Foundations and Trends® in
Theoretical Computer Science
Vol. 5, No. 2 (2009) 73–205

DOI: 10.1561/0400000029

now
the essence of knowledge

Algorithmic and Analysis Techniques in Property Testing

By Dana Ron

Contents

1 Introduction **75**

1.1 Settings in Which Property Testing is Beneficial 76
1.2 A Brief Overview 78
1.3 Property Testing and "Classical" Approximation 80
1.4 Property Testing and Learning 81
1.5 Organization of this Survey 82
1.6 Related Surveys 83

2 Preliminaries **84**

2.1 Basic Definitions and Notations 84
2.2 Testing Graph Properties 86

3 The Self-correcting Approach **89**

3.1 Linearity 90
3.2 Low-Degree Polynomials 96
3.3 Implications of Self-correction 99

4 The Enforce-and-Test Approach **109**

4.1 Testing Whether a Graph is a Biclique 109
4.2 Testing Bipartiteness in the Dense-Graphs Model 111
4.3 Other Applications of the Enforce-and-Test Approach 119

5 Testing by Implicit Learning **121**

5.1 A Building Block: Testing Juntas 122
5.2 The Algorithm for Testing by Implicit Learning 129

6 The Regularity Lemma **135**

6.1 Background 135
6.2 Statement of the Lemma 136
6.3 Testing Triangle-Freeness 137

7 Local-Search Algorithms **141**

7.1 Connectivity 142
7.2 k-Edge Connectivity 146
7.3 k-Vertex Connectivity. 150
7.4 Minor-Closed Properties 150
7.5 Other Local-Search Algorithms 152

8 Random Walks Algorithms **154**

8.1 Testing Bipartiteness in Bounded-Degree Graphs 154
8.2 Testing Expansion 159

9 Lower Bounds **161**

9.1 A Lower Bound for Testing Triangle-Freeness 161
9.2 A Lower Bound for Testing Bipartiteness of Constant Degree Graphs 166

10 Other Results **170**

10.1 Testing Monotonicity 170
10.2 Testing in the General-Graphs Model 178
10.3 Testing Membership in Regular Languages and Other Languages 180

11 Extensions, Generalizations, and Related Problems **183**

11.1 Distribution-Free Testing 183
11.2 Testing in the Orientation Model 184
11.3 Tolerant Testing and Distance Approximation 186
11.4 Testing and Estimating Properties of Distributions 187

Acknowledgments **196**

References **197**

Foundations and Trends® in
Theoretical Computer Science
Vol. 5, No. 2 (2009) 73–205

DOI: 10.1561/0400000029

now
the essence of knowledge

Algorithmic and Analysis Techniques in Property Testing

Dana Ron*

School of EE, Tel-Aviv University, Ramat Aviv, 10027, Israel,
danar@eng.tau.ac.il

Abstract

Property testing algorithms are "ultra"-efficient algorithms that decide whether a given object (e.g., a graph) has a certain property (e.g., bipartiteness), or is significantly different from any object that has the property. To this end property testing algorithms are given the ability to perform (local) queries to the input, though the decision they need to make usually concerns properties with a global nature. In the last two decades, property testing algorithms have been designed for many types of objects and properties, amongst them, graph properties, algebraic properties, geometric properties, and more.

In this monograph we survey results in property testing, where our emphasis is on common analysis and algorithmic techniques. Among the techniques surveyed are the following:

- The *self-correcting* approach, which was mainly applied in the study of property testing of algebraic properties;

* This work was supported by the Israel Science Foundation (grant number 246/08).

- The *enforce-and-test* approach, which was applied quite extensively in the analysis of algorithms for testing graph properties (in the dense-graphs model), as well as in other contexts;
- Szemerédi's *Regularity Lemma*, which plays a very important role in the analysis of algorithms for testing graph properties (in the dense-graphs model);
- The approach of *Testing by implicit learning*, which implies efficient testability of membership in many functions classes; and
- Algorithmic techniques for testing properties of sparse graphs, which include *local search* and *random walks*.

1

Introduction

Property testing algorithms are algorithms that perform a certain type of *approximate decision*. Namely, standard (exact) decision algorithms are required to determine whether a given input is a YES instance (has a particular property) or is a NO instance (does not have the property). In contrast, property testing algorithms are required to determine (with high success probability) whether the input has the property (in which case the algorithm should *accept*) or is *far* from having the property (in which case the algorithm should *reject*). In saying that the input is *far* from having the property we mean that the input should be modified in a non-negligible manner so that it obtains the property.

To be precise, the algorithm is given a *distance* parameter, denoted ϵ, and should reject inputs that are ϵ-far from having the property (according to a prespecified distance measure). If the input neither has the property nor is far from having the property, then the algorithm can either accept or reject. In other words, if the algorithm accepts, then we know (with high confidence) that the input is close to having the property, and if it rejects, then we know (with high confidence) that the input does not have the property.

Since a property testing algorithm should perform only an approximate decision and not an exact one, we may expect it to be (*much*) more efficient than any exact decision algorithm for the same property. In particular, as opposed to exact decision algorithms, which are considered efficient if they run in time that is polynomial in the size of the input (and the best we can hope for is linear-time algorithms), property testing algorithms may run in time that is *sublinear* in the size of the input (and hence we view them as being "ultra"-efficient). In such a case they cannot even read the entire input. Instead, they are given *query access* to the input, where the form of the queries depends on the type of input considered.

Since property testing algorithms access only a small part of the input, they are naturally allowed to be randomized and to have a small probability of error (failure). In some cases they have a non-zero error probability only on inputs that are far from having the property (and *never* reject inputs that have the property). In such a case, when they reject an input, they always provide (small) *evidence* that the input does not have the property.

By the foregoing discussion, when studying a specific property testing problem, one should define a distance measure over inputs (which determines what inputs should be rejected), and one should define the queries that the algorithm is allowed. For example, when dealing with functions and their properties (e.g., linearity), the distance measure is usually defined to be the Hamming distance normalized by the size of the domain, and queries are simply queries for values of the function at selected elements of the domain. In other cases, such as graph properties, there are several different natural models for testing (see Section 2.2 for details).

1.1 Settings in Which Property Testing is Beneficial

In addition to the intellectual interest in relating global properties to local patterns, property testing algorithms are beneficial in numerous situations. A number of such settings are discussed next.

1. *Applications that deal with huge inputs.* This is the case when dealing with very large databases in applications related to

computational biology, astronomy, study of the Internet, and more. In such cases, reading the entire input is simply infeasible. Hence, some form of approximate decision, based on accessing only a small part of the input, is crucial.

2. *Applications in which the inputs are not huge, but the property in question is $\mathcal{NP}$-hard.* Here too some form of approximation is necessary, and property testing algorithms provide one such form. In fact, while "classical" approximation algorithms are required to run in time polynomial in the size of the input, here we require even more of the algorithm: It should provide an approximately good answer, but is allowed only sublinear time. For example, there is a property testing algorithm that can be used to obtain a $(1 \pm \epsilon)$-factor approximation of the size of the maximum cut in a dense graph, whose running time depends only on ϵ, and does not depend at all on the size of the graph. (In Section 1.3 we further discuss the relation between the notion of approximation provided by property testing and more "classical" notions.)
3. *Applications in which the inputs are not huge and the corresponding decision problem has a polynomial-time algorithm, but we are interested in ultra-efficient algorithms, and do not mind sacrificing some accuracy.* For example, we may not mind accepting a graph that is not *perfectly* bipartite, but is *close* to being bipartite (that is, it has a two-way partition with relatively few "violating edges" within the two parts).
4. *Scenarios similar to the one described in the previous item except that the final decision must be exact* (though a small probability of failure is allowed). In such a case we can first run the testing algorithm, and only if it accepts do we run the exact decision procedure. Thus, we save time whenever the input is far from having the property, and this is useful when typical (but not all) inputs are far from having the property. A related scenario, discussed in Section 1.4, is the application of property testing as a preliminary step to learning.

Thus, employing a property testing algorithm yields a certain loss in terms of accuracy, but our gain, in terms of efficiency, is in many cases dramatic. Furthermore, in many cases the loss in accuracy is inevitable either because the input is huge or the problem is hard.

1.2 A Brief Overview

Property testing first appeared (implicitly) in the work of Blum et al. [35], who designed the well-known *Linearity testing algorithm*. It was first explicitly defined in the work of Rubinfeld and Sudan [123], who considered testing whether a function is a low-degree polynomial. The focus of these works was on testing algebraic properties of functions, and they, together with other works, had an important role in the design of *Probabilistically Checkable Proofs* ($\mathcal{PCP}$) systems (cf. [19, 20, 21, 22, 57, 66, 67, 123]).

The study of property testing in a more general context was initiated by Goldreich et al. [72]. They gave several general results, among them results concerning the relation between testing and learning, and then focused on testing properties of graphs (in what we refer to as the *dense-graphs* model). Following this work, property testing has been applied to many types of inputs and properties.[1] In particular, the study of algebraic properties of functions continued to play an important role, partly because of the relation to the area of *error correcting codes* (for a short explanation concerning this relation, see the beginning of Section 3). The study of graph properties was significantly extended since the work of Goldriech et al. [72]. This includes a large number of works in the dense-graphs model, as well as the introduction of other models (more suitable for graphs that are sparse or that are neither dense nor sparse), and the design of algorithms that work within these models. There has also been progress in the last few years on the design of testing algorithms for properties of functions that can be viewed as *logical* rather than algebraic (such as functions that have a small DNF representation). The study of such properties is of interest from the point of view of learning theory (see Section 1.4). Other families of properties to

[1] In what follows in this subsection we do not give references to relevant works. These references can be found in the body of this monograph when each specific result is mentioned.

which the framework of property testing has been applied include Geometric properties and "clusterability" of ensembles of points, properties defined by restricted languages (e.g., regular languages), properties of distributions, and more.

In some cases the algorithms designed are extremely efficient: The number of operations they perform *does not depend* at all on the size of the input, but only on the distance parameter ϵ. In other cases the dependence is some sublinear function of the size of the input (e.g., polylog(n) or $\sqrt{n}$, for inputs of size n), where in many of the latter cases there are matching (or almost matching) lower bounds that justify this dependence on the size of the input.

While each algorithm has features that are specific to the property it tests, there are several common algorithmic and analysis techniques. Perhaps, the two better-known analysis techniques are the *self-correcting* approach, which is applied in the analysis of many testing algorithms of algebraic properties, and Szemerédi's Regularity Lemma [124], which is central to the analysis of testing graph properties in the dense-graphs model. Other techniques include the *enforce-and-test* approach (that is also applied in the analysis of testing algorithms in the dense-graphs model, as well as in testing certain metric properties and clustering properties), and the approach of *testing by implicit learning* whose application gives a variety of results (among them testing of small DNF formula). Indeed, as the title of this monograph suggests, we organize the results presented according to such common techniques.

In addition to the extension of the scope of property testing, there have been several extensions and generalizations of the basic notion of property testing. One extension (which was already introduced in [72] but for which positive results appeared several years later) is allowing the underlying distribution (with respect to which the distance measure is defined) to be different from the uniform distribution (and in particular to be unknown — this is referred to as *distribution-free* testing). Another natural extension is to *tolerant testing*. In tolerant testing the algorithm is given two distance parameters: ϵ_1 and ϵ_2, and it must distinguish between the case that the object is ϵ_1-close to having the property (rather than perfectly having the property as in the original definition of property testing) and the case that the object is ϵ_2-far from

having the property. A related notion is that of *distance approximation* where the task is to obtain an estimate of the distance to having the property.

1.3 Property Testing and "Classical" Approximation

Consider for example the problem of deciding whether a given graph $G = (V, E)$ has a clique of size at least k, for $k = \rho n$ where ρ is a fixed constant and $n = |V|$. The "classical" notion of an approximation algorithm for this problem requires the algorithm to distinguish between the case that the max-clique in the graph has size at least ρn and, say, the case in which the max-clique has size at most $\rho n/2$.

On the other hand, when we talk of testing the "ρ-Clique" property, the task is to distinguish between the case that the graph has a clique of size ρn and the case in which it is ϵ-far from the any n-vertex graph that has a clique of size ρn. Since this property is relevant only to dense graphs (where $|E| = \Theta(n^2)$), our notion of ϵ-far in this context is that more than ϵn^2 edges should be added to the graph so that it has a clique of size ρn. This is equivalent to the *dual approximation* task (cf., [89, 90]) of distinguishing between the case that an n-vertex graph has a clique of size ρn and the case that in any subset of ρn vertices, the number of missing edges (between pairs of vertices in the subset) is more than ϵn^2.

The above two tasks are vastly different: Whereas the former task is $\mathcal{NP}$-hard, for $\rho < 1/4$ [30, 88], the latter task can be solved in $\exp(O(1/\epsilon^2))$-time, for any $\rho, \epsilon > 0$ [72]. We believe that there is no absolute sense in which one of these approximation tasks is better than the other: Each of these tasks is relevant in some applications and irrelevant in others. We also mention that in some cases the two notions coincide. For example, consider the problem of deciding whether a graph has a cut of size at least k for $k = \rho n^2$ (where ρ is a fixed constant). Then a testing algorithm for this problem will distinguish (with high probability) between the case that the max-cut in the graph is of size at least ρn^2 and the case in which the max-cut is of size less than $(\rho - \epsilon)n^2$ (which for $\epsilon = \gamma\rho$ gives a "classical" $(1 - \gamma)$-factor approximation to the size of the max-cut).

Finally, we note that while property testing algorithms are decision algorithms, in many cases they can be transformed into optimization algorithms that actually construct approximate solutions. To illustrate this, consider the two aforementioned properties, which we refer to as ρ-Clique and ρ-Cut. For the first property, suppose the graph has a clique of size at least ρn. Then, building on the testing algorithm, it is possible to obtain (with high probability (w.h.p.)), in time that grows only linearly in n, a subset of ρn vertices that is close to being a clique. (That is, the number of missing edges between pairs of vertices in the subset is at most ϵn^2.) Similarly, for the second property, if the graph has a cut of size at least ρn^2, then it is possible to obtain (w.h.p.), in time linear in n, a cut of size at least $(\rho - \epsilon)n^2$. In both cases the dependence on $1/\epsilon$ in the running time is exponential (whereas a polynomial dependence cannot be obtained unless $\mathcal{P} = \mathcal{NP}$).

For these problems and other partition problems (e.g., k-colorability), the testing algorithm (when it accepts the input) actually defines an *implicit* partition. That is, after the execution of the testing algorithm, it is possible to determine for each vertex (separately) to which part it belongs in the approximately good partition, in time poly$(1/\epsilon)$.

1.4 Property Testing and Learning

Following standard frameworks of learning theory, and in particular the PAC learning model of Valiant [125] and its variants, when we say *learning* we mean outputting a good estimate of a function to which we have query access (or from which we can obtain random labeled examples). Thus, another view of property testing is as a relaxation of learning (with queries and under the uniform distribution).[2] Namely, instead of asking that the algorithm output a good estimate of the (target) function (which is possibly assumed to belong to a particular class of functions $\mathcal{F}$), we only require that the algorithm decide whether the function belongs to $\mathcal{F}$ or is far from any function in $\mathcal{F}$. Given

[2] Testing under non-uniform distributions and testing with random examples (only) have been considered (and we discuss the former in this monograph), but most of the work in property testing deals with testing under the uniform distributions and with queries.

this view, a natural motivation for property testing is to serve as a preliminary step before learning: We can first run the testing algorithm in order to decide whether to use a particular class of functions as our hypothesis class.

In this context too we are interested in testing algorithms that are more efficient than the corresponding learning algorithms. As observed in [72], property testing is no harder than *proper* learning (where the learning algorithm is required to output a hypothesis from the same class of functions as the target function). Namely, if we have a proper learning algorithm for a class of functions $\mathcal{F}$, then we can use it as a subroutine to test the property of membership in $\mathcal{F}$.

We also note that property testing is related to *hypothesis testing* (see e.g., [101, Chap. 8]). For a short discussion of this relation, see the introduction of [121].

1.5 Organization of this Survey

In this monograph we have chosen to present results in property testing with an emphasis on analysis techniques and algorithmic techniques. Specifically:

- In Section 3 we discuss results whose analysis follows the *Self-correcting* approach (e.g., testing linearity), and mention several implications of this approach.
- In Section 4 we discuss results whose analysis follows the *enforce-and-test* approach (e.g., testing bipartiteness in the dense-graphs model). In many cases this approach implies that the testing algorithm can be transformed into an efficient approximate optimization algorithm (as discussed in Section 1.3).
- The approach of *Testing by Implicit Learning*, whose application leads to efficient testing of many function classes (e.g., DNF formula with a bounded number of terms), is described in Section 5.
- The *Regularity Lemma* of Szemerédi [124], which is a very important tool in the analysis of testing algorithms in the

dense-graphs model, is presented in Section 6, together with its application to testing triangle-freeness (in this model).
- In Section 7 we discuss algorithms for testing properties of sparse graphs that are based on *local search*.
- The use of *random walks* by testing algorithms for properties of sparse graphs is considered in Section 8.
- In Section 9 we present two examples of lower bound proofs for property testing algorithms, so as to give a flavor of the type of arguments used in such proofs.
- A small selection of other families of results, which did not fit naturally in the previous sections (e.g., testing monotonicity of functions), is discussed in Section 10.
- We conclude the monograph in Section 11 with a discussion of several extensions and generalizations of property testing (e.g., tolerant testing).

1.6 Related Surveys

There are several surveys on property testing ([58, 69, 120], and the more recent [121]), which have certain overlaps with the current survey. In particular, the recent survey [121] of the current author presents property testing from a learning theory perspective. Thus, the emphasis in that survey is mainly on testing properties of functions (that is, testing for membership in various function classes). Though the perspective taken in the current monograph is different, there are naturally several results that appear in both articles, possibly with different levels of detail.

For the broader context of sublinear-time approximation algorithms see [104, 47]. For a survey on *Streaming* (where the constraint is sublinear *space* rather than time), see [107].

2

Preliminaries

2.1 Basic Definitions and Notations

For any positive integer k, let $[k] = \{1, \ldots, k\}$. For a string $x = x_1, \ldots, x_n \in \{0,1\}^n$, we use $|x|$ to denote the number of indices i such that $x_i = 1$. We use "$\cdot$" to denote multiplication (e.g., $a \cdot b$) whenever we believe that it aids readability.

Since many of the results we survey deal with testing properties of functions (or functional representations of objects, such as graphs), we start with several definitions and notations pertaining to functions.

For two functions $f, g : X \to R$ over a finite domain X we let

$$\mathrm{dist}(f,g) \stackrel{\mathrm{def}}{=} \Pr_{x \in X}[f(x) \neq g(x)] \tag{2.1}$$

denote the distance between the functions, where the probability is taken over a uniformly selected $x \in X$.

When we use the term "with high probability", we mean with probability at least $1 - \delta$ for a small constant δ. When the claim is for higher success probability (e.g., $1 - \mathrm{poly}(1/n)$ where n is the input size), then this is stated explicitly. When considering the probability of a certain event we usually denote explicitly over what the probability is taken

(e.g., $\Pr_{x\in X}[f(x) \neq g(x)]$), unless it is clear from the context (in which case we may write $\Pr[f(x) \neq g(x)]$).

Let $\mathcal{P}$ be a *property* of functions (from domain X to range R). That is, $\mathcal{P}$ defines a subset of functions, and so we shall use the notation $g \in \mathcal{P}$ to mean that function g has the property $\mathcal{P}$. For a function $f : X \to R$ we define

$$\mathrm{dist}(f,\mathcal{P}) \stackrel{\mathrm{def}}{=} \min_{g\in\mathcal{P}}\{\mathrm{dist}(f,g)\}, \tag{2.2}$$

where there may be more than one function g that attains the minimum on the right-hand side. If $\mathrm{dist}(f,\mathcal{P}) = \epsilon$, then we shall say that f is *at distance ϵ from (having) $\mathcal{P}$* (or *has distance ϵ to $\mathcal{P}$*).

Definition 2.1 (Testing (Function Properties)). A testing algorithm for property $\mathcal{P}$ (of functions from domain X to range R) is given a distance parameter ϵ and query access to an unknown function $f : X \to R$.

- If $f \in \mathcal{P}$ then the algorithm should accept with probability at least 2/3; and
- If $\mathrm{dist}(f,\mathcal{P}) > \epsilon$ then the algorithm should reject with probability at least 2/3.

We shall be interested in bounding both the query complexity and the running time of the testing algorithm. In some cases our focus will be on the query complexity, putting aside the question of time-complexity. We observe that the choice of a success probability of 2/3 is arbitrary and can clearly be improved to $1 - \delta$, for any $\delta > 0$ at a multiplicative cost of $\log(1/\delta)$ in the complexity of the algorithm. We say that a testing algorithm has *one-sided error* if it accepts every $f \in \mathcal{P}$ with probability 1. Otherwise, it has *two-sided error*.

One may consider variations of the abovementioned notion of testing. In particular, the underlying distribution (which determines the distance in Equation (2.1), and hence in Equation (2.2)) may be an arbitrary and unknown distribution (rather than the uniform distribution). We refer to this as *distribution-free* testing, and discuss it in

Section 11.1. Another variant requires that testing be performed based on random (uniform) examples alone; that is, queries cannot be performed. We shall not discuss this variant in the current survey (and there are actually only few positive results known in this model [100]).

2.2 Testing Graph Properties

Much of the work in property testing deals with testing properties of graphs, where several models have been studied. The first two models, described next, correspond to representations of graphs as functions, and hence essentially coincide with Definition 2.1. In all that follows, the number of graph vertices is denoted by n. Unless stated otherwise, we consider undirected, simple graphs (that is, with no multiple edges and no self-loops). For a vertex v we let $\Gamma(v)$ denote its set of neighbors, and we let $\deg(v) = |\Gamma(v)|$ denote its degree.

2.2.1 The Dense-Graphs (Adjacency-Matrix) Model

The first model, introduced in [72], is the *adjacency-matrix* model. In this model the algorithm may perform queries of the form: "is there an edge between vertices u and v in the graph?" That is, the algorithm may probe the adjacency matrix representing the tested graph $G = (V(G), E(G))$, which is equivalent to querying the function $f_G : V \times V \to \{0,1\}$, where $f_G(u,v) = 1$ if and only if $(u,v) \in E$. We refer to such queries as *vertex-pair* queries. The notion of distance is also linked to this representation: A graph is said to be ϵ-far from having property $\mathcal{P}$ if more than ϵn^2 edge modifications should be performed on the graph so that it obtains the property. We note that since each edge appears twice in the functional representation (and there are no self-loops), to be exactly consistent with the functional view point, we should have said that a graph is ϵ-far from having $\mathcal{P}$ if more than $\epsilon\binom{n}{2}$ edge modifications have to be be performed so that the graph obtains the property. However, it will be somewhat simpler to work with the slightly different definition given here. This model is most suitable for *dense* graphs in which the number of edges m is $\Theta(n^2)$. For this reason we shall also refer to it as the *dense-graphs* model.

