Property Testing and its connection to Learning and Approximation*

Oded Goldreich† Shafi Goldwasser‡ Dana Ron§

Abstract – *We study the question of determining whether an unknown function has a particular property or is ϵ-far from any function with that property. A property testing algorithm is given a sample of the value of the function on instances drawn according to some distribution, and possibly may query the function on instances of its choice. First, we establish some connections between property testing and problems in learning theory. Next, we focus on testing graph properties, and devise algorithms to test whether a graph has properties such as being k-colorable or having a ρ-clique (clique of density ρ w.r.t the vertex set). Our graph property testing algorithms are probabilistic and make assertions which are correct with high probability, utilizing only $\mathrm{poly}(1/\epsilon)$ edge-queries into the graph, where ϵ is the distance parameter. Moreover, the property testing algorithms can be used to efficiently (i.e., in time linear in the number of vertices) construct partitions of the graph which correspond to the property being tested, if it holds for the input graph.*

1. Introduction

We are interested in the following general question of Property Testing: Let P be a fixed property of functions, and f be an unknown function. Our goal is to determine (possibly probabilistically) if f has property P or if it is far from any function which has property P, where distance between functions is measured with respect to some distribution D on the domain of f. Towards this end, we are given examples of the form $(x, f(x))$, where x is distributed according to D. We may also be allowed to query f on instances of our choice.

The problem of testing properties emerges naturally in the context of program checking and probabilistically checkable proofs as applied to multi-linear functions or low-degree polynomials [14, 7, 6, 19, 21, 36, 5, 4, 10, 11, 8, 9]. Property testing *per se* was considered in [36, 35]. Our definition of property testing is inspired by the PAC learning model [37]. It allows the consideration of arbitrary distributions rather than uniform ones, and of testers which utilize randomly chosen instances only (rather than being able to query instances of their own choice).

We believe that property testing is a natural notion whose relevance to applications goes beyond program checking, and whose scope goes beyond the realm of testing algebraic properties. Firstly, in some cases one may be merely interested in whether a given function, modeling an environment, (resp. a given program) possesses a certain property rather than be interested in learning the function (resp. checking that the program computes a specific function correctly). In such cases, learning the function (resp., checking the program) as means of ensuring that it satisfies the property may be an over-kill. Secondly, learning algorithms work under the postulation that the function (representing the environment) belongs to a particular class. It may be more efficient to test this postulation first before trying to learn the function (and possibly failing when the postulation is wrong). Similarly, in the context of program checking, one may choose to test that the program satisfies certain properties before checking that it computes a specified function. This paradigm has been followed both in the theory of program checking [14, 36] 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. Thirdly, we show how to apply property testing to the domain of graphs by considering several classical graph properties. This, in turn, offers a new perspective on approximation problems as discussed below.

THE RELEVANT PARAMETERS. Let $\mathcal{F}$ be the class of functions which satisfy property P. Then, testing property P corresponds to testing membership in the class $\mathcal{F}$. The two parameters relevant to property testing are the permitted distance, ϵ, and the desired confidence, δ. We require the tester to accept each function in $\mathcal{F}$ and reject every function which is further than ϵ away from any function in $\mathcal{F}$. We allow the tester to be probabilistic and make incorrect positive and negative assertions with probability at most δ. The complexity measures we focus on are the *sample complexity* (the number of examples of the function's values that the tester requires), the *query complexity* (the number of function queries made – if at all), and the *running time* of the tester.

1.1. Property Testing and Learning Theory

As noted above, our formulation of testing mimics the standard frameworks of learning theory. In both cases one is given access to an unknown *target* function (either in the form of random instances accompanied by the function values or in the form of oracle access to the function). A semantic differ-

*Full version available from `http://theory.lcs.mit.edu/~oded/`

†Dept. of Computer Science and Applied Math., Weizmann Institute of Science, ISRAEL. E-mail: `oded@wisdom.weizmann.ac.il`. On sabbatical leave at LCS, MIT.

‡Laboratory for Computer Science, MIT, 545 Technology Sq., Cambridge, MA 02139. E-mail: `shafi@theory.lcs.mit.edu`.

§Laboratory for Computer Science, MIT, 545 Technology Sq., Cambridge, MA 02139. E-mail: `danar@theory.lcs.mit.edu`. Supported by an NSF postdoctoral fellowship.

ence is that, for sake of uniformity, even in case the functions are Boolean, we refer to them as functions rather than concepts. However, there are two important differences between property testing and learning. Firstly, the goal of a learning algorithm is to *find* a good approximation to the target function $f \in \mathcal{F}$, whereas a testing algorithm should only *determine* whether the target function is in $\mathcal{F}$ or is far away from it. This makes the task of the testing seem easier than that of learning. On the other hand, a learning algorithm should perform well only when the target function belongs to $\mathcal{F}$ whereas a testing algorithm must perform well also on functions far away from $\mathcal{F}$. Furthermore, (non-proper) learning algorithms may output an approximation $\tilde{f}$ of the target $f \in \mathcal{F}$ so that $\tilde{f} \notin \mathcal{F}$.

We show that the relation between learning and testing is non-trivial. On one hand, proper (representation dependent) learning implies testing. On the other hand, there are function classes for which testing is harder than (non-proper) learning, provided $\mathcal{NP} \not\subseteq \mathcal{BPP}$. Nonetheless, there are also function classes for which testing is much easier than learning. Further details are given in Subsection 2.2. In addition, the graph properties discussed below provide a case where testing (with queries) is much easier than learning (also with queries).

1.2. Testing Graph Properties

In the main technical part of this paper, we focus our attention on testing graph properties. We view graphs as Boolean functions on pairs of vertices, the value of the function representing the existence of an edge. We mainly consider testing algorithms which use queries and work under the uniform distribution. That is, a testing algorithm for graph property P makes queries of the form "is there an edge between vertices u and v" in an unknown graph G. It then decide whether G has property P or is "ϵ-away" from any graph with property P, and is allowed to err with probability $1/3$. Distance between two N-vertex graphs is defined as the fraction of vertex-pairs which are adjacent in one graph but not in the other.

We present algorithms of $\mathrm{poly}(1/\epsilon)$ query-complexity and running-time[1] at most $\exp(\tilde{O}(1/\epsilon^3))$ for testing the following graph properties:

k-Colorability for any fixed $k \geq 2$. (Here the query-complexity is $\mathrm{poly}(k/\epsilon)$, and for $k = 2$ the running-time is $\tilde{O}(1/\epsilon^3)$.)

ρ-Clique for any $\rho > 0$. That is, does the N-vertex graph has a clique of size ρN.

ρ-CUT for any $\rho > 0$. That is, does the N-vertex graph has a cut of size at least ρN^2. A generalization to k-way cuts works within query-complexity $\mathrm{poly}((\log k)/\epsilon)$.

ρ-Bisection for any $\rho > 0$. That is, does the N-vertex graph have a bisection of size at most ρN^2.

[1] Here and throughout the paper, we consider a RAM model in which trivial manipulation of vertices (e.g., reading/writing a vertex name and ordering vertices) can be done in constant time.

Furthermore:

- 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 $\mathrm{poly}(1/\epsilon) \cdot N$-time, a partition which approximately obeys the property. For example, for ρ-CUT, we can construct a partition with at least $(\rho - \epsilon)N^2$ crossing edges.
- Except for Bipartite (2-Colorability) testing, running-time of $\mathrm{poly}(1/\epsilon)$ is unlikely, as it will imply $\mathcal{NP} \subseteq \mathcal{BPP}$.
- None of these properties can be tested without queries when using $o(\sqrt{N})$ random examples.
- The k-Colorability tester has one-sided error: it always accepts k-colorable graphs. Furthermore, when rejecting a graph, this tester always supplies a $\mathrm{poly}(1/\epsilon)$-size subgraph which is not k-colorable. All other algorithms have two-sided error, and this is unavoidable within $o(N)$ query-complexity.
- Our algorithms for k-Colorability, ρ-Clique and ρ-Cut can be easily extended to provide testers with respect to *product distributions*: that is, distributions $\Pi : \mathrm{V}(G)^2 \mapsto [0, 1]$ of the form $\Pi(u, v) = \pi(u) \cdot \pi(v)$, where $\pi : \mathrm{V}(G) \mapsto [0, 1]$ is a distribution on the vertices. In contrast, it is not possible to test any of the graph properties discussed above in a distribution-free manner.

