Electronic Colloquium on Computational Complexity, Report No. 15 (2007)

On the Randomness Complexity of Property Testing*

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

Or Sheffet[‡]
24 Tirza street
Ramat Gan
Israel.
or.sheffet@gmail.com

February 27, 2007

Abstract

We initiate a general study of the randomness complexity of property testing, aimed at reducing the randomness complexity of testers without (significantly) increasing their query complexity. One concrete motovation for this study is provided by the observation that the product of the randomness and query complexity of a tester determine the actual query complexity of implementing a version of this tester that utilizes a weak source of randomness (through a randomness-extractor). We present rather generic upper- and lower-bounds on the randomness complexity of property testing and study in depth the special case of testing bipartiteness in two standard property testing models.

Keywords: Property Testing, Randomness Complexity, Weak Sources of Randomness, Randomness Extractors, Sampling.

*This work is based on the M.Sc. thesis of the second author, which was completed under the supervision of the first author.

[†]Partially supported by the Israel Science Foundation (grant No. 460/05).

[‡]Work done while Or was a graduate student at the Weizmann Institute of Science.

ISSN 1433-8092

1 Introduction

Property testing [RS, GGR] is concerned with a relaxed type of decision problems; specifically, for a fixed property (resp., a set) Π, the task is to distinguish between objects that have property Π (resp., are in Π) and objects that are "far" from have property Π (resp., are "far" from any object in Π). The focus of property testing is on sublinear-time algorithms, which in particular cannot examine the entire object. Instead, these algorithms, called *testers*, may obtain bits in the representation of the object by issuing adequate queries. Indeed, in this case, the query complexity of testers becomes a measure of central interest.

For natural properties, testers of sub-linear query-complexity must be randomized (see articulation in Section 2.1). This is a qualitative assertion, and the corresponding quantitative question arises naturally: for any fixed property Π and a sub-linear function q, *what is the randomness-complexity of testers for* Π *that have query-complexity* q*?*

In addition to the natural appeal of the foregoing question, there are concrete reasons to care about it. Firstly, the randomness-complexity of tester determines the length of PCPs that are constructed on top of these testers. Indeed, this was the motivation for the interest of [GS, BSVW] in reducing the randomness complexity of low-degree testing. Secondly, the randomness-complexity of a tester affects the complexity of implementing a version of this tester while utilizing a weak source of randomness. This motivation is further discussed in Section 1.2.

Indeed, the randomness-complexity of testers was considered in some prior work, starting in [GS]. This subject is the pivot of [BSVW] and the main topic of [SW]. However, all these works refer to specific (algebraic) tasks (i.e., testing low-degree polynomials and group homomorphisms). In contrast, our focus in this paper is either on general properties (see Section 1.4) or on specific combinatorial properties (see Section 1.3).

1.1 The Perspective of Average-Estimation

Property testing is a vast generalization of the task of estimating the average value of a function. Specifically, consider the task of distinguishing between functions $f : \{0,1\}^n \to \{0,1\}$ having average value exceeding 0.5 and functions that are ϵ-far from having this property (i.e., functions having average value below $0.5 - \epsilon$). Clearly, this task can be solved by a randomized algorithm that queries the function at $O(1/\epsilon^2)$ (random) points. This query-complexity is optimal and any algorithm achieving it, called a *sampler*, must be randomized (see [CEG]). Furthermore, a quantitative study of the randomness-complexity of samplers in terms of their query-complexity was also carried out in [CEG]. The current paper may be viewed as extending this study to the domain of general property testing.

Note that estimating the average value of a function corresponds to very restricted properties of functions. In particular, these properties are *symmetric* (i.e., are invariant under any relabeling of the inputs to the function). In contrast, most of the study of property testing refers to properties that are not symmetric (e.g., being a low-degree polynomial, monotonicity, representing a graph that has a certain graph property, etc). Furthermore, while all symmetric properties of Boolean functions are easily testable by straightforward sampling, this cannot be said about property testing in general (nor about the numerous special cases that were studied in the last decade [F, R]).

1.2 A Concrete Motivation: Using Weak Sources of Randomness

In the standard context of randomized algorithms, a concrete motivation for minimizing the randomness-complexity is provided by the exponential effect of the latter measure on the time-complexity of a possible derandomization. In contrast, in the context of property testing, derandomization is typically infeasible, because (as noted above) deterministic testers cannot have sub-linear query complexity. Instead, a different motivation, advocated in [G] becomes very relevant in this context.

We refer to the effect of the randomness-complexity on the overhead involved in implementing the tester when using only weak sources of randomness (rather than perfect ones). Specifically, we refer to the paradigm of implementing randomized algorithms by using (a single sample from) such a weak source, and trying all possible seeds to an adequate randomness extractor (see below). We shall see that the overhead created by this method is determined by the randomness-complexity of the original algorithm.

Recall that a *randomness extractor* is a function $E : \{0,1\}^s \times \{0,1\}^n \to \{0,1\}^r$ that uses an s-bit long random seed in order to transform an n-bit long (outcome of a) weak source of randomness into an r-bit long string that is almost uniformly distributed in $\{0,1\}^r$. Specifically, we consider arbitrary weak sources that are restricted (only) in the sense that, for a parameter k, no string appears as the source outcome with probability that exceeds 2^{-k}. Such sources are called (n,k)-sources (and k is called the min-entropy). Now, E is called a (k,ϵ)-extractor if for any (n,k)-source X it holds that $E(U_s,X)$ is ϵ-close to U_r, where U_m denotes the uniform distribution over m-bit strings (and the term 'close' refers to the statistical distance between the two distributions). For further details about (k,ϵ)-extractors, the interested reader is referred to Shaltiel's survey [Shal].

Next we recall the standard paradigm of implementing randomized algorithms while using sources of weak randomness. Suppose that the algorithm A has time-complexity t and randomness-complexity $r \leq t$. Recall that, typically, the analysis of algorithm A refers to what happens when A obtains its randomness from a perfect random source (i.e., for each possible input α, we consider the behavior of $A(\alpha,U_r)$, where $A(\alpha,\omega)$ denotes the output of A on input α when given randomness ω). Now, suppose that we have at our disposal only a weak source of randomness; specifically, a (n,k)-source for $n \gg k \gg r$ (e.g., $n = 10k$ and $k = 2r$). Then, using a (k,ϵ)-extractor $E : \{0,1\}^s \times \{0,1\}^n \to \{0,1\}^r$, we can transform the n-bit long outcome of the weak source into 2^s strings, each of length r, and use the resulting 2^s strings (which are "random on the average") in 2^s corresponding invocations of the algorithm A. That is, upon obtaining the outcome $x \in \{0,1\}^n$ from the source, we invoke the algorithm A for 2^s times such that in the $i^{\rm th}$ invocation we provide A with randomness $E(i,x)$. The results of these 2^s invocations are processed in the natural manner. For example, if A is a decision procedure, then we output the majority vote obtained in the 2^s invocations (i.e., when given the input α, we output the majority vote of the sequence $\langle A(\alpha,E(i,x))\rangle_{i=1,\ldots,2^s}$). As shown in Appendix A.1.1, this decision will be correct with probability at least $1-2(p+\epsilon)$, where p denotes the error probability of $A(\alpha,U_r)$.

Let us consider the cost of the foregoing implementation. We assume, for simplicity, that the running-time of the randomness extractor is dominated by the running-time of A. Then, algorithm A can be implemented using a weak source, while incurring an overhead factor of 2^s. Recalling that $s > \log_2(n-k)$ and $n > k > r$ must hold (cf. [Shal]), it follows that for $k = n - \Omega(n)$ the aforementioned overhead is at least linear in r. On the other hand, for $n = O(k) = O(r)$ (resp., $n = {\rm poly}(k) = {\rm poly}(r)$) efficient randomness-extractors using $s = (1+o(1))\log_2 n$ (resp., $s = O(\log n)$) are known (see Appendix A.1.2). This establishes our claim that the time-complexity of implementing randomized algorithms when using weak sources is related to the randomness-complexity of these algorithms. The same applies to the query complexity of testers. Specifically, for $n = O(k) = O(r)$ (resp., $n = {\rm poly}(k) = {\rm poly}(r)$) the query-complexity of implementing a tester is almost linear in $r \cdot q$ (resp., is ${\rm poly}(r) \cdot q$), where q is the query-complexity of the original tester (which use a perfect source of randomness).