2.2.2 The Bounded-Degree (Incidence-Lists) Model

The second model, introduced in [76], is the *bounded-degree incidence-lists* model. In this model, the algorithm may perform queries of the form: "who is the i-th neighbor of vertex v in the graph?" That is, the algorithm may probe the incidence lists of the vertices in the graph, where it is assumed that all vertices have degree at most d for some fixed degree-bound d. This is equivalent to querying the function $f_G : V \times [d] \to V \cup \{\Gamma\}$ that is defined as follows: For each $v \in V$ and $i \in [d]$, if the degree of v is at least i then $f_G(v,i)$ is the i-th neighbor of v (according to some arbitrary but fixed ordering of the neighbors), and if v has degree smaller than i, then $f_G(v,i) = \Gamma$. We refer to these queries as *neighbor* queries.

Here too the notion of distance is linked to the representation: A graph is said to be ϵ-far from having property $\mathcal{P}$ if more than ϵdn edge modifications should be performed on the graph so that it obtains the property. In this case ϵ measures the fraction of entries in the incidence lists representation (the domain of f_G, which has size dn), that should be modified. This model is most suitable for graphs with $m = \Theta(dn)$ edges; that is, whose maximum degree is of the same order as the average degree. In particular, this is true for *sparse* graphs that have *constant degree*. We shall refer to it in short either as the *bounded-degree* model or as the *incidence-lists* model.

2.2.3 The Sparse-Graphs Model and the General-Graphs Model

In [112] it was first suggested to decouple the questions of representation and type of queries allowed from the definition of distance to having a property. Specifically, it was suggested that distance be measured simply with respect to the number of edges, denoted m, in the graph (or an upper bound on this number). Namely, a graph is said to be ϵ-far from having a property, if more than ϵm edge modifications should be performed so that it obtains the property. In [112] (where the focus was on sparse graphs), the algorithm is allowed the same type of queries as in the bounded-degree incidence-lists model, and it can also query the degree of any given vertex.

The main advantage of the [112] model over the bounded-degree incidence-lists model is that it is suitable for sparse graphs whose degrees may vary significantly. Hence we refer to it as the *sparse-graphs* model. We note that while it may seem that the sparse-graphs model is (strictly) more general than the bounded-degree model, this is not exactly true. The reason is that for some properties a graph may be far from having the property in the bounded-degree model but close to having it in the sparse-graphs model because it is far from any graph that has the property *and has degree at most d*, but is close to a graph that has the property but doesn't have the degree limitation.

More generally, when the graph is not necessarily sparse (and not necessarily dense), we may allow vertex-pair queries in addition to neighbor queries and degree queries. This model was first studied by Kaufman et al. [96], and is referred to as the *general-graphs* model.

3

The Self-correcting Approach

Recall that the goal of a testing algorithm for a particular property $\mathcal{P}$ is to distinguish between the case that the tested object (function f) has the property $\mathcal{P}$ and the case that it is far from any function that has $\mathcal{P}$. To this end many testing algorithms run several independent executions of some local test. For example, in the case of linearity, the algorithm tests whether $f(x) + f(y) = f(x + y)$ for uniformly selected pairs x and y in the domain of f. The local tests are such that if the function has the property, then they always pass. In order to show that the testing algorithm rejects (with high constant probability) functions that are far from having the property, the contrapositive statement is established. Namely, that if the testing algorithm accepts a function f with sufficiently large constant probability (that is, the probability that a random local test doesn't pass is sufficiently low), then f is close to having the property.

For linearity and several other properties, this is done by defining a *self-corrected* version of f. The self-corrected version is defined based on the values of f (hence the usage of *self*), and the local tests. For example in the case of linearity, the self-corrected version, $g^f(\cdot)$, is such that $g^f(x)$ is the majority (or plurality) value of $f(x + y) - f(y)$, taken

over all points y in the domain. Showing that g^f is close to f tends to be relatively easy, and the crux of the proof is in showing that g^f indeed has the tested property (e.g., is a linear function).

A coding-theory perspective. The results described in this section also have an interpretation from the point of view of coding theory. Namely, each of the properties (function classes) corresponds to a code (or family of codes): The Hadamard code, Reed–Solomon codes, Reed–Muller codes, and Generalized Reed–Muller codes, respectively. If we view functions as words (e.g., for the domain $\{0,1\}^n$, the word is of length 2^n), then the test distinguishes between codewords and words that are ϵ-far from every codeword. This is referred to as *local testing of codes* (see, e.g., [70]). Taking this point of view, the self-corrected version of a word that is not too far from being a codeword corresponds to the closest codeword.

3.1 Linearity

For the sake of simplicity we consider functions from $\{0,1\}^n$ to $\{0,1\}$. The result extends to functions $f : G \to H$, where G and H are groups. Thus, here addition is modulo 2, and for $x, y \in \{0,1\}^n$, $x + y$ is the bitwise sum (XOR) of the two strings, that is, it is the string $z \in \{0,1\}^n$ such that $z_i = x_i + y_i$. For the sake of simplicity, here when we say "linear functions" we mean linear functions that do not have a free term (as defined next). In order to allow a free term, the test (Algorithm 3.1) should be slightly modified. Thus, strictly speaking, the algorithm is actually a *homomorphism* testing algorithm.

Definition 3.1 (Linearity). We say that $f : \{0,1\}^n \to \{0,1\}$ is a linear function if there exist coefficients $b_1, \ldots, b_n \in \{0,1\}$ such that for $x = x_1, \ldots, x_n \in \{0,1\}^n$, $f(x) = \sum_{i=1}^n b_i x_i$. In other words, there exists a subset $S \subseteq \{1, \ldots, n\}$ such that $f(x) = \sum_{i \in S} x_i$.

Linearity testing is essentially the first property testing problem studied, though the term "Property Testing" was not yet explicitly defined at the time. Linearity testing was first studied by

Blum et al. [35] in the context of *Program Testing*. Namely, they were interested in designing algorithms (program-testers) that, given access to a program that is supposed to compute a *particular* function f, distinguish between the case that the program computes f correctly on all inputs and the case that it errs on at least a certain fraction ϵ of the domain elements. The program-tester should be *much simpler* than the program itself, and is typically based on calls to the program and some basic operations on the resulting outputs.

In the case of testing whether a program computes a particular linear function, the program-tester first distinguishes between the case that the program computes *some* linear function and the case that the function it computes is far from any linear function. That is, it first performs property testing of linearity. The starting point of the BLR test is the following characterization of linear functions, which is not hard to verify (and some would actually use it as a definition of linear functions).

Fact 3.1. A function $f : \{0,1\}^n \to \{0,1\}$ is linear if and only if $f(x) + f(y) = f(x+y)$ for every $x, y \in \{0,1\}^n$.

The BLR test is given in Figure 3.1.

Before we prove the correctness of the algorithm, we remark on its complexity: the algorithm performs only $O(1/\epsilon)$ queries. In particular, its query complexity is *independent* of n. This is in contrast to the query complexity of any learning algorithm for the class of linear

Algorithm 3.1: Linearity Test

1. *Repeat the following* $\Theta(1/\epsilon)$ *times.*
   (a) *Uniformly and independently select* $x, y \in \{0,1\}^n$.
   (b) *If* $f(x) + f(y) \neq f(x+y)$ *then output* reject *(and exit).*
2. *If no iteration caused rejection then output* accept.

Fig. 3.1 The BLR linearity testing algorithm.

(*parity*) functions, which is $\Omega(n)$. This is true simply because every two linear functions have distance 1/2 between them (under the uniform distribution), and a linear function is not uniquely determined by fewer than n labeled points. We note that the difference in the running time between testing and learning is less dramatic (linear in n versus quadratic in n), since the testing algorithm reads all n bits of each sampled string.

Theorem 3.1. Algorithm 3.1 is a one-sided error testing algorithm for linearity. Its query complexity is $O(1/\epsilon)$.

Let $\mathcal{L}$ denote the class of linear functions over $\{0,1\}^n$. By Fact 3.1, Algorithm 3.1 accepts every function $f \in \mathcal{L}$ with probability 1. We turn to proving that if $\text{dist}(f,\mathcal{L}) > \epsilon$ then the algorithm rejects with probability at least 2/3. Let $\epsilon_{\mathcal{L}}(f)$ denote the distance of f to being linear. Namely, if we let $\mathcal{L}$ denote the set of all linear functions, then $\epsilon_{\mathcal{L}}(f) \stackrel{\text{def}}{=} \text{dist}(f,\mathcal{L})$. We would like to prove that for every given $\epsilon > 0$, if $\epsilon > \epsilon_{\mathcal{L}}(f)$, then the probability that the test rejects is at least 2/3. This will follow from showing that if the constraint $f(x) + f(y) = f(x+y)$ is violated for relatively few pairs (x,y), then f is close to some linear function. In other words (using the terminology of [35, 123]), the characterization provided by Fact 3.1 is *robust*. To this end we define:

$$\eta(f) \stackrel{\text{def}}{=} \Pr_{x,y}[f(x) + f(y) \neq f(x+y)], \tag{3.1}$$

where in Equation (3.1) and elsewhere in this subsection, the probability is taken over a uniform choice of points in $\{0,1\}^n$. That is, $\eta(f)$ is the probability that a single iteration of Algorithm 3.1 "finds evidence" that f is not a linear function. We shall show that $\eta(f) \geq \epsilon_{\mathcal{L}}(f)/c$ for some constant $c \geq 1$ (this can actually be shown for $c = 1$ but the proof uses Discrete Fourier analysis [29] while the proof we show builds on first principles). It directly follows that if $\epsilon_{\mathcal{L}}(f) > \epsilon$ and the number of iterations is at least $2c/\epsilon$, then the probability that the test rejects is at least

$$1 - (1 - \eta(f))^{2c/\epsilon} > 1 - e^{-2c\eta(f)/\epsilon} \geq 1 - e^{-2} > 2/3, \tag{3.2}$$

thus establishing Theorem 3.1.

Somewhat unintuitively, showing that $\eta(f) \geq \epsilon_{\mathcal{L}}(f)/c$ is easier if $\epsilon_{\mathcal{L}}(f)$ is not too large. Specifically, it is not hard to prove the following claim.

Claim 3.2. For every function f it holds that $\eta(f) \geq 3\epsilon_{\mathcal{L}}(f)(1 - 2\epsilon_{\mathcal{L}}(f))$. In particular, if $\epsilon_{\mathcal{L}}(f) \leq \frac{1}{4}$ then $\eta(f) \geq \frac{3}{2}\epsilon(f)$ (and more generally, if $\eta(f) = \frac{1}{2} - \gamma$ for $\gamma > 0$, then $\eta(f) \geq 6\gamma\epsilon_{\mathcal{L}}(f)$, which gives a weak bound as $\eta(f)$ approaches $1/2$).

It remains to prove that even when $\epsilon_{\mathcal{L}}(f)$ is not bounded away (from above) from $1/2$ then still $\eta(f) \geq \epsilon_{\mathcal{L}}(f)/c$ for a constant c. To this end we define the following *majority* function: for each fixed choice of $x \in \{0,1\}^n$,

$$g^f(x) = \begin{cases} 0 & \text{if } \Pr_y[f(x+y) - f(y) = 0] \geq 1/2 \\ 1 & \text{otherwise.} \end{cases} \tag{3.3}$$

Let

$$V_y^f(x) \stackrel{\text{def}}{=} f(x+y) - f(y) = f(y) + f(x+y) \tag{3.4}$$

be the *Vote* that y *casts* on the value of x. By the definition of $g^f(x)$ it is the majority vote taken over all y. Note that if f is linear then $V_y^f(x) = f(x)$ for every $y \in \{0,1\}^n$.

We shall prove two lemmas, stated next.

Lemma 3.3. $\text{dist}(f, g^f) \leq 2\eta(f)$.

Lemma 3.4. If $\eta(f) \leq \frac{1}{6}$ then g^f is a linear function.

By combining Lemmas 3.3 and 3.4 we get that $\eta(f) \geq \frac{1}{6}\epsilon_{\mathcal{L}}(f)$. To see why this is true, observe first that if $\eta(f) > \frac{1}{6}$, then the inequality clearly holds because $\epsilon_{\mathcal{L}}(f) \leq 1$. (In fact, since it can be shown that $\epsilon_{\mathcal{L}}(f) \leq 1/2$ for every f, we actually have that $\eta(f) \geq \frac{1}{3}\epsilon_{\mathcal{L}}(f)$.) Otherwise ($\eta(f) \leq \frac{1}{6}$), since g^f is linear and $\text{dist}(f, g^f) \leq 2\eta(f)$, we have that $\epsilon_{\mathcal{L}}(f) \leq \text{dist}(f, g^f) \leq 2\eta(f)$, so that $\eta(f) \geq \epsilon_{\mathcal{L}}(f)/2$, and we are done.

Since g^f is defined only based on f (and it is a linear function close to f), we view it as the *self-corrected* version of f (with respect to linearity).

Proof of Lemma 3.3. Let $\Delta(f,g^f) = \{x : g^f(x) \neq f(x)\}$ be the set of points on which f and g^f differ. By the definition of $g^f(x)$, it is the majority value of $V_y^f(x)$ taken over all y. Hence, for every fixed choice of $x \in \Delta(f,g^f)$ we have that $\Pr_y[V_y^f(x) \neq f(x)] \geq 1/2$. Therefore,

$$\begin{aligned}\Pr_{x,y}[f(x) \neq V_y^f(x)] &\geq \Pr_x[x \in \Delta(f,g^f)] \\ &\quad \cdot \Pr_y[f(x) \neq V_y^f(x) \,|\, x \in \Delta(f,g^f)] \\ &\geq \frac{1}{2}\Pr_x[g^f(x) \neq f(x)]. \end{aligned} \tag{3.5}$$

Since $\Pr_{x,y}[f(x) \neq V_y^f(x)] = \eta(f)$, it must hold that $\Pr_x[g^f(x) \neq f(x)] \leq 2\eta(f)$. □

Proof of Lemma 3.4. In order to prove this lemma, we first prove the next claim.

Claim 3.5. For every $x \in \{0,1\}^n$ it holds that $\Pr_y[g^f(x) = V_y^f(x)] \geq 1 - 2\eta(f)$.

Note that by the definition of g^f as the "majority-vote function", $\Pr_y[g^f(x) = V_y^f(x)] \geq \frac{1}{2}$. Claim 3.5 says that the majority is actually "stronger" (for small $\eta(f)$).

Proof. Fixing x, let $p_0(x) = \Pr_y[V_y^f(x) = 0]$, and let $p_1(x) = \Pr_y[V_y^f(x) = 1]$. We are interested in lower bounding $p_{g^f(x)}(x)$, where, by the definition of g^f, $p_{g^f(x)}(x) = \max\{p_0(x), p_1(x)\}$. Now,

$$p_{g^f(x)}(x) = p_{g^f(x)}(x) \cdot (p_0(x) + p_1(x)) \geq (p_0(x))^2 + (p_1(x))^2 \,. \tag{3.6}$$

Since $(p_0(x))^2 + (p_1(x))^2 = \Pr_{y,z}[V_y^f(x) = V_z^f(x)]$, in order to lower bound $p_{g^f(x)}(x)$, it suffices to lower bound $\Pr_{y,z}[V_y^f(x) = V_z^f(x)]$, which is what we do next. In what follows we shall use the fact that the range

of f is $\{0,1\}$.

$$\begin{aligned}
&\Pr_{y,z}[V_y^f(x) = V_z^f(x)] \\
&\quad = \Pr_{y,z}[V_y^f(x) + V_z^f(x) = 0] \\
&\quad = \Pr_{y,z}[f(y) + f(x+y) + f(z) + f(x+z) = 0] \\
&\quad = \Pr_{y,z}[f(y) + f(x+z) + f(y+x+z) \\
&\qquad + f(z) + f(x+y) + f(z+x+y) = 0] \\
&\quad \geq \Pr_{y,z}[f(y) + f(x+z) + f(y+x+z) = 0 \\
&\qquad \wedge f(z) + f(x+y) + f(z+x+y) = 0] \\
&\quad = 1 - \Pr_{y,z}[f(y) + f(x+z) + f(y+x+z) = 1 \\
&\qquad \vee f(z) + f(x+y) + f(z+x+y) = 1] \\
&\quad \geq 1 - (\Pr_{y,z}[f(y) + f(x+z) + f(y+x+z) = 1] \\
&\qquad + \Pr_{y,z}[f(z) + f(x+y) + f(z+x+y) = 1]) \\
&\quad = 1 - 2\eta(f). \qquad \square
\end{aligned}$$

In order to complete the proof of Lemma 3.4, we show that for any two given points $a, b \in \{0,1\}^n$, $g^f(a) + g^f(b) = g^f(a+b)$. We prove this by the probabilistic method. Specifically, we show that there exists a point y for which the following three equalities hold simultaneously:

(1) $g^f(a) = f(a+y) - f(y)$ $(= V_y^f(a))$.
(2) $g^f(b) = f(b + (a+y)) - f(a+y)$ $(= V_{a+y}^f(b))$.
(3) $g^f(a+b) = f(a+b+y) - f(y)$ $(= V_y^f(a+b))$.

But in such a case,

$$g^f(a) + g^f(b) = f(b+a+y) - f(y) = g^f(a+b), \tag{3.7}$$

and we are done. To see why there exists such a point y, consider selecting y uniformly at random. For each of the above three equalities, by Claim 3.5, the probability that the equality does not hold is at most $2\eta(f)$. By the union bound, the probability (over a uniform selection of y) that any one of the three does not hold is at most $6\eta(f)$. Since $\eta(f) < 1/6$, this is bounded away from 1, and so the probability that there *exists* a point y for which all three equalities hold simultaneously is greater than 0, implying that such a point y indeed exists. $\square$

3.1.1 Self-correction in Its Own Right

In the foregoing discussion we presented self-correction as an analysis technique. However, the argument introduced directly implies that if f is not too far from being linear, then it can be *constructively* self-corrected (which was also a task studied in [35]). Namely, for any x of our choice, if we want to know the value, on x, of the linear function closest to f (or, in the coding theory view, we want to know the correct bit in the position corresponding to x in the closest codeword), then we do the following. We select, uniformly at random, $y_1, \ldots, y_t$ and take the majority vote of $V^f_{y_1}(x), \ldots, V^f_{y_t}(x)$ (where the choice of t determines the probability that the majority is correct). The fact that self-correction can be done constructively has several implications, which we discuss in Section 3.3.

3.2 Low-Degree Polynomials

Self-correcting is also applied in several results on testing low-degree polynomials over finite fields [11, 66, 92, 98, 123]. Consider first the univariate case, that is, testing whether a function $f : F \to F$ for a finite field F is of the form $f(x) = \sum_{i=0}^{d} C^f_i x^i$ for a given degree-bound d (where the coefficients C^f_i belong to F). In this case, the testing algorithm [123] works by simply trying to interpolate the function f on $\Theta(1/\epsilon)$ collections of $d+2$ uniformly selected points, and checking whether the resulting functions are all polynomial of degree at most d. Thus the algorithm essentially works by trying to learn the function f (and the interpolated function obtained is the self-corrected version of f).[1]

When dealing with the more general case of multivariate polynomials, the results vary according to the relation between the size of the field $|F|$ and the degree-bound d. In what follows we give the high-level idea of the results, and note where self-correcting comes into play.

[1] In fact, a slightly more efficient version of the algorithm would select $d+1$ arbitrary points, find (by interpolating), the unique polynomial g^f of degree d that agrees with f on these points, and then check that g^f agrees with f on an additional sample of $\Theta(1/\epsilon)$ uniformly selected points.

The case of large fields. In the first result, of Rubinfeld and Sudan [123] (which builds in part on [66]), it is assumed that $|F| \geq d+2$ (and that F is a prime field). The idea of the algorithm is to select random *lines* in F^n, and to verify that the restriction of f to each line is a (univariate) polynomial of degree at most d. To be precise, the algorithm does not query all points on the line, but rather $d+2$ evenly spaced points of the form $f(x+i\cdot y)$ (for uniformly selected $x, y \in F^n$), and verifies that they obey a certain linear constraint.

Here the self-corrected version of f (denoted g^f) is defined (for each $x \in F^n$) as the *plurality* value taken over all $y \in F^n$ of the *vote* $V_y^f(x)$ of y on the value of x. This vote is the value that $f(x)$ "should have", so that the restriction of f to the line defined by x and y will indeed be a univariate polynomial of degree at most d (conditioned on the values that f has on $x+i\cdot y$ for $i \neq 0$). This value is simply a linear combination of $f(x+i\cdot y)$ for $1 \leq i \leq d+1$. Similarly to the analysis of the linearity testing algorithm, it is shown that if the test accepts with sufficiently high probability, then g^f is a polynomial of degree at most d and is close to f.

Small fields and the general case. The case that $|F| < d+2$ was first studied by Alon et al. [11] for the special case of $|F| = 2$ (which corresponds to the well-known Reed–Muller codes), and was later generalized to $|F| > 2$ in [98, 92] (where the two works, [98] and [92], differ somewhat in the scope and the techniques). A main building block of the analysis of the general case in [98] is the following characterization of degree-d multivariate polynomials over finite fields.

Theorem 3.6. Let $F = \mathrm{GF}(q)$ where $q = p^s$ and p is prime. Let d be an integer, and let $f : F^n \to F$. The function f is a polynomial of degree at most d if and only if its restriction to every affine subspace of dimension $\ell = \left\lceil \frac{d+1}{q-q/p} \right\rceil$ is a polynomial of degree at most d.

Theorem 3.6 generalizes the characterization result of Friedl and Sudan [66] (on which the aforementioned algorithm of [123] builds) which refers to the case $q - q/p \geq d+1$. That is, the size of the field

F is sufficiently larger than the degree d, and the affine subspaces considered are of dimension $\ell = 1$.