GENERAL GRAPH PARTITION. All of the above properties are special cases of the General Graph k-Partition property, parameterized by a set of lower and upper bounds. The parameterized property holds if there exists a partition of the vertices into k disjoint subsets so that the number of vertices in each subset as well as the number of edges between each pair of subsets is within the specified lower and upper bounds. We present a testing algorithm for the above general property. The algorithm uses $\tilde{O}(k^2/\epsilon)^{k+5}$ queries, runs in time exponential in its query-complexity, and makes two-sided error. Approximating partitions, if they exist, can be efficiently constructed in this general case as well. We note that the specialized algorithms perform better than the general algorithm with the appropriate parameters.

OTHER GRAPH PROPERTIES. Going beyond the general graph partition problem, we remark that there are graph properties which are very easy to test (e.g., Connectivity, Hamiltonicity, and Planarity). On the other hand, there are graph properties in $\mathcal{NP}$ which are extremely hard to test; namely, any testing algorithm must inspect at least $\Omega(N^2/\log N)$ of the vertex pairs. In view of the above, we believe that providing a characterization of graph properties according to the complexity of testing them may not be easy.

OUR TECHNIQUES. Our algorithms share some underlying ideas. The first is the uniform selection of a small sample and the search for a *suitable* partition of this sample. In case of

k-Colorability certain k-colorings of the subgraph induced by this sample will do, and these are found by k-coloring a slightly augmented graph. In the other algorithms we exhaustively try all possible partitions. This is reminiscent of the *exhaustive sampling* of Arora *et. al.* [3], except that the partitions considered by us are always directly related to the combinatorial structure of the problem. We show how each possible partition of the sample induces a partition of the entire graph so that the following holds. If the tested graph has the property in question then, with high probability over the choice of the sample, there exists a partition of the sample which induces a partition of the entire graph so that the latter partition approximately satisfies the requirements established by the property in question. For example, in case the graph has a ρ-cut, there exists a 2-way-partition of the sample inducing a partition of the entire graph with $(\rho - \epsilon)N^2$ crossing edges. On the other hand, if the graph should be rejected by the test, then by definition no partition of the entire graph (and in particular none of the induced partitions) approximately obeys the requirements.

The next idea is to use an additional sample to approximate the quality of each such induced partition of the graph, and discover if at least one of these partitions approximately obeys the requirements of the property in question. An important point is that since the first sample is small (i.e., of size $\mathrm{poly}(1/\epsilon)$), the total number of partitions it induces is only $\exp \mathrm{poly}(1/\epsilon)$. Thus, the additional sample must approximate only these many partitions (rather than all possible partitions of the entire graph) and it suffices that this sample be of size $\mathrm{poly}(1/\epsilon)$,

The difference between the various algorithms is in the way in which partitions of the sample induce partitions of the entire graph. The simplest case is in testing Bipartiteness. For a partition $(\mathrm{S}_1, \mathrm{S}_2)$ of the sample, all vertices in the graph which have a neighbor in S_1 are placed on one side, and the rest of the vertices are placed on the other side. In the other algorithms the induced partition is less straightforward. For example, in case of ρ-Clique, a partition $(\mathrm{S}_1, \mathrm{S}_2)$ of the sample S with $|\mathrm{S}_1| \approx \rho|\mathrm{S}|$, induces a candidate clique roughly as follows. Consider the set T of graph vertices each neighboring all of S_1. Then the candidate clique consists of the ρN vertices with the highest degree in the subgraph induced by T. In the, ρ-Cut, ρ-Bisection and General Partition testing algorithms, we use auxiliary guesses which are implemented by exhaustive search.

1.3. Testing Graph Properties and Approximation

The relation of testing graph properties to approximation is best illustrated in the case of Max-CUT. A tester for the class ρ-Cut, working in time $T(\epsilon, N)$, yields an algorithm for approximating the maximum cut in an N-vertex graph, up to additive error ϵN^2, in time $\frac{1}{\epsilon} \cdot T(\epsilon, N)$. Thus, for any constant $\epsilon > 0$, we can approximate the size of the max-cut to within ϵN^2 in constant time. This yields a constant time approximation scheme (i.e., to within any constant relative error) for dense graphs, improving on Arora *et. al.* [3] and de la Vega [17] who solved this problem in polynomial-time ($O(N^{1/\epsilon^2})$–time and $\exp(\tilde{O}(1/(\epsilon^2))) \cdot N^2$–time, respectively). In both works the problem is solved by actually constructing approximate max-cuts. Finding an approximate max-cut does not seem to follow from the mere existence of a tester for ρ-Cut; yet, as mentioned above, our tester can be used to find such a cut in time linear in N (i.e., $\tilde{O}(1/\epsilon^3) \cdot N + \exp(\tilde{O}(1/\epsilon^3))$–time).

One can turn the question around and ask whether approximation algorithms for dense instances can be transformed into corresponding testers as defined above. In several cases this is possible. For example, using some ideas from our work, the Max-CUT algorithm of [17] can be transformed into a tester of complexity comparable to ours. We do not know whether the same is true with respect to the algorithms in [3]. Results on testing graph properties can be derived also from work by Alon *et. al.* [1]. That paper proves a constructive version of the Regularity Lemma of Szemerédi, and obtains from it a polynomial-time algorithm that given an N-vertex graph, $\epsilon > 0$ and $k \geq 3$, either finds a subgraph of size $f(\epsilon, k)$ which is not k-colorable, or omits at most ϵN^2 edges and k-colors the rest. Noga Alon has observed that the analysis can be modified to yield that almost all subgraphs of size $f(\epsilon, k)$ are not k-colorable, which in turn implies a tester for k-Colorability. In comparison with our k-Colorability Tester, which takes a sample of $O(k^2 \log k/\epsilon^3)$ vertices, the k-colorability tester derived (from [1]) takes a much bigger sample of size equaling a tower of $(k/\epsilon)^{20}$ exponents (i.e., $\log^* f(\epsilon, k) = (k/\epsilon)^{20}$).

A DIFFERENT NOTION OF APPROXIMATION FOR MAX-CLIQUE. Our notion of ρ-Clique Testing differs from the traditional notion of Max-Clique Approximation. When we talk of testing "ρ-Cliqueness", the task is to distinguish the case in which an N-vertex graph has a clique of size ρN from the case in which it is ϵ-far from the class of N-vertex graphs having a clique of size ρN. On the other hand, traditionally, when one talks of approximating the size of Max-Clique, the task is to distinguish the case in which the max-clique has size at least ρN from, say, the case in which the max-clique has size at most $\rho N/2$. Whereas the latter problem is NP-Hard, for $\rho \leq 1/64$ (see [9, Sec. 3.9]), we've shown that the former problem can be solved in $\exp(O(1/\epsilon^2))$-time, for any $\rho, \epsilon > 0$. Furthermore, Arora *et. al.* [3] showed that the "dense-subgraph" problem, a generalization of ρ-cliqueness, has a polynomial-time approximation scheme (PTAS) for dense instances.