1.3 Specific Algorithms

The motivation discussed in Section 1.2 is best illustrated by our results regarding testing bipartiteness *in the bounded-degree model* of [GR1]. Specifically, fixing a degree bound d, the task is to distinguish (N-vertex) bipartite graphs of maximum degree d from (N-vertex) graphs of maximum degree d that are ϵ-far from bipartite (for some parameter ϵ), where ϵ-far means that $\epsilon \cdot dN$ edges have to be omitted from the graph in order to yield a bipartite graph. It is easy to see that no deterministic algorithm of $o(N)$ time-complexity can solve this problem. Yet, there exists a probabilistic algorithm of time-complexity $\tilde{O}(\sqrt{N}\text{poly}(1/\epsilon))$ that solves this problem correctly (with probability 2/3). This algorithm makes $q \stackrel{\text{def}}{=} \tilde{O}(\sqrt{N}\text{poly}(1/\epsilon))$ incidence-queries to the graph, and (as described in the work [GR2]) has randomness-complexity $r > q > \sqrt{N}$ (yet $r < q \cdot \log_2 N$).[1]

Let us now turn to the question of implementing the foregoing tester in a setting where we have access only to a weak source of randomness. In this case, the implementation calls for invoking the original tester $\tilde{O}(r)$ times, which yields a total running time of $\tilde{O}(r) \cdot \tilde{O}(\sqrt{N}\text{poly}(1/\epsilon)) > N$ (and the same bound holds for its query-complexity). But in such a case we better use the standard (deterministic) decision procedure for bipartiteness!

Fortunately, a randomness-efficient implementation of the original tester of [GR2] is possible. This implementation (presented in Section 3.2) has randomness-complexity $r' = \text{poly}(\epsilon^{-1} \log N)$ (rather than $r = \text{poly}(\epsilon^{-1} \log N) \cdot \sqrt{N}$). Thus, the cost of the implementation that uses a weak source of randomness is related to $r' \cdot s = \tilde{O}(\sqrt{N}\text{poly}(1/\epsilon))$, which matches the original bound (up to differences hidden in the $\tilde{O}()$ and poly() notation).

The randomness-efficient implementation of the [GR2]-tester presented in Section 3.2 is based on pin-pointing the "random features" used in the original analysis, and providing an alternative implementation that satisfies the same features. In contrast, the randomness-efficient tester presented in Section 3.1 is based on new ideas.

In Section 3.1 we consider testers for graph properties *in the adjacency matrix model* of [GGR]. Specifically, we consider the task of testing bipartiteness. Recall that the tester presented in [GGR] works by selecting a random set of $\tilde{O}(\epsilon^{-2})$ vertices and inspecting the (corresponding) induced subgraph. In fact, as shown [GGR], it suffices to make $\tilde{O}(\epsilon^{-3})$ queries. A randomness-efficient implementation of the "random features" used in the original analysis, allows reducing the randomness-complexity to $\tilde{O}(\epsilon^{-1}) \cdot \log_2 N$, where N denotes the number of vertices. In contrast, using an alternative approach, we present a tester of randomness-complexity $O(\log(1/\epsilon)) \cdot \log_2 N$, while maintaining a query-complexity bound of $\tilde{O}(\epsilon^{-3})$. The latter randomness-efficient tester is the main technical contribution of this work. In the next paragraph, we provide an extremely high-level description of the principles underlying its design.

The original tester works by first selecting a random sample of $t = \tilde{O}(\epsilon^{-1})$ vertices, and the analysis refers to 2^t candidate 2-colorings that are induced by all possible 2-partitions of this sample. The tester then selects an auxiliary sample of $\tilde{O}(t/\epsilon)$ vertex-pairs and checks whether this sample rules out all these 2^t candidate 2-colorings. The analysis boils down to showing that if the graph is ϵ-far from bipartite then, with high probability, all these candidate 2-colorings are ruled out. This is shown by applying a union bound on this set of 2^t candidate 2-colorings, which means that each candidate has to be ruled out with probability at least $1 - 2^{-t}$. Thus, the randomness complexity of any implementation of this tester must exceed t. Seeking to achieve randomness-complexity that is linearly related to $\log t$, we performs a preliminary step aimed at obtaining a single 2-partition of the initial t-vertex sample that induces a single candidate 2-coloring, which will be checked as

[1]We comment that $\Omega(\sqrt{N})$ is a lower-bound on the query-complexity of any property tester of bipartiteness (in the bounded-degree model; see [GR1]).

in the original tester. The preliminary step obtains such a 2-partition by collecting constraints on the mutual placements of pair of vertices. These constraints are found using the same mechanism that underlies the checking of candidates in the original tester. The punch-line is that here we are dealing with $\binom{t}{2}$ (rather than 2^t) events, which allows us to work with an error probability of $t^{-2}/O(1)$ (rather than $2^{-t}/O(1)$) per each event.

1.4 Generic Bounds

In contrast to the specific algorithms described in Section 1.3, we now consider quite generic lower- and upper-bounds on the randomness-complexity of property testers as a function of their query-complexity. We stress that these results do not refer to the time-complexity of the testers, which makes the lower-bounds stronger (and the upper-bound weaker).

Loosely speaking, we show that, for a wide class of properties of functions defined over a domain of size D, *the randomness-complexity of testing with q queries is essentially* $\log_2(D/q)$. Needless to say, the dependence on the query-complexity is essential, because deterministic testers of query-complexity D exist for any property. Furthermore, the randomness-complexity of any tester can be decreased by an additive term of t while increasing the query complexity by a factor of 2^t.

The lower-bounds established in Section 2.1 are exactly of the foregoing form, and they apply to two general and natural classes of properties. In particular, these lower-bounds apply to testing low-degree polynomials (cf., e.g., [BLR, RS]), locally-testable codes (cf., e.g., [GS]), testing graph properties (in both the adjacency matrix and incidence-list models, see [GGR, GR1], resp.), testing monotonicity (cf., e.g., [GGLRS]), and testing of clustering (cf., e.g., [ADPR]). The upper-bound established in Section 2.2 refers to any property but is actually of the form $\log_2 D + \log_2\log_2 R + O(1)$ (rather than $\log_2(D/q)$), where R is the size of the range of the functions we refer to.

2 Generic Bounds

We consider testing properties of functions from D to R. Fixing a set of such functions Π, we say that a randomized oracle machine T is an ϵ-tester for Π if the following two conditions hold:

1. For every $f \in \Pi$ it holds that $\Pr[T^f = 1] \geq 2/3$.
2. For every f that is ϵ-far from Π it holds that $\Pr[T^f = 1] \leq 1/3$, where f is ϵ-far from Π if for every $g \in \Pi$ it holds that $\Pr_{x\in D}[f(x) \neq g(x)] > \epsilon$.

In case the first condition holds with probability 1, we say that T has one-sided error. The query and randomness complexities of T are defined in the natural manner. A tester is called non-adaptive if it determines its queries based solely on its internal coin-tosses (and independently of the answers to prior queries).

Note that we have defined property testers for finite properties and a fixed value of the proximity parameter ϵ. The definition may be extended to infinite properties and varying ϵ, by providing the tester with $|D|, |R|$ and ϵ as inputs (and assuming $D = [D]$).[2]

2.1 Lower Bounds

We provide lower-bounds on the randomness complexity of testing two general classes of properties.

[2]Occasionally, we shall assume that $\epsilon \geq |D|^{-1}$; otherwise, ϵ-testers coincide with standard decision procedures.

2.1.1 Strongly evasive properties

We first consider properties that are "strongly evasive" in the sense that determining the values of some function at a constant fraction of the domain leaves the promise problem (of distinguishing between yes-instances and "far from yes"-instances) undetermined.[3] That is, for fixed parameters ϵ and ρ, the property Π is called strongly evasive if there exists a function $f : D \to R$ such that for every $D' \subset D$ of density ρ, there exists $f_1 \in \Pi$ and $f_0 : D \to R$ that is ϵ-far from Π such that for every $x \in D'$ it holds that $f_1(x) = f_0(x) = f(x)$. Many natural properties that are strongly evasive (with respect to various pairs of parameters); see examples below. The following result can be easily proved by extending a similar result regarding samplers (which is presented in [CEG]).

Theorem 1 *Let Π be strongly evasive with respect to ϵ and ρ. Then any ϵ-tester for Π that has query complexity q, must have randomness complexity greater than $\log_2(\rho|D|/q)$.*

Proof: Let T be an arbitrary ϵ-tester of query-complexity q and randomness-complexity r, and f be a function witnessing the fact that Π is strongly evasive. For every $\omega \in \{0,1\}^r$, we consider the set of queries made by T when the outcome of T's coin-tosses equals ω *and T is given oracle access to f*. Denoting the latter set by Q_ω, we let $D' = \cup_{\omega\in\{0,1\}^r} Q_\omega$. Clearly, $|D'| \leq 2^r \cdot q$. The theorem follows by proving that $|D'| > \rho \cdot |D|$.

Suppose towards the contradiction that $|D'| \leq \rho \cdot |D|$. Then there exists $f_1 \in \Pi$ and $f_0 : D \to R$ that is ϵ-far from Π such that for every $x \in D'$ it holds that $f_1(x) = f_0(x) = f(x)$. It follows that T^{f_1} and T^{f_0} behaves exactly as T^f (because all these functions agree on D'), which yields a contradiction because T must accept f_1 with probability at least $2/3$ and accept f_0 with probability at most $1/3$. ■

Some applications. Many graph properties are strongly evasive, but since such properties will be at the focus of Section 2.1.2, we mention first a few examples that refer to different types of properties.