The testing algorithm of [98] utilizes the characterization in Theorem 3.6 (which is shown to be robust). Specifically, the algorithm selects random affine subspaces (of dimension ℓ as defined in Theorem 3.6), and checks that the restriction of the function f to each of the selected subspaces is indeed a polynomial of degree at most d. Such a check is implemented by verifying that various linear combinations of the values of f on the subspace sum to 0. Here too the self-corrected version of f, g^f, is defined for each $x \in F^n$ as the plurality value of a certain vote. In this case the vote is taken over all ℓ-tuples $y_1, \ldots, y_\ell$, which are linearly independent points in F^n. Each such tuple, together with x, determines an affine subspace of dimension ℓ, and the vote is the value that $f(x)$ "should have" so that the restriction of f to the subspace be a polynomial of degree at most d (conditioned on the values of f on the other points in the subspace).

The query complexity and running times of the above algorithms depend on the relation between $|F|$ and d. Roughly speaking, for any degree d, as the field size $|F|$ increases, the complexity decreases from being exponential in d (e.g., when $|F| = 2$) to being polynomial in d when F is of the same order as d (or larger). This behavior can be shown to be fairly tight by almost matching lower bounds. More details on these algorithms and their analyses can be found in [121, Section 3].

Extending the results for testing low-degree polynomials. The testability of low-degree polynomials was significantly extended by Kaufman and Sudan [99]. Using invariance properties of algebraic function classes, they give sufficient conditions for efficient testing. These conditions imply previously known results as well as new ones (e.g., subfamilies of polynomials with degree that is linear in n). Self-correcting plays a role in their analysis as well.

Other techniques for testing algebraic properties. One of the analysis techniques that was used early on in the study of testing linearity by Bellare et al. [29] is *Fourier analysis*. Bellare et al. [29] reveal a relation between the Fourier coefficients of (an appropriate transfor-

mation of) a function f and its distance to linearity as well as a relation between these coefficients and the probability that the BLR test [35] rejects f. Using these relations they gain better understanding of the behavior of the linearity test.

Another technique that was applied more recently by Kaufman and Litsyn [97] for testing certain families of "almost-orthogonal" codes (e.g., dual-BCH) is the *weight distribution* (*spectrum*) of a code and its dual.

3.3 Implications of Self-correction

3.3.1 Self-correcting and Distribution-Free testing

One interesting implication of self-correction is in the context of *distribution-free testing*. In distribution-free testing there is an unknown underlying distribution D over the domain X, and distance is defined with respect to this distribution. That is, for two functions $f, g : X \to R$ we let

$$\mathrm{dist}_D(f,g) \stackrel{\mathrm{def}}{=} \Pr_{x\sim D}[f(x) \neq g(x)], \tag{3.8}$$

and for a function $f : X \to R$ and a property (family of functions) $\mathcal{P}$ we let

$$\mathrm{dist}_D(f,\mathcal{P}) \stackrel{\mathrm{def}}{=} \min_{g\in\mathcal{P}}\{\mathrm{dist}_D(f,g)\}. \tag{3.9}$$

As in the "standard" definition of testing (when the underlying distribution is uniform), the algorithm is given query access to the tested function f. In addition, the algorithm is given access to examples $x \in X$ distributed according to D. The algorithm should still accept with probability at least 2/3 if[2] $f \in \mathcal{P}$, but now it should reject (with probability at least 2/3) if $\mathrm{dist}_D(f,\mathcal{P}) > \epsilon$.

The notion of distribution-free testing was introduced in [72]. However, in that paper it was only observed that distribution-free (proper) learning implies distribution-free testing. Other than that, in [72] there

[2] An alternative definition would require that the algorithm accept (with high probability) if $\mathrm{dist}_D(f,\mathcal{P}) = 0$. We adopt the requirement that $f \in \mathcal{P}$ since the known results are under this definition.

were only negative results about distribution-free testing of graph properties, which have very efficient standard testing algorithms (that is, that work under the uniform distribution). The first positive results for distribution-free testing (with queries) were given by Halevy and Kushilevitz [81, 84]. Here we describe their general result for obtaining distribution-free testing algorithms from standard testing algorithms when the function class has a (property) self-corrector.

Halevy and Kushilevitz introduce the notion of a *property self-corrector*, which generalizes the notion of a self-corrector, introduced by Blum et al. [35].

Definition 3.2. A γ-self-corrector for a class of functions $\mathcal{F}$ is a probabilistic oracle machine M, which is given oracle access to an arbitrary function $f : X \to R$ and satisfies the following conditions (where M^f denotes the execution of M when given oracle access to f):

- If $f \in \mathcal{F}$ then $\Pr[M^f(x) = f(x)] = 1$ for every $x \in X$.
- If there exists a function $g \in \mathcal{F}$ such that $\text{dist}(f, g) \leq \gamma$, then $\Pr[M^f(x) = g(x)] \geq 2/3$ for every $x \in X$.

In this definition, the distance (i.e., the measure $\text{dist}(\cdot, \cdot)$) is defined with respect to the uniform distribution. However, it will be useful for distribution-free testing (when the distance ($\text{dist}_D(\cdot, \cdot)$) is measured with respect to some fixed but unknown distribution (D)). Observe that the second condition in Definition 3.2 implies that g must be unique.

Theorem 3.7. Let $\mathcal{F}$ be a class of functions that has a standard testing algorithm T and a γ-self-corrector M. Let $Q_T(\cdot)$ be the query complexity of T (as a function of the distance parameter ϵ) and let Q_M be the query complexity of M (that is, the number of queries performed in order to determine $M^f(x)$). Then there exists a distribution-free testing algorithm for $\mathcal{F}$ with query complexity $O(Q_T(\min\{\epsilon, \gamma\}) + Q_M/\epsilon)$.

In Figure 3.2 we give the distribution-free testing algorithm referred to in Theorem 3.7. We assume that the distance parameter ϵ is smaller than γ (or else we set ϵ to γ).

Algorithm 3.2: Distribution-free test based on self-correction

1. *Run the standard testing algorithm T on f, 24 (independent) times with the distance parameter ϵ. If T outputs* reject *in at least half of these executions then halt and output* reject.
2. *Repeat $2/\epsilon$ times:*
   (a) *Sample a point $x \in X$ according to the underlying distribution D.*
   (b) *Repeat twice: Compute $M^f(x)$ and query $f(x)$. If $M^f(x) \neq f(x)$ then output* reject *(and exit).*
3. *If no iteration caused rejection then output* accept.

Fig. 3.2 The distribution-free testing algorithm that is based on self-correction.

Proof of Theorem 3.7. Clearly, the query complexity of Algorithm 3.2 is as stated in Theorem 3.7. Hence we turn to proving its correctness. Consider first the case that $f \in \mathcal{F}$. In such a case the standard testing algorithm T should accept with probability at least 2/3, and the probability that it rejects in at least half of its 24 independent executions is less than 1/3. Assume such an event did not occur. By the first condition in Definition 3.2, for every $x \in X$, we have that $M^f(x) = f(x)$ with probability 1. Hence the second step of the algorithm never causes rejection. It follows that the algorithm accepts with probability at least 2/3. (Note that if T has one-sided error then so does Algorithm 3.2.)

In what follows, in order to distinguish between the case that distance is measured with respect to the uniform distribution and the case that it is measured with respect to the underlying distribution D, we shall use the terms (ϵ, U)-close (or far) and (ϵ, D)-close (or far), respectively. Assume now that f is (ϵ, D)-far from $\mathcal{F}$. If f is also (ϵ, U)-far from $\mathcal{F}$ then it is rejected by T with probability at least 2/3, and is therefore rejected by the algorithm in its first step with probability at least 2/3. Hence assume that f is (ϵ, U)-close to $\mathcal{F}$.

In such a case, by the second condition in Definition 3.2, for every $x \in X$, $\Pr[M^f(x) = g(x)] \geq 2/3$, where g is a fixed function in $\mathcal{F}$ that is (γ, U)-close to f and the probability is taken over the internal coin flips of M (recall that $\epsilon \leq \gamma$ so such a function g exists). In particular, for any point x such that $f(x) \neq g(x)$ we have that $\Pr[M^f(x) \neq f(x)] \geq 2/3$. Thus, if in one of the $(2/\epsilon)$ iterations of the second step of the algorithm we obtain such a point x, then the algorithm rejects with probability at least $1 - (1/3)^2 = 8/9$ (since it computes $M^f(x)$ twice). But since f is (ϵ, D)-far from $\mathcal{F}$, for every function $h \in \mathcal{F}$, we have that $\Pr_{x \sim D}[f(x) \neq h(x)] > \epsilon$, and in particular this is true of g. Hence the probability that the algorithm does not obtain any point x for which $f(x) \neq g(x)$ is at most $(1 - \epsilon)^{2/\epsilon} < \exp(-2) < 1/6$. It follows that the algorithm rejects with probability at least $1 - (1/9 + 1/6) > 2/3$, as required. □

In particular, Theorem 3.7 can be applied to obtain distribution-free property testing algorithms for all properties described in this section. Other properties (function classes) include singletons (since they are a subclass of the class of linear functions), and k-juntas (since they are a subclass of degree-k multivariate polynomials).

3.3.2 Self-correcting and Testing Subclasses of Functions

Two other (related) results that build on self-correcting are testing singletons (also known as *dictator functions*) and testing monomials.

Definition 3.3 (Singletons and Monomials). A function $f : \{0,1\}^n \to \{0,1\}$ is a singleton function if there exists an $i \in [n]$ such that $f(x) = x_i$ for every $x \in \{0,1\}^n$ or $f(x) = \bar{x}_i$ for every $x \in \{0,1\}^n$.

We say that f is a monotone k-monomial for $1 \leq k \leq n$ if there exist k indices $i_1, \ldots, i_k \in [n]$ such that $f(x) = x_{i_1} \wedge \cdots \wedge x_{i_k}$ for every $x \in \{0,1\}^n$. If we allow some of the x_{i_j}s above to be replaced with $\bar{x}_{i_j}$, then f is a k-monomial. The function f is a monomial if it is a k-monomial for some $1 \leq k \leq n$.

Here we describe the algorithm for testing singletons and explain how self-correcting comes into play. The testing algorithm for k-monomials generalizes the algorithm for testing singletons and also builds on

self-correcting. We actually describe an algorithm for testing whether a function f is a *monotone* singleton. In order to test whether f is a singleton we can check whether either f or $\bar{f}$ passes the monotone singleton test. For the sake of succinctness, in what follows we refer to monotone singletons simply as singletons.

For $x, y \in \{0,1\}^n$ we shall use $x \wedge y$ to denote the bitwise "AND" of the two strings. That is, $z = x \wedge y$ satisfies $z_i = x_i \wedge y_i$ for every $1 \leq i \leq n$.

The following characterization of monotone k-monomials motivates our algorithm.

Lemma 3.8. Let $f : \{0,1\}^n \to \{0,1\}$. The function f is a monotone k-monomial if and only if the following two conditions hold:

(1) $\Pr[f(x) = 1] = \frac{1}{2^k}$; and
(2) $f(x \wedge y) = f(x) \wedge f(y)$ for all $x, y \in \{0,1\}^n$.

In what follows we shall say that a pair of points $x, y \in \{0,1\}^n$ are *violating with respect to* f if $f(x \wedge y) \neq f(x) \wedge f(y)$.

Proof. If f is a k-monomial then clearly the conditions hold. We turn to prove the other direction. We first observe that the two conditions imply that $f(x) = 0$ for all $|x| < k$, where $|x|$ denotes the number of ones in x. In order to verify this, assume in contradiction that there exists some x such that $|x| < k$ but $f(x) = 1$. Now consider any y such that $y_i = 1$ whenever $x_i = 1$. Then $x \wedge y = x$, and therefore $f(x \wedge y) = 1$. But by the second item, since $f(x) = 1$, it must also hold that $f(y) = 1$. However, since $|x| < k$, the number of such points y is strictly greater than 2^{n-k}, contradicting the first item.

Next let $F_1 \stackrel{\text{def}}{=} \{x : f(x) = 1\}$, and let $y = \bigwedge_{x \in F_1} x$. Using the second item in the claim we get:

$$f(y) = f\left(\bigwedge_{x \in F_1} x\right) = \bigwedge_{x \in F_1} f(x) = 1 \,. \tag{3.10}$$

However, we have just shown that $f(x) = 0$ for all $|x| < k$, and thus $|y| \geq k$. Hence, there exist k indices $i_1, \ldots, i_k$ such that $y_{i_j} = 1$ for all $1 \leq$

$j \leq k$. But $y_{i_j} = \bigwedge_{x \in F_1} x_{i_j}$. Hence, $x_{i_1} = \ldots = x_{i_k} = 1$ for every $x \in F_1$. The first item now implies that $f(x) = x_{i_1} \wedge \ldots \wedge x_{i_k}$ for every $x \in \{0,1\}^n$. □

Given Lemma 3.8, a natural candidate for a testing algorithm for singletons would take a sample of uniformly selected pairs (x, y), and for each pair verify that it is not violating with respect to f. In addition, the test would check that $\Pr[f(x) = 0]$ is roughly $1/2$ (or else any monotone k-monomial would pass the test). As shown in [116], the correctness of this testing algorithm can be proved as long as the distance between f and the closest singleton is bounded away from $1/2$. It is an open question whether this testing algorithm is correct in general.

We next describe a modified version of this algorithm, which consists of two stages. In the first stage, the algorithm tests whether f belongs to (is close to) a more general class of functions (that contains all singleton functions). In the second stage it applies a slight variant of the original test (as described in the previous paragraph). Specifically, the more general class of functions is the class $\mathcal{L}$ of linear Boolean functions over $\{0,1\}^n$, which was discussed in Section 3.1. Clearly, every singleton function $f(x) = x_i$ is a linear function. Hence, if f is a singleton function, then it passes the first stage of the test (the linearity test) with probability 1. On the other hand, if it is far from any linear function, then it will be rejected already by the linearity test. As we shall see, if f is far from every singleton function, *but* it is close to some linear function that is not a singleton function (so that it may pass the linearity test), then we can prove that it will be rejected in the second stage of the algorithm with high probability.

In order to motivate the modification we introduce in the aforementioned "natural" singleton test, we state the following lemma and discuss its implications.

Lemma 3.9. Let $S \subseteq [n]$, and let $g_s(x) = \sum_{i \in S} x_i$ (where the sum is taken modulo 2). If $|S|$ is even then

$$\Pr_{x,y}[g_s(x \wedge y) = g_s(x) \wedge g_s(y)] = \frac{1}{2} + \frac{1}{2^{|S|+1}}$$

and if $|S|$ is odd then

$$\Pr_{x,y}[g_s(x \wedge y) = g_s(x) \wedge g_s(y)] = \frac{1}{2} + \frac{1}{2^{|S|}}.$$

Proof. Let $s = |S|$, and let x, y be two strings such that (i) x has $0 \leq i \leq s$ ones in S, that is, $|\{\ell \in S : x_\ell = 1\}| = i$; (ii) $x \wedge y$ has $0 \leq k \leq i$ ones in S; and (iii) y has a total of $j + k$ ones in S, where $0 \leq j \leq s - i$.

If $g_s(x \wedge y) = g_s(x) \wedge g_s(y)$, then either (1) i is even and k is even, or (2) i is odd and j is even. Let $Z_1 \subset \{0,1\}^n \times \{0,1\}^n$ be the subset of pairs x, y that obey the first constraint, and let $Z_2 \subset \{0,1\}^n \times \{0,1\}^n$ be the subset of pairs x, y that obey the second constraint. Since the two subsets are disjoint,

$$\Pr_{x,y}[g_s(x \wedge y) = g_s(x) \wedge g_s(y)] = 2^{-2n}(|Z_1| + |Z_2|). \tag{3.11}$$

It remains to compute the sizes of the two sets. Since the coordinates of x and y outside S do not determine whether the pair x, y belongs to one of these sets, we have

$$|Z_1| = 2^{n-s} \cdot 2^{n-s} \cdot \left(\sum_{i=0, i \text{ even}}^{s} \binom{s}{i} \sum_{k=0, k \text{ even}}^{i} \binom{i}{k} \sum_{j=0}^{s-i} \binom{s-i}{j} \right) \tag{3.12}$$

and

$$|Z_2| = 2^{n-s} \cdot 2^{n-s} \cdot \left(\sum_{i=0, i \text{ odd}}^{s} \binom{s}{i} \sum_{k=0}^{i} \binom{i}{k} \sum_{j=0, j \text{ even}}^{s-i} \binom{s-i}{j} \right). \tag{3.13}$$

The right-hand side of Equation (3.12) equals

$$2^{2n-2s} \cdot (2^{2s-2} + 2^{s-1}) = 2^{2n-2} + 2^{2n-s-1} = 2^{2n} \cdot (2^{-2} + 2^{-(s+1)}). \tag{3.14}$$

The right-hand side of Equation (3.13) equals $2^{2n} \cdot (2^{-2} + 2^{-(s+1)})$ if s is odd and 2^{2n-2} if s is even. The lemma follows by combining Equations (3.12) and (3.13) with Equation (3.11). □

Hence, if f is a linear function that is not a singleton and is not the all-0 function, that is, $f = g_s$ for $|S| \geq 2$, then the probability that

a uniformly selected pair x, y is violating with respect to f is at least $1/8$. In this case, a sample of 16 such pairs will contain a violating pair with probability at least $1 - (1 - 1/8)^{16} \geq 1 - e^{-2} > 2/3$.

However, what if f passes the linearity test but is only close to being a linear function? Let g denote the linear function that is closest to f and let δ be the distance between them. (Note that g is unique, given that f is sufficiently close to a linear function). What we would like to do is check whether g is a singleton, by selecting a sample of pairs x, y and checking whether it contains a violating pair with respect to g. Observe that, since the distance between functions is measured with respect to the uniform distribution, for a uniformly selected pair x, y, with probability at least $(1 - \delta)^2$, both $f(x) = g(x)$ and $f(y) = g(y)$. However, we cannot make a similar claim about $f(x \wedge y)$ and $g(x \wedge y)$, since $x \wedge y$ is *not* uniformly distributed. Thus, it is not clear that we can replace the violation test for g with a violation test for f. In addition we need to verify that g is not the all-0 function.

The solution is to use a *self-corrector* for linear functions, essentially as defined in Definition 3.2. Namely, given query access to a function $f : \{0,1\}^n \to \{0,1\}$, which is strictly closer than $1/4$ to some linear function g, and an input $x \in \{0,1\}^n$, the procedure Self-Correct(f, x) returns the value of $g(x)$, with probability at least $9/10$. The query complexity of the procedure is constant. The testing algorithm for singletons is given in Figure 3.3.

Theorem 3.10. Algorithm 3.3 is a one-sided error testing algorithm for monotone singletons. The query complexity of the algorithm is $O(1/\epsilon)$.

Proof. Since the linearity testing algorithm has a one-sided error, if f is a singleton function, then it always passes the linearity test. In this case the self-corrector always returns the value of f on every given input point. In particular, Self-Correct$(f, \vec{1}) = f(\vec{1}) = 1$, since every monotone singleton has value 1 on the all-1 vector. Similarly, no violating pair can be found in Step 3.3.2. Hence, Algorithm 3.3.2 always accepts a singleton.

Algorithm 3.3: Test for Singleton Functions

1. *Apply the linearity test (Algorithm 3.1) to f with distance parameter* $\min(1/5, \epsilon)$. *If the test rejects then output* reject *(and exit).*
2. *If Self-Correct*$(f, \vec{1}) = 0$ *(where $\vec{1}$ is the all-1 vector), then output* reject *(and exit).*
3. *Uniformly and independently select $m = 64$ pairs of points x, y.*
   - *For each such pair, let $b_x =$ Self-Correct(f, x), $b_y =$ Self-Correct(f, y) and $b_{x \wedge y} =$ Self-Correct$(f, x \wedge y)$.*
   4. *Check that $b_{x \wedge y} = b_x \wedge b_y$.*

(1) *If one of the checks fails then output* reject. *Otherwise output* accept.

Fig. 3.3 The testing algorithm for singletons (that is based on self-correction).

Assume, without loss of generality, that $\epsilon \leq 1/5$. Consider the case in which f is ϵ-far from any singleton. If it is also ϵ-far from any linear function, then it will be rejected with probability at least 9/10 in the first step of the algorithm. Otherwise, there exists a unique linear function g such that f is ϵ-close to g. If g is the all-0 function, then f is rejected with probability at least 9/10 (in Step 3.3.2).

Otherwise, g is a linear function of at least two variables. By Lemma 3.9, the probability that a uniformly selected pair x, y is a violating pair with respect to g is at least 1/8. Given such a pair, the probability that the self-corrector returns the value of g on all the three calls (that is, $b_x = g(x)$, $b_y = g(y)$, and $b_{x \wedge y} = g(x \wedge y)$), is at least $(1 - 1/10)^3 > 7/10$. The probability that Algorithm 3.3.2 obtains a violating pair with respect to g *and* all calls to the self-corrector return the correct value, is greater than 1/16. Therefore, a sample of 64 pairs will ensure that a violation $b_{x \wedge y} \neq b_x \wedge b_y$ will be found with probability at

least 9/10. The total probability that f is accepted, despite being ϵ-far from any singleton, is hence at most $3 \cdot (1/10) < 1/3$.

The query complexity of the algorithm is dominated by the query complexity of the linearity tester, which is $O(1/\epsilon)$. The second stage takes constant time. □

4

The Enforce-and-Test Approach

In order to introduce the idea of the "enforce-and-test" approach, we start by giving a very simple example: testing whether a graph is a *biclique*. We later present the slightly more involved analysis for the more general problem of testing whether a graph is bipartite, and shortly discuss other properties for which the enforce-and-test approach is applied. We note that this approach was most commonly (though not solely) applied when testing properties of graphs in the dense-graphs model.

4.1 Testing Whether a Graph is a Biclique

A graph $G = (V, E)$ is a *biclique* if there exists a partition (V_1, V_2) of the graph vertices such that $E = V_1 \times V_2$ (that is, V_1 and V_2 are independent sets and there is a complete bipartite graph between V_1 and V_2). Recall that by the definition of the dense-graphs model, a graph is ϵ-far from being a biclique (and hence should be rejected with probability at least 2/3) if more than ϵn^2 edge-modification (additions and/or deletions) should be performed on the graph so that it becomes a biclique. This is equivalent to saying that for *every* partition (V_1, V_2), the size of the symmetric difference $(E \setminus V_1 \times V_2) \cup (V_1 \times V_2 \setminus E)$ is greater than ϵn^2.

Consider the following algorithm. It first selects an arbitrary vertex v_0. It then uniformly and independently selects $s = 2/\epsilon$ pairs of vertices $(u_1, w_1), \ldots, (u_s, w_s)$ and queries each pair (u_j, w_j) as well as (v_0, u_j) and (v_0, w_j). If the algorithm encounters evidence that the graph is not a biclique (that is, for some $1 \leq j \leq s$ we have that (u_j, w_j), (v_0, u_j), and (v_0, w_j) are all edges or exactly one of them is an edge), then it rejects. Otherwise it accepts. Since the algorithm only rejects when it finds evidence that the graph is not a biclique, it accepts every biclique with probability 1.