TESTING k-COLORABILITY VS. APPROXIMATING k-COLORABILITY. Petrank has shown that it is NP-Hard to distinguish 3-colorable graphs from graphs in which every 3-partition of the vertex set violates at least a constant fraction of the edges [30]. In contrast, our k-Colorability Tester implies that solving the same promise problem is easy *for dense graphs*, where by dense graphs we mean N-vertex graphs with $\Omega(N^2)$ edges. This is the case since, for every $\epsilon > 0$, our tester can distinguish, in $\exp(k^2/\epsilon^3)$-time, between k-colorable N-vertex graphs and N-vertex graphs which remain non-k-colorable even if one omits at most ϵN^2 of their edges.[2]

We note that deciding k-colorability even for N-vertex graphs of minimum degree at least $\frac{k-3}{k-2} \cdot N$ is NP-complete (cf., Edwards [18]). On the other hand, Edwards also gave a polynomial-time algorithm for k-coloring k-colorable N-vertex graphs of minimum degree at least αN, for any constant $\alpha > \frac{k-3}{k-2}$.

1.4. Other Related Work

PROPERTY TESTING IN THE CONTEXT OF PCP: Property testing plays a central role in the construction of PCP systems. Specifically, the property tested is being a codeword with respect to a specific code. This paradigm explicitly introduced in [6] has shifted from testing codes defined by low-degree polynomials [6, 19, 5, 4] to testing Hadamard codes [4, 10, 11, 8], and recently to testing the "long code" [9].

PROPERTY TESTING IN THE CONTEXT OF PROGRAM CHECKING: There is an immediate analogy between program self-testing [14] and property-testing *with queries*. The difference is that in self-testing, a function f (represented by a program) is tested for being close to a fully specified function g, whereas in property-testing the test is whether f is close to any function in a function class $\mathcal{G}$. Interestingly, many self-testers [14, 36] work by *first* testing that the program satisfies some properties which the function it is supposed to compute satisfies (and only then checking that the program satisfies certain constraints specific to the function). Rubinfeld and Sudan [36] defined property testing, under the uniform distribution and using queries, and related it to their notion of Robust Characterization. Rubinfeld [35] focuses on property testing as applied to properties which take the form of functional equations of various types.

PROPERTY TESTING IN THE CONTEXT OF LEARNING THEORY: Departing from work in Statistics regarding the classification of distributions (e.g., [24, 16, 41]), Ben-David [12] and Kulkarni and Zeitouni [28] considered the problem of classifying an unknown function into one of two classes of functions, given labeled examples. Ben-David studied this classification problem in the limit (of the number of examples), and Kulkarni and Zeitouni studied it in a PAC inspired model. For any fixed ϵ, the problem of testing the class $\mathcal{F}$ with distance parameter ϵ can be casted as such a classification problem (with $\mathcal{F}$ and the set of functions ϵ-away from $\mathcal{F}$ being the two classes). A different variant of the problem was considered by Yamanishi [39].

TESTING GRAPH PROPERTIES. Our notion of testing a graph property P is a *relaxation* of the notion of *deciding the graph property* P which has received much attention in the last two decades [29]. In the classical problem there are no margins of error, and one is required to accept all graphs having property P and reject all graphs which lack it. In 1975 Rivest and Vuillemin [33] resolved the Aanderaa–Rosenberg Conjecture [34], showing 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. The query complexity of randomized decision procedures was conjectured by Yao to be $\Omega(N^2)$. Progress towards this goal was made by Yao [40], King [27] and Hajnal [23] culminating in an $\Omega(N^{4/3})$ lower bound. Our results, that some non-trivial monotone graph properties can be tested by examining a constant number of random locations in the matrix, stand in striking contrast to all of the above.

APPROXIMATION IN DENSE GRAPHS. As stated previously, Arora *et. al.* [3] and de la Vega [17] presented PTAS for dense instances of Max-CUT. The approach of Arora *et. al.* uses Linear Programming and Randomized Rounding, and applies to other problems which can be casted as a "smooth" Integer Programs.[3] The methods of de la Vega [17] are purely combinatorial and apply also to similar graph partition problems. Following the approach of Alon *et. al.* [1], but using a modification of the regularity Lemma (and thus obtaining much improved running times), Frieze and Kannan [20] devise PTAS for several graph partition problems such as Max-Cut and Bisection. We note that compared to all the above results, our respective graph partitioning algorithms have better running-times. Like de la Vega, our methods use elementary combinatorial arguments related to the problem at hand. Still our methods suffice for dealing with the General Graph Partition Problem.

Important Note: In this extended abstract, we present only two of our results on testing graph properties: the k-Colorability and the ρ-Clique testers. The definition and theorem regarding the General Graph Partition property appears in Subsection 3.3. All other results as well as proofs and further details can be found in our report [22].

2. General Definitions and Observations

2.1. Definitions

Let $\mathcal{F} = \{\mathcal{F}_n\}$ be a parameterized class of functions, where the functions[4] in $\mathcal{F}_n$ are defined over $\{0,1\}^n$ and let $\mathcal{D} = \{D_n\}$ be a corresponding class of distributions (i.e., D_n is a distribution on $\{0,1\}^n$). We say that a function f defined on $\{0,1\}^n$ is *ϵ-close* to $\mathcal{F}_n$ with respect to D_n if there exists a function $g \in \mathcal{F}_n$ such that

[2] As noted by Noga Alon, similar results, alas with much worse dependence on ϵ, can be obtained by using the results of Alon et. al. [1].

[3] In [2], the approach of [3] is extended to other problems, such as Graph Isomorphism, using a new rounding procedure for the Assignment Problem.

[4] The range of these functions may vary and for many of the results and discussions it suffices to consider Boolean function.

$$\mathrm{Prob}_{x\sim D_n}[f(x) \neq g(x)] \leq \epsilon \ . \quad (1)$$

Otherwise, f is ϵ-*far* from $\mathcal{F}_n$ (with respect to D_n).
We shall consider several variants of testing algorithms, where the most basic one is defined as follows.

Definition 2.1 (property testing): *Let $\mathcal{A}$ be an algorithm which receives as input a size parameter n, a distance parameter $0 < \epsilon < 1$, and a confidence parameter $0 < \delta < 1/2$. Fixing an arbitrary function f and distribution D_n over $\{0,1\}^n$, the algorithm is also given access to a sequence of f-*labeled examples*, $(x_1, f(x_1)), (x_2, f(x_2)), \ldots$, where each x_i is independently drawn from the distribution D_n. We say that $\mathcal{A}$ is a* property testing algorithm *(or simply a* testing algorithm*) for the class of functions $\mathcal{F}$ if for every n, ϵ and δ and for every function f and distribution D_n over $\{0,1\}^n$ the following holds*

- *if $f \in \mathcal{F}_n$ then with probability at least $1-\delta$ (over the examples drawn from D_n and the possible coins tosses of $\mathcal{A}$), $\mathcal{A}$* accepts *f (i.e., outputs 1);*
- *if f is ϵ-far from $\mathcal{F}_n$ (with respect to D_n) then with probability at least $1-\delta$, $\mathcal{A}$* rejects *f (i.e., outputs 0).*

The sample complexity *of $\mathcal{A}$ is a function of n, ϵ and δ bounding the number of labeled examples examined by $\mathcal{A}$ on input (n, ϵ, δ).*

Though it was not stated explicitly in the definition, we shall also be interested in bounding the running time of a property testing algorithm (as a function of the parameters n, δ, ϵ, and in some case of a complexity measure of the class $\mathcal{F}$). We consider the following variants of the above definition: (1) D_n may be a specific distribution which is known to the algorithm. In particular, we shall be interested in testing with respect to the uniform distribution; (2) D_n may be restricted to a known class of distributions (e.g., product distributions); (3) The algorithm may be given access to an *oracle* for the function f, which when queried on $x \in \{0,1\}^n$, returns $f(x)$. In this case we refer to the number of queries made by $\mathcal{A}$ (which is a function of n, ϵ, and δ), as the *query complexity* of $\mathcal{A}$.