1. *Multi-variate polynomial.* For every m and d, we consider the set of m-variate polynomial of total degree d (over a finite field F). To see that this set of functions is strongly evasive consider the all-zero function, f, and let $f_1 = f$. Then, for every D' of density $1/2$, let $f_0(x) = 0$ if $x \in D'$ and $f_0(x) = 1$ otherwise. Assuming $|F| > 4d$, it follows that f_0 is $1/4$-far from any degree d polynomial.

2. *Codes of linear distance.* A binary code $C \subset \{0,1\}^n$ of distance $d = \Omega(n)$, is viewed as a set of functions of the form $f : [n] \to \{0,1\}$, where each function corresponds to a codeword. To see that this set is strongly evasive consider any codeword f, and let $f_1 = f$. Then, for every D' of density $1 - (d/2n)$, let $f_0(x) = f(x)$ if $x \in D'$ and $f_0(x) = 1 - f(x)$ otherwise. Clearly, f_0 is $(d/2n)$-far from any codeword.

3. *Monotone functions.* 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). To see that the set of monotone functions is strongly evasive consider the all-one function f, and let $f_1 = f$. Then, for every D' of density

[3] This notion of "strongly evasive" is incomparable to the standard definition of evasiveness (cf. [LY]). On one hand, strong evasiveness has a non-deterministic flavor and furthermore it refers to the relaxation of property testing. On the other hand, we shall focus on $\rho \ll 1$, whereas standard evasiveness refers to $\delta = 1 - |D|^{-1}$.

$1/4$, let $f_0(\sigma z) = f(\sigma z)$ if $\{0z, 1z\} \cap D' \neq \emptyset$ and $f_0(\sigma z) = 1 - \sigma$ otherwise. Note that if $\{0z, 1z\} \cap D' = \emptyset$ then f_0 must be modified at either $0z$ or $1z$ in order to obtain a monotone function. Thus, f_0 is $1/4$-far from being monotone.

Turning back to graph properties, we focus on the bounded incidence lists model (of [GR1]) because the results of Section 2.1.2 do not apply to it. We mention a few properties of bounded-degree graphs that are strongly evasive in the (bounded) incidence lists model. Examples include connectivity and being Eulerian (or Hamiltonian), which can be demonstrated to be strongly evasive by starting with the N-cycle (and omitting edges). Additional examples such as planarity and bipartiteness can be demonstrated to be strongly evasive by starting with the empty graph (and adding edges).

2.1.2 Relabeling-invariant properties

We now consider properties that are invariant under some "nice" relabeling of D. Specifically, for any set S_D of permutations over D, we say that the property Π is S_D-invariant if for every $f : D \to R$ and every $\pi \in S_D$ it holds that $f \in \Pi$ if and only $(f \circ \pi) \in \Pi$, where $(f \circ \pi)(x) = f(\pi(x))$. We consider only sets S_D that correspond to a transitive group of permutations over D; that is, S_D is permutation group and for every $x, y \in D$ there exists a permutation $\pi \in S_D$ such that $\pi(x) = y$. Needless to say, the set of all permutations is a transitive group of permutations, but so are also many other permutation groups (e.g., the group of all cyclic permutations).

Theorem 2 *Let S_D be a transitive group of permutations over D, and Π be a non-empty and S_D-invariant property of functions from D to R. Suppose that, for some $\sigma \in R$, the all-σ function is 2ϵ-far from Π. Then any* non-adaptive *ϵ-tester for Π that has query complexity q, must have randomness complexity at least* $\log_2(|D|/q) - 1$.

Proof: Like the proof of Theorem 1, the current proof is based on deriving a contradiction from the hypothesis that the tester never examines most of the function (i.e., $|D'| \ll |D|$). The difference is in the way that this contradiction is derived, since we can no longer take the straightforward route offered by strong evasiveness.

Let T be an ϵ-tester for Π, and denote its query-complexity and randomness-complexity by q and r respectively. Since T is non-adaptive, its queries are oblivious of the oracle. For every $\omega \in \{0,1\}^r$, we denote by Q_ω the set of queries made by T when the outcome of its coin-tosses equals ω, and let $D' = \cup_{\omega \in \{0,1\}^r} Q_\omega$. Again, $|D'| \leq 2^r \cdot q$, and the theorem follows by proving that $|D'| > |D|/2$.

Let $f : D \to R$ be a function in Π with the maximum number of σ values, among all functions in Π. By the hypothesis, $|\{x \in D : f(x) \neq \sigma\}| > 2\epsilon|D|$. Suppose, for a moment, that $|\{x \in D \setminus D' : f(x) \neq \sigma\}| \geq \epsilon|D|$, and let h be defined such that $h(x) = f(x)$ if $x \in D'$ and $h(x) = \sigma$ otherwise. Then (by the maximility of f), h is ϵ-far from Π. However, T^h behaves exactly as T^f (because h and f agree on D'), which yields a contradiction because T must accept f with probability at least $2/3$ and accept h with probability at most $1/3$.

It is left to prove that if $|D \setminus D'| \geq |D|/2$ then $|\{x \in D \setminus D' : f(x) \neq \sigma\}| \geq \epsilon|D|$. This does not necessarily hold, but we shall show that it holds when replacing f by another function in Π that also has a maximum number of σ values. Here we use the hypothesis that Π is a S_D-invariant property, where S_D is a transitive group of permutations over D. Specifically, consider a random permutation $\pi \in S_D$, and let $f' = (f \circ \pi) \in \Pi$. Then, $f' \in \Pi$ and $|\{x \in D : f'(x) \neq \sigma\}| > 2\epsilon|D|$. On the other hand, since S_D is a transitive group of permutations over D, for every $x, y \in D$ it holds that $\Pr_{\pi\in S_D}[\pi(x)=y] = 1/|D|$. It follows that, for a random permutation $\pi \in S_D$, the *expected size* of $\{x \in D \setminus D' : f'(x) \neq \sigma\}$ equals

$$|D \setminus D'| \cdot \frac{|D \setminus f^{-1}(\sigma)|}{|D|} \geq \epsilon|D|\,,$$

where the inequality is due to the hypotheses $|D \setminus D'| \geq |D|/2$ and $|D \setminus f^{-1}(\sigma)| > 2\epsilon|D|$. Thus, there exists a $f' \in \Pi$ such that $|\{x \in D \setminus D' : f'(x) \neq \sigma\}| \geq \epsilon|D|$, and the theorem follows. ■

Main application. As hinted in Section 2.1.1, the most appealing application of Theorem 2 is to testing graph properties in the adjacency matrix model (of [GGR]). In this model, N-vertex graphs are represented by Boolean functions defined over $[N] \times [N]$. For technical reasons, we prefer to represent such graphs as Boolean functions defined over the set of the $\binom{N}{2}$ (unordered) vertex-pairs, which is actually more natural (as well as non-redundant). Note that the set of all permutations over $[N]$ induces a transitive group of permutations over these pairs, where the permutation $\pi : [N] \to [N]$ induces a permutation that maps pairs of the form $\{i, j\}$ to $\{\pi(i), \pi(j)\}$. Indeed, any graph property is invariant under this group, and Theorem 2 can be applied whenever either the empty graph or the complete graph is far from the property. We note that all the graph properties considered in [GGR] fall into the latter category (and that the testers of [GGR] are all non-adaptive).[4]

Other applications. We note that any property that refers to sets of objects (e.g., sets of points as in [ADPR]) is invariant under the group of all permutations. Another application domain consists of matrix-properties that are preserved under row and column permutations.

Generalizations. Theorem 2 can be generalized to properties that are S_D-invariant under a set of permutations that is "sufficiently mixing" in the sense that a permutation selected uniformly in S_D maps each element of the domain to a distribution that has high min-entropy. For example, for a parameter $\alpha \geq 1$, it suffices that for every $x \in D$ and $y \in R$ it holds that $\Pr_{\pi\in S_D}[\pi(x) = y] \leq \alpha/|D|$. In this case, we shall prove that $|D'| > |D|/2\alpha$, and a lower-bound of $\log_2(|D|/q) - \log_2(2\alpha)$ on the randomness-complexity follows. A different generalization is obtained by replacing σ with a set of values $S \subset R$ and referring to properties for which every function $f : D \to S$ is 2ϵ-far from the property.

2.2 Upper Bounds

We start with a totally generic bound, and later focus on testing graph properties.

2.2.1 A generic bound

Recall that we refer to properties of functions from D to R. The following result can be easily proved by extending a similar result regarding samplers (presented in [CEG]), which in turn is proved using well-known techniques.