In order to prove if the tested graph is ϵ-far from being a biclique, then the algorithm rejects it with probability at least 2/3, we do the following. We view v_0 as *enforcing* a partition of all graph vertices in the following manner. On one side of the partition (V_1) we put v_0 together with all vertices that it does not neighbor, and on the other side (V_2), we put all the neighbors of v_0. The vertex v_0 *enforces* this partition in the sense that if the graph is indeed a biclique then this is the only partition that obeys the biclique conditions. On the other hand, recall that if the graph is ϵ-far from being a biclique, then for every partition (V_1, V_2) we have that $|E \setminus V_1 \times V_2| + |V_1 \times V_2 \setminus E| > \epsilon n^2$. In particular this is true of the aforementioned partition where $V_1 = V \setminus \Gamma(v_0)$ and $V_2 = \Gamma(v_0)$ (recall that $\Gamma(v_0)$ denotes the set of neighbors of v_0).

Therefore, with probability at least $1 - (1 - \epsilon)^s > 1 - \exp(-\epsilon s) > 2/3$, among the s sampled pairs $(u_1, w_1), \ldots, (u_s, w_s)$ there will be at least one pair (u_j, w_j) either in $E \setminus V_1 \times V_2$ or in $V_1 \times V_2 \setminus E$. In the former case either u_j and w_j both belong to V_1, and so the subgraph induced by u_j, w_j, and v_0 contains a single edge (u_j, w_j), or u_j and w_j both belong to V_2, and so the subgraph induced by u_j, w_j, and v_0 contains all three edges. In the latter case this subgraph contains a single edge (between v_0 and either u_j or w_j). For an illustration, see Figure 4.1.

The General Idea. As exemplified by the problem of testing whether a graph is a biclique, the high-level idea behind the design and analysis of algorithms that follows the "enforce-and-test" approach is roughly the following. The algorithm takes a sample from the tested object (e.g., a small random subgraph), and checks whether the sample

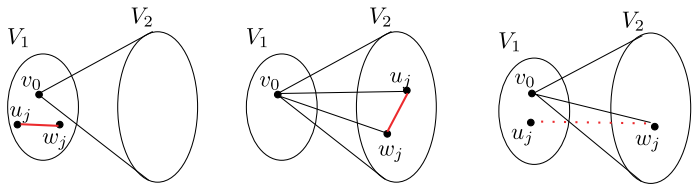

Fig. 4.1 Illustrations of the three cases in the analysis of the biclique tester. On the left is an illustration for the case that $(u_j, w_j) \in E \setminus V_1 \times V_2$ and $u_j, w_j \in V_1$; in the middle is an illustration for the case that $(u_j, w_j) \in E \setminus V_1 \times V_2$ and $u_j, w_j \in V_2$; and on the right is an illustration for the case that $(u_j, w_j) \in V_1 \times V_2 \setminus E$, and $u_j \in V_1$, $w_j \in V_2$. In the last case the "missing edge" between u_j and w_j is marked by a dotted line.

has a particular property, which is possibly, but not necessarily, the property tested. The analysis views the sample as consisting of two parts. The first part is the "enforcing" part, and the second is the "testing" part. The goal of the enforcing part is to implicitly induce certain constraints over the structure of the (yet unseen portion) of the object. The constraints are such that if the object is far from having the property, then with high probability over the choice of the testing part it will contain evidence that (together with the enforce part) "proves" that the object does not have the tested property.

4.2 Testing Bipartiteness in the Dense-Graphs Model

Recall that a graph $G = (V, E)$ is *bipartite* if there exists a partition (V_1, V_2) of the vertices where there are no edges (u, w) such that $u, w \in V_1$ or $u, w \in V_2$. We say in such a case that the partition is *bipartite*. If a partition (V_1, V_2) is not bipartite, then we shall say that the edges $(u, w) \in E$ such that $u, w \in V_1$ or $u, w \in V_2$ are *violating edges* with respect to (V_1, V_2). Recall that we can decide (exactly) whether a graph is bipartite in linear time by running a Breadth First Search (BFS). By the definition of the dense-graphs model, a graph G is ϵ-far from (being) bipartite in this model if (and only if) it is necessary to remove more than ϵn^2 edges to make it bipartite.

The algorithm is very simple and is given in Figure 4.2. Note that the number of queries performed is *independent* of the size of the graph, and only depends (polynomially) on $1/\epsilon$. Clearly, if the graph G is

Algorithm 4.1: Bipartiteness Test

1. *Take a sample S of $\Theta\left(\epsilon^{-2} \cdot \log(1/\epsilon)\right)$ vertices, selected uniformly at random.*
2. *Ask vertex-pair queries for all pairs in the sample, thus obtaining the induced subgraph G_S.*
3. *Run a Breadth First Search (BFS) on G_S: if it is bipartite then* accept*, otherwise,* reject*.*

Fig. 4.2 The bipartiteness testing algorithm (for dense graphs).

bipartite then it is accepted by Algorithm 4.1 with probability 1, and when the algorithm rejects a graph it provides evidence "against" the graph in the form of a small subgraph (G_S) that is not bipartite. Hence, from this point on assume G is ϵ-far from being bipartite, and we will show that it is rejected with probability at least 2/3.

If G is ϵ-far from bipartite then this means that *for every* partition (V_1, V_2) of V, there are more than ϵn^2 violating edges with respect to (V_1, V_2). Consider the following initial attempt of analyzing the algorithm: If we consider a single partition (V_1, V_2) (that has more than ϵn^2 violating edges, since the graph is ϵ-far from bipartite), then it is easy to see that a sample of $s = \Theta\left(\epsilon^{-1} \cdot \log(1/\delta)\right)$ vertices will "hit" the two end-points of such an edge (i.e., that is violating with respect to (V_1, V_2)) with probability at least $1 - \delta$. The natural idea would be to take a union bound over all partitions. The problem is that there are 2^n possible partitions and so in order for the union bound to work we would have to take $\delta < 2^{-n}$, implying that the sample should have size linear in n.

Instead, we shall think of the sample as consisting of two disjoint parts, U (the "enforce" part) and W (the "test" part). The intuition is that in some sense U will introduce constraints that will effectively reduce the number of "relevant" partitions of V to a much smaller number than 2^n, and then W will be used to "test" only them. We let $|U| = \Theta\left(\epsilon^{-1} \cdot \log(1/\epsilon)\right)$ and $|W| = \Theta\left(\epsilon^{-1} \cdot \log 2^{|U|}\right) = \Theta\left(\epsilon^{-2} \cdot \log(1/\epsilon)\right)$.

We first introduce a couple of additional definitions:

Definition 4.1. For any fixed partition (U_1, U_2) of U, we shall say that W is not compatible with (U_1, U_2) if there is no partition (W_1, W_2) of W such that $(U_1 \cup W_1, U_2 \cup W_2)$ is a bipartite partition.

We would like to show that (since G is ϵ-far from bipartite), with high probability over the choice of U and W, no matter how we partition U into (U_1, U_2), the subset W will not be compatible with (U_1, U_2) (implying that there is no bipartite partition of both U and W, which causes the algorithm to reject).

Definition 4.2. Let (U_1, U_2) be a (bipartite) partition of U. We shall say that a vertex w is a witness against (U_1, U_2) if there exist $u_1 \in U_1$ and $u_2 \in U_2$ such that $(w, u_1), (w, u_2) \in E$. We shall say that a pair w_1 and w_2 are witnesses against (U_1, U_2) if $(w_1, w_2) \in E$ and there exist $u_1, u_2 \in U$ such that $u_1, u_2 \in U_1$ or $u_1, u_2 \in U_2$ and (w_1, u_1), $(w_2, u_2) \in E$.

For an illustration of the notion of witnesses, see Figure 4.3.

Observation: If W contains a vertex w that is a witness against (U_2, U_2) or a pair of vertices w_1 and w_2 that are witnesses against (U_1, U_2) then W is not compatible with (U_1, U_2). Hence, we would like to show that with high probability over U and W, there are witnesses in W against *every* partition of U.

Simplifying assumption: We first continue the analysis under the assumption that U is such that *every* $v \in V$ has at least one neighbor

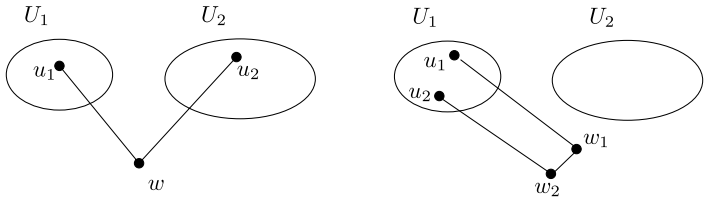

Fig. 4.3 An illustration of a witness w, and a pair of witnesses w_1, w_2, both with respect to the partition (U_1, U_2) of U.

in U. (We later remove this assumption.) Under this assumption, given a bipartite partition (U_1, U_2) of U, we define a partition of all of V. For $u \in U$ we put u in V_1 if $u \in U_1$ and we put u in V_2 if $u \in U_2$. For $v \in V \setminus U$ (that is, almost all vertices are considered here) if v has a neighbor in U_1 then we put v in V_2 and otherwise (it has a neighbor in U_2), then we put it in V_1. For an illustration, see Figure 4.4.

Now, each one of these at most $2^{|U|}$ partitions of V contains more than ϵn^2 violating edges. Since (U_1, U_2) is bipartite, and we put each vertex in $V \setminus U$ opposite its neighbor, these edges are of the form $(w_1, w_2) \in E$ where w_1 and w_2 both have a neighbor in U_1 or both have a neighbor in U_2, or they are of the form (w, u_2) where $u_2 \in U_2$ and w has a neighbor $u_1 \in U_1$ (so it was put in V_2). But this exactly coincides with our definition of witnesses against (U_1, U_2). Therefore, if we catch such a vertex (pair), then W in not compatible with (U_1, U_2). For simplicity of the analysis, even in the case that w is a witness because it was put in V_1 but it has a neighbor $u_2 \in U_2$, we shall think of (u_2, w) as a pair of witnesses, and so it won't be considered sufficient that $w \in W$ but we'll require that $u_2, w \in W$.

We shall think of the uniform sample W as a sample over uniformly selected pairs of vertices. Since the probability that we catch a pair of

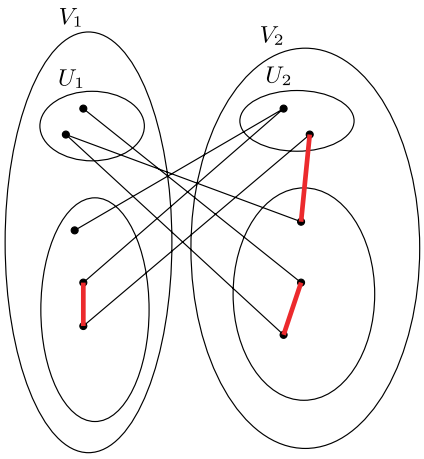

Fig. 4.4 An illustration of the partition of V that is defined based on (U_1, U_2) when we make the simplifying assumption that every vertex in V has a neighbor in U. Violating edges (which correspond to witnesses) are marked by bold lines.

witnesses in a single trial is more than $\frac{\epsilon n^2}{n^2} = \epsilon$, the probability that we *don't* catch any pair of witnesses in W is at most $(1-\epsilon)^{|W|/2}$. If we take $|W| = \Theta(|U|/\epsilon)$ then this is less than $(1/6) \cdot 2^{-|U|}$. By a union bound over all two-way partitions of U, the probability that for some (U_1, U_2), we have that W is compatible with (U_1, U_2) is hence at most $1/3$. In other words, with probability at least $5/6$ there is no bipartite partition of $U \cup W$.

It remains to remove the assumption that every vertex in V has a neighbor in U.

Definition 4.3. We say that a vertex in V has high degree if its degree is at least $(\epsilon/4)n$. Otherwise it has low degree.

Lemma 4.1. With probability at least $5/6$ over the choice of $(4/\epsilon) \cdot \log(24/\epsilon)$ vertices (denoted U), all but at most $(\epsilon/4)n$ of the high degree vertices in V have a neighbor in U.

We prove this lemma momentarily, but first show how to modify the argument based on the lemma. Assume U is as stated in the lemma (where we later take into account the probability of $1/6$ that this is not the case). Then, given a partition (U_1, U_2) of U, we define a partition of all vertices similarly to what we did before. In particular, the vertices in U and their neighbors are partitioned as before. All remaining vertices, which do not have a neighbor in U and whose set is denoted R, are put arbitrarily in V_1. For an illustration, see Figure 4.5. Once again, for every bipartite partition (U_1, U_2), the partition of V just defined contains more than ϵn^2 violating edges. Now, some of these violating edges might not correspond to witnesses. In particular, some of these edges might be incident to vertices in R. However, the total number of edges that are incident to vertices in R is at most $n \cdot (\epsilon/4)n + (\epsilon/4)n \cdot n = (\epsilon/2)n^2$. Hence, there are at least $(\epsilon/2)n^2$ violating edges that correspond to witnesses, and we shall catch one with high constant probability.

More precisely, if $|W| = \Theta(\epsilon^{-1} \cdot |U|) = \Theta(\epsilon^{-2} \cdot \log(1/\epsilon))$, then, conditioned on U being as in Lemma 4.1, with probability at least $5/6$

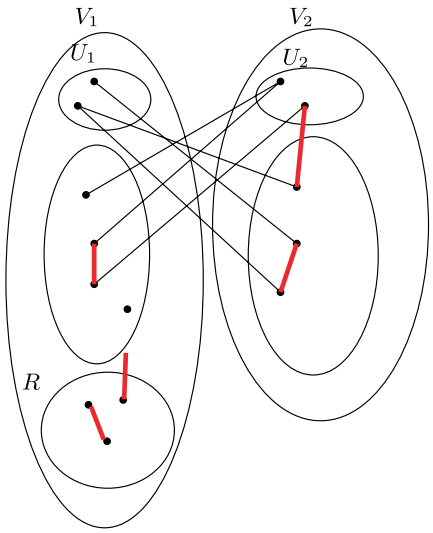

Fig. 4.5 An illustration of the partition of V that is defined based on (U_1, U_2) when we remove the simplifying assumption that every vertex in V has a neighbor in U. Violating edges that are incident to R are marked by dashed lines while violating edges which correspond to witnesses are marked by bold lines.

over the choice of W, there is a pair of witnesses in W against every partition of U. The probability that either U is not as in Lemma 4.1, or W does not include a witness against some partition of U, is at most 1/3. It follows that with probability at least 2/3 (over the choice of $S = U \cup W$) the algorithm rejects (since there is no bipartite partition of S). It remains to prove Lemma 4.1.

Proof of Lemma 4.1. Consider any fixed high degree vertex v. The probability that U does not contain any neighbor of v is at most $(1 - (\epsilon/4))^{|U|} < \epsilon/24$. Therefore, the expected fraction of high degree vertices in V that do not have a neighbor in U is at most $\epsilon/24$. By Markov's inequality, the probability that there is more than an $\epsilon/4$ fraction of such vertices in V (that is, more than six times the expected value), is at most 1/6. □

4.2.1 Reducing the Number of Queries

We first observe that by the foregoing analysis, we can modify the algorithm (see Figure 4.6) so as to reduce the query complexity and

Algorithm 4.2: Bipartiteness Test (Version II)

1. *Take a sample U of $\Theta\left(\epsilon^{-1} \cdot \log(1/\epsilon)\right)$ vertices, $u_1, \ldots, u_s$, selected uniformly, independently, at random, and a sample W of $\Theta\left(\epsilon^{-2} \cdot \log(1/\epsilon)\right)$ vertices $w_1, \ldots, w_t$ selected uniformly, independently, at random.*
2. *Ask vertex-pair queries for all pairs $(u_i, u_j) \in U \times U$, $(u_i, w_k) \in U \times W$ and for all pairs $(w_{2\ell-1}, w_{2\ell})$ where $1 \leq \ell \leq \lfloor t/2 \rfloor$. Let the subgraph obtained be denoted H.*
3. *Run a Breadth First Search (BFS) on H: if it is bipartite then* accept*, otherwise,* reject*.*

Fig. 4.6 The bipartiteness testing algorithm (version II).

running time to $\Theta(\epsilon^{-3} \cdot \log^2(1/\epsilon))$. The basic observation is that we can actually partition the sample into two parts, U and W (as described in the analysis), and we don't need to perform all vertex-pair queries on pairs of vertices in W, but rather only on a linear (in $|W|$) number of disjoint pairs.

A more sophisticated analysis of Alon and Krivelevich [10] shows that a sample of vertices having size $\Theta(\epsilon^{-1} \cdot \log(1/\epsilon))$ suffices for the original algorithm (Algorithm 4.1), so that the number of queries performed is $\Theta(\epsilon^{-2} \cdot \log^2(1/\epsilon))$. The result of Alon and Krivelevich is optimal in terms of the *number of vertices* that the tester inspects [10]. A natural question addressed by Bogdanov and Trevisan [37] is whether $\Omega(\epsilon^{-2})$ queries are necessary. Bogdanov and Trevisan showed that $\Omega(\epsilon^{-2})$ queries are indeed necessary for any *non-adaptive* tester. For *adaptive* testers they showed that $\Omega(\epsilon^{-3/2})$ queries are necessary.[1] This result still left open the question whether an adaptive tester can indeed have query complexity that is $o(\epsilon^{-2})$, and possibly even $O(\epsilon^{-3/2})$. This question was answered affirmatively in [79] for the case that (almost) all vertices have degree $O(\epsilon n)$, where the lower bounds of [37] hold

[1] A non-adaptive tester must choose all its queries in advance whereas an adaptive tester may choose its queries based on answers to previous queries. In the dense-graphs model, for any fixed property the gap in the query complexity between adaptive and non-adaptive testing algorithms is at most quadratic [7, 78].

under this condition. The algorithm of [79] works by importing ideas from testing in the bounded-degree model to the dense-graphs model. In [79] it was also shown that $O(\epsilon^{-3/2})$ queries are sufficient when (almost) all vertices have degree $\Omega(\epsilon^{1/2}n)$. The general question regarding the exact complexity of adaptively testing bipartiteness for general graphs (in the dense-graphs model) is still open. We note that the power of adaptivity in the dense-graphs model was further studied in [77].

4.2.2 Constructing an Approximately Good Bipartition

One interesting implication of the analysis of the bipartiteness tester is that if the graph is indeed bipartite then it is possible to use the tester to obtain (with high constant probability) auxiliary information that lets us construct an approximately good bipartition in time linear in n. To be precise, we say that a partition (V_1, V_2) is *ϵ-good* if there are at most ϵn^2 violating edges in G with respect to (V_1, V_2). Now suppose G is bipartite and we run Algorithm 4.1, where we view the sample as consisting of two parts: U and W (or we run Algorithm 4.2 for which the partition of the sample is explicit).

As shown in Lemma 4.1, with high constant probability, all but at most $(\epsilon/4)n$ of the high degree vertices in V have a neighbor in U (where we said that a vertex has high degree if it has at least $(\epsilon/4)n$ neighbors). We shall say in such a case that U is an $(\epsilon/4)$-*dominating-set*. Assume from this point on that U is indeed an $(\epsilon/4)$-dominating set (where we take into account the probability that this is not the case in our failure probability).

For each bipartite partition (U_1, U_2) of U, consider the partition $(U_1 \cup (V \setminus \Gamma(U_1)), U_2 \cup \Gamma(U_1))$ of V (as defined in the analysis of the tester). By the argument used to prove the correctness of the tester in the case that the graph is ϵ-far from being bipartite, we have the following. With high constant probability over the choice of W, for every (U_1, U_2) such that $(U_1 \cup (V \setminus \Gamma(U_1)), U_2 \cup \Gamma(U_1))$ is not ϵ-good, there will be no bipartite partition $(U_1 \cup W_1, U_2 \cup W_2)$ of the sample. Assume this is in fact the case (where we add the probability that this is not the case to our failure probability). Since G is bipartite, the BFS executed by the tester will find a bipartite partition $(U_1 \cup W_1, U_2 \cup$

W_2), implying that the partition $(U_1 \cup (V \setminus \Gamma(U_1)), U_2 \cup \Gamma(U_1))$ must be ϵ-good.

We can hence use the partition (U_1, U_2) to determine, for every vertex $v \in V$ to which side it belongs in the ϵ-good partition $(U_1 \cup (V \setminus \Gamma(U_1)), U_2 \cup \Gamma(U_1))$ by simply performing all queries between v and $u \in U$.

4.3 Other Applications of the Enforce-and-Test Approach

There are also similar (though somewhat more complex) analyses of algorithms in the dense-graphs model for testing k-colorability, ρ-Clique (having a clique of size ρN), ρ-Cut (having a cut of size at least ρN^2), and in general for the family of all *partition properties* [72]. Namely, these properties are defined by upper and lower bounds on the sizes of some constant number k of parts, and upper and lower bounds on the edge-densities between these parts and within each part. The number of queries performed in all cases is polynomial in $1/\epsilon$ and exponential in k. The time-complexity is exponential in $1/\epsilon$, but this is inevitable (assuming $\mathcal{P} \neq \mathcal{NP}$) since partition problems include $\mathcal{NP}$-hard problems.

As in the case of bipartiteness, for all these properties, when the graph has the desired property, with high probability the testing algorithm outputs some auxiliary information that lets us construct, in $\text{poly}(1/\epsilon) \cdot N$ time, a partition that approximately obeys the property (recall that the number of parts, k, is assumed to be a constant). For example, for ρ-Clique, the algorithm will find a subset of vertices of size ρN, such that at most ϵN^2 edges need to be added so that it becomes a clique. In the case of ρ-Cut, the algorithm will construct a partition with at least $(\rho - \epsilon)N^2$ crossing edges (so that if we run the algorithm with $\epsilon = \gamma \cdot \rho$, we get a cut of size at least $(1 - \gamma)$ times the optimal). As in the case of bipartiteness, the basic idea is that the partition of the sample that caused the algorithm to accept is used to partition the whole graph.

Returning to the property of bipartiteness, we observe that the construction algorithm for ρ-Cut (which constructs a partition with at least $(\rho - \epsilon)N^2$ crossing edges) can be applied to get an ϵ-good bipartition even when the graph is not bipartite but rather is close (say, $\epsilon/2$-close)

to being bipartite. More generally, the construction algorithm for the general partition problem can be used to construct approximately good partitions even when the graph does not have a corresponding "perfect" partition.