2.2. Property Testing and PAC Learning

A *Probably Approximately Correct* (PAC) learning algorithm [37] works in the same framework as that described in Definition 2.1 except for the following (crucial) differences: (1) It is given a *promise* that the unknown function f (referred to as the *target* function) belongs to $\mathcal{F}$; (2) It is required to output (with probability at least $1-\delta$) a *hypothesis* function h which is ϵ-close to f, where closeness is as defined in Equation (1) (and ϵ is usually referred to as the *approximation* parameter). Note that the differences pointed out above effect the tasks in opposite directions. Namely, the absence of a promise makes testing potentially harder than learning, whereas deciding whether a function belongs to a class rather than finding the function may make testing easier.

In the learning literature, a distinction is made between *proper* (or *representation dependent*) learning and *non-proper* learning [31]. In the former model, the hypothesis output by the learning algorithm is required to belong to the same function class as the target function f, i.e. $h \in \mathcal{F}$, while in the latter model, no such restriction is made. We stress that a proper learning algorithm (for $\mathcal{F}$) may either halt without output or output a function in $\mathcal{F}$, but it may not output functions not in $\mathcal{F}$.[5] There are numerous variants of PAC learning (including learning with respect to specific distributions, and learning with access to an oracle for the target function f). Unless stated otherwise, whenever we refer in this section to PAC learning we mean the *distribution-free no-query model* described above. The same is true for references to property testing. In addition, apart from one example, we shall restrict our attention to classes of Boolean functions.

TESTING IS NOT HARDER THAN PROPER LEARNING.

Proposition 2.1 *If a function class $\mathcal{F}$ has a* proper *learning algorithm $\mathcal{A}$, then $\mathcal{F}$ has a property testing algorithm $\mathcal{A}'$ such that $m_{\mathcal{A}'}(n,\epsilon,\delta) = m_{\mathcal{A}}(n, \epsilon/2, \delta/2) + O(\log(1/\delta)/\epsilon)$. Furthermore, the same relation holds between the running times of the two algorithm.*

The proof of this proposition, as well as of all other propositions in this section, can be found in our report [22]. The above proposition implies that if for every n, $\mathcal{F}_n$ has polynomial (in n) VC-dimension [38, 15], then $\mathcal{F}$ has a tester whose sample complexity is $\mathrm{poly}(n/\epsilon)\cdot\log(1/\delta)$. The reason is that classes with polynomial VC-dimension can be properly learned from a sample of the above size [15]. However, the running time of such a proper learning algorithm, and hence of the resulting testing algorithm might be exponential in n.

Corollary 2.2 *Every class which is learnable with a $\mathrm{poly}(n/\epsilon)$ sample is testable with a $\mathrm{poly}(n/\epsilon)$ sample (in at most exponential time).*

TESTING MAY BE HARDER THAN LEARNING. In contrast to Proposition 2.1 and to Corollary 2.2, we show that there are classes which are efficiently learnable (though not by a proper learning algorithm) but are not efficiently testable. This is proven by observing that many hardness results for proper learning (*cf.* [31, 13, 32]) actually establish the hardness of testing (for the same classes). Furthermore, we believe that it is more natural to view these hardness results as referring to testing. Thus, the separation between efficient learning and efficient proper learning translates to a separation between efficient learning and efficient testing.

[5] We remark that in case the function is $\mathcal{F}$ have an easy to recognize representation, one can easily guarantee that the algorithm never outputs a function not in $\mathcal{F}$. Standard classes considered in works on proper learning typically have this feature.

Proposition 2.3 *If $\mathcal{NP} \not\subseteq \mathcal{BPP}$ then there exist function classes which are not* $\mathrm{poly}(n/\epsilon)$*-time testable but are* $\mathrm{poly}(n/\epsilon)$*-time (non-properly) learnable.*

We stress that while Proposition 2.1 generalizes to learning and testing under specific distributions, and to learning and testing with queries, the proof of Proposition 2.3 uses the premise that the testing (or proper learning) algorithm works for any distribution and does not make queries.

TESTING MAY BE EASIER THAN LEARNING.

Proposition 2.4 *There exist function classes $\mathcal{F}$ such that $\mathcal{F}$ has a property testing algorithm whose sample complexity and running time are $O(\log(1/\delta)/\epsilon)$, yet any learning algorithm for $\mathcal{F}$ must have sample complexity exponential in n.*

The impossibility of learning the function class in Proposition 2.4 is due to its exponential VC-dimension, (i.e., it is a pure information theoretic consideration). We now turn to function classes of exponential (rather than double exponential) size. Such classes are always learnable with a polynomial sample, the question is whether they are learnable in polynomial-time. We present a function class which is easy to test but cannot be learned in polynomial-time (even under the uniform distribution), provided trapdoor one-way permutations exist (*e.g.,* factoring is intractable).

Proposition 2.5 *If there exist trapdoor one-way permutations then there exists a family of functions which can be tested in* $\mathrm{poly}(n/\epsilon)$*-time but can not be learned in* $\mathrm{poly}(n/\epsilon)$*-time, even with respect to the uniform distribution. Furthermore, the functions can be computed by* $\mathrm{poly}(n)$*-size circuits.*

The class presented in Proposition 2.5 consists of multi-valued functions. We leave it as an open problem whether a similar result holds for a class of Boolean functions.

LEARNING AND TESTING WITH QUERIES (under the uniform distribution). Invoking known results on linearity testing [14, 7, 19, 10, 11, 8] we conclude that there is a class of 2^n functions which can be tested within query complexity $O(\log(1/\delta)/\epsilon)$, and yet learning it requires at least n queries. Similarly, using results on low-degree testing [7, 6, 21, 36], there is a class of $\exp(2^n)$ function which can be tested within query complexity $O(\frac{\log(1/\delta)}{\epsilon} \cdot n)$, and yet learning it requires $\exp(n)$ many queries.

AGNOSTIC LEARNING AND TESTING. In a variant of PAC learning, called *Agnostic* PAC learning [26], there is no promise concerning the target function f. Instead, the learner is required to output a hypothesis h from a certain hypothesis class $\mathcal{H}$, such that h is ϵ-close to the function in $\mathcal{H}$ which is closest to f. The absence of a promise makes agnostic learning closer in spirit to property testing than basic PAC learning. In particular, agnostic learning with respect to a hypothesis class $\mathcal{H}$ implies proper learning of the class $\mathcal{H}$ and thus property testing of $\mathcal{H}$.

LEARNING AND TESTING DISTRIBUTIONS. The context of learning (cf., [25]) and testing distributions offers a dramatic demonstration to the importance of a promise (*i.e.*, the fact that the learning algorithm is required to work only when the target belongs to the class, whereas the testing algorithm needs to work for all targets which are either in the class or far away from it).

Proposition 2.6 *There exist distribution classes which are efficiently learnable (in both senses mentioned above) but cannot be tested with a subexponential sample (regardless of the running-time).*

3. Testing Graph Properties

We concentrate on testing graph properties using queries and with respect to the uniform distribution.