[4]Note that q adaptive Boolean queries can always be replaced by 2^q non-adaptive Boolean queries. We warn that the more query-efficient transformation provided in [GT] is inapplicable here, because this transformation does not preserve the randomness-complexity.

Theorem 3 *If* Π *has an* ϵ*-tester that makes* q *queries then it has an* ϵ*-tester that makes* $O(q)$ *queries and tosses* $\log_2|D| + \log_2\log_2|R| + O(1)$ *coins. Furthermore, one-sided error and/or non-adaptivity are preserved.*

Proof: Let T be a tester as in the hypothesis, and suppose that it tosses r coins. Consider an 2^r-by-$|R|^{|D|}$ matrix in which the rows correspond to r-bit strings (representing possible outcomes of T's coin tosses) and the columns correspond to possible functions such that the entry (ω, f) equals the verdict of $T^f(\omega)$ (i.e., when T uses randomness ω and has oracle access to the function f). Note that the average values in any column that corresponds to a function in Π (resp., a function that is ϵ-far from Π) is at least $2/3$ (resp., at most $1/3$).

Using the probabilistic method, we will show that there exists a multi-set Ω of $O(|D|\log|R|)$ rows such that, for each column, the average of this column *taken only over the rows in* Ω is $1/15$-close to the average over the entire column. Using this set Ω, we consider the oracle machine that, when given access to any function f, selects uniformly $\omega \in \Omega$ and emulates $T^f(\omega)$. This machine accepts every $f \in \Pi$ with probability at least $(2/3) - (1/15) = 3/5$, rejects every f that is ϵ-far from Π with probability at least $3/5$, and its randomness complexity is $\log_2|\Omega| = \log_2|D| + \log_2\log_2|R| + O(1)$. Using randomness-efficient error-amplification (e.g., using the neighbors of a random vertex in an expander), we obtain the desired tester.

The probabilistic argument proceeds via a union bound over all possible $|R|^{|D|}$ functions. Fixing any function f, we consider the probability that, for a uniformly distributed multi-set Ω of size s, the following bad event occurs:

$$\left| 2^{-r} \cdot \sum_{\omega \in \{0,1\}^r} T^f(\omega) - s^{-1} \cdot \sum_{\omega \in \Omega} T^f(\omega) \right| > \frac{1}{15} \tag{1}$$

Using Chernoff Bound, the probability that the bad event in Eq. (1) holds is at most $\exp(-\Omega(s))$. Thus, for $s = O(|D|\log|R|)$, we conclude that there exists a multi-set of size s such that, for every f, the bad event in Eq. (1) does not hold. The theorem follows. ■

Corollary. Applying Theorem 3 to testers of graph properties in the adjacency matrix model (of [GGR]), we conclude that *if a property of* N*-vertex graphs is* ϵ*-testable using* q *queries then it has an* ϵ*-tester that makes* $O(q)$ *queries and tosses* $2\log_2 N + O(1)$ *coins.* We further discuss this model in Section 2.2.2.

2.2.2 Bounds for canonical testers of graph properties

The proof of Theorem 3 shows that for every tester T (of randomness complexity r) there exists a small set of coin-sequences Ω_T ($\subset \{0,1\}^r$) that is essentially as good as the original set of coin-sequences used by this tester (i.e., $\{0,1\}^r$). This raises the question of whether there may exists a universal set Ω that is good for all testers (of randomness complexity r). Needless to say, the latter formulation is too general and is doomed to yield a negative answer (e.g., by considering, for any Ω, a patalogical tester that behaves badly when fed with any sequence in Ω). Still such universal sets may exist for naturally restricted classes of testers.

One adequate class of testers was suggested in [GT], and it refers to testing graph properties in the adjacency matrix model. A canonical ϵ-tester for a property Π of N-vertex graphs is determined by an integer k and a property Π' of k-vertex graphs. Such a tester, sometimes referred to as k-canonical, selects uniformly a set of k vertices in the input graph G and accepts G if and only if the corresponding induced (k-vertex) subgraph has the property Π'. It was shown in [GT] that if Π is ϵ-testable with query complexity q then Π has a k-canonical ϵ-tester with $k = O(q)$. Thus, it is natural to consider the notion of a "universal set" of k-subsets of $[N]$ that is good for all k-canonical testers.

Definition 4 *A set* $\Omega \subseteq \{S \subset [N] : |S| = k\}$ *is called* (ϵ, k)-universal *if for every property* Π *of* N*-vertex graphs and for every* k*-canonical* ϵ*-tester for* Π*, denoted* T*, the following holds:*

1. *For every* G *that has property* Π*, it holds that* $\Pr_{\omega \in \Omega}[T^G(\omega) = 1] \geq 3/5$*, where* $T^G(\omega)$ *denotes the execution of* T *when given the coin-sequence* ω *and oracle access to* G*.*

2. *For every* G *that is* ϵ*-far from property* Π*, it holds that* $\Pr_{\omega \in \Omega}[T^G(\omega) = 1] \leq 2/5$*.*

Using an (ϵ, k)-universal set, we can reduce the randomness complexity of any k-canonical ϵ-tester T by selecting uniformly $\omega \in \Omega$ and emulating $T(\omega)$. The residual oracle machine, denoted T', is essentially an ϵ-tester for the same property, except that T' may err with probability at most $2/5$ (rather than $1/3$) Neeedless to say, T' has randomness complexity $\log_2 |\Omega|$ and query complexity $\binom{k}{2}$. Furthermore, T' preserves the possible one-sided error of T.

Needless to say, the set of all k-subsets is (ϵ, k)-universal, because using this set coincides with the definition of a k-canonical ϵ-tester. We seek (ϵ, k)-universal sets that are much smaller; specifically, by prior results we may hope to have (ϵ, k)-universal sets of size $O(N^2)$. By extending the proof of Theorem 3, we can prove the following result.

Theorem 5 *There exist* (ϵ, k)*-universal sets* (of subsets of $[N]$) *having size* $2^{k^2} \cdot N^2$.

The randomness complexity of the derived ϵ-tester is $k^2 + 2\log_2 N$. For relatively small k and in particular for k that only depends on ϵ (as in [GGR, AFKS, AFNS]), this is much smaller than the randomness complexity of the k-canonical ϵ-tester (i.e., $k \log_2 N$).

Proof: The key observation is that a k-canonical tester is determined by the property Π' that it decides (for the induced k-vertex subgraph), while Π' can be described by $K = 2^{\binom{k}{2}} < 2^{k^2/2}$ bits which determine for each k-vertex graph whether it is in Π'. Thus, when applying a union bound as in the proof of Theorem 3, the number of k-canonical testers that we need to consider is less than 2^K. This means that the set of random-sequences should be larger by a factor of K than in the proof of Theorem 3. Using $O(K) < 2^{k^2}$, the claim follows. ■

Open problems. Can the upper-bound of Theorem 5 be improve; in particular, do there exist (ϵ, k)-universal sets (of subsets of $[N]$) having size $O(\mathrm{poly}(k) \cdot N^2)$ or even $O(N^2)$? Can universal sets of small size (e.g., as in Theorem 5) be efficiently constructed?

3 Specific Algorithms: The Case of Bipartiteness

We consider two standard models for testing graph properties: the adjacency matrix model (introduced in [GGR]) and the bounded-degree model (introduced in [GR1]). We focus on the problem of testing bipartiteness in these models. Further details and additional testers are provided in [Shef]. We make extensive use of randomness-efficient hitters as defined and discussed in Appendix A.2.

3.1 In the Adjacency Matrix Model

In the adjacency matrix model an N-vertex graph $G = (V, E)$ is represented by the Boolean function $g : [N] \times [N] \to \{0, 1\}$ such that $g(u, v) = 1$ if and only if u and v are adjacent in G (i.e., $\{u, v\} \in E$). In this section we present a randomness-efficient Bipartite Tester for the adjacency matrix model. This tester is strongly influenced by the tester of [GGR], but differs from it in significant ways. Still, it is instructive to start with a description of the tester of [GGR].

3.1.1 The tester of [GGR]

Essentially, the bipartite tester of [GGR] selects a random set of $\tilde{O}(\epsilon^{-2})$ vertices, inspects the subgraph of G induces by this set, and accepts if and only if this induced subgraph is bipartite. The analysis in [GGR] actually refers to the following description, which also has a lower query-complexity.