Other property testing problems that are solved using the enforce-and-test approach include testing metric properties [113] and testing of clustering [5]. In these cases it also holds that the testing algorithms can be extended to solve approximate versions of the corresponding search problems (e.g., finding good clusterings of all but a small fraction of the points). As we discuss in Section 8, the analysis of the bipartiteness tester in the bounded-degree model can also be viewed as following the enforce-and-test approach, though this is perhaps less evident than in other cases.

The enforce-and-test approach is also related to a framework introduced by Czumaj and Sohler [46], in which the notion of *Abstract Combinatorial Programs* is defined, and based on these programs, several (old and new) property testing algorithms are derived.

5

Testing by Implicit Learning

In this subsection we describe the results of Diakonikolas et al. [50]. They present a general method for testing whether a function has a concise representation (e.g., an s-term DNF or an s-node decision tree). Here we mostly focus on the Boolean case, though the technique in [50] extends to general domains and ranges. The query complexity is always polynomial in the size parameter s, and is quadratic in $1/\epsilon$. The running time grows exponentially[1] with s.

The approach taken in [50] uses ideas from the junta testing algorithm(s) of Fischer et al. [61] and ideas from learning theory (a k-junta is a function that depends on at most k variables). As noted in the introduction, it was observed in [72] that if we have a proper learning algorithm for a class of functions $\mathcal{F}$, then we can use it as a subroutine to test the property of membership in $\mathcal{F}$. However, for all the properties considered in [50], proper learning requires a number of queries that grow at least logarithmically with the number of variables, n. Therefore, a more sophisticated approach is required in order to obtain algorithms whose query complexity does not depend on n.

[1] In recent work [51] the dependence of the running time on s in the case of s-term polynomials over $GF(2)$ was reduced to polynomial.

The first key observation behind the general algorithm of [50] is that many classes of functions that have a concise representation are "well-approximated" by small juntas that belong to the class. That is, every function in the class is close to some other function in the class that is a small junta. For example, for any choice of δ, every s-term DNF is δ-close to an s-term DNF that depends only on at most $s\log(s/\delta)$ variables. This is true since by removing a term that has more than $\log(s/\delta)$ variables, the error incurred is at most δ/s (recall that the underlying distribution is uniform).

Given this observation, the algorithm works roughly as follows. It first finds a collection of subsets of variables such that each subset contains a single variable on which the function depends (in a non-negligible manner). If the number of such subsets is larger than some threshold k, then the algorithm rejects. Otherwise, the algorithm creates a sample of labeled examples, where the examples are points in $\{0,1\}^k$, that is, over the variables that the function depends on. It is important to stress that the algorithm creates this sample *without* actually identifying the relevant variables. Finally, the algorithm checks whether there exists a function of the appropriate form over the small set of variables that is consistent with the sample. Roughly speaking, the algorithm works by attempting to learn the *structure* of the junta that f is close to (without actually identifying its variables). This is the essence of the idea of "testing by implicit learning".

Since the results of [51] build on testing juntas, we first describe an algorithm for testing whether a function is a small junta [61].

5.1 A Building Block: Testing Juntas

We start with a formal definition.

Definition 5.1 (Juntas). A function $f : \{0,1\}^n \to \{0,1\}$ is a k-junta for an integer $k \leq n$ if f is a function of at most k variables. Namely, there exists a set $J \subseteq [n]$ where $|J| \leq k$ such that $f(x) = f(y)$ for every $x, y \in \{0,1\}^n$ that satisfy $x_i = y_i$ for each $i \in J$. We say in such a case that J dominates the function f.

The main result of [61] is stated next.

Theorem 5.1. For every fixed k, the property of being a k-junta is testable using $\text{poly}(k)/\epsilon$ queries.

Fischer et al. [61] establish Theorem 5.1 by describing and analyzing several algorithms. The algorithms vary in the polynomial dependence on k (ranging between[2] $\tilde{O}(k^4)$ to $\tilde{O}(k^2)$), and in two properties: whether the algorithm is non-adaptive or adaptive (that is, queries may depend on answers to previous queries), and whether it is has one-sided error or two-sided error. They also prove a lower bound of $\tilde{\Omega}(\sqrt{k})$ for non-adaptive algorithms, which was later improved to an $\Omega(k)$ lower bound for adaptive algorithms by Chockler and Guttreund [43], thus establishing that a polynomial dependence on k is necessary. While we focus here on the domain $\{0,1\}^n$ and on the case that the underlying distribution is uniform, Theorem 5.1 holds for other domains and when the underlying distribution is a product distribution.

In order to describe and analyze the testing algorithm, we first introduce some definitions and notations. The domain of the functions we consider is always $\{0,1\}^n$ and it will be convenient to assume that the range of the functions is $\{1,-1\} = \{(-1)^0,(-1)^1\}$ (rather than $\{0,1\}$).

Partial Assignments. For a subset $S \subseteq [n]$ we denote by $\mathcal{A}(S)$ the set of *partial assignments* to the variables x_i where $i \in S$. Each $w \in \mathcal{A}(S)$ can be viewed as a string in $\{0,1,*\}^n$, where for every $i \in S$, $w_i \in \{0,1\}$, and for every $i \notin S$, $w_i = *$. In particular, $\mathcal{A}([n]) = \{0,1\}^n$. For two *disjoint* subsets $S, S' \subset [n]$, and for partial assignments $w \in \mathcal{A}(S)$ and $w' \in \mathcal{A}(S')$, we let $w \sqcup w'$ denote the partial assignment $z \in \mathcal{A}(S \cup S')$ defined by: $z_i = w_i$, for every $i \in S$, $z_i = w'_i$ for every $i \in S'$, and $z_i = w_i = w'_i = *$ for every $i \in [n] \setminus \{S \cup S'\}$. In particular, we shall consider the case $S' = [n] \setminus S$, so that $w \sqcup w' \in \{0,1\}^n$ is a complete assignment (and $f(w \sqcup w')$ is well defined). Finally, for $x \in \{0,1\}^n$ and $S \subseteq [n]$, we let $x_{|S}$ denote the partial assignment $w \in \mathcal{A}(S)$ defined by

[2] The notation $\tilde{O}(g(t))$ for a function g of a parameter t means $O(g(t) \cdot \text{polylog}(g(t)))$.

$w_i = x_i$ for every $i \in S$, and $w_i = *$ for every $i \notin S$. For the sake of conciseness, we shall use $\overline{S}$ as a shorthand for $[n] \setminus S$.

Variation. For a function $f : \{0,1\}^n \to \{1,-1\}$ and a subset $S \subset [n]$, we define the *variation* of f on S (or the variation of S with respect to f), denoted $\mathrm{Vr}_f(S)$, as the probability, taken over a uniform choice of $w \in \mathcal{A}(\overline{S})$ and $z_1, z_2 \in \mathcal{A}(S)$, that $f(w \sqcup z_1) \neq f(w \sqcup z_2)$. That is[3]:

$$\mathrm{Vr}_f(S) \stackrel{\mathrm{def}}{=} \mathrm{Pr}_{w \in \mathcal{A}(\overline{S}), z_1, z_2 \in \mathcal{A}(S)}[f(w \sqcup z_1) \neq f(w \sqcup z_2)]. \tag{5.1}$$

The simple but important observation is that if f does not depend on any variable x_i where $i \in S$, then $\mathrm{Vr}_f(S) = 0$, and otherwise it must be non-zero (though possibly small). One useful property of the variation is that it is *monotone*. Namely, for any two subsets $S, T \subseteq [n]$,

$$\mathrm{Vr}_f(S) \leq \mathrm{Vr}_f(S \cup T). \tag{5.2}$$

Another property is that it is *subadditive*, that is, for any two subsets $S, T \subseteq [n]$,

$$\mathrm{Vr}_f(S \cup T) \leq \mathrm{Vr}_f(S) + \mathrm{Vr}_f(T). \tag{5.3}$$

As we show next, the variation can also be used to bound the distance that a function has to being a k-junta.

Lemma 5.2. Let $f : \{0,1\}^n \to \{1,-1\}$ and let $J \subset [n]$ be such that $|J| \leq k$ and $\mathrm{Vr}_f(\overline{J}) \leq \epsilon$. Then there exists a k-junta g that is dominated by J and is such that $\mathrm{dist}(f,g) \leq \epsilon$.

Proof. We define the function g as follows: for each $x \in \{0,1\}^n$ let

$$g(x) \stackrel{\mathrm{def}}{=} \mathrm{majority}_{u \in \mathcal{A}(\overline{J})}\{f(x_{|J} \sqcup u)\}. \tag{5.4}$$

That is, for each $w \in \mathcal{A}(J)$, the function g has the same value on all strings $x \in \{0,1\}^n = \mathcal{A}([n])$ such that $x_{|J} = w$, and this value is simply the majority value of the function f taken over all strings of this form.

[3] We note that in [61] a more general definition is given (for real-valued functions). For the sake of simplicity we give only the special case of $\{1,-1\}$-valued function, and we slightly modify the definition by removing a factor of 2.

We are interested in showing that $\Pr[f(x) = g(x)] \geq 1 - \epsilon$. That is,

$$\Pr_{w \in \mathcal{A}(J), z \in \mathcal{A}(\overline{J})}[f(w \sqcup z) = \text{majority}_{u \in \mathcal{A}(\overline{J})}\{f(w \sqcup u)\}] \geq 1 - \epsilon. \quad (5.5)$$

Similarly to what was shown in the proof of Claim 3.5, this probability is lower bounded by $\Pr_{w \in \mathcal{A}(J), z_1, z_2 \in \mathcal{A}(\overline{J})}[f(w \sqcup z_1) = f(w \sqcup z_2)]$, which is simply $1 - \text{Vr}_f(\overline{J}) \geq 1 - \epsilon$. □

5.1.1 An Algorithm for Testing Juntas

Here we describe an algorithm for testing k-juntas, which has one-sided error, is non-adaptive, and has query complexity $\tilde{O}(k^4/\epsilon)$. In [61] there are actually two algorithms with this complexity. We have chosen to describe the one on which the more efficient algorithms (mentioned previously) are based, and which also plays a role in the results described in Section 5.2. We assume that $k > 1$, since 1-juntas are simply singletons, for which we already know there is a testing algorithm. The idea behind the algorithm is simple: It randomly partitions the variables into $\Theta(k^2)$ disjoint subsets. For each subset it checks whether it contains any variable on which the function depends. If there are more than k subsets for which such a dependence is detected, then the algorithm rejects. Otherwise it accepts. The algorithm is given in Figure 5.1.

Theorem 5.3. Algorithm 5.1 is a one-sided error testing algorithm for k-juntas. Its query complexity is $O(k^4 \log k/\epsilon)$.

The bound on the query complexity of the algorithm is $O(r \cdot h) = O(k^4 \log k/\epsilon)$. The dependence test declares that f depends on a set S_j only if it has found evidence of such a dependence and the algorithm rejects only if there are more than k disjoint sets for which such evidence is found. Therefore, the algorithm never rejects a k-junta. We next turn to proving that if f is ϵ-far from being a k-junta, then it is rejected with probability at least $2/3$.

Let $\tau = (\log(k+1) + 4)/h$ and note that by the definition of h, $\tau \leq \epsilon/(4r)$ (recall that r is the number of sets in the random partition selected by the algorithm and h is the number of applications of the dependence test). Define $J = J_\tau(f) \stackrel{\text{def}}{=} \{i \in [n] : \text{Vr}_f(\{i\}) > \tau\}$. Thus $\overline{J}$

Algorithm 5.1: *k*-Junta Test

1. *For $r = \Theta(k^2)$ select a random partition $\{S_1, \ldots, S_r\}$ of $[n]$ by assigning each $i \in [n]$ to a set S_j with equal probability.*
2. *For each $j \in [r]$, perform the following* dependence test *at most*

$$h = 4(\log(k+1) + 4)r/\epsilon = \Theta(k^2 \log k/\epsilon)$$

times:

   - *Uniformly and independently select $w \in \mathcal{A}(\overline{S}_j)$ and $z_1, z_2 \in \mathcal{A}(S_j)$. If $f(w \sqcup z_1) \neq f(w \sqcup z_2)$ then declare that f depends on at least one variable in S_j (and continue to $j+1$).*

3. *If f is found to depend on more than k subsets S_j, then output* reject*, otherwise output* accept*.*

Fig. 5.1 Testing algorithm for juntas.

consists of all i such that $\mathrm{Vr}_f(\{i\}) \leq \tau$. We claim:

Lemma 5.4. If $\mathrm{Vr}_f(\overline{J}) > \epsilon$ then Algorithm 5.1 rejects with probability at least 2/3.

Lemma 5.5. If $|J| > k$ then Algorithm 5.1 rejects with probability at least 2/3.

By Lemma 5.2, if f is ϵ-far from any k-junta, then either $\mathrm{Vr}_f(\overline{J}) > \epsilon$ or $|J| > k$ (or both). By Lemmas 5.4 and 5.5 this implies that the algorithm rejects with probability at least 2/3. Both lemmas rely on the following claim regarding the dependence test.

Claim 5.6. For any subset S_j, if $\mathrm{Vr}_f(S_j) \geq \tau$, then the probability that Step 2 in Algorithm 5.1 declares that f depends on at least one variable in S_j is at least $1 - 1/(e^4(k+1))$.

Proof. By the definition of the dependence test, the probability that a single application of the test finds evidence that f depends on S_j is exactly $\mathrm{Vr}_f(S_j)$. Since $\tau = (\log(k+1)+4)/h$, if $\mathrm{Vr}_f(S_j) \geq \tau$, the probability that the test fails to find such evidence in h independent applications is at most $(1-\tau)^h < \exp(-\tau h) < e^{-4}/(k+1)$, as claimed. □

We now prove Lemma 5.5, which is quite simple, and later sketch the proof of Lemma 5.4, which is more complex.

Proof of Lemma 5.5. First observe that if $|J| > k$, then the probability, over the choice of the partition, that there are fewer than $k+1$ sets S_j such that $S_j \cap J \neq \emptyset$, is $O(k^2/r)$. Since $r = ck^2$ where c is a constant, for an appropriate choice of c, this probability is at most $1/6$. Assume from this point on that are at least $k+1$ sets S_j such that $S_j \cap J \neq \emptyset$ (where we later take into account the probability that this is not the case).

By the monotonicity of the variation (Equation (5.2)) and since $\mathrm{Vr}_f(\{i\}) > \tau$ for each $i \in J$, if a set S_j satisfies $S_j \cap J \neq \emptyset$, then $\mathrm{Vr}_f(S_j) \geq \tau$. By Claim 5.6 and the union bound, the probability that the algorithm finds evidence of dependence for fewer than $k+1$ sets is less than $1/6$. Summing this probability with the probability that there are fewer than $k+1$ sets S_j such that $S_j \cap J \neq \emptyset$, the lemma follows. □

Proof Sketch of Lemma 5.4. By the premise of the lemma, $\mathrm{Vr}_f(\overline{J}) > \epsilon$. Since the variation is subadditive (Equation (5.3)), for any partition $\{S_1, \ldots, S_r\}$, $\sum_{j=1}^{r} \mathrm{Vr}_f(S_j \cap \overline{J}) > \epsilon$. Since the subsets in the partition are equally distributed, we have that for each fixed choice of j, $\mathbb{E}[\mathrm{Vr}_f(S_j \cap \overline{J})] > \epsilon/r$. The main technical claim (whose proof we omit) is that with high probability $\mathrm{Vr}_f(S_j \cap \overline{J})$ is not much smaller than its expected value. To be precise, for each fixed choice of j, with probability at least $3/4$ (over the random choice of the partition), $\mathrm{Vr}_f(S_j \cap \overline{J}) \geq \epsilon/(4r)$. Recall that by the definition of τ (and of h as a function of r), we have that $\epsilon/(4r) \geq \tau$.

Using this claim, we now show how Lemma 5.4 follows. Recall that by monotonicity of the variation, $\mathrm{Vr}_f(S_j) \geq \mathrm{Vr}_f(S_j \cap \overline{J})$. We shall say

that a set S_j is *detectable*, if $\mathrm{Vr}_f(S_j) \geq \tau$. Thus, the expected number of detectable subsets is at least $(3/4)r$. Let α denote the probability that there are fewer than $r/8$ detectable subsets. Then $\alpha \leq 2/7$ (as the expected number of detectable subsets is at most $\alpha(r/8)+(1-\alpha)r$). Equivalently, with probability at least $5/7$, there are at least $r/8 = \Omega(k^2) > k+1$ detectable subsets. Conditioned on this event, by Claim 5.6 (and the union bound), the probability that the algorithm detects dependence for fewer than $k+1$ subsets is at most $1/e^4$. Adding this to the probability that there are fewer than $k+1$ detectable sets, the lemma follows. □

5.1.2 More Efficient Algorithms

By allowing the algorithm to be adaptive, it is possible to reduce the query complexity to $O(k^3 \log^3(k+1)/\epsilon)$, and by allowing the algorithm to have two-sided error, it can be reduced to $O(k^2 \log^3(k+1)/\epsilon)$ (without the need for adaptivity). Here we give the high-level ideas for the more efficient algorithms.

Both algorithms start by partitioning the variables into $r = \Theta(k^2)$ disjoint subsets $\{S_1, S_2, \ldots, S_r\}$ as done in Algorithm 5.1. The main idea used in the first improvement (the adaptive algorithm) is to speed up the detection of subsets S_j that have non-negligible variation $\mathrm{Vr}_f(S_j)$, in the following manner of divide and conquer. Instead of applying the dependence test to each subset separately, it is applied to *blocks*, each of which is a union of several subsets. If f is not found to depend on a block, then all the variables in the block are declared to be "variation free". Otherwise (some dependence is detected), the algorithm partitions the block into two equally sized sub-blocks, and continues the search on them.

The two-sided error test also applies the dependence test to blocks of subsets, only the blocks are chosen differently and in particular, may overlap. The selection of blocks is done as follows. For $s = \Theta(k \log r) = \Theta(k \log k)$, the algorithm picks s random subsets of coordinates $I_1 \ldots, I_s \subseteq [r]$ of size k, independently, each by uniformly selecting (without repetitions) k elements of $[n]$. For each $1 \leq \ell \leq s$, block B_ℓ is defined as $B_\ell = \bigcup_{j \in I_\ell} S_j$. The dependence test is then applied h times

to each block (where h is as in Algorithm 5.1). For each subset S_j, the algorithm considers the blocks that contain it. The algorithm declares that f depends on S_j, if it found that f depends on all blocks that contain S_j. If there are more than k such subsets, or if f depends on at least a half of the blocks, the algorithm rejects, otherwise, it accepts. For further details of the analysis, see [61].

An almost optimal tester for juntas. In a recent work [34] Blais improves the dependence on k and gives an almost optimal one-sided error tester for k-juntas whose query complexity is $O(k/\epsilon + k \log k)$ (recall that there is a $\Omega(k)$ lower bound [43] for this problem). This algorithm works for functions with arbitrary finite product domains and arbitrary finite ranges, as well as with respect to any underlying product distribution.

5.2 The Algorithm for Testing by Implicit Learning

Before describing the algorithm in more detail, we give a central definition, and state the main theorem.

Definition 5.2. Let $\mathcal{F}$ be a class of Boolean functions over $\{0,1\}^n$. For $\delta > 0$, we say that a subclass $\mathcal{F}_\delta \subseteq \mathcal{F}$ is a (δ, k_δ)-approximator for $\mathcal{F}$ if the following two conditions hold.

- The subclass $\mathcal{F}_\delta$ is closed under permutations of the variables.
- For every function $f \in \mathcal{F}$ there is a function $f' \in \mathcal{F}_\delta$ such that $\text{dist}(f', f) \leq \delta$ and f' is a k_δ-junta.

Returning to the case that $\mathcal{F}$ is the class of s-term DNF functions, we may take $\mathcal{F}_\delta$ to be the subclass of $\mathcal{F}$ that consists of s-term DNF where each term is of size at most $\log(s/\delta)$, so that $k_\delta = s \log(s/\delta)$. Note that k_δ may be a function of other parameters determining the function class $\mathcal{F}$.

We shall use the notation $\widehat{\mathcal{F}}_\delta$ for the subset of functions in $\mathcal{F}_\delta$ that depend on the variables $x_1, \ldots, x_{k_\delta}$. Moreover, we shall view these functions as taking only k_δ arguments, that is, being over $\{0,1\}^{k_\delta}$.

We now state the main theorem of [50] (for the Boolean case).

Theorem 5.7. Let $\mathcal{F}$ be a class of Boolean functions over $\{0,1\}^n$. Suppose that for each choice of $\delta > 0$, $\widehat{\mathcal{F}}_\delta \subseteq \mathcal{F}$ is a (δ, k_δ) approximator for $\mathcal{F}$. Suppose also that for every $\epsilon > 0$ there is a δ satisfying

$$\delta \leq \frac{c\epsilon^2}{k_\delta^2 \cdot \log^2(k_\delta) \cdot \log^2 |\widehat{\mathcal{F}}_\delta| \cdot \log\log(k_\delta) \cdot \log(\log |\widehat{\mathcal{F}}_\delta|/\epsilon)}, \tag{5.6}$$

where c is a fixed constant. Let δ^* be the largest value of δ that satisfies Equation (5.6). Then there is a two-sided error testing algorithm for $\mathcal{F}$ that makes $\tilde{O}(k_{\delta^*}^2 \log^2 |\widehat{\mathcal{F}}_{\delta^*}|/\epsilon^2)$ queries.

We note that Theorem 5.7 extends to function classes with domain Ω^n and any range, in which case there is a dependence on $\log|\Omega|$ in Equation (5.6) and in the query complexity of the algorithm.

All results from [50] that appear in Table 5.1 are obtained by applying Theorem 5.7. In all these applications, k_δ grows logarithmically with $1/\delta$, and $\log|\widehat{\mathcal{F}}_\delta|$ is at most polynomial in k_δ. This ensures that Equation (5.6) can be satisfied. The most typical case in the applications is that for a class $\mathcal{F}$ defined by a size parameter s, we have that $k_\delta \leq \text{poly}(s)\log(1/\delta)$ and $\log|\widehat{\mathcal{F}}_\delta| \leq \text{poly}(s)\text{polylog}(1/\delta)$. This yields $\delta^* = \tilde{O}(\epsilon^2)/\text{poly}(s)$, and so the query complexity of the algorithm is $\text{poly}(s)/\tilde{\Theta}(\epsilon^2)$.