We consider undirected, simple graphs (no multiple edges or self-loops). For a simple graph G, we denote by $\mathrm{V}(\mathrm{G})$ its vertex set and assume, without loss of generality, that $\mathrm{V}(\mathrm{G}) = \{1, ..., |\mathrm{V}(\mathrm{G})|\}$. The graph G is represented by the (symmetric) Boolean function $g : \mathrm{V}(\mathrm{G}) \times \mathrm{V}(\mathrm{G}) \mapsto \{0, 1\}$ where $g(u, v) = 1$ if and only if there is an edge between u and v in G. This brings us to associated undirected graphs with directed graphs, where each edge in the undirected graph is associated with a pair of anti-parallel edges. Specifically, for a graph G, we denote by $\mathrm{E}(\mathrm{G})$ the set of ordered pairs which correspond to edges in G (i.e., $(u, v) \in \mathrm{E}(\mathrm{G})$ iff there is an edge between u and v in G). The distance between two N-vertex graphs, G_1 and G_2, is defined as the number of entries $(u, v) \in [N]^2$ ($[N] \stackrel{\mathrm{def}}{=} \{1, ..., N\}$) which are in the symmetric difference of $\mathrm{E}(\mathrm{G}_1)$ and $\mathrm{E}(\mathrm{G}_2)$. We denote

$$\mathrm{dist}(\mathrm{G}_1, \mathrm{G}_2) \stackrel{\mathrm{def}}{=} \frac{|(\mathrm{E}(\mathrm{G}_1) \setminus \mathrm{E}(\mathrm{G}_2)) \cup (\mathrm{E}(\mathrm{G}_2) \setminus \mathrm{E}(\mathrm{G}_1))|}{N^2}$$

This notation is extended naturally to a set, $\mathcal{C}$, of N-vertex graphs; that is, $\mathrm{dist}(\mathrm{G}, \mathcal{C}) \stackrel{\mathrm{def}}{=} \min_{\mathrm{G}' \in \mathcal{C}}\{\mathrm{dist}(\mathrm{G}, \mathrm{G}')\}$.

3.1. Testing k-Colorability

In this subsection we present an algorithm for testing the k-Colorability property for any given k. Namely, we are interested in determining if the vertices of a graph G can be colored by k colors so that no two adjacent vertices are colored by the same color, or if any k-partition of the graph has at least ϵN^2 violating edges (i.e. edges between pairs of vertices which belong to the same side of the partition).

The test itself is straightforward. We uniformly select a sample, denoted X, of $O\left(\frac{k^2(\log(k/\delta)}{\epsilon^3}\right)$ vertices of the graph, query all pairs of vertices in X to find which are edges in G, and check if the induced subgraph is k-Colorable. In lack of efficient algorithms for k-Colorability, for $k \geq 3$, we use the obvious exponential-time algorithm on the induced subgraph. The resulting algorithm is called the k-Colorability Testing Algorithm. Towards analyzing it, we define violating edges and good k-partitions.[6]

Definition 3.1.1 (violating edges and good k-partitions): *We say that an edge $(u, v) \in \mathrm{E(G)}$ is a* violating *edge with respect to a k-partition $\pi : \mathrm{V(G)} \to [k]$ if $\pi(u) = \pi(v)$. We shall say that a k-partition is* ϵ-good *if it has at most ϵN^2 violating edges (otherwise it is* ϵ-bad*). The partition is* perfect *if it has no violating edges.*

Theorem 3.1 *The* k-Colorability Testing Algorithm *is a property testing algorithm for the class of k-Colorable graphs whose query complexity is $\mathrm{poly}(k \log(1/\delta)/\epsilon)$ and whose running time is exponential in its query complexity. If the tested graph G is k-Colorable, then it is accepted with probability 1, and with probability at least $1-\delta$ (over the choice of the sampled vertices), it is possible to construct an ϵ-good k-partition of $\mathrm{V(G)}$ in time $\mathrm{poly}(k \log(1/\delta)/\epsilon) \cdot |\mathrm{V(G)}|$.*

Proof: If G is k-Colorable then every subgraph of G is k-Colorable, and hence G will always be accepted. The crux of the proof is to show that every G which is ϵ-far from the class of k-Colorable graphs, denoted $\mathcal{G}_k$, is rejected with probability at least $1-\delta$. We establish this claim by proving its counter-positive. Namely, that every G which is accepted with probability greater than δ, must have an ϵ-good k-partition (and is thus ϵ-close to $\mathcal{G}_k$). This is done by giving a (constructive) proof of the existence of an ϵ-good k-partition of $\mathrm{V(G)}$. Hence, in case $\mathrm{G} \in \mathcal{G}_k$, we also get an efficient probabilistic procedure for finding an ϵ-good k-partition of $\mathrm{V}(G)$. Note that if the test rejects G then we have a certificate that $\mathrm{G} \notin \mathcal{G}_k$, in form of the (small) subgraph induced by X which is not k-colorable.

We view the set of sampled vertices X as a union of two disjoint sets U and S, where U is a union of ℓ (disjoint) sets $\mathrm{U}^1, \ldots, \mathrm{U}^\ell$, each of size m. The size of S is m as well, where $m = O((\ell \log(k/\delta))/\epsilon)$ and $\ell = 4k/\epsilon$. The set U (or rather a k-partition of U) is used to define a k-partition of $\mathrm{V}(G)$. The set S ensures that with high probability, the k-partition of U which is induced by the perfect k-partition of $\mathrm{X} = \mathrm{U} \cup \mathrm{S}$, defines an ϵ-good partition of $\mathrm{V}(G)$.

In order to define a k-partition of $\mathrm{V(G)}$ given a k-partition of U, we first introduce the notion of a *clustering* of the vertices in $\mathrm{V(G)}$ with respect to this partition of U. More precisely, we define the clustering based on the k-partition of a subset $\mathrm{U}' \subset \mathrm{U}$, where this partition, denoted $(\mathrm{U}'_1, \ldots, \mathrm{U}'_k)$, is the one induced by the k-partition of U. The clustering is defined so that vertices in the same cluster have neighbors on the same sides of the partition of U'. For every $A \subseteq [k]$, the A-cluster, denoted C_A, contains all vertices in $\mathrm{V(G)}$ which have neighbors in U'_i for every $i \in A$ (and do not have neighbors in the other U'_i's). The clusters impose restrictions on possible extensions of the partition of U' to partitions $(\mathrm{V}_1, \ldots, \mathrm{V}_k)$ of all $\mathrm{V}(G)$, which do not have violating edges incident to vertices in U'. Namely, vertices in C_A should not be placed in any V_i such that $i \in A$. As a special case, $C_\emptyset$ is the set of vertices that do not have any neighbors in U' (and hence can be put on any side of the partition). In the other extreme, $C_{[k]}$ is the set of vertices that in any extension of the partition of U' will cause violations. For each i, the vertices in $C_{[k]\setminus\{i\}}$ are *forced* to be put in V_i, and thus are easy to handle. It is more difficult to deal with the the clusters C_A where $|A| < k-1$.[7]

Definition 3.1.2 (clusters): *Let U' be a set of vertices, and let π' be a perfect k-partition of U'. Define $U'_i \stackrel{\mathrm{def}}{=} \{v \in \mathrm{U} : \pi'(v) = i\}$. For each subset $A \subseteq [k]$ we define the A-*cluster with respect to π' *as follows:*

$$C_A \stackrel{\mathrm{def}}{=} \left(\bigcap_{i \in A} \Gamma(\mathrm{U}'_i)\right) \setminus \left(\bigcup_{i \notin A} \Gamma(\mathrm{U}'_i)\right) . \qquad (2)$$

The relevance of the above clusters becomes clear given the following definitions of extending and consistent partitions.

Definition 3.1.3 (consistent extensions): *Let U' and π' be as above. We say that a k-partition π of $\mathrm{V(G)}$* extends *a k-partition π' of U' if $\pi(u) = \pi'(u)$ for every $u \in \mathrm{U}'$. An extended partition π is* consistent *with π' if $\pi(v) \neq \pi'(u)$ for every $u \in \mathrm{U}'$ and $v \in \Gamma(u) \setminus C_{[k]}$, where $C_{[k]}$ is the $[k]$-cluster w.r.t π'.*

Thus, each vertex v in the cluster C_A (w.r.t π' defined on U') is forced to satisfy $\pi(v) \in \bar{A} \stackrel{\mathrm{def}}{=} [k] \setminus A$, for every k-partition π which extends π' in a consistent manner. There are no restrictions regarding vertices in $C_\emptyset$ and vertices in $C_{[k]}$ (the latter is guaranteed artificially in the definition and the consequences will have to be treated separately). For $v \in C_{[k]-\{i\}}$ the consistency condition forces $\pi(v) = i$.