Algorithm 6 *On input parameters N and ϵ, and oracle access to an adjacency predicate of an N-vertex graph, $G = (V, E)$, proceed as follows:*

1. *Uniformly select a sample U of $\tilde{O}(\epsilon^{-1})$ vertices.*
2. *Uniformly select a sample S of $\tilde{O}(\epsilon^{-2})$ vertex-pairs.*
3. *For each $u \in U$ and $(v_1, v_2) \in S$, check whether $\{u, v_1\}, \{u, v_2\}$ and $\{v_1, v_2\}$ are edges.*
4. *Accept if and only if the subgraph viewed in Step 3 is bipartite.*

Clearly, this algorithm never rejects a bipartite graph, and thus its analysis focuses on the case that G is ϵ-far from being bipartite. One key observation is that each 2-partition, (U_1, U_2), of U induces a 2-partition of the entire graph in which all neighbors of U_1 are on one side and all the other vertices are on the other side. A pair of vertices (v_1, v_2) detects that the latter partition is not a valid 2-coloring of G if there exists $u_1, u_2 \in U_1$ (resp., $u_1, u_2 \in U_2$) such that $\{u_1, v_1\}, \{v_1, v_2\}$ and $\{v_2, u_2\}$ are all edges of G. In such a case, we call the pair (v_1, v_2) a witness against (U_1, U_2). The analysis in [GGR] shows that if G is ϵ-far from being bipartite then, with high probability, for every 2-partition of U there exists a pair in S that is a witness against this 2-partition. Let us briefly recall how this is done.

The first step is proving that, with high probability (say, with probability at least 5/6), the set U dominates[5] all but an $\epsilon/8$ fraction of the vertices of G that have degree at least $\epsilon N/8$. This step is quite straightforward. The next step is proving that this implies that *for every 2-partition of U there exists at least $\epsilon N^2/2$ (ordered) vertex-pairs that are each a witness against this 2-partition.* The implication is proved by confronting the following two facts:

1. Since G is ϵ-far from being bipartite, the 2-partition of V induced by any 2-partition of U has at least ϵN^2 (ordered) vertex-pairs that reside on the same side of the partition and yet are connected by an edge.
2. The number of (ordered) vertex-pairs (v_1, v_2) such that $\{v_1, v_2\} \in E$ but either v_1 or v_2 is not dominated by U is at most $\epsilon N^2/2$, because each low-degree vertex contributes at most $\epsilon N/4$ such (ordered) pairs and there are at most $\epsilon N/8$ high-degree vertices that are not dominated by U.

[5] We say that a set U dominates a vertex v in the graph G if v is adjacent to some vertex in U.

Having established the existence of at least $\epsilon N^2/2$ vertex-pairs that constitute a witness against any fixed 2-partition of U, it is clear that each random pair of vertices will be a witness with probability at least $\epsilon/2$, and selecting enough random pairs will do the job. The point, however, is that we need to rule out each of the $2^{|U|}$ possible 2-partitions of U. Thus, the number of selected pairs is set such that the probability that we do not find a witness against any specific 2-partition is smaller than $2^{-|U|}$. Indeed, setting $|S| = O(|U|/\epsilon)$ will do. This completes our review of [GGR].

As stated in Section 1.3, the problem with the foregoing approach is that it is impossible to implement it using randomness-complexity below $|U|$, which in turn is $\Omega(\epsilon^{-1})$. However, our aim is to obtain randomness-complexity that is linearly related to $O(\log(1/\epsilon))$.

3.1.2 A warm-up: randomness-efficient tester of query complexity $\tilde{O}(\epsilon^{-4})$

A closer look at the foregoing argument reveals that a pair (v_1, v_2) such that $\{u_1, v_1\}, \{v_1, v_2\}$ and $\{v_2, u_2\}$ are all edges of G is not merely a witness against a *specific* 2-partition of U that places u_1 and u_2 on the same side. It is actually a witness against *any* 2-partition of U that places u_1 and u_2 on the same side. Viewed from a different perspective, such a pair (v_1, v_2) imposes a constraint on the "relevant" 2-partition of U; *the constraint being that u_1 and u_2 should not be placed on the same side*. It will be useful to consider the graph of these constraints, which has the vertex-set U and edges between each pair of vertices to which such a constraint is applied (i.e., there is an edge between u_1 and u_2 if there exists a pair $(v_1, v_2) \in V \times V$ that imposes a constraint on the pair (u_1, u_2)). Indeed, the 2-partitions of U that satisfy the set of these constraints are exactly the 2-colorings of this auxiliary graph.

The foregoing perspective suggests that it may be useful to try to accumulate constraints. At the very extreme, the graph of constraints will not be bipartite, which definitely allows us to reject (because it indicates that there are witnesses against each 2-partition of U). Discarding this case, we consider another extreme case in which the graph of constraints is connected, leaving us with a single allowed 2-partition of U (i.e., a single 2-coloring of the constraint graph), which can be checked as in Algorithm 6. The point, however, is that in this case it will suffice to set $|S| = O(\epsilon^{-1})$ and more importantly to have a sample that rules out the remaining partition with constant probability (rather than with probability $2^{-|U|}$). This opens the door to a randomness-efficient implementation.

But what if the graph of constraints that we found is not connected? Unless this event is due to sheer lack of luck, it indicates that there are few pairs in $V \times V$ that impose constraints regarding vertex-pairs in $U \times U$ that are in different connected components of the constraint graph. This implies that, for every 2-partition of U that is consistent with the constraint graph (i.e., every 2-coloring of this graph), there are many pairs in $V \times V$ that constitute a witness against the 2-partition of some of the connected components. That is, each such pair imposes a constraint that refers to vertices that reside in the same connected component, and furthermore this constraint contradicts the constraints that are already present regarding this connected component.

Needless to say, for the foregoing to work, we should determine adequate thresholds for the notion of "*few* pairs in $V \times V$ that impose a constraint regarding vertex-pairs" (in $U \times U$). Let us start by spelling out the notion of imposing (or rather forcing) a constraint. We say that the pair $(v_1, v_2) \in V \times V$ constrains the pair $(u_1, u_2) \in U \times U$ if $\{u_1, v_1\}, \{v_1, v_2\}$ and $\{v_2, u_2\}$ are all edges of G. Next, we say that a pair $(u_1, u_2) \in U \times U$ is ρ-constrained if there are at least $\rho \cdot N^2$ vertex-pairs in $V \times V$ that constrain (u_1, u_2). Leaving ρ unspecified for a moment, we make the following observations:

1. Using a sample of $O(\rho^{-1} \cdot \log |U|)$ vertex-pairs in $V \times V$, with high probability, it holds that *for every ρ-constrained pair $(u_1, u_2) \in U \times U$, the sample contains a pair that constrains (u_1, u_2)*.

This holds even if the sample is generated using a randomness-efficient hitter (which hits any set of density ρ with probability at least $1-(|U|^{-2}/10)$, using randomness-complexity $O(\log|V|+\log|U|) = O(\log|V|))$. The point is that there are at most $|U|^2$ relevant pairs (i.e., pairs that are ρ-constrained), and we may apply a Union Bound as long as we fail on each such pair with probability at most $|U|^{-2}/10$ (or so).

2. Consider the graph $G_{U,\rho}$ consisting of the vertex-set U and edges corresponding to the ρ-constrained pairs of vertices. Then, the number of vertex-pairs in $V\times V$ that constrain some pair of vertices (in U) that does not belong to the same connected component of $G_{U,\rho}$ is at most $|U|^2\cdot\rho N^2$.

   Recall that if G is ϵ-far from bipartite and U is good (i.e., U dominates almost all high-degree vertices) then, for every 2-partition of U, there are at least $\epsilon N^2/2$ pairs that constrain some pair of vertices that are on the same side of this 2-partition. It follows that at least $((\epsilon/2)-|U|^2\rho)\cdot N^2$ of these pairs constrain pairs that are in the same connected component of $G_{U,\rho}$. Setting $\rho=\epsilon/(4|U|^2)$, we need to hit a set of density $\epsilon/4$, which is easy to do using a randomness-efficient hitter.

This analysis lead to an algorithm that resembles Algorithm 6, except that it uses a secondary sample S that has different features than in the original version. In Algorithm 6 the set S had to hit any fix set of density $\epsilon/2$ with probability at least $1-2^{-|U|}$. Here the set S needs to hit any fix set of density $\rho=\epsilon/(4|U|^2)<\epsilon^{-3}$ with probability at least $1-(|U|^{-2}/10)$. Thus, while in Algorithm 6 we used $|S|=O(|U|/\epsilon)$ but generating the set S required at least $|U|$ random bits, here $|S|=O(|U|^2/\epsilon)=\tilde{O}(\epsilon^{-3})$ but generating the set S can be done using $O(\log N)$ random bits. (The set U is generated with the same aim as in Algorithm 6; that is, hitting a set of density ϵ with probability at least $1-\epsilon^{-1}$. Such a set can be generated using $O(\log N)$ random bits.)

Thus, we obtain a (computational efficient) ϵ-tester with randomness-complexity $O(\log N)$ and query-complexity $O(|U|\cdot|S|)=\tilde{O}(\epsilon^{-4})$. Our aim in the next section is to reduce the query-complexity to $\tilde{O}(\epsilon^{-3})$ while essentially maintaining the randomness-complexity.