In particular, returning to the case that $\mathcal{F}$ is the class of s-term DNF, we have that $k_\delta = s\log(s/\delta)$ and $|\widehat{\mathcal{F}}_\delta| \leq (2s\log(s/\delta))^{s\log(s/\delta)}$. This

Table 5.1. Results obtained by [50] using the implicit learning approach.

Class of functions	Number of Queries
decision lists	$\tilde{O}(1/\epsilon^2)$
size-s decision trees, size-s branching programs	$\tilde{O}(s^4/\epsilon^2)$
s-term DNF, size-s Boolean formulas	$\Omega(\log s/\log\log s)$
s-sparse polynomials over $GF(2)$	$\tilde{O}(s^4/\epsilon^2)$, $\tilde{\Omega}(\sqrt{s})$
size-s Boolean circuits	$\tilde{O}(s^6/\epsilon^2)$
functions with Fourier degree $\leq d$	$\tilde{O}(2^{6d}/\epsilon^2)$, $\tilde{\Omega}(\sqrt{d})$
s-sparse polynomials over a general field F	$\tilde{O}((s\|F\|)^4/\epsilon^2)$, $\tilde{\Omega}(\sqrt{s})$ for $\|F\| = O(1)$
size-s algebraic circuits and computation trees over F	$\tilde{O}(s^4\|F\|^3/\epsilon^2)$

implies that $\delta^* = \tilde{O}(\epsilon^2/s^4)$, from which the upper bound of $\tilde{O}(s^4/\epsilon^2)$ on the query complexity follows. As another example, consider the case that $\mathcal{F}$ is the class of all decision lists. Then, for every δ, if we let $\widehat{\mathcal{F}}_\delta$ be the subclass of decision lists with length $\log(1/\delta)$, and we set $k_\delta = \log(1/\delta)$, then $\widehat{\mathcal{F}}_\delta$ is a (δ, k_δ)-approximation for $\mathcal{F}$. Since $|\widehat{\mathcal{F}}_\delta| \leq 2 \cdot 4^{\log(1/\delta)}(\log(1/\delta))!$, we get that $\delta^* = \tilde{O}(\epsilon^2)$, from which the bound of $\tilde{O}(1/\epsilon^2)$ on the query complexity follows.

The algorithm. The testing algorithm consists of three procedures. The first procedure, named Identify-Critical-Subsets, is a slight variant of the two-sided error junta test of [61] (described in Section 5.1.2). This variant is executed with $k = k_{\delta^*}$, where δ^* is as defined in Theorem 5.7 and with slightly larger constants than the original [61] algorithm. The main modification is that instead of returning accept in the case of success, the procedure returns the at most k_{δ^*} subsets of variables among $S_1, \ldots S_r$ that the function f was found to depend on by the test. In the case of failure, it outputs reject like the two-sided error junta test.

The analysis of the two-sided error test can be slightly modified so as to ensure the following. If $f \in \mathcal{F}$, so that it is δ^*-close to a k_{δ^*}-junta $f' \in \mathcal{F}_{\delta^*}$, then with high probability, Identify-Critical-Subsets completes successfully and outputs $\ell \leq k_{\delta^*}$ subsets of variables $S_{i_1}, \ldots S_{i_\ell}$. On the other hand, it is still true that if f is far from any k_{δ^*}-junta, then Identify-Critical-Subsets outputs reject with high probability. Moreover, if f is such that with probability at least $1/3$ the procedure completes successfully and outputs $\ell \leq k_{\delta^*}$ subsets $S_{i_1}, \ldots, S_{i_\ell}$, then these subsets satisfy the following conditions. (1) For $\tau = \Theta(\epsilon/k_{\delta^*})$, each variable x_i for which $\mathrm{Vr}_f(\{i\}) \geq \tau$ occurs in one of the subsets S_{i_j}, and each of these subsets contains at most one such variable. (2) The total variation of all other variables is $O(\epsilon/\log|\widehat{\mathcal{F}}_{\delta^*}|)$.

We now turn to the second procedure, which is referred to as Construct-Sample. This procedure receives as input the subsets $S_{i_1}, \ldots, S_{i_\ell}$ that were output by Identify-Critical-Subsets. Assume that indeed the subsets satisfy the aforementioned conditions. For the sake of the discussion, let us make the stronger assumption that every variable has either non-negligible variation with respect to f or zero variation. This implies that each subset S_{i_j} output by Identify-Critical-Subsets

contains exactly one relevant variable (and there are no other relevant variables).

Given a point $z \in \{0,1\}^n$, we would like to find the restriction of z to its $\ell \leq k_{\delta^*}$ relevant variables (without actually determining these variables). Consider a subset S_{i_j} output by Identify-Critical-Subsets, and let x_p, for $p \in S_{i_j}$, denote the relevant variable in S_{i_j}. We would like to know whether $z_p = 0$ or $z_p = 1$. To this end, we partition the variables in S_{i_j} into two subsets: $S^0_{i_j}(z) = \{q \in S_{i_j} : z_q = 0\}$ and $S^1_{i_j}(z) = \{q \in S_{i_j} : z_q = 1\}$. Now all we do is run the dependence test (as defined in Algorithm 5.1) sufficiently many times so as to ensure (with high probability) that we determine whether $p \in S^0_{i_j}(z)$ (so that $z_p = 0$), or $p \in S^1_{i_j}(z)$ (so that $z_p = 1$). The pseudo-code for the procedure appears in Figure 5.2.

The third procedure, Check-Consistency, is given as input the sample output by Construct-Sample. If some function $f' \in \widehat{\mathcal{F}}_{\delta^*}$ is consistent with

Procedure Construct-Sample(Input: $S_{i_1}, \ldots, S_{i_\ell}$)
Let $m = \Theta(\log|\widehat{\mathcal{F}}_{\delta^*}|/\epsilon)$. For $t = 1, \ldots, m$ construct a labeled example (x^t, y^t), where $x^t \in \{0,1\}^{k_{\delta^*}}$ as follows:

1. Uniformly select $z^t \in \{0,1\}^n$, and let $y^t = f(z^t)$.
2. For $j = 1, \ldots, \ell$ do:
   (a) For $b \in \{0,1\}$, let $S^b_{i_j}(z^t) = \{q \in S_{i_j} : z^t_q = b\}$.
   (b) For $g = \Theta(k_{\delta^*} \log(m \cdot k_{\delta^*})/\epsilon) = \Theta((k_{\delta^*}/\epsilon)\log(\log|\mathcal{F}_{\delta^*}|k_{\delta^*}/\epsilon))$, run the dependence test on $S^0_{i_j}(z^t)$ and on $S^1_{i_j}(z^t)$, g times (each).
   (c) If there is evidence that f depends on both $S^0_{i_j}(z^t)$ and $S^1_{i_j}(z^t)$, then output reject (and exit). If there is evidence that f depends on $S^b_{i_j}(z^t)$ for $b = 0$ or $b = 1$, then set $x^t_j = b$. Otherwise set x^t_j uniformly at random to be either 0 or 1.
3. For $j = \ell + 1, \ldots, k_{\delta^*}$, set x^t_j uniformly at random to be either 0 or 1.

Fig. 5.2 The procedure for constructing a labeled sample.

the sample, then the procedure outputs accept. Otherwise it outputs reject.

Proof Sketch of Theorem 5.7. Consider first the case that $f \in \mathcal{F}$, so that it is δ^*-close to some function $f' \in \widehat{\mathcal{F}}_{\delta^*}$ where f' is a k_{δ^*}-junta. The parameter δ^* is selected to be sufficiently small so that we can essentially assume that $f = f'$. Thus, we shall make this assumption in this proof sketch. For $\tau = \Theta(\epsilon/k_{\delta^*})$, each variable x_i such that $\mathrm{Vr}_{f'}(\{i\}) \geq \tau$ will be referred to as *highly relevant*. As discussed previously, with high probability, the procedure Identify-Critical-Subsets outputs $\ell \leq k_{\delta^*}$ subsets $S_{i_1}, \ldots, S_{i_\ell}$ that satisfy the following conditions: (1) each highly relevant variable occurs in one of these subsets; (2) each of the subsets contains at most one highly relevant variable of f' (in fact, exactly one relevant variable of f'); and (3) all other variables are "very irrelevant" (have small total variation).

Assuming the subsets output by Identify-Critical-Subsets are as specified above, consider the construction of $x^t \in \{0,1\}^{k_{\delta^*}}$ for any $1 \leq t \leq m$. Since each S_{i_j} contains exactly one relevant variable, if this variable is highly relevant, then the following holds with high probability: one of the executions of the dependence test finds evidence that either this variable is in $S^0_{i_j}(z^t)$ or that it is in $S^1_{i_j}(z^t)$, and x^t_j is set accordingly. If the variable is not highly relevant, then either x^t_j is set correctly, as in the highly relevant case, or x^t_j is set randomly to 0 or 1. Since the total variation of all non-highly relevant variables is small, with high probability $f'(x^t_j) = y^t$ (recall that $y^t = f(z^t)$). Thus, with high probability, we get a random sample of points in $\{0,1\}^{k_{\delta^*}}$ that is labeled by the k_{δ^*}-junta f'. Since $f' \in \widehat{\mathcal{F}}_{\delta^*}$, in such a case the procedure Check-Consistency will output accept, as required (recall that $\widehat{\mathcal{F}}_{\delta^*}$ is closed under permutations of the variables).

We now turn to the case that f is ϵ-far from $\mathcal{F}$. If it is also $(\epsilon/2)$-far from every k_{δ^*}-junta, then Identify-Critical-Subsets detect this with high probability, and rejects. Otherwise, f is $(\epsilon/2)$-close to a k_{δ^*}-junta. Note that f can still be rejected by either Identify-Critical-Subsets or by Create-Sample. If this occurs with high probability, then we are done. Otherwise, by the properties of these two procedures, with high probability there won't be any function in $\widehat{\mathcal{F}}_{\delta^*}$ that is consistent with

the sample output by Create-Sample (based on the subsets output by Identify-Critical-Subsets). This is true since otherwise it would imply that there is a function $f'' \in \widehat{\mathcal{F}}_{\delta^*} \subseteq \mathcal{F}$ that is $(\epsilon/2)$-close to a k_{δ^*}-junta f' such that $\text{dist}(f, f') \leq \epsilon/2$. But this would contradict the fact that f is ϵ-far from $\mathcal{F}$. □

6

The Regularity Lemma

One of the most powerful tools for analyzing property testing algorithms in the dense-graphs model is Szemerédi's Regularity Lemma [124] and variants of it.

6.1 Background

The first property testing result that uses (a variant of) the Regularity Lemma is implicit in work of Alon et al. [6]. Their result implies that k-colorability is testable with query complexity that is independent of n, where the dependence on $1/\epsilon$ is a tower of $\text{poly}(1/\epsilon)$ exponents. The first explicit testing result that uses the Regularity Lemma is in the work of Alon et al. [7]. Alon et al. [7] give algorithms for the class of *first-order* graph properties. These are properties that can be formulated by first-order expressions about graphs. This covers a large class of graph properties (in particular coloring and subgraph-freeness properties). Here too the application of the Regularity Lemma implies that the dependence on $1/\epsilon$ is a tower of $\text{poly}(1/\epsilon)$ exponents.

A sequence of works by Alon and Shapira [18, 17, 16], together with the work of Fischer and Newman [64] culminated in a characterization

of all graph properties that are testable (in the dense-graphs model) using a number of queries that is independent of n [8]. As the title of the paper says: "It's all about regularity". To be a little more precise, the characterization says (roughly) that a graph property $\mathcal{P}$ is testable using a number of queries that is independent of n *if and only if* testing $\mathcal{P}$ can be reduced to testing the property of satisfying one of a finitely many Szemerédi-partitions [124]. A different characterization, based on graph limits, was proved independently by Borgs et al. [40].

Variants of the regularity lemma were also used to derive property testing results for directed graphs [15] and for hypergraphs [102, 14, 59]. In what follows we first state the lemma and then give an example of its application by analyzing a testing algorithm for *triangle-freeness*.

6.2 Statement of the Lemma

In order to state the lemma, we need some definitions and notations. For any two non-empty disjoint sets of vertices, A and B, we let $E(A,B)$ denote the set of edges between A and B, and we let $e(A,B) = |E(A,B)|$. The *edge-density* of the pair is defined as:

$$d(A,B) \stackrel{\text{def}}{=} \frac{e(A,B)}{|A| \cdot |B|}. \tag{6.1}$$

We say that a pair A,B is γ-*regular* for some $\gamma \in [0,1]$ if for every two subsets $A' \subseteq A$ and $B' \subseteq B$ satisfying $|A'| \geq \gamma|A|$ and $|B'| \geq \gamma|B|$ we have that $|d(A',B') - d(A,B)| < \gamma$. Note that if we consider a random bipartite graph between A and B (where there is an edge between each pair of vertices $v \in A$ and $u \in B$ with constant probability p), then it will be regular w.h.p. for some constant γ. In what follows, when we refer to an equipartition $\mathcal{A} = \{V_1, \ldots, V_k\}$ of V, we mean that for every $1 \leq j \leq k$, $||V_i| - |V_j|| \leq 1$.

Lemma 6.1 (Regularity Lemma). For every integer ℓ_0 and for every $\gamma \in (0,1]$, there exists a number $u_0 = u_0(\ell_0, \gamma)$ with the following property: Every graph $G = (V,E)$ with $n \geq u_0$ vertices has an equipartition $\mathcal{A} = \{V_1, \ldots, V_k\}$ of V where $\ell_0 \leq k \leq u_0$ for which all pairs (V_i, V_j) but at most $\gamma \cdot \binom{k}{2}$ of them are γ-regular.

6.3 Testing Triangle-Freeness

For a graph $G = (V, E)$ and a triple of distinct vertices (u, v, w), we say that (u, v, w) is a triangle in G if all three pairs, (u, v), (u, w), and (v, w) are edges in the graph. A graph $G = (V, E)$ is *triangle-free* if it contains no triangles. The algorithm for testing triangle-freeness simply takes a sample of size $m = m(\epsilon)$ (which will be set later), queries all pairs of vertices in the sample to obtain the induced subgraph, and accepts or rejects depending on whether it sees a triangle in the induced subgraph. Clearly, if the graph is triangle-free, then the algorithm accepts with probability 1. It remains to prove that if the graph is ϵ-far from triangle-free, then (for sufficiently large $m = m(\epsilon)$), the sample will contain a triangle with high constant probability.

An important note is in place concerning the size of m. Alon [3] has shown that (as opposed to bipartiteness and other partition problems) it does *not* suffice to take m that is polynomial in $1/\epsilon$. That is, there exist graphs that are ϵ-far from being triangle-free but for which a poly$(1/\epsilon)$-size sample will not show any triangle. In other words, it is possible that the fraction of edges that need to be removed in order to make a graph triangle-free is greater than ϵ, but the fraction of triples of vertices that are triangles (among all n^3 triples) is smaller than poly(ϵ). We discuss this in more detail in section 9.1. As we shall see, the sample size m that we can show suffices for our needs, is significantly higher than the lower bound, so there is quite a big gap between the upper and lower bounds, and indeed it is an interesting open problem to reduce this gap.

Suppose we apply the regularity lemma with $\ell_0 = 8/\epsilon$ and $\gamma = \epsilon/8$. Our first observation is that for this setting, the total number of edges in G that are between pairs of vertices that belong to the same part V_i of the partition is at most

$$k \cdot \left(\frac{n}{k}\right)^2 = \frac{1}{k} \cdot n^2 \leq \frac{1}{\ell_0} \cdot n^2 = \frac{\epsilon}{8} n^2. \tag{6.2}$$

It follows that if we define G_1 as the graph that is the same as G except that we remove all edges within the parts of the regular partition, then G_1 is at least $(7/8)\epsilon$-far from being triangle-free.

Next, since there are at most $\frac{\epsilon}{8} \cdot \binom{k}{2} < \frac{\epsilon}{16}k^2$ non-regular pairs, the total number of edges between non-regular pairs in the partition is at most

$$\frac{\epsilon}{16}k^2 \cdot \left(\frac{n}{k}\right)^2 = \frac{\epsilon}{16}n^2. \tag{6.3}$$

Therefore, if we continue by removing all these edges from G_1, and let the resulting graph be denoted G_2, then G_2 is at least $(3/4)\epsilon$-far from being triangle-free.

We shall perform one more step of this kind (for an illustration of all three steps, see Figure 6.1). Consider all pairs (V_i, V_j) such that $d(V_i, V_j) < \frac{\epsilon}{2}$. That is, $e(V_i, V_j) < \frac{\epsilon}{2} \cdot \left(\frac{n}{k}\right)^2$. Since there are at most $k^2/2$ such pairs, the total number of edges between such pairs is at most $\frac{\epsilon}{4}n^2$. Therefore, if we remove all these edges from G_2, and let the resulting graph be denoted G_3, then G_3 is at least $(\epsilon/2)$-far from being triangle-free. In particular this means that there exists at least one

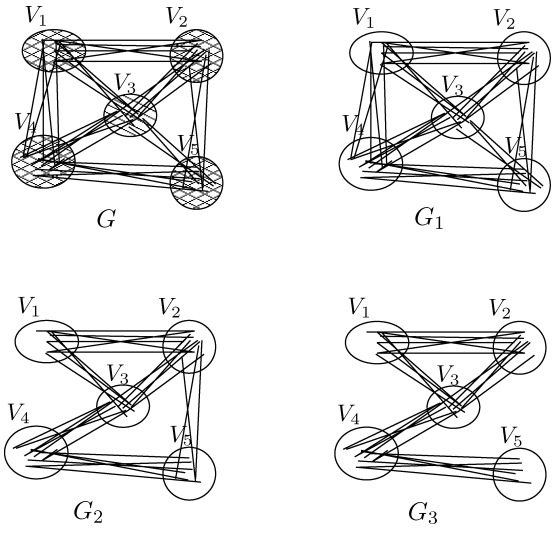

Fig. 6.1 An illustration of the three steps in the modification of the graph G, given its regular partition. In the transformation from G to G_1 we remove all edges internal to the parts in the partition. In the transformation from G_1 to G_2 we remove all edges between non-regular pairs of parts (i.e., $E(V_1, V_4)$ and $E(V_3, V_5)$), and in the transformation from G_2 to G_3 we remove all edges between pairs of parts whose edge density is relatively small (i.e., $E(V_2, V_5)$).

triplet (V_i, V_j, V_ℓ) such that all three edge-densities, $d(V_i, V_j)$, $d(V_j, V_\ell)$, and $d(V_i, V_\ell)$ are at least $\epsilon/2$ in G_3. (If no such triplet existed then G_3 would be triangle-free.) We shall show that since all three pairs are $(\epsilon/8)$-regular, there are "many real triangles" $(u, v, w) \in V_i \times V_j \times V_\ell$, so that a sufficiently large sample will catch one.

For simplicity we denote the three subsets by V_1, V_2, V_3. For each vertex $v \in V_1$, we let $\Gamma_2(v)$ denote the set of neighbors that v has in V_2, and by $\Gamma_3(v)$ the set of neighbors that v has in V_3 (for an illustration see Figure 6.2). We shall say that v is *helpful* if both $|\Gamma_2(v)| \geq \frac{\epsilon}{4}\left(\frac{n}{k}\right)$ and $|\Gamma_3(v)| \geq \frac{\epsilon}{4}\left(\frac{n}{k}\right)$. Since (V_2, V_3) is a regular pair,

$$e(\Gamma_2(v), \Gamma_3(v)) \geq (d(V_2, V_3) - \gamma)\left(\frac{\epsilon}{4}\right)^2\left(\frac{n}{k}\right)^2 \geq \frac{\epsilon^3}{c \cdot k^2} n^2 \tag{6.4}$$

for some constant c. It follows that if we get a helpful vertex v from V_1, and then we take an additional sample of $\Theta((u_0)^2/\epsilon^3)$ pairs of vertices (recall that $k \leq u_0$), then we shall obtain a triangle with high constant probability. It remains to show that there are relatively many helpful vertices in V_1.

Consider any vertex $z \in V_1$ that is *not* helpful. We shall say that it is *unhelpful of type* 2 if $|\Gamma_2(v)| < \frac{\epsilon}{4}\left(\frac{n}{k}\right)$, and that it is *unhelpful of type* 3 if $|\Gamma_3(v)| < \frac{\epsilon}{4}\left(\frac{n}{k}\right)$. Without loss of generality, assume that there are more unhelpful vertices of type 2. Suppose that at least half the vertices in V_1 are unhelpful. Then at least a fourth of the vertices are unhelpful of type 2. Let V_1' consist of all these vertices, so that $|V_1'| > \gamma|V_1|$ (recall

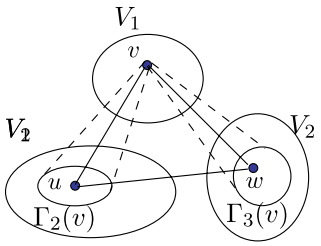

Fig. 6.2 An illustration of three parts such that all three pairs of parts are γ-regular, and the edge-density between each pair is at least $\epsilon/2$. For every helpful vertex v in V_1, there are relatively many edges (u, w) such that $u \in \Gamma_2(v)$ and $w \in \Gamma_3(v)$.

that $\gamma = \epsilon/8 \geq 1/8$). Let $V_2' = V_2$. But then,

$$d(V_1', V_2) \leq \frac{|V_1'| \cdot \frac{\epsilon}{4}\left(\frac{n}{k}\right)}{|V_1'| \cdot |V_2'|} = \frac{\epsilon}{4} < d(V_1, V_2) - \gamma \tag{6.5}$$

and we have reached a contradiction to the regularity of the pair (V_1, V_2). Hence, at least a half of the vertices in V_1 are helpful (that is, $\Omega\left(\frac{n}{k}\right)$ vertices), and so a sample of size $\Theta(u_0)$ will contain a helpful vertex with high probability. Therefore, if we take a sample of vertices having size $\Theta(u_0) + \Theta((u_0)^2/\epsilon^3) = \Theta((u_0)^2/\epsilon^3)$, then the induced subgraph will contain a triangle with high probability.

The above analysis may seem somewhat wasteful, but unless we find a way to remove the dependence on the number of parts of the partition, given that this number is a tower of height poly$(1/\epsilon)$, it does not make much of a difference if our dependence on this number is linear or quadratic.