We now focus on the main problem of the analysis. Given a k-partition of U, what is a good way to define a k-partition of $\mathrm{V}(G)$? Our main idea is to claim that with high probability the set U contains a subset U' so that the clusters with respect to the induced k-partition of U' determine whatever needs to be determined. That is, if these clusters allow to place some vertex on a certain side of the partition, then doing so does not introduce too many violating edges. The first step in implementing this idea is the notion of a *restricting* vertex.

Definition 3.1.4 (restricting vertex): *A pair (v, i), where $v \notin C_{[k]}$ and $i \in [k]$, is said to be* restricting *with respect to a k-partition π'* (of U') *if v has at least $\frac{\epsilon}{4}N$ neighbors*

[6] k-partitions are associated with mappings of the vertex set into the canonical k-element set $[k]$. The partition associated with $\pi : \mathrm{V(G)} \to [k]$ is $(\mathrm{V}_1 \stackrel{\mathrm{def}}{=} \pi^{-1}(1), \ldots, \mathrm{V}_k \stackrel{\mathrm{def}}{=} \pi^{-1}(k))$. We shall use the mapping notation π, and the explicit partition notation $(\mathrm{V}_1, \ldots, \mathrm{V}_k)$, interchangeably.

[7] In the Bipartite case, this is easy too (since $C_\emptyset$ is likely to contain few vertices of high degree).

in $\cup_{B:i\notin B} C_B$. Otherwise, (v,i) is non-restricting. *A vertex $v \in C_A$, where $A \neq [k]$, is* restricting *with respect to π' if for every $i \in \bar{A}$ the pair (v,i) is restricting. Otherwise, v is* non-restricting. *As always, the clusters are with respect to π'.*

Thus, a vertex $v \in C_A$ is restricting if for every $i \in \bar{A}$, adding v to U'_i (and thus to U') will cause may of its neighbors to move to a cluster corresponding to a bigger subset. That is, v's neighbors in the B-cluster (w.r.t $(\mathrm{U}'_1, \ldots, \mathrm{U}'_k)$) move to the $(B \cup \{i\})$-cluster (w.r.t $(U'_1, \ldots, U'_i \cup \{v\}, \ldots, \mathrm{U}'_k)$).

Given a prefect k-partition of U, we construct U' in steps starting with the empty set. At step j we add to U' a vertex $u \in \mathrm{U}^j$ (recall that $U = U^1 \cup \cdots \cup U^\ell$), which is a restricting vertex with respect to the k-partition of the current set U'. If no such vertex exists, the procedure terminates. When the procedure terminates (and as we shall see it must terminate after at most ℓ steps), we will be able to define, based on the k-partition of the final U', an ϵ-good k-partition of $\mathrm{V(G)}$. The procedure defined below is viewed at this point as a mental experiment. Namely, it is provided in order to show that with high probability there exists a subset U' of U with certain desired properties (which we later exploit).

Restriction Procedure (Construction of U')
Input: a perfect k-partition of $\mathrm{U} = \mathrm{U}^1 \cup \cdots \cup \mathrm{U}^\ell$.

1. $\mathrm{U}' \leftarrow \emptyset$.
2. For $j = 1, 2, \ldots$ do the following. Consider the current set U' and its partition π' (induced by the perfect k-partition of U).
   - If there are less than $(\epsilon/8)N$ restricting vertices with respect to π' then halt and output U'.
   - If there are at least $(\epsilon/8)N$ restricting vertices but there is no restricting vertex in U^j, then halt and output error.
   - Otherwise (there is a restricting vertex in U^j), add the first (by any fixed order) restricting vertex to U'.

Claim 3.1.5 *For every* U *and a perfect k-partition of* U*, after at most $\ell = 4k/\epsilon$ iterations, the Restriction Procedure halts and outputs either* U' *or* error.

The proof of this claim, as well as all other missing proofs, can be found in our report [22]. Before we show how U' can be used to define a k-partition π of $\mathrm{V(G)}$, we need to ensure that with high probability, the restriction procedure in fact outputs a set U' and not error. To this end, we first define the notion of a *covering set*.

Definition 3.1.6 (covering sets – for k-coloring): *We say that* U *is a* covering set *for* $\mathrm{V(G)}$*, if for* every *perfect k-partition of* U*, the Restriction Procedure, given this partition as input, halts with an output* $\mathrm{U}' \subset \mathrm{U}$ *(rather than an error message).*

In other words, U is such that for every perfect k-partition of U and for each of the at most ℓ iterations of the procedure, if there exist at least $(\epsilon/8)N$ restricting vertices with respect to the current partition of U', then U^j will include at least one such restricting vertex.

Lemma 3.1.7 *With probability at least $1 - \frac{\delta}{2}$, a uniformly chosen set of size $\ell \cdot m = O\left(\frac{k^2 \log(k/\delta)}{\epsilon^3}\right)$ is a covering set.*

Definition 3.1.8 (closed partitions): *Let U' be a set and π' a k-partition of it. We call (U', π')* closed *if there are less than $(\epsilon/8)N$ restricting vertices with respect to π'.*

Clearly, if the Restriction Procedure outputs a set U' then this set together with its (induced) partition are closed. If (U', π') is closed, then most of the vertices in $\mathrm{V(G)}$ are non-restricting. Recall that a non-restricting vertex v, belonging to a cluster C_A, $A \neq [k]$, has the following property. There exists at least one index $i \in \bar{A}$, such that (v,i) is non-restricting. It follows from Definition 3.1.4 that for every consistent extension of π' to π which satisfies $\pi(v) = i$ there are at most $\frac{\epsilon}{2}N$ violating edges incident to v.[8] However, even if v is non-restricting there might be indices $i \in \bar{A}$ such that (v,i) is restricting, and hence there may exist a consistent extensions of π' to π which satisfies $\pi(v) = i$ in which there are more than $\frac{\epsilon}{2}N$ violating edges incident to v. Therefore, we need to define for each vertex its set of *forbidden* indices which will not allow to have $\pi(v) = i$ for a restricting pair (v,i).

Definition 3.1.9 (forbidden sets): *Let (U', π') be closed and consider the clusters with respect to π'. For each $v \in V(G) \setminus \mathrm{U}'$ we define the* forbidden set of *v, denoted F_v, as the smallest set satisfying*

- *$F_v \supseteq A$, where $v \in C_A$.*
- *For every $i \in \bar{A}$, if v has at least $(\epsilon/4)N$ neighbors in the clusters C_B for which $i \notin B$, then i is in F_v.*

For $u \in \mathrm{U}'$, define $F_u = [k] \setminus \{\pi'(u)\}$.

Lemma 3.1.10 *Let (U', π') be an arbitrary closed pair and F_v's be as in Definition 3.1.9. Then:*

1. *$|\{v : (v \notin C_{[k]}) \wedge (F_v = [k])\}| \leq \frac{\epsilon}{8}N$.*
2. *Let π be any k-partition of $\mathrm{V(G)} \setminus \{v : F_v = [k]\}$ such that $\pi(v) \notin F_v$, for every $v \in V(G)$. Then, the number of edges $(v,v') \in \mathrm{E(G)}$ for which $\pi(v) = \pi(v')$ is at most $(\epsilon/2)N^2$.*

[8] First note that by definition of a consistent extension no vertex in cluster C_B, where $i \in B$, can have π-value i. Thus, all violated edges incident to v are incident to vertices in clusters C_B so that $i \notin B$. Since the pair (v,i) is non-restricting, there are at most $\frac{\epsilon}{2}N$ such edges.