3.1.3 The actual algorithm: randomness-efficient tester of query complexity $\tilde{O}(\epsilon^{-3})$

The query-complexity bottleneck in Section 3.1.2 is due to the size of S, which in turn needs to hit sets of density $\rho=O(\epsilon^3)$. Our improvement will follow by using a larger value of the threshold ρ (essentially $\rho=O(\epsilon^2)$). Recall that in Section 3.1.2 we used $\rho=O(\epsilon^3)$ in order to bound the total number of pairs that constrain pairs that are not ρ-constrained. Thus, using $\rho=O(\epsilon^3)$ seems inherent to an analysis that refers to each pair separately, and indeed we shall deviate from that paradigm in this section.

The planned deviation is quite natural. After all, we not not care about having specific edges in our constraint graph, but rather care about the connected components of that graph. For example, looking at any vertex $u\in U$, any pair in $V\times V$ that constrains any pair (u,u'), where $u'\in U\setminus\{u\}$, increases the connected component in which u resides. That is, let $\gamma(u_1,u_2)$ denote the fraction of vertex-pairs in $V\times V$ that constrain (u_1,u_2), and recall that a pair (u_1,u_2) was called ρ-constrained if $\gamma(u_1,u_2)\geq\rho$. Thus, we (tentatively) say that $u\in U$ is ρ-constrained if $\sum_{u'\in U\setminus\{u\}}\gamma(u,u')\geq\rho$. Let us now see what happens.

1. Using a sample of $O(\rho^{-1}\cdot\log|U|)$ vertex-pairs in $V\times V$, with high probability, it holds that for every ρ-constrained vertex $u\in U$, the sample contains a pair that constrains (u,u'), for some $u'\in U\setminus\{u\}$. Again, this holds even if the sample is generated using a randomness-efficient hitter.

2. The number of vertex-pairs in $V \times V$ that constrain some pair of vertices $(u_1, u_2) \in U \times U$ such that either u_1 or u_2 is not ρ-constrained is at most $2|U| \cdot \rho N^2$. This means that we can ignore such vertex-pairs (in $V \times V$) even when setting $\rho = O(\epsilon/|U|)$ or so.

Thus, taking a sample S' as in Item 1, will result in having a constraint graph $G_{U,S'}$ in which each ρ-constrained vertex resides in non-singleton connected components. In particular, the number of non-singleton connected components is at most $|U|/2$.

Note, however, that unlike in Section 3.1.2, the foregoing facts do not yield an upper-bound on the number of vertex-pairs in $V \times V$ that constrain some pair of vertices (in U) that does not belong to the same connected component of $G_{U,S'}$. Loosely speaking, we shall iterate the same process on the non-singleton connected components of $G_{U,S'}$, while recalling that the only vertices that form singleton connected components in $G_{U,S'}$ are not ρ-constrained (and thus can be ignored). This suggests an iterative process, which will halt after at most $\log_2 |U|$ iterations in a situation analogous to having no ρ-constrained vertices. At this point we may proceed with a final sample of pairs that, with high probability, will yield a constraint that conflicts with the existing ones.

Clarifying the foregoing iterative process requires generalizing the notion of ρ-constrained vertices such that it will apply to the connected components determined in the previous iteration. Consider a partition of U, denoted $\overline{U} = (U^{(0)}, U^{(1)}, ..., U^{(k)})$, where $U^{(0)}$ may be empty and k may equal 0, but for every $i \in [k]$ it holds that $U^{(i)} \neq \emptyset$. In the first iteration, we use $\overline{U} = (\emptyset, \{u_1\}, ..., \{u_t\})$, where $U = \{u_1, ..., u_t\}$. In later iterations, $U^{(1)}, ..., U^{(k)}$ will correspond to connected components of the current constraint graph and $U^{(0)}$ will contain vertices that were cast aside at some point.

Definition 7 (being constrained w.r.t a partition): *For $i \in \{0, 1, ..., k\}$, we say that $u \in U^{(i)}$ is ρ-*constrained w.r.t *$\overline{U}$ if $\sum_{u' \in U'} \gamma(u, u') \geq \rho$, where $U' = \cup_{j \in [k] \setminus \{i\}} U^{(j)}$. Recall that $\gamma(u_1, u_2)$ denote the fraction of vertex-pairs in $V \times V$ that constrain (u_1, u_2), where the pair $(v_1, v_2) \in V \times V$* constrains *the pair $(u_1, u_2) \in U \times U$ if $\{u_1, v_1\}, \{v_1, v_2\}$ and $\{v_2, u_2\}$ are all edges of G.*

We stress that the foregoing sum does not include vertices in either $U^{(0)}$ or $U^{(i)}$. Our analysis will refer to the following algorithm, which *can be implemented within randomness-complexity $O(\log(1/\epsilon)) \cdot \log_2 N$ and query-complexity $\tilde{O}(\epsilon^{-3})$.*

Algorithm 8 (The Bipartite Tester, revised):

1. *Select a sample U of $\tilde{O}(\epsilon^{-1})$ vertices by using a hitter that hits any set of density $\epsilon/8$ with probability at least $1 - (\epsilon/100)$.*

2. *For $i = 1, ..., \ell + 1$, where $\ell = \log_2 |U|$, select a sample S_i of $\tilde{O}(\epsilon^{-2})$ vertex-pairs by using a hitter that hits any set of density $\rho = \epsilon/\tilde{O}(|U|)$ with probability at least $1 - \tilde{O}(|U|)^{-1}$.* (This hitter has randomness-complexity $O(\log N + \log |U|) = O(\log N)$.) *Let $S = \cup_{i=1}^{\ell+1} S_i$.*

3. *For each $u \in U$ and $(v_1, v_2) \in S$, check whether $\{u, v_1\}, \{u, v_2\}$ and $\{v_1, v_2\}$ are edges.*

4. *Accept if and only if the subgraph viewed in Step 3 is bipartite.*

Needless to say, the peculiar way in which S is selected is aimed to support the analysis.

Lemma 9 *If G is ϵ-far from being bipartite then Algorithm 8 rejects with probability at least 2/3.*

Proof: We may assume that U is good in the sense that it dominates all but $\epsilon N/8$ of the vertices that have degree at least $\epsilon N/8$. As argued above (and shown in [GGR]), there are at most $\epsilon N^2/2$ vertex pairs that have an endpoint that is not dominated by $U = \{u_1, ..., u_t\}$. Starting with $\overline{U} = (\emptyset, \{u_1\}, ..., \{u_t\})$, we shall proceed in iterations proving that in each iteration one of the following two events occur:

1. There are $\Omega(\epsilon N^2)$ vertex pairs that form constraints that contradicts the existing constraints. In this case, with very high probability, the algorithm will select such a pair and will reject (because the subgraph that it sees is not 2-colorable).
2. There exist ρ-constrained vertices with respect to the current partition $\overline{U} = (U^{(0)}, U^{(1)}, ..., U^{(k)})$, where $U^{(1)}, ..., U^{(k)}$ are connected components of the current constraint graph and $U^{(0)}$ contains vertices that were cast aside in previous iterations. We shall also show that ρ-constrained (w.r.t $\overline{U}$) vertices cannot be in $U^{(0)}$. In this case, with very high probability, the algorithm will find new constraints and in particular it will find such a constraint between every ρ-constrained (w.r.t $\overline{U}$) vertex and some vertex that is in one of the other k connected components.

We shall shortly take a closer look at what happens in the second case (i.e., Case 2) and prove that indeed at least one of the foregoing cases must hold. But before doing so, we note that the second case (i.e., Case 2) becomes impossible once we reach a situation in which $k = 1$, at which point the algorithm must reject due to the first case (i.e., Case 1).

Let us first take a closer look at what happens in Case 2. Suppose that $u \in U^{(i)}$ is ρ-constrained w.r.t the current $\overline{U}$. Then by the foregoing, due to a newly found constraint, vertex u gets connected to some vertex in $\cup_{j\in[k]\setminus\{i\}} U^{(j)}$. This means that each $U^{(i)}$ ($i \neq 0$) that contains some ρ-constrained vertex gets merged to some $U^{(j)}$ ($j \neq 0$ and $j \neq i$). We will not add any constraint that refers to vertices that were cast aside (i.e., those in $U^{(0)}$). Thus, vertices that were cast aside in the past (since they were not ρ-constrained w.r.t a previous partition) will remain in $U^{(0)}$, and indeed they are also not ρ-constrained w.r.t any later partition.[6] For $i \neq 0$, if $U^{(i)}$ was not merged with any other $U^{(j)}$ ($j \neq 0$ and $j \neq i$) then it contains no ρ-constrained vertex, and we cast it aside (i.e., move it to the new $U^{(0)}$). Thus, in each iteration, the number of connected components not cast aside (i.e., k) shrinks by a factor of at least two.