Other results based on the Regularity lemma. As noted previously, there are many other results that build on the Regularity Lemma. While their analysis may be more complex than that of triangle-freeness, the core of these arguments is the same. Specifically, what the regularity lemma essentially says is that for any given γ, every graph G corresponds to a small graph, G^γ over $k(\gamma)$ vertices whose edges have weights in $[0,1]$. The correspondence is such that for every vertex in the small graph there is a subset of vertices in G, where the subsets have (almost) the same size, the edge-densities between the subsets in G equal the weights of the edges in G^γ, *and* all but a γ-fraction of the pairs of subsets are γ-regular. It can then be shown that: (1) If G is ϵ-far from a certain property then for $\gamma = \gamma(\epsilon)$, G^γ is relatively far from a related property (where the distance measure takes the edge-weights of G^γ into account); and (2) If G^γ is far from this property then, due to the regularity of almost all pairs of subsets in G, a sufficiently large (i.e., that depends on $k(\gamma)$) sample in G will provide evidence that G does not have the original property considered.

7

Local-Search Algorithms

Recall that when dealing with sparse graphs (where here this will refer to graphs with $O(n)$ edges, though some of the results are more general), we consider two models. In both models the algorithm can perform queries of the form: "who is the i-th neighbor of vertex v", where if v has less than i neighbor, then a special symbol is returned. The difference between the two models is in whether there is a fixed degree-bound, denoted d, on the degrees of all vertices or whether no such bound is assumed. In the first model, referred to as the *bounded-degree* model, a graph is said to be ϵ-far from having a specified property if more than ϵdn edges modifications should be performed so that the graph obtains the property (where its degree remains bounded by d). In the second model, referred to as the *sparse-graphs* model, a graph is said to be ϵ-far from having a specified property if more than ϵm edges modifications should be performed so that the graph obtains the property, where m is the number of edges in the graph (or a given upper bound on this number).

In what follows we first present what is probably the simplest local-search type algorithm: the algorithm for testing connectivity [76]. We then discuss the extension of this simple algorithm to testing k-edge

connectivity, and very shortly discuss what is known about testing k-vertex connectivity. Testing minor-closed properties [33, 87] is considered in Section 7.4, and in Section 7.5 we briefly mention other testing algorithms that are based on local search.

7.1 Connectivity

One of the most basic properties of graphs is *connectivity*. A graph is *connected* if there is a path between every two vertices. Otherwise, the vertices in the graph are partitioned into *connected components*, which are maximal subgraphs that are connected. The basic simple observation is that if a graph is not connected, then the minimum number of edges that should be added to it in order to make it connected is simply $\kappa(G) - 1$, where $\kappa(G)$ is the number of connected components in G. The implication of this simple observation for the sparse-graphs model is:

Claim 7.1. If a graph is ϵ-far from being connected (as defined in the sparse-graphs model), then it contains more than $\epsilon m + 1$ connected components.

Proof. Assume, contrary to the claim that there are at most $\epsilon m + 1$ connected components, denoted $C_1, \ldots, C_\ell$. Then, by adding, for each $1 \leq j \leq \ell - 1$ an edge between some vertex in C_j and some vertex in C_{j+1}, we obtain a connected graph. Since the number of edges added is at most ϵm, we reach a contradiction. □

In the bounded-degree model there is a small subtlety: Because of the degree constraint, we can't immediately get a contradiction as in the proof of Claim 7.1. That is, it might be the case that in some connected component C_j, all vertices have the maximum degree d, and we cannot simply add edges incident to them. However, it is not hard to show that in such a case (since d must be at least 2 or else the graph cannot be connected), it is possible to remove an edge from C_j without disconnecting it, and thus obtain two vertices with degree less than d. Therefore, in the bounded-degree model it also holds that if the graph is

far from being connected then it contains many connected components. For the sake of simplicity we continue with the sparse-graphs model. Before doing so note that since $\kappa(G) < n$, so that $\kappa(G)/n^2 < 1/n$, in the dense-graphs model, every graph is ϵ-close to being connected for $\epsilon > 1/n$. In other words, it is trivial to test for connectivity in the dense-graphs model.

Let $d_{\text{avg}} = \frac{m}{n}$ (the average degree up to a factor of 2). Then Claim 7.1 implies the next claim.

Claim 7.2. If a graph is ϵ-far from being connected, then it contains more than $\frac{\epsilon}{2} d_{\text{avg}} n$ connected components of size at most $\frac{2}{\epsilon d_{\text{avg}}}$ each.

An implicit implication of Claim 7.2 is that if $\epsilon \geq \frac{2}{d_{\text{avg}}}$, then every graph is ϵ-close to being connected, so that the algorithm can immediately accept. Hence we assume from this point on that $\epsilon < \frac{2}{d_{\text{avg}}}$.

Proof. By Claim 7.1, the graph contains more than $\epsilon d_{\text{avg}} n$ connected components. Assume, contrary to the claim, that there are at most $\frac{\epsilon}{2} d_{\text{avg}} n$ components of size at most $\frac{2}{\epsilon d_{\text{avg}}}$ each. We shall refer to such components as *small* components. Otherwise they are *big*. Consider all other (big) components. Since they each contain more than $\frac{2}{\epsilon d_{\text{avg}}}$ vertices, and they are disjoint, there can be at most $\frac{n}{2/(\epsilon d_{\text{avg}})} = \frac{\epsilon}{2} d_{\text{avg}} n$ such big connected components. Adding the number of small connected components we get a contradiction. □

Claim 7.1 suggests the algorithm given in Figure 7.1.

Correctness of the algorithm. Clearly, if the graph is connected, then the algorithm accepts with probability 1. On the other hand, if the graph is ϵ-far from being connected, then by Claim 7.2 there are at least $\frac{\epsilon}{2} d_{\text{avg}} n$ vertices that belong to small connected components. If the algorithm selects such a vertex in its first step, then it rejects. The probability that the algorithm does not select such a vertex is $(1 - \frac{\epsilon d_{\text{avg}}}{2})^{4/(\epsilon d_{\text{avg}})} < e^{-2} < 1/3$.

Algorithm 7.1: Connectivity Testing Algorithm I

1. *Uniformly and independently select* $\frac{4}{\epsilon d_{\text{avg}}}$ *vertices.*[a]
2. *For each selected vertex* s *perform a BFS starting from* s *until* $\frac{2}{\epsilon d_{\text{avg}}}$ *vertices have been reached or no more new vertices can be reached (a small connected component has been found).*
3. *If any of the above searches finds a small connected component, then the algorithm* rejects*, otherwise it* accepts.

[a] If ϵ is so small so that the sample size is of the order of n, then the algorithm will just run a BFS on the graph.

Fig. 7.1 Connectivity testing algorithm (version I).

The query complexity (and running time). Consider first the case that the maximum degree in the graph is of the same order as the average degree d_{avg}. In this case the complexity is:

$$O\left(\frac{1}{\epsilon\, d_{\text{avg}}} \cdot \frac{1}{\epsilon\, d_{\text{avg}}} \cdot d_{\text{avg}}\right) = O\left(\frac{1}{\epsilon^2 d_{\text{avg}}}\right), \tag{7.1}$$

and in particular, when $d_{\text{avg}} = O(1)$ we get $O(1/\epsilon^2)$. More generally, the cost of the BFS is the total number of edges observed in the course of the BFS, which is at most $\left(\frac{2}{\epsilon\, d_{\text{avg}}}\right)^2$ and so we get an upper bound of $O\left(\frac{1}{\epsilon^3 (d_{\text{avg}})^3}\right)$.

Improving the query complexity. Note that there was a certain "waste" in our counting. On one hand we performed, for each vertex selected, a BFS that may go up to $\frac{2}{\epsilon\, d_{\text{avg}}}$ vertices, since in the worst case all small components have this number of vertices. On the other hand, when we counted the number of vertices that belong to small connected components, then we assumed a worst-case scenario in which there is just one vertex in each small component. These two worst-case scenarios cannot occur together. Specifically, if all small components indeed contain $\frac{2}{\epsilon\, d_{\text{avg}}}$ vertices, then (since there are at least $\frac{\epsilon}{2 d_{\text{avg}}} n$ small components), every vertex belongs to a small component so the probability of selecting a vertex from a small component is 1. In the other extreme,

if all small components are of size 1, then we will need to take a sample of size $\Theta(1/\epsilon d_{\text{avg}})$, but then it will be enough to ask a constant number of queries for each of these vertices.

More generally, let us partition the (at least $\frac{\epsilon}{2d_{\text{avg}}}n$) small connected components according to their size. Specifically, for $i = 1, \ldots, \ell$ where $\ell = \log(2/(\epsilon d_{\text{avg}})) + 1$ (where for simplicity we ignore floors and ceilings), let B_i consist of all connected components that contain at least 2^{i-1} vertices and at most $2^i - 1$ vertices. By our lower bound on the total number of small connected components, $\sum_{i=1}^{\ell} |B_i| \geq \frac{\epsilon}{2} d_{\text{avg}} n$. Hence, there exists an index j for which $|B_j| \geq \frac{1}{\ell} \cdot \frac{\epsilon}{2} d_{\text{avg}} n$. By definition of B_j, the number of vertices that belong to connected components in B_j is at least $2^{j-1} \cdot |B_j| \geq 2^{j-1} \cdot \frac{1}{\ell} \cdot \frac{\epsilon}{2} d_{\text{avg}} n$, and if we uniformly and independently select a vertex, then the probability that it belongs to a component in B_j is $2^{j-1} \cdot \frac{1}{\ell} \cdot \frac{\epsilon}{2} d_{\text{avg}}$. Therefore, if we select $\Theta\left(\frac{\ell}{2^{j-1}\epsilon d_{\text{avg}}}\right)$ vertices, uniformly at random, and from each we perform a BFS until we reach 2^j vertices or a small connected component is detected, then we'll find evidence of a small connected component in B_j with constant probability.

Since we don't know what is the (an) index j for which $|B_j| \geq \frac{1}{\ell} \cdot \frac{\epsilon}{2} d_{\text{avg}} n$, we run over all possibilities. Namely, we get the algorithm in Figure 7.2.

Algorithm 7.2: Connectivity Testing Algorithm II

1. *For* $i = 1$ *to* $\ell = \log(2/(\epsilon d_{\text{avg}})) + 1$ *do*
   (a) *Uniformly and independently select* $t_i = \frac{4\ell}{2^{i-1}\epsilon d_{\text{avg}}}$ *vertices.*
   (b) *For each selected vertex* s *perform a BFS starting from* s *until* 2^i *vertices have been reached or no more new vertices can be reached (a small connected component has been found).*
2. *If any of the above searches finds a small connected component, then the algorithm* rejects*, otherwise it* accepts.

Fig. 7.2 Connectivity testing algorithm (version II).

The correctness of the algorithm follows from the foregoing discussion. The query complexity is bounded as follows:

$$\sum_{i=1}^{\ell} t_i \cdot 2^{2i} = c \cdot \sum_{i=1}^{\ell} \frac{\ell}{2^{i-1}\epsilon\, d_{\text{avg}}} \cdot 2^{2i} \tag{7.2}$$

$$= c' \cdot \frac{\ell}{\epsilon\, d_{\text{avg}}} \sum_{i=1}^{\ell} 2^i \tag{7.3}$$

$$\leq c' \cdot \frac{\ell}{\epsilon\, d_{\text{avg}}} \cdot 2^{\ell+1} \tag{7.4}$$

$$= c'' \cdot \frac{\log(1/(\epsilon d_{\text{avg}}))}{\epsilon^2 d_{\text{avg}}}. \tag{7.5}$$

Therefore, we have saved a factor of $\Theta\left(\frac{\log(1/(\epsilon\, d_{\text{avg}}))}{\epsilon\, d_{\text{avg}}}\right)$ (as compared to Algorithm 7.1).

7.2 k-Edge Connectivity

Recall that a graph is k-edge connected (or in short, k-connected), if between every pair of vertices in the graph there are k edge-disjoint paths. An equivalent definition is that the size of every cut in the graph is at least k. Recall that the connectivity (1-connectivity) testing algorithm is based on the observation that if a graph is far from being connected, then it contains many small connected components (cuts of size 0). This statement generalizes as follows to k-connectivity for $k > 1$ (based on e.g., [53, 52, 109]) where we use the shorthand $(C, \overline{C})$ for $|E(C, V \setminus C)|$. If a graph is far from being k-connected, then it contains *many* subsets C of vertices that are *small* and such that: (1) $(C, \overline{C}) = \ell < k$; (2) for every $C' \subset C$, $(C', \overline{C'}) > \ell$. We say in this case that the subset C is ℓ-*extreme*.

As in the case of connectivity, we shall uniformly select a sufficient number of vertices and for each we shall try and detect whether it belongs to a small ℓ-extreme set C for $\ell < k$. The algorithmic question is how to do this in time that depends only on the size of C and possibly d (or d_{avg}) and k. There are special purpose algorithms for $k = 2$ and $k = 3$ (that are more efficient than the general algorithm), but here we shall discuss how to deal with the general case of $k \geq 3$.

The problem is formalized as follows: Given a vertex $v \in C$ where C is an ℓ-extreme set for $\ell < k$ and $|C| \leq t$, describe a (possibly randomized) procedure for finding C (with high probability), when given access to neighbor queries in the graph. Here it will actually be somewhat simpler to work in the bounded-degree model (though the algorithm can be easily modified to work in the sparse-graphs (unbounded-degree) model).

The suggested procedure is an iterative procedure: at each step it has a current subset of vertices S and it adds a single vertex to S until $|S| = t$ or a cut of size less than k is detected. To this end the procedure maintains a cost that is assigned to every edge incident to vertices in S. Specifically, initially $S = \{v\}$. At each step, the procedure considers all edges in the cut $(S, \overline{S})$. If an edge was not yet assigned a cost, then it is assigned a cost uniformly at random from $[0, 1]$. Then the edge in $(S, \overline{S})$ that has the minimum cost among all cut edges is selected. If this edge is (u, v) where $u \in S$ and $v \in \overline{S}$, then v is added to S. The procedure is repeated $\Theta(t^2)$ times. Our goal is to prove that a single iteration of the procedure succeeds in reaching $S = C$ with probability at least t^{-2} or possibly reaching $S = C'$ such that the cut $(C', \overline{C'})$ has size less than k (recall that ℓ may be strictly smaller than k). Before doing so observe that the total running time is upper bounded by $O(t^2 \cdot t \cdot d\log(td)) = \tilde{O}(t^3 \cdot d)$. Since it is sufficient to consider t that is polynomial in k and $1/(\epsilon d)$, we obtain an algorithm whose complexity is polynomial in k and $1/\epsilon$.

For our purposes it will be convenient to represent $\overline{C}$ by a single vertex x that has ℓ neighbors in C. Since, if the procedure traverses an edge in the cut $(C, \overline{C})$, we account for this as a failure in detecting the cut, we are not interested in any other information regarding $\overline{C}$. Let this new graph, over at most $t + 1$ vertices, be denoted by G_C. Note that since C is an ℓ-extreme set, every vertex $v \in C$ has degree greater than ℓ. The first observation is that though our procedure assigns costs in an online manner, we can think of the random costs being assigned ahead of time, and letting the algorithm "reveal" them as it goes along.

Consider any spanning tree T of the subgraph induced by C (this is the graph G_C minus the "outside" vertex x). We say that T is *cheaper than the cut* $(C, \overline{C})$ if all $t - 1$ edges in T have costs that are lower than all costs of edges in the cut $(C, \overline{C})$.

Claim 7.3. Suppose that the subgraph induced by C has a spanning tree that is cheaper than the cut $(C, \overline{C})$. Then the search process succeeds in finding the cut $(C, \overline{C})$ or a cut $(C', \overline{C'})$ that has size less than k.

Proof. We prove, by induction on the size of the current S, that $S \subseteq C$. Since the procedure stops when it finds a cut of size less than k, it will stop when $S = C$, if it doesn't stop before that. Initially, $S = \{v\}$ so the base of the induction holds. Consider any step of the procedure. By the induction hypothesis, at the start of the step $S \subseteq C$. If $S = C$, then we are done. Otherwise, $S \subset C$. But this means that there is at least one edge from the spanning tree in the current cut $(S, \overline{S})$ (that is, an edge connecting $v \in S$ to $u \in C \setminus S$). But since all edges in $(C, \overline{C})$ have a greater cost, one of the spanning tree edges must be selected, and the induction step holds. □

Karger's Algorithm. It remains to prove that with probability $\Omega(t^{-2})$, the subgraph induced by C has a spanning tree that is cheaper than the cut $(C, \overline{C})$. To this end we consider a randomized algorithm for finding a minimum cut in a graph known as "Karger's min-cut algorithm"[95], and its analysis.

Karger's algorithm works iteratively as follows. It starts from the original graph (in which it wants to find a min-cut) and at each step it modifies the graph and in particular decreases the number of vertices in the graph. An important point is that the intermediate graphs may have parallel edges (even if the original graph does not). The modification in each step is done by *contracting* two vertices that have at least one edge between them. After the contraction we have a single vertex instead of two, but we keep all edges to other vertices. That is, if we contract u and v into a vertex w, then for every edge (u, z) such that $z \neq v$ we have an edge (w, z) and similarly for (v, z), $z \neq u$. The edges between u and v are discarded (i.e., we don't keep any self-loops). The algorithm terminates when only two vertices remain: each is the contraction of one side of a cut, and the number of edges between them is exactly the size of the cut.

The contraction in Karger's algorithm is performed by selecting, uniformly at random, an edge in the current graph, and contracting its

two end-points. Recall that we have parallel edges, so the probability of contracting u and v depends on the number of edges between them. An equivalent way to describe the algorithm is that we first uniformly select a random ordering (permutation) of the edges, and then, at each step we contract the next edge (that is still in the graph) according to this ordering. To get a random ordering we can simply assign random costs in $[0,1]$ to the edges in the graph (which induces a random ordering of the edges).

Now, as a thought experiment, consider an execution of Karger's min-cut algorithm on G_C (whose min-cut is $(C,\{x\})$). If the algorithm succeeds in finding this cut (that is, it ends when C is contracted into a single vertex and no edge between C and x is contracted), then the edges it contracted along the way constitute a spanning tree of C and this spanning tree is cheaper than the cut. So the probability that Karger's algorithm succeeds is a lower bound on the probability that there exists a spanning tree that is cheaper than the cut, which is a lower bound on the success probability of the local-search procedure. Hence, it remains to lower bound the success probability of Karger's algorithm. (Observe that our local-search procedure also defines a spanning tree, but its construction process is similar to Prim's algorithm while Karger's algorithm is similar to Kruskal's algorithm.)

Returning to the analysis of Karger's algorithm, the simple key point is that, since C is an ℓ-extreme set, at *every step* of the algorithm the degree of every vertex is at least $\ell+1$ (recall that a vertex corresponds to a contracted subset $C' \subset C$). Thus, at the start of the i-th contraction step the current graph contains at least $\frac{(n-(i-1))\cdot(\ell+1)+\ell}{2}$ edges. Hence, the probability that no cut edge is contracted is at least

$$\prod_{i=1}^{t-1}\left(1-\frac{2\ell}{(t-(i-1))\cdot(\ell+1)+\ell}\right)$$

$$=\prod_{i=0}^{t-2}\left(1-\frac{2\ell}{(t-i)\cdot(\ell+1)+\ell}\right) \tag{7.6}$$

$$=\prod_{i=0}^{t-2}\frac{(t-i)(\ell+1)-\ell}{(t-i)(\ell+1)+\ell} \tag{7.7}$$

$$= \prod_{j=2}^{t} \frac{j - \ell/(\ell+1)}{j + \ell/(\ell+1)} \tag{7.8}$$

$$> \prod_{j=2}^{t} \frac{j-1}{j+1} > \frac{6}{t^2}. \tag{7.9}$$

(If ℓ is small, e.g., $\ell = 1$, then the probability is even higher.) Thus, the success probability of Karger's algorithm, and hence of our local-search algorithm, is $\Omega(t^{-2})$.

A Comment. Note that the local-search procedure *does not* select to traverse at each step a random edge in the cut $(S, \overline{S})$. To illustrate why this would not be a good idea, consider the case in which $k = 1$, C is a cycle, and there is one edge from a vertex v in C to x. If we executed the alternative algorithm, then once v would be added to S, at each step the probability that the cut edge is traversed, would be $1/3$, and the probability this doesn't happen would be exponential in t. On the other hand, the way our procedure works, it succeeds with probability $\frac{1}{t}$ because that is the probability that the cut edge gets the maximum cost.

7.3 k-Vertex Connectivity.

A graph is k-vertex connected if between every two vertices there are k vertex-disjoint paths. First note that being 1-vertex connected is the same as being 1-edge connected, a property we have already discussed. Testing k-vertex connectivity for $k = 2$ and $k = 3$ was studied in the bounded-degree model in [73], and the algorithms proposed have query complexity (and running time) $\tilde{O}(\epsilon^{-2}d^{-1})$ and $\tilde{O}(\epsilon^{-3}d^{-2})$, respectively. The general case of $k > 3$ was studied by Ito and Yoshida [91], and their algorithm has complexity $\tilde{O}(d(ck/(\epsilon d))^k)$ (for a constant c).

7.4 Minor-Closed Properties

A graph property is said to be *minor closed* if it is closed under removal of edges, removal of vertices, and contraction of edges.[1] All

[1] When an edge (u, v) is contracted, the vertices u and v are replaced by a single vertex w, and the set of neighbors of w is the union of the sets of neighbors of u and v.

properties defined by a forbidden minor[2] (or minors) are minor closed, and in particular this is true of planarity, outerplanarity, having a bounded tree-width, and more. A graph property is *hereditary* if it is closed under removal of vertices (so that every minor-closed property is hereditary, but also other properties such as k-colorability). Czumaj et al. [44] proved that every hereditary property is testable if the input graph belongs to a family of graphs that is hereditary and "non-expanding" (that is, it does not contain graphs with expansion greater than $1/\log^2 n$). The query complexity of their testing algorithm is doubly-exponential in $\text{poly}(1/\epsilon)$.

Building on some ideas from [44], Benjamini et al. [33] proved that every minor-closed property is testable (without any condition on the tested graph), using a number of queries that is triply-exponential in $\text{poly}(1/\epsilon)$. This result was improved by Hassidim et al. [87] who reduced the complexity to singly-exponential in $\text{poly}(1/\epsilon)$. We note that the special case of testing cycle-freeness (which is equivalent to testing whether a graph is K_3-minor-free) is considered in [76]. That work includes an algorithm for testing cycle-freeness in the bounded-degree model with query complexity $\text{poly}(1/\epsilon)$. We also mention that testing cycle-freeness in the sparse (unbounded-degree) model requires $\Omega(\sqrt{n})$ queries [112].