The lemma can be thought of as saying that any k-partition which respects the forbidden sets is good (i.e., does not have many violating edges). However, the partition applies only to vertices for which the forbidden set is not $[k]$. The first item tells us that there cannot be many such vertices which do not belong to the cluster $C_{[k]}$. We next show that, with high probability over the choice of S, the k-partition π' of U' (induced by the k-partition of $\mathrm{U} \cup \mathrm{S}$) is such that $C_{[k]}$ is small. This implies that all the vertices in $C_{[k]}$ (which were left out of the partition in the previous lemma) can be placed in any side without contributing too many violating edges (which are incident to them).

Definition 3.1.11 (useful k-partitions): *We say that a pair* (U', π') *is* ϵ-useful *if* $|C_{[k]}| < \frac{\epsilon}{8}N$. *Otherwise it is* ϵ-unuseful.

The next claim directly follows from our choice of m and the above definition.

Claim 3.1.12 *Let* U' *be a fixed set of size* ℓ *and* π' *be a fixed* k*-partition of* U' *so that* (U', π') *is* ϵ*-unuseful. Let* S *be a uniformly chosen set of size* m*. Then, with probability at least* $\frac{\delta}{2}k^{-\ell}$*, there exists no perfect* k*-partition of* $\mathrm{U}' \cup \mathrm{S}$ *which extends* π'.

The following is a corollary to the above claim and to the fact that the number of possible closed pairs (U', π') determined by all possible k-partitions of U is at most k^{ℓ}.

Corollary 3.1.13 *If all closed pairs* (U', π') *which are determined by all possible* k*-partitions of* U *are unuseful, then with probability at least* $1 - \delta/2$ *over the choice of* S*, there is no perfect* k*-partition of* $\mathrm{X} = \mathrm{U} \cup \mathrm{S}$.

We can now wrap up the proof of Theorem 3.1. If G is accepted with probability greater than δ, then by Lemma 3.1.7, the probability that it is accepted and U is a covering set is greater than $\delta/2$. In particular, there must exist at least one covering set U, such that if U is chosen then G is accepted with probability greater than $\delta/2$ (with respect to the choice of S). That is, (with probability greater than $\delta/2$) there exists a perfect partition of $\mathrm{U} \cup \mathrm{S}$. But in such a case (by applying Corollary 3.1.13), there must be a useful closed pair (U', π') (where $\mathrm{U}' \subset \mathrm{U}$). If we now partition $\mathrm{V}(G)$ as described in Lemma 3.1.10, where vertices with forbidden set $[k]$ are placed arbitrarily, then from the two items of Lemma 3.1.10 and the usefulness of (U', π') it follows that there are at most ϵN^2 violating edges with respect to this partition. This completes the main part of the proof. ■ **(Theorem 3.1)**

3.2. Testing Max-Clique

Let $\omega(\mathrm{G})$ denote the size of the largest clique in graph G, and $\mathcal{C}_\rho \stackrel{\mathrm{def}}{=} \{\mathrm{G} : \omega(\mathrm{G}) \geq \rho \cdot |\mathrm{V}(\mathrm{G})|\}$ be the set of graphs having cliques of density at least ρ. The main result of this subsection is:

Theorem 3.2 *Let* $\ell \stackrel{\mathrm{def}}{=} O(\log(1/\epsilon\delta))$*. There exists a property testing algorithm,* $\mathcal{A}$*, for the class* $\mathcal{C}_\rho$ *whose edge-query complexity is* $O(\ell^2\rho^2/\epsilon^6)$ *and whose running time is* $\exp(\ell\rho/\epsilon^2)$*. In particular,* $\mathcal{A}$ *uniformly selects* $O(\ell^2\rho^2/\epsilon^4)$ *vertices in* G *and queries the oracle only on the existence of edges between these vertices. In case* $\mathrm{G} \in \mathcal{C}_\rho$*, one can also retrieve in time* $O(\ell^2\rho^2/\epsilon^4) \cdot |\mathrm{V}(\mathrm{G})|$ *a set of* $\rho \cdot |\mathrm{V}(\mathrm{G})|$ *vertices in* G *which is almost a clique (in the sense that it lacks at most* $\epsilon \cdot |\mathrm{V}(\mathrm{G})|^2$ *edges to being a clique).*

Theorem 3.2 is proven by presenting a seemingly unnatural algorithm/tester (see below). However, as a corollary, we observe that "the natural" algorithm, which uniformly selects $\mathrm{poly}(\log(1/\delta)/\epsilon)$ many vertices and accepts iff they induce a subgraph with a clique of density $\rho - \frac{\epsilon}{2}$, is a valid $\mathcal{C}_\rho$-tester as well.

Corollary 3.3 *Let* $m = \mathrm{poly}(1/\epsilon)$ *and let* R *be a uniformly selected set of* m *vertices in* $V(\mathrm{G})$*. Let* $\mathrm{G_R}$ *be the subgraph (of* G*) induced by* R*. Then,*

- *if* $\mathrm{G} \in \mathcal{C}_\rho$ *then* $\mathrm{Prob_R}[\omega(\mathrm{G_R}) > (\rho - \frac{\epsilon}{2}) \cdot m] > \frac{2}{3}$.
- *if* $\mathrm{dist}(\mathrm{G}, \mathcal{C}_\rho) > \epsilon$ *then* $\mathrm{Prob_R}[\omega(\mathrm{G_R}) \leq (\rho - \frac{\epsilon}{2}) \cdot m] > \frac{2}{3}$.

In the rest of this subsection we provide a motivating discussion to the algorithm asserted in Theorem 3.2. Recall that $N = |\mathrm{V}(\mathrm{G})|$ denotes the number of vertices in G.

Our first idea is to select at random a small sample U of $\mathrm{V}(\mathrm{G})$ and to consider all subsets U' of size $\frac{\ell}{2} \cdot |\mathrm{U}|$ of U where $|\mathrm{U}| = \mathrm{poly}(1/\epsilon)$. For each U' let $\mathrm{T}(\mathrm{U}')$ be the set of all vertices which neighbor every vertex in U' (i.e., $\mathrm{T}(\mathrm{U}') = \cap_{u \in \mathrm{U}'} \Gamma(u)$). In the subgraph induced by $\mathrm{T}(\mathrm{U}')$, consider the set $\mathrm{Y}(\mathrm{U}')$ of ρN vertices with highest degree in the induced subgraph. Clearly, if G is ϵ-far from $\mathcal{C}_\rho$, then $\mathrm{Y}(\mathrm{U}')$ misses at least ϵN^2 edges to being a clique (for every choice of U and U'). On the other hand, we show that if G has a clique C of size ρN then, with high probability over the choice of U, there exists a subset $\mathrm{U}' \subset \mathrm{U}$ such that $\mathrm{Y}(\mathrm{U}')$ misses at most $(\epsilon/3)N^2$ to being a clique (in particular, $\mathrm{U}' \subseteq \mathrm{C} \cap \mathrm{U}$ will do).

Assume that for any fixed U' we could sample the vertices in $\mathrm{Y}(\mathrm{U}')$ and perform edge queries on pairs of vertices in this sample. Then, a sample of $O(t/\epsilon^2)$ vertices (where $t = |\mathrm{U}|$) suffices for approximating the edge density in $\mathrm{Y}(\mathrm{U}')$ to within an $\epsilon/3$ fraction with probability $1 - O(2^{-t})$. In particular a sample can distinguish between a set $\mathrm{Y}(\mathrm{U}')$ which is far from being a clique and a set $\mathrm{Y}(\mathrm{U}')$ which is almost a clique. The point is that we need only consider $\binom{|\mathrm{U}|}{|\mathrm{U}'|} < 2^t$ possible sets $\mathrm{Y}(\mathrm{U}')$, where t is only a polynomial in $1/\epsilon$.