We now prove that at least one of the two aforementioned conditions must hold. Looking at the current partition $\overline{U}$, we first note that if one of the connected components (including those contained in $U^{(0)}$) is not bipartite then we already have a set of constraints that is self-contradictory (i.e., does not allow a 2-coloring of the subgraph we have seen so far). This situation is a special case of Case 1, and indeed in this sub-case the algorithm rejects. Disposing of this sub-case, we now consider an arbitrary 2-coloring of the constraint graph, and the 2-partition that it induces on the rest of G (i.e., we put on the first side all the vertices that are *dominated by some vertex of U that was colored by the second color*). Then, there are at least ϵN^2 vertex-pairs that are adjacent and were put on the same side, and at least $\epsilon N^2/2$ of these vertex-pairs have both its vertices dominated by U. Each such (v_1, v_2) is of one of the following two types.

(i) The vertex-pair (v_1, v_2) constrains a pair of vertices (u_1, u_2) where both vertices are in the same connected component of the constraint graph. As showed next, such a pair imposes a constraint that contradicts the constraints of the current graph. Thus, this pair contributes to the pairs counted in Case 1.

[6] In a later partition, some components get merged and some move to $U^{(0)}$. This can only decrease the count towards being ρ-constrained.

To see that the said constraint contradicts the constraints of the current graph, recall that since (v_1, v_2) constrains the pair $(u_1, u_2) \in U \times U$ it holds that the edges $\{u_1, v_1\}$, $\{v_1, v_2\}$, and $\{v_2, u_2\}$ form an odd-length path between u_1 and u_2. On the other hand, v_1 and v_2 were placed on the same side of the 2-partition of V, which implies that u_1 and u_2 were assigned the same color by a 2-coloring of the current constraint graph. Since u_1 and u_2 are in the same connected component of that graph, it follows that they are connected by an even-length path (which reflects an even-length path in G). Thus, the new set of constraints form an odd-length cycle.

(ii) The vertex-pair (v_1, v_2) constrains a pair of vertices (u_1, u_2) that belong to different connected component of the constraint graph. As showed next, the existence of more than $\epsilon N^2/4$ such pairs implies Case 2 (i.e., the existence of ρ-constrained vertices, which in particular are not in $U^{(0)}$).

We first recall that a vertex in $U^{(0)}$ can not be ρ-constrained with respect to the current partition, because it is not ρ-constrained with respect to some previous partition and because the previous partition allows more pairs to be counted.

As for the main claim, note that each pair of the current type is counted towards determining whether u_1 (resp., u_2) is ρ-constrained with respect to the current partition. The total "pair count" of vertices that are not ρ-constrained is smaller than ρN^2, which implies that Thus, for $\rho = \epsilon/(4|U|)$, there are less than $|U| \cdot \rho N^2 = \epsilon N^2/4$ pairs of the current type that refer to vertices that are not ρ-constrained. It follows if there are more than $\epsilon N^2/4$ pairs of the current type, then ρ-constrained vertices must exist, which imply that Case 2 holds.

We conclude that either there are more than $\epsilon N^2/4$ vertices of type (ii), which imply that Case 2 holds, or there are more than $\epsilon N^2/4$ vertices of type (i), which imply that Case 1 holds.

Recall that if Case 2 holds then the number of non-discarded connected components (i.e., k) shrinks by a factor of at least 2. Thus, after $\log_2 |U|$ iterations, the current partition must satisfy $k \leq 1$, and thus Case 2 cannot hold in the next iteration. The lemma follows. ■

Open problem. Needless to say, we are aware of the Bipartite Tester of [AK], which has better query-complexity than the tester of [GGR] (as well as ours). Specifically, the query-complexity of the tester of [AK] is $\tilde{O}(\epsilon^{-2})$ rather than $\tilde{O}(\epsilon^{-3})$. Theorem 3 implies that the tester of [AK] has a randomness-efficient implementation, but it does not provide an explicit one. We conjecture that there exists a randomness-efficient bipartite tester that has query-complexity $\tilde{O}(\epsilon^{-2})$ and time-complexity $\text{poly}(\epsilon^{-1} \log N)$.

3.2 In the Bounded-Degree Model

The bounded-degree model refers to a fixed degree bound, denoted d. An N-vertex graph $G = (V, E)$ (of maximum degree d) is represented in this model by a function $g : [N] \times [d] \to \{0, 1, ..., N\}$ such that $g(v, i) = u \in [N]$ if u is the i^{th} neighbor of v and $g(v, i) = 0$ if v has less than i neighbors. In this section we provide a randomness-efficient implementation of the Bipartite Tester of [GR2], which refers to the bounded-degree model. Thus, we start with a description of that tester.

Algorithm 10 (The Bipartite Tester of [GR2]): *On input parameters N, d, ϵ, and oracle access to an incidence function for an N-vertex graph, $G = (V, E)$, of degree bound d, repeat $T \stackrel{\text{def}}{=} \Theta(\frac{1}{\epsilon})$ times:*

1. *Uniformly select a* ("start") *vertex s in V.*

2. (Try to find an odd-length cycle through vertex s):

   (a) *Perform* $K \stackrel{\text{def}}{=} \text{poly}((\log N)/\epsilon) \cdot \sqrt{N}$ *random walks starting from vertex s, where each walk is of length* $L \stackrel{\text{def}}{=} \text{poly}((\log N)/\epsilon)$.

   (b) *Let* R_0 (respectively, R_1) *denote the set of vertices that were reached from vertex s in an even* (respectively, odd) *number of steps in any of these walks.*

   (c) *If* $R_0 \cap R_1$ *is not empty then* reject.

If the algorithm did not reject in any one of the above T iterations, then it accepts.

Clearly, this algorithm never rejects a bipartite graph. Indeed, the analysis of [GR2] focuses on the case that the graph G is ϵ-far from bipartite, and shows that the algorithm will reject G with high probability. The rather involved analysis breaks down to two complimentary facts that refer to a notion of a good start vertex. Loosely speaking, a start vertex is called good if, when the tester selects it in Step 1, the probability that the tester finds an odd-length cycle in Step 2 is somewhat small (say, below 1/10). We note that the actual definition of a good vertex refers to the probability of finding an odd-length cycle *when taking two independent random walks from this vertex.*

Most of [GR2] is devoted to establishing the fact that if G is ϵ-far from bipartite then an $\Omega(\epsilon)$ fraction of the vertices are not good. It is crucial for us that this technically involved analysis does not refer at all to the algorithm; it rather refers to the definition of a good vertex, which (as stressed above) refers to a mental experiment in which one takes two independent random walks from this vertex. Thus, this analysis remains intact regardless of how we chose to implement Algorithm 10.

The complimentary fact regarding good vertices is that when the tester selects a vertex that is not good (in Step 1), the probability that it finds an odd-length cycle in Step 2 is not too small (say, at least 1/10). Indeed, this fact refers to Algorithm 10 itself, but its rather simple proof (provided in [GR2]) only presumes that the K random walks are distributed in a 4-wise independent manner. Specifically, the analysis defines a random variable for each pair of walks such that this random variable represents the event of finding an odd-length cycle via the corresponding two walks. Then, Chebyshev's Inequality is applied while relying on the expectation and variance of the sum of these random variables. As one may guess, the said expectation and variance are computed by only relying on the expectation of the individual random variables and the co-variances of all possible pairs of random variables. Thus, the analysis remains valid as long as the said expectation and co-variance maintain their value, which is definitely the case provided that each pair of random variables maintains its behavior. Noting that each pair of random variables refers to at most four different random walks, we establish our claim that the analysis of [GR2] only presumes that the K random walks are distributed in a 4-wise independent manner.

The foregoing discussion suggests the following implementation of Algorithm 10. For Step 1 use a randomness-efficient hitter that hits any set of density $\Omega(\epsilon)$ with constant probability. More importantly, for Step 2 use a randomness-efficient construction of K four-wise independent random strings, each specifying a random walk of length L (i.e., each being a string of length $L \log_2 d$). By the foregoing discussion, this implementation preserves the performance guarantees of Algorithm 10; that is, this implementation is also an ϵ-test for bipartiteness. The crucial point, however, is that Step 2 is now implemented using $4 \cdot L \log_2 d$ (rather than $K \cdot L \log_2 d$) random coins. Thus, we obtain:

Theorem 11 *There exists a Bipartite tester* (in the incidence function model) *of time-complexity* $\text{poly}((\log N)/\epsilon) \cdot \sqrt{N}$ *and randomness-complexity* $\text{poly}((\log N)/\epsilon)$. *Furthermore, as Algorithm 10, this tester always accepts a Bipartite graph, and in case of rejection it provides a witness of length* $\text{poly}((\log N)/\epsilon)$ (that the graph is not bipartite).

References

[ADPR] N. Alon, S. Dar, M. Parnas, and D. Ron. Testing of Clustering. *SIAM Journal on Discrete Mathematics*, Vol. 16 (3), pages 393–417, 2003.

