT_s^d and functions that are far from any function in $\mathrm{DT}_{s'}^d$, where $s' > s$ (though for constant d, s' is not much larger). Second, it does not work for every distance parameter ϵ, but only for values bounded away from 1/2 (and so can be seen as analogous to *weak learning*).

Theorem 9 *For any size s, dimension d and constant $C \geq 1$, let $s' = s'(s,d,C) \stackrel{\mathrm{def}}{=} 2^{d+1}(2s)^{1+1/C}$. Then there exists an algorithm that uses uniformly distributed examples, and with probability at least 2/3 accepts functions $f \in \mathrm{DT}_s^d$ and rejects functions that are $\left(\frac{1}{2} - \frac{1}{2^{d+5}(Cd)^d}\right)$-far from any decision tree in $\mathrm{DT}_{s'}^d$. The algorithm uses $\tilde{O}\left((2Cd)^{2.5d} \cdot s^{\frac{1}{2}(1+1/C)}\right)$ examples, and its running time is at most $(2\log(2s))^d$ times the number of examples used.*

A version of the algorithm that performs queries, has query complexity and running time $O\left((2Cd)^{2d+1} \cdot \log(s)^{d+1}\right)$.

References

[ADL+94] N. Alon, R. A. Duke, H. Lefmann, V. Rodl, and R. Yuster. The algorithmic aspects of the regularity lemma. *Journal of Algorithms*, 16:80–109, 1994.

[ADPR00] N. Alon, S. Dar, M. Parnas, and D. Ron. Testing of clustering. In *Proceedings of the Forty-First Annual Symposium on Foundations of Computer Science*, pages 240 – 251, 2000.

[AFKS99] N. Alon, E. Fischer, M. Krivelevich, and M Szegedy. Efficient testing of large graphs. In *Proceedings of the Fortieth Annual Symposium on Foundations of Computer Science*, pages 645–655, 1999.

[AK99] N. Alon and M. Krivelevich. Testing k-colorability. Manuscript, 1999.

[AKNS99] N. Alon, M. Krivelevich, I. Newman, and M Szegedy. Regular languages are testable with a constant number of queries. In *Proceedings of the Fortieth Annual Symposium on Foundations of Computer Science*, pages 656–666, 1999.

[ALM+98] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and intractability of approximation problems. *JACM*, 45(3):501–555, 1998.

[AS92] N. Alon and J. H. Spencer. *The Probabilistic Method.* John Wiley & Sons, Inc., 1992.

[AS97] S. Arora and S. Sudan. Improved low degree testing and its applications. In *Proceedings of the Thirty-First Annual ACM Symposium on the Theory of Computing*, pages 485–495, 1997.

[AS98] S. Arora and S. Safra. Probabilistic checkable proofs: A new characterization of NP. *JACM*, 45(1):70–122, 1998.

[BCH+95] M. Bellare, D. Coppersmith, J. Håstad, M. Kiwi, and M. Sudan. Linearity testing in characteristic two. In *Proceedings of the Thirty-Sixth Annual Symposium on Foundations of Computer Science*, pages 432–441, 1995.

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

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

[BGLR93] M. Bellare, S. Goldwasser, C. Lund, and A. Russell. Efficient probabilistically checkable proofs and applications to approximation. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on the Theory of Computing*, pages 294–304, 1993.

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

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

[BR00] M. Bender and D. Ron. Testing acyclicity of directed graphs in sublinear time. In *Proceedings of ICALP*, pages 809–820, 2000.

[BS94] M. Bellare and M. Sudan. Improved non-approximability results. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on the Theory of Computing*, pages 184–193, 1994.

[CSZ00] A. Czumaj, C. Sohler, and M. Ziegler. Property testing in computational geometry. In *Proceedings of the 8th Annual European Symposium on Algorithms (ESA)*, pages 155–166, 2000. Lecture Notes in Computer Science edited by M. Paterson, Springer-Verlag, Berlin.

[DGL+99] Y. Dodis, O. Goldreich, E. Lehman, S. Raskhodnikova, D. Ron, and A. Samorodnitsky. Improved testing algorithms for monotonocity. In *Proceedings of RANDOM*, pages 97–108, 1999.

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

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

[EKK+98] F. Ergun, S. Kannan, S. R. Kumar, R. Rubinfeld, and M. Viswanathan. Spot-checkers. In *Proceedings of the Thirty-Second Annual ACM Symposium on the Theory of Computing*, pages 259–268, 1998.