The only problem which remains is how to sample from $\mathrm{Y}(\mathrm{U}')$. Certainly, we can sample $\mathrm{T} = \mathrm{T}(\mathrm{U}')$, by sampling $\mathrm{V}(\mathrm{G})$ and testing membership in T, but how do we decide which vertex is among those of highest degree? The first idea is to estimate the degrees of vertices in T using an additional sample, denoted W. Thus, instead of considering the ρN

vertices of highest degree in T, we consider the ρN vertices in T having the most neighbors in $\mathrm{T} \cap \mathrm{W}$. The second idea is that we can sample T, order vertices in this sample according to the number of neighbors in $\mathrm{T} \cap \mathrm{W}$, and take the ρ fraction with the most such neighbors.

3.3. The General Partition Problem

The following General Graph Partition property generalizes all properties considered in previous subsections. In particular, it captured any graph property which requires the existence of partitions satisfying certain fixed density constraints. These constraints may refer both to the number of vertices on each side of the partition and to the number of edges between each pair of sides.

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 $\varrho_{j,j'}^{\text{LB}} \leq \varrho_{j,j'}^{\text{UB}}$ $(\forall j, j')$. Let $\mathcal{GP}_\Phi$ be the class of graphs which have a k-way partition $(\mathrm{V}_1, \ldots, \mathrm{V}_k)$ such that

$$\forall j, \;\; \rho_j^{\text{LB}} \cdot N \;\leq\; |\mathrm{V}_j| \;\leq\; \rho_j^{\text{UB}} \cdot N \;, \tag{3}$$

$$\forall j, j', \;\; \varrho_{j,j'}^{\text{LB}} \cdot N^2 \;\leq\; |\mathrm{E}(\mathrm{V}_j, \mathrm{V}_{j'})| \;\leq\; \varrho_{j,j'}^{\text{UB}} \cdot N^2 \;, \tag{4}$$

where $\mathrm{E}(\mathrm{V}_j, \mathrm{V}_{j'})$ denotes the set of edges with one endpoint in V_j and one in $\mathrm{V}_{j'}$. That is, Eq. (3) places lower and upper bounds on the relative sizes of the various parts; whereas Eq. (4) imposes lower and upper bounds on the density of edges among the various pairs of parts. For example, k-colorability is expressed by setting $\varrho_{j,j}^{\text{UB}} = 0$ for every j (and setting $\rho_j^{\text{LB}} = 0$, $\rho_j^{\text{UB}} = 1$, and similarly setting the $\varrho_{j,j'}^{\text{xx}}$'s for $j' \neq j$).

Theorem 3.4 *There exists an algorithm $\mathcal{A}$ such that for every given set of parameters Φ, algorithm $\mathcal{A}$ is a property testing algorithm for the class $\mathcal{GP}_\Phi$ with query complexity $(O(k^2)/\epsilon)^{k+5} \cdot k^2 \log(k/\epsilon\delta)$, and running time $\exp\left((O(k^2)/\epsilon)^{k+2} \cdot \log(k/\epsilon\delta)\right)$.*

Recall that better complexities for Max-CUT and Bisection (as well as for k-Colorability and ρ-Clique), are obtained by custom-made algorithms.

Acknowledgments

We wish to thank Noga Alon, Ravi Kannan, David Karger and Madhu Sudan for useful discussions.

References

[1] 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.

[2] S. Arora, A. Frieze, and H. Kaplan. A new rounding procedure for the assignment problem with applications to dense graph arrangement problems. In *these proceedings*, 1996.

[3] S. Arora, D. Karger, and M Karpinski. Polynomial time approximation schemes for dense instances of NP-hard problems. In *27th STOC*, pages 284–293, 1995.

[4] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and intractability of approximation problems. In *33rd FOCS*, pages 14–23, 1992.

[5] S. Arora and S. Safra. Probabilistic checkable proofs: A new characterization of NP. In *33rd FOCS*, pages 1–13, 1992.

[6] L. Babai, L. Fortnow, L. Levin, and M. Szegedy. Checking computations in polylogarithmic time. In *23nd STOC*, pages 21–31, 1991.

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

[8] M. Bellare, D. Coppersmith, J. Håstad, M. Kiwi, and M. Sudan. Linearity testing in characteristic two. In *36th FOCS*, pages 432–441, 1995.

[9] M. Bellare, O. Goldreich, and M. Sudan. Free bits, pcps and non-approximability – towards tight results. In *36th FOCS*, pages 422–431, 1995.

[10] M. Bellare, S. Goldwasser, C. Lund, and A. Russell. Efficient probabilistically checkable proofs and applications to approximation. In *25th STOC*, pages 294–304, 1993.

[11] M. Bellare and M. Sudan. Improved non-approximability results. In *26th STOC*, pages 184–193, 1994.

[12] S. Ben-David. Can finite samples detect singularities of real-valued functions? In *24th STOC*, pages 390–399, 1992.

[13] A. Blum and R. Rivest. Training a 3-node neural network is NP-complete. In *Advances in Neural Information Processing Systems I*, pages 494–501, 1989.

[14] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47:549–595, 1993.

[15] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *JACM*, 36(4):929–965, October 1989.

[16] T. M. Cover. On determining the rationality of the mean of a random variable. *Annals of Statistics*, 1:862–871, 1973.

[17] W. F. de la Vega. MAX-CUT has a randomized approximation scheme in dense graphs. To appear in *Random Structures and Algorithms*, 1994.

[18] K. Edwards. The complexity of colouring problems on dense graphs. *Theoretical Computer Science*, 43:337–343, 1986.

[19] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Approximating clique is almost NP-complete. In *32nd FOCS*, pages 2–12, 1991.

[20] A. Frieze and R. Kanan. The regularity lemma and approximation schemes for dense problems. In *these proceedings*, 1996.

[21] P. Gemmell, R. Lipton, R. Rubinfeld, M. Sudan, and A. Wigderson. Self-testing/correcting for polynomials and for approximate functions. In *23nd STOC*, pages 32–42, 1991.

[22] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. Available from `http://theory.lcs.mit.edu/~oded/ggr.html`.

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

[24] W. Hoeffding and J. Wolfowitz. Distinguishability of sets of distributions. *Annals of Mathematical Statistics*, 29:700–718, 1958.

[25] M. J. Kearns, Y. Mansour, D. Ron, R. Rubinfeld, R. E. Schapire, and L. Sellie. On the learnability of discrete distributions. In *The 25th Annual ACM Symposium on Theory of Computing*, pages 273–282, 1994.

[26] M. J. Kearns, R. E. Schapire, and L. M. Sellie. Toward efficient agnostic learning. In *5th COLT*, pages 341–352, 1992.

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

[28] S. R. Kulkarni and O. Zeitouni. On probably correct classification of concepts. In *6th COLT*, pages 111–116, 1993.

[29] L. Lovász and N. Young. Lecture notes on evasiveness of graph properties. Technical Report TR–317–91, Princeton University, Computer Science Department, 1991.

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

[31] L. Pitt and L. G. Valiant. Computational limitations on learning from examples. *JACM*, 35(4):965–984, October 1988.

[32] L. Pitt and M. K. Warmuth. The minimum consistent DFA problem cannot be approximated within any polynomial. *JACM*, 40(1):95–142, January 1993.

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

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

[35] R. Rubinfeld. Robust functional equations and their applications to program testing. In *35th FOCS*, 1994.

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

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

[38] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its applications*, 17(2):264–280, 1971.

[39] K. Yamanishi. Probably almost discriminative learning. *Machine Learning*, 18:23–50, 1995.

[40] A. C. C. Yao. Lower bounds to randomized algorithms for graph properties. In *28th FOCS*, pages 393–400, 1987.

[41] O. Zeitouni and S. R. Kulkarni. A general classification rule for probability measures. To appear in Annals of Statistics, 1991.