[AFKS] N. Alon, E. Fischer, M. Krivelevich, and M. Szegedy. Efficient testing of large graphs. In *Proc. of 40th IEEE Symposium on Foundations of Computer Science*, pages 645–655, 1999.

[AFNS] N. Alon, E. Fischer, I. Newman, and A. Shapira. A Combinatorial Characterization of the Testable Graph Properties: It's All About Regularity. In *38th ACM Symposium on the Theory of Computing*, pages 251–260, 2006.

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

[BSVW] E. Ben-Sasson, M. Sudan, S. Vadhan, and A. Wigderson. Randomness-efficient low degree tests and short PCPs via epsilon-biased sets. In *Proc. 35th ACM Symposium on the Theory of Computing*, June 2003, pp. 612–621.

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

[CEG] R. Canetti, G. Even and O. Goldreich. Lower Bounds for Sampling Algorithms for Estimating the Average. *Information Processing Letters*, Vol. 53, pages 17–25, 1995.

[F] E. Fischer. The art of uninformed decisions: A primer to property testing. *Bulletin of the European Association for Theoretical Computer Science*, Vol. 75, pages 97–126, 2001.

[G] O. Goldreich. Another motivation for reducing the randomness complexity of algorithms. Position paper, *ECCC*, 2006.

[GGR] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, pages 653–750, July 1998.

[GGLRS] O. Goldreich, S. Goldwasser, E. Lehman, D. Ron, and A. Samorodnitsky. Testing Monotinicity. *Combinatorica*, Vol. 20 (3), pages 301–337, 2000.

[GR1] O. Goldreich and D. Ron. Property testing in bounded degree graphs. *Algorithmica*, pages 302–343, 2002.

[GR2] O. Goldreich and D. Ron. A sublinear bipartite tester for bounded degree graphs. *Combinatorica*, Vol. 19 (3), pages 335–373, 1999.

[GS] O. Goldreich and M. Sudan. Locally testable codes and PCPs of almost linear length. In *Proc.* 43*rd IEEE Symposium on Foundations of Computer Science*, Nov. 2002, pp. 13–22. (See ECCC Report TR02-050, 2002).

[GT] O. Goldreich and L. Trevisan. Three theorems regarding testing graph properties. *Random Structures and Algorithms*, Vol. 23 (1), pages 23–57, August 2003.

[LY] L. Lovász and N. Young. Lecture Notes on Evasiveness of Graph Properties. Technical Report TR-317-91, Princeton University, Computer Science Department, 1991.

[R] D. Ron. Property testing. In *Handbook on Randomization, Volume II*, pages 597–649, 2001. (Editors: S. Rajasekaran, P.M. Pardalos, J.H. Reif and J.D.P. Rolim.)

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

[Shal] R. Shaltiel. Recent Developments in Explicit Constructions of Extractors. *Bulletin of the EATCS 77*, pages 67–95, 2002.

[SU] R. Shaltiel and C. Umans. Simple Extractors for All Min-Entropies and a New Pseudo-Random Generator. In *32nd IEEE Symposium on Foundations of Computer Science*, pages 648–657, 2001.

[Shef] O. Sheffet. Reducing the Randomness Complexity of Property Testing, with an Emphasis on Testing Bipartiteness. M.Sc. Thesis, Weizmann Institute of Science, December 2006. Available from `http://www.wisdom.weizmann.ac.il/~oded/msc-os.html`

[SW] A. Shpilka and A. Wigderson. Derandomizing Homomorphism Testing in General Groups SIAM Journal on Computing, Vol. 36-4, pages 1215–1230, 2006.

[TZS] A. Ta-Shma, D. Zuckerman, and S. Safra. Extractors from Reed-Muller Codes. In *32nd IEEE Symposium on Foundations of Computer Science*, pages 638–647, 2001.

Appendix: Preliminaries

A.1 On Using Weak Random Sources

A.1.1 Analysis of the standard use of extractors

The following claim is meaningful only if $p + \epsilon < 1/4$ (e.g., $p, \epsilon < 1/8$).

Claim 12 *Let A be a randomized decision procedure of randomness-complexity r and error probability p, and $E : \{0,1\}^s \times \{0,1\}^n \to \{0,1\}^r$ be an (k, ϵ)-extractor. Consider the algorithm A' that, on input α, obtains a single sample x from an (n, k)-source and rules according to the majority value in $\langle A(\alpha, E(i, x))\rangle_{i=1,\ldots,2^s}$. Then, A' has error probability at most $2(p + \epsilon)$.*

Proof: The analysis of the foregoing implementation is based on the fact that "on the average" the 2^s strings extracted from the source approximate a perfect r-bit long source (i.e., a random setting of the s-bit seed yields an almost uniformly distributed r-bit string). Specifically, by definition, if X is a (n, k)-source then $E(U_s, X)$ is ϵ-close to U_r. It follows that the probability that $A(\alpha, E(U_s, X))$ errs is at most $p + \epsilon$. By Markov Inequality, the probability that the majority of the values in $\langle A(\alpha, E(i, X))\rangle_{i=1,\ldots,2^s}$ are wrong is at most $2(p + \epsilon)$. The claim follows. ■

Comment. We note that randomized procedures with one-sided error probability p can be implemented using a weak random source as long as $p + \epsilon < 1$. An important case is of search problems for which the randomized algorithm finds a correct solution with probability $1 - p$ and halts without solution otherwise. When implementing such an algorithm, we may output any solution obtained in any of the invocations of the original algorithm, which means that we "rule by or" rather than "ruling by majority".

A.1.2 On two main types of weak sources

We stress that the two types of (n, k)-sources that were mentioned in Section 1.2 (i.e., $k = \Omega(n)$ and $k = n^{\Omega(1)}$) are the most natural types of weak sources and indeed most research on extractors has focused on them. Let us take a closer look at these two cases. Recall that r denotes the number of bits that we need to extract from such a source (in order to feed our algorithm, denoted A). Furthermore, it suffices to set the deviation parameter of the extractor (i.e., ϵ) to a small constant (e.g., $\epsilon = 1/10$ will do). The two cases we consider are:

1. *Linearly related n, k and r*: that is, for some constants $c > c' > 1$, it holds that $n = c \cdot r$ and $k = c' \cdot r$. In other words, *we refer to sources having a constant rate of min-entropy.*

   In this case, efficient randomness extractors that use $s = \log n + O(\log\log n) = \log_2 \tilde{O}(n)$ are known (cf. [TZS, Shal]). Using these extractors, we obtain an implementation of A (using such weak sources) with overhead factor $\tilde{O}(r)$. Thus, *a tester of query-complexity q and randomness-complexity r will be implemented using a number of queries that is $\tilde{O}(r) \cdot q$.*

2. *Polynomially related n, k and r*: that is, for some $c > c' > 1$, it holds that $n = r^c$ and $k = r^{c'}$. In other words, we refer to a source having min-entropy that is polynomially related to its length.

   In this case, efficient randomness extractors that use $s = \log \tilde{O}(n) = c \log_2 \tilde{O}(r)$ are known (cf. [SU, Shal]). Using these extractors, we obtain an implementation of A (using such weak sources) with overhead factor $\tilde{O}(r^c)$.

In both cases, the overhead factor is approximately linear in the length of the source's outcome (which, in turn, is linearly or polynomially related to r).

A.2 Randomness-Efficient Hitters

The hitting problem is a one-sided version of the Boolean sampling problem. Given parameters n (length), ϵ (density) and δ (error), and oracle access to any function $f:\{0,1\}^n \to \{0,1\}$ such that $|\{x : f(x)=1\}| \geq \epsilon 2^n$, the task is to find a string that is mapped to 1.

Definition 13 (hitter): *A* hitter *is a randomized algorithm that on input parameters n, ϵ and δ, and oracle access to* any *function $f:\{0,1\}^n \to \{0,1\}$ such that $|f^{-1}(1)| \geq \epsilon 2^n$, satisfies*

$$\Pr[\text{hitter}^f(n,\epsilon,\delta) \in f^{-1}(1)] \;>\; 1-\delta$$

For any constant $\delta > 0$, using a pairwise-independent sequence of length $O(1/\epsilon)$, we obtain a hitter for sets of density ϵ and error probability δ. Thus, this hitter has query-complexity $O(1/\epsilon)$ and randomness-complexity $2n$. An alternative hitter based on the neighborhood of a random vertex in an expander graph has query-complexity $O(1/\epsilon)$ and randomness-complexity n. Combining any of these hitters with a random walk (of length $O(\log(1/\delta))$) on an expander graph, we obtain a hitter for sets of density ϵ and any desired error probability δ such that this hitter has query-complexity $O(\epsilon^{-1}\log(1/\delta))$ and randomness-complexity $r + O(\log(1/\delta))$, where $r \in \{n, 2n\}$ depending on the basic hitter we use.

ECCC ISSN 1433-8092
http://eccc.hpi-web.de/