[FGL+96] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Approximating clique is almost NP-complete. *JACM*, 43(2):268–292, 1996.

[FK99] A. Frieze and R. Kanan. Quick approximation to matrices and applications. *Combinatorica*, 19(2):175–220, 1999.

[GGL+00] O. Goldreich, S. Goldwasser, E. Lehman, D. Ron, and A. Samordinsky. Testing monotonicity. *Combinatorica*, 20(3):301–337, 2000.

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

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

[GR97] O. Goldreich and D. Ron. Property testing in bounded degree graphs. In *Proceedings of the Thirty-First Annual ACM Symposium on the Theory of Computing*, pages 406–415, 1997. To appear in *Algorithmica*. A long version is available from `http://www.eng.tau.ac.il/~danar/papers.html`.

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

[Haj91] P. Hajnal. An $\Omega(n^{4/3})$ lower bound on the randomized complexity of graph properties. *Combinatorica*, 11(2):131–144, 1991.

[Hås96] J. Håstad. Testing of the long code and hardness for clique. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing*, pages 11–19, 1996.

[Hås97] J. Håstad. Getting optimal in-approximability results. In *Proceedings of the Thirty-First Annual ACM Symposium on the Theory of Computing*, pages 1–10, 1997.

[HS87] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *JACM*, 34(1):144–162, January 1987.

[HS88] D. S. Hochbaum and D. B. Shmoys. A polynomial approximation scheme for machine scheduling on uniform processors: Using the dual approximation approach. *SIAM Journal on Computing*, 17(3):539–551, 1988.

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

[Kin91] V. King. An $\Omega(n^{5/4})$ lower bound on the randomized complexity of graph properties. *Combinatorica*, 11(1):23–32, 1991.

[Kiw96] M. Kiwi. *Probabilistically Checkable Proofs and the Testing of Hadamard-like Codes.* PhD thesis, MIT, 1996.

[KR98] M. Kearns and D. Ron. Testing problems with sub-learning sample complexity. In *Proceedings of the Eleventh Annual ACM Conference on Computational Learning Theory*, pages 268–277, 1998.

[KSS94] M. J. Kearns, R. E. Schapire, and L. M. Sellie. Toward efficient agnostic learning. *Machine Learning*, 17(2-3):115–141, 1994.

[Mih89] M. Mihail. Conductance and convergence of Markov chains - A combinatorial treatment of expanders. In *Proceedings 30th Annual Conference on Foundations of Computer Science*, pages 526–531, 1989.

[New00] I. Newman. Testing of functions that have small width branching programs. In *Proceedings of the Forty-First Annual Symposium on Foundations of Computer Science*, pages 251–258, 2000.

[PR99a] M. Parnas and D. Ron. Testing the diameter of graphs. In *Proceedings of RANDOM*, pages 85–96, 1999.

[PR99b] M. Parnas and D. Ron. Testing the diameter of graphs. Available from `http://www.eng.tau.ac.il/~danar`, 1999.

[PR00] M. Parnas and D. Ron. Testing metric properties. Unpublished manuscript, 2000.

[Ros73] A. L. Rosenberg. On the time required to recognize properties of graphs: A problem. *SIGACT News*, 5:15–16, 1973.

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

[RS97] R. Raz and S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proceedings of the Thirty-First Annual ACM Symposium on the Theory of Computing*, pages 475–484, 1997.

[Rub99] R. Rubinfeld. Robust functional equations and their applications to program testing. *SIAM Journal on Computing*, 28(6):1972–1997, 1999.

[RV76] R. L. Rivest and J. Vuillemin. On recognizing graph properties from adjacency matrices. *Theoretical Computer Science*, 3:371–384, 1976.

[Sud99] M. Sudan. Private communications, 1999.

[Sze78] E. Szemeredi. Regular partitions of graphs. In *Proc. Colloque Inter. CNRS*, pages 399–401, 1978.

[Tre98] L. Trevisan. Recycling queries in pcps and in linearity tests. In *Proceedings of the Thirty-Second Annual ACM Symposium on the Theory of Computing*, pages 299–308, 1998.

[Val84] L. G. Valiant. A theory of the learnable. *CACM*, 27(11):1134–1142, November 1984.

[Yao87] A. C. C. Yao. Lower bounds to randomized algorithms for graph properties. In *Proceedings of the Twenty-Eighth Annual Symposium on Foundations of Computer Science*, pages 393–400, 1987.