All the abovementioned algorithms perform local search. The [44] algorithm searches for evidence that the graph does not have the property, where this evidence is in the form of a forbidden induced subgraph (of bounded size). Thus, their algorithm has one-sided error. Finding such evidence by performing a number of queries that does not depend on n is not possible in general graphs (even for the simple case of the (minor-closed) property of cycle-freeness there is a lower bound of $\Omega(\sqrt{n})$ on the number of queries necessary for any one-sided error algorithm [76]). Instead, the algorithm of [33] uses local search in order to estimate the number of different subgraphs of a bounded size, and it is shown that such an estimate can distinguish (with high probability)

[2] A graph H is a minor of a graph G if H can be obtained from G by vertex removals, edge removals, and edge contractions. Robertson and Seymour [119] have shown that every minor-closed property can be expressed via a constant number of forbidden minors (where it is possible to find such a minor if it exists, in cubic time [118]).

between graphs that have any particular minor-closed property and graphs that are far from having the property.

The algorithm of [87] has both a "one-sided error part" and a "two-sided error part". That is, it may reject either because it found "hard" evidence in the form of a small subgraph that contains a forbidden H-minor, or because it found certain "circumstantial evidence". The latter type of evidence is based on the Separator Theorem of Alon et al. [13]. This theorem implies that if a graph (with degree at most d) is H-minor free, then by removing at most ϵdn edges it is possible to obtain connected components that are all of size $O(1/\epsilon^2)$ (where the constant in the $O(\cdot)$ notation depends on H). The algorithm of [87] first tries to estimate the number of edges in G between the subsets of vertices that correspond to these connected components. This is done by implementing what they call a *Partition Oracle* (where in the case that the graph is minor closed, the parts of the partition correspond to small connected subsets that are separated by a relatively small number of edges).

If the graph has the minor-closed property in question, then with high probability the estimate on the number of edges between parts will be sufficiently small. On the other hand, if the graph is far from the property, then one of the following two events will occur (with high probability): (1) the estimate obtained is large, so that the algorithm may reject; or (2) the estimate obtained is small (because the graph can be separated to small connected components by removing few edges (even though the graph is far from being minor closed)). In the latter case it can be shown that in the second part of the algorithm, where the algorithm searches for a forbidden minor in the close vicinity of a sample of vertices, a forbidden minor will be detected with high probability (since many forbidden minors must reside within the small parts).

7.5 Other Local-Search Algorithms

There are also local-search algorithms for the following properties: being Eulerian, subgraph-freeness [76], and having a diameter of a bounded size [112]. The dependence on $1/\epsilon$ in all these cases is polynomial (and there is no dependence on the number, n, of graph vertices).

In some cases (subgraph-freeness) the algorithm works only in the bounded-degree model (and there is a lower bound of $\Omega(\sqrt{n})$ on the number of queries required in the sparse (unbounded-degree) model). In the other cases there are algorithms that work in both models. All of these algorithms are based on local search, though naturally, the local search may be somewhat different, and once the local search is performed, different checks are performed. For example, in the case of the diameter testing algorithm, it is important to account for the depth of the BFS. That is, we are interested in the number of vertices reached when going to a certain distance from the selected vertices.

8

Random Walks Algorithms

In this section we discuss two algorithms that work in the bounded-degree model and are based on random walks. The first is for testing bipartiteness and the second is for testing expansion. The algorithm for testing bipartiteness was extended to the general model (and as a special case, to the sparse-graphs model), in [96]. We shortly discuss the [96] result in Section 10.

8.1 Testing Bipartiteness in Bounded-Degree Graphs

Recall that a graph $G = (V, E)$ is bipartite if there exists a partition (V_1, V_2) of V such that $E(V_1, V_1) \cup E(V_2, V_2) = \emptyset$. A graph is ϵ-far from being bipartite in the bounded-degree model if more than ϵdn edges should be removed in order to make it bipartite. In other words, for every partition (V_1, V_2) of V we have that $|E(V_1, V_1) \cup E(V_2, V_2)| > \epsilon dn$. It was first shown in [76] that as opposed to the dense-graphs model, any algorithm for testing bipartiteness in the bounded-degree model must perform $\Omega(\sqrt{n})$ queries (for constant ϵ). (See Section 9.2 for a high-level idea of the proof.)

The algorithm we present [74] almost matches this lower bound in terms of its dependence on n. The query complexity and running time

of the algorithm are $O(\sqrt{n} \cdot \text{poly}(\log n/\epsilon))$. The algorithm is based on performing *random walks* of the following (*lazy*) form: in each step of the random walk on a graph of degree at most d, if the current vertex is v, then the walk continues to each of v's neighbors with equal probability $\frac{1}{2d}$ and remains in place with probability $1 - \frac{\deg(v)}{2d}$. An important property of such random walks on connected graphs is that, no matter in which vertex we start, if we consider the distribution reached after t steps, then for sufficiently large t, the distribution is very close to uniform. The sufficient number of steps t depends on properties of the graph (and of course on how close we need to be to the uniform distribution). In particular, if it suffices that t be relatively small (i.e., logarithmic in n), then we say that the graph is *rapidly mixing*. In particular, *expander* graphs (in which subsets of vertices have relatively many neighbors outside the subset) are rapidly mixing. We describe the algorithm in Figure 8.1, and give a sketch of the proof of its correctness for the case of rapidly mixing graphs.

Note that while the number of steps performed in each walk is exactly the same (L), the lengths of the paths they induce (i.e., removing steps in which the walk stays in place) vary. Hence, in particular, there are even-length paths and odd-length paths. Also note that if the graph is bipartite, then it is always accepted, and so we only need to show that if the graph is ϵ-far from bipartite, then it is rejected with probability at least 2/3.

The rapidly mixing case. Assume the graph is rapidly mixing (with respect to L). That is, from each starting vertex s in G, and for every $v \in V$, the probability that a random walk of length $L = \text{poly}((\log n)/\epsilon)$ ends at v is at least $\frac{1}{2n}$ and at most $\frac{2}{n}$ — i.e., approximately the probability assigned by the stationary distribution. (Recall that this ideal case occurs when G is an expander.) Let us fix a particular starting vertex s. For each vertex v, let p_v^0 be the probability that a random walk (of length L) starting from s, ends at v, and corresponds to an even-length path. Define p_v^1 analogously for odd-length paths. Then, by our assumption on G, for every v, $p_v^0 + p_v^1 \geq \frac{1}{2N}$.

We consider two cases regarding the sum $\sum_{v \in V} p_v^0 \cdot p_v^1$ — in case the sum is (relatively) "small", we show that there exists a partition

Algorithm 8.1: Test-Bipartite (for bounded-degree graphs)

- *Repeat* $T = \Theta(\frac{1}{\epsilon})$ *times:*
  1. *Uniformly select* s *in* V.
  2. *If* odd-cycle*(*s*)* *returns* found*, then* reject.
- *In case the algorithm did not reject in any one of the above iterations, it accepts.*

Procedure odd-cycle(s)

1. Let $K \stackrel{\text{def}}{=} \text{poly}((\log n)/\epsilon) \cdot \sqrt{n}$, and $L \stackrel{\text{def}}{=} \text{poly}((\log n)/\epsilon)$.
2. Perform K random walks starting from s, each of length L.
3. If some vertex v is reached (from s) both on a prefix of a random walk corresponding to an even-length path and on a prefix of a walk corresponding to an odd-length path, then return found. Otherwise, return not-found.

Fig. 8.1 The algorithm for testing bipartiteness in bounded-degree graphs.

(V_0, V_1) of V that is ϵ-good, and so G is ϵ-close to being bipartite. Otherwise (i.e., when the sum is not "small"), we show that $\Pr[\textsf{odd-cycle}(s) = \textsf{found}]$ is constant. This implies that if G is ϵ-far from being bipartite, then necessarily the sum is not "small" (or else we would get a contradiction by the first case). But this means that $\Pr[\textsf{odd-cycle}(s) = \textsf{found}]$ is a constant for every s, so that if we select a constant number of starting vertices, with high probability for at least one we'll detect a cycle.

Consider first the case in which $\sum_{v \in V} p_v^0 \cdot p_v^1$ is smaller than $\frac{\epsilon}{c \cdot n}$ for some suitable constant $c > 1$ ($c > 300$ should suffice). Let the partition (V_0, V_1) be defined as follows: $V_0 = \{v : p_v^0 \geq p_v^1\}$ and $V_1 = \{v : p_v^1 > p_v^0\}$. Consider a particular vertex $v \in V_0$. By definition of V_0 and our rapid-mixing assumption, $p_v^0 \geq \frac{1}{4n}$. Assume v has neighbors in V_0, and denote this set of neighbors by $\Gamma_0(v)$, and their number by $d_0(v)$. Then for each

such neighbor $u \in \Gamma_0(v)$, $p_u^0 = p_u^0(L) \geq \frac{1}{4n}$ as well. This can be shown to imply that $p_u^0(L-1) \geq \frac{1}{32n}$. (The high-level idea is to map walks of L steps that end at u and correspond to even-length paths to walks of $L-1$ steps that are obtained by removing one step of staying in place (and hence also correspond to even-length paths).) However, since there is a probability of $\frac{1}{2d}$ of taking a transition from u to v in walks on G, we can infer that each neighbor u contributes $\frac{1}{2d} \cdot \frac{1}{32n}$ to the probability p_v^1. In other words,

$$p_v^1 = \sum_{u \in \Gamma_0(v)} \frac{1}{2d} \cdot p_u^0(L-1) \geq d_0(v) \cdot \frac{1}{64dn}. \tag{8.1}$$

Therefore, if we denote by $d_1(v)$ the number of neighbors that a vertex $v \in V_1$ has in V_1, then

$$\sum_{v \in V} p_v^0 \cdot p_v^1 \geq \sum_{v \in V_0} \frac{1}{4n} \cdot \frac{d_0(v)}{64dn} + \sum_{v \in V_1} \frac{1}{4n} \cdot \frac{d_1(v)}{64dn} \tag{8.2}$$

$$= \frac{1}{c'dn^2} \cdot \left(\sum_{v \in V_0} d_0(v) + \sum_{v \in V_1} d_1(v) \right), \tag{8.3}$$

where c' is a constant. Thus, if there were many (more than ϵdn) violating edges with respect to (V_0, V_1), then the sum $\sum_{v \in V} p_v^0 \cdot p_v^1$ would be at least $\frac{\epsilon}{c'n}$, contradicting our case hypothesis (for $c > c'$).

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

We note that the above analysis can be viewed as following the enforce-and-test approach: a selected vertex s *enforces* a partition, and the walks taken from it *test* the partition.

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

To overcome this difficulty, the analysis of [74] proceeds in a different manner. Let us call a vertex s *good*, if the probability that odd-cycle(s) returns found is at most 0.1. Then, assuming G is accepted with probability greater than $\frac{1}{3}$, all but at most $\frac{\epsilon}{16}$ of the vertices are *good*. It is possible to define a partition in stages as follows. In the first stage we pick any *good* vertex s. What can be shown is that not only is there a set of vertices S that are reached from s with high probability and can be partitioned without many violations (due to the "goodness" of s), but also that there is a small cut between S and the rest of the graph. Thus, no matter how we partition the rest of the vertices, there cannot be many violating edges between S and $V \setminus S$. We therefore partition S (as above), and continue with the rest of the vertices in G.

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

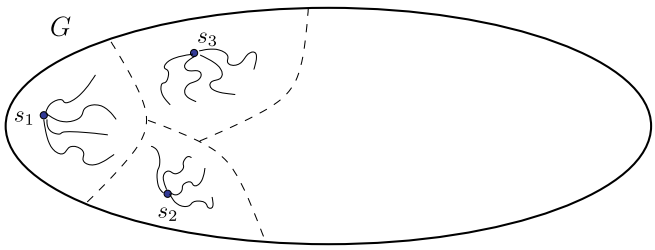

Fig. 8.2 An illustration of several stages of the partitioning process in the case that the graph G passes the test with sufficiently high probability. The vertices s_1, s_2, and s_3 are "good" vertices, where the random walks from each are used to obtain a partition with few violating edges in a part of a graph. The "boundaries" between these parts are marked by broken lines, where the number of edges crossing between parts is relatively small.

between pairs of vertices such that one vertex belongs to S and one to $V(H) \setminus S$. Adding it all together, the total number of violating edges with respect to the final partition is at most ϵdn. For an illustration see Figure 8.2. The core of the proof is hence in proving that indeed for most good vertices s there exists a subset S as defined above. The analysis builds in part on techniques of Mihail [106] who proved that the existence of large cuts (good expansion) implies rapid mixing.

8.2 Testing Expansion

We have seen in the previous subsection that knowing that a graph is an expander can simplify the analysis of a testing algorithm. Here we shortly discuss the problem of testing whether a graph is an expander (using random walks). We say that a graph $G = (V, E)$ is a (γ, α)-*expander* if for every subset $U \subset V$ such that $|U| \geq \gamma n$ (where $n = |V|$), we have that the number of neighbors of vertices in U that are outside of U (i.e., $\big|\{v : v \in V \setminus U, \text{ and } \exists u \in U \text{ s.t. } (u, v) \in E\}\big|$) is at least $\alpha \cdot |U|$. (A closely related definition sets a lower bound on the number of *edges* going out of U relative to $|U|$, but here we use the *vertex-expansion* definition.) When γ is not explicitly specified, then it is assumed to be $1/2$.

The first result concerning testing expansion was a negative one: it was shown in [76] that testing expansion requires $\Omega(\sqrt{n})$ queries (for constant ϵ, γ and α). The lower bound establishes that it is hard to

distinguish in less than $\sqrt{n}/c$ queries (for some constant c), between a random 3-regular graph (which is a very good expander with high probability) and a graph that consists of several such disjoint subgraphs (which is far from being even a weak expander).

In [75] it was conjectured that there is an almost matching upper bound in terms of the dependence on n. Specifically, in [75] a random-walks-based algorithm was proposed. The basic underlying idea of the algorithm is that if a graph is a sufficiently good expander, then the distribution induced by the end-points of random walks (of length that grows logarithmically with n and also depends on the expansion parameter α) is close to uniform. The algorithm performs $\sqrt{n} \cdot \mathrm{poly}(1/\epsilon)$ random walks, and counts the number of collisions (that is, the number of times that the same vertex appears as an end-point of different walks). The algorithm rejects only if this number is above a certain threshold. It was shown in [75] that the algorithm indeed accepts every sufficiently good expander.[1] The harder direction of proving that the algorithm rejects graphs that are far from being good (or even reasonably good) expanders was left open.

Several years later, Czumaj and Sohler [48] made progress on this problem and showed that the algorithm of [75] (with an appropriate setting of the parameters) can distinguish with high probability between an α-expander (with degree-bound d) and a graph that is ϵ-far from being an α' expander for $\alpha' = O\left(\alpha^2/(d^2 \log(n/\epsilon))\right)$. The query complexity and running time of the algorithm are $O\left(d^2\sqrt{n}\log(n/\epsilon)\alpha^{-2}\epsilon^{-3}\right)$. This result was improved by Kale and Seshadhri [94] and by Nachmias and Shapira [108] (who build on an earlier version of [93]) so that $\alpha' = O(\alpha^2/d^2)$ (with roughly the same complexity — the dependence on n is slightly higher, and the dependence on $1/\epsilon$ is lower). It is still open whether it is possible to improve the result further so that it holds for α' that depends linearly on α (thus decreasing the gap between accepted instances and rejected instances).

[1] The notion of expansion considered in [75] was actually the algebraic one, based on the second largest eigenvalue of the graph, but this notion can be translated to vertex expansion.

9

Lower Bounds

We have mentioned several lower bound results along the way, but haven't given any details. Here we give some details for two lower bounds so as to provide some flavor of the types of constructions and arguments used. Note that when dealing with lower bounds, there are two main issues. One is whether or not we allow the algorithm to be adaptive, that is, whether its queries may depend on previous answers, and the second is whether it is allowed to have two-sided error or only one-sided error. Clearly, the strongest type of lower bound is one that holds for adaptive algorithms that are allowed two-sided error, though weaker results may be informative as well.

9.1 A Lower Bound for Testing Triangle-Freeness

Recall that in Section 6 we showed that there is an algorithm for testing triangle-freeness of dense graphs that has a dependence on $1/\epsilon$ that is quite high. While there is no known matching lower bound, we shall show that a super-polynomial dependence on $1/\epsilon$ is necessary [3]. Here we give the proof only for the one-sided error case, and note that it can be extended to two-sided error algorithms [15]. Namely, we shall show

how to construct dense graphs that are ϵ-far from being triangle-free but for which it is necessary to perform a super-polynomial number of queries in order to see a triangle. As shown in [7, 78], if we ignore quadratic factors, we may assume without loss of generality that the algorithm takes a uniformly selected sample of vertices and makes its decision based on the induced subgraph. This "gets rid" of having to deal with adaptivity. Furthermore, since we are currently considering one-sided error algorithms, the algorithm may reject only if it obtains a triangle in the sample.

The construction of [3] is based on graphs that are known as *Behrend Graphs*. These graphs are defined by sets of integers that include no three-term arithmetic progression (abbreviated as 3AP). Namely, these are sets $X \subset \{1,\ldots,m\}$ such that for every three elements $x_1, x_2, x_3 \in X$, if $x_2 - x_1 = x_3 - x_2$ (i.e., $x_1 + x_3 = 2x_2$), then necessarily $x_1 = x_2 = x_3$. Below we describe a construction of such sets that are large (relative to m), and later explain how such sets determine Behrend graphs.

Lemma 9.1. For every sufficiently large m there exists a set $X \subset \{1,\ldots,m\}$, $|X| \geq m^{1-g(m)}$ where $g(m) = o(1)$, such that X contains no three-term arithmetic progression.

In particular, it is possible to obtain $g(m) = c/\sqrt{\log m}$ for a small constant c. We present a simpler proof that gives a weaker bound of $g(m) = O(\log\log\log m/\log\log m)$, but gives the idea of the construction.

Proof. Let $b = \log m$ and $k = \left\lfloor \frac{\log m}{\log b} \right\rfloor - 1$. Since $\log m/\log b = \log m/\log\log m$ we have that $k < b/2$ for every $m \geq 8$. We arbitrarily select a subset of k different numbers $\{x_1,\ldots,x_k\} \subset \{0,\ldots,b/2-1\}$ and define

$$X = \left\{ \sum_{i=1}^{k} x_{\pi(i)} b^i \ : \ \pi \text{ is a permutation of } \{1,\ldots,k\} \right\}. \tag{9.1}$$

By the definition of X we have that $|X| = k!$. By using $z! > (z/e)^z$, we get that

$$\begin{aligned}
|X| = k! &= \left(\left\lfloor \frac{\log m}{\log\log m} \right\rfloor - 1\right)! \\
&> \frac{1}{(\log m/\log\log m)^2} \cdot \frac{\log m}{\log\log m}! && (9.2)\\
&> \left(\frac{\log\log m}{\log m}\right)^2 \cdot \left(\frac{\log m}{e \cdot \log\log m}\right)^{\frac{\log m}{\log\log m}} && (9.3)\\
&= 2^{2(\log\log\log m - \log\log m)} \cdot 2^{\log m \cdot \frac{\log\log m - \log e - \log\log\log m}{\log\log m}} \\
&> m^{1-\frac{\log\log\log m+4}{\log\log m}}. && (9.4)
\end{aligned}$$

Consider any three elements $u, v, w \in X$ such that $u + v = 2w$. By definition of X, these elements are of the form $u = \sum_{i=1}^{k} x_{\pi_u(i)} b^i$, $v = \sum_{i=1}^{k} x_{\pi_v(i)} b^i$, and $w = \sum_{i=1}^{k} x_{\pi_w(i)} b^i \in X$, where π_u, π_v, π_w are permutations over $\{1, \ldots, k\}$. Since $u + v = \sum_{i=1}^{k} (x_{\pi_u(i)} + x_{\pi_v(i)}) b^i$ and $x_i < b/2$ for every $1 \leq i \leq k$, it must be the case that for every i,

$$x_{\pi_u(i)} + x_{\pi_v(i)} = 2x_{\pi_w(i)}. \tag{9.5}$$

This implies that for every i:

$$x^2_{\pi_u(i)} + x^2_{\pi_v(i)} \geq 2x^2_{\pi_w(i)}, \tag{9.6}$$

where the inequality in Equation (9.6) is strict unless $x_{\pi_u(i)} = x_{\pi_v(i)} = x_{\pi_w(i)}$. (This follows from the more general fact that for every convex function f, $\frac{1}{n}\sum_{i=1}^{n} f(a_i) \geq f(\frac{1}{n}\sum_{i=1}^{n} a_i)$.) If we sum over all indices i and there is at least one index i for which the inequality in Equation (9.6) is strict we get that

$$\sum_{i=1}^{k} x^2_{\pi_u(i)} + \sum_{i=1}^{k} x^2_{\pi_v(i)} > \sum_{i=1}^{k} 2x^2_{\pi_w(i)}, \tag{9.7}$$

which is a contradiction since we took permutations of the same numbers. Thus, we get that $u = v = w$. □

Remark. The better construction has a similar form. Define

$$X_{b,B} = \left\{ \sum_{i=1}^{k} x_i b^i \,:\, 0 \le x_i < \frac{b}{2} \text{ and } \sum_{i=0}^{k} x_i^2 = B \right\},$$

where $k = \left\lfloor \frac{\log m}{\log b} \right\rfloor - 1$ as before. Then it can be shown that there exists a choice of b and B for which $|X_{b,B}| \ge \frac{m}{\exp(\sqrt{\log m})} = m^{1-\Theta(1/\sqrt{\log m})}$.

Behrend graphs. Given a set $X \subset \{1,\ldots,m\}$ with no three-term arithmetic progression we define the Behrend graph BG_X as follows. It has $6m$ vertices that are partitioned into three parts: V_1, V_2, and V_3 where $|V_i| = i \cdot m$. For each $i \in \{1,2,3\}$ we associate with each vertex in V_i a different integer in $\{1,\ldots,i \cdot m\}$. The edges of the graph are defined as follows (for an illustration see Figure 9.1):

- The edges between V_1 and V_2. For every $x \in X$ and $j \in \{1,\ldots,m\}$ there is an edge between $j \in V_1$ and $(j+x) \in V_2$;
- The edges between V_2 and V_3. For every $x \in X$ and $j \in \{1,\ldots,2m\}$ there is an edge between $(j+x) \in V_2$ and $(j+2x) \in V_3$; and
- The edges between V_1 and V_3. For every $x \in X$ and $j \in \{1,\ldots,m\}$ there is an edge between $j \in V_1$ and $(j+2x) \in V_3$.

(It is also possible to construct a "nicer", regular graph, by working modulo m.) We shall say that an edge between $j \in V_1$ and $j' \in V_2$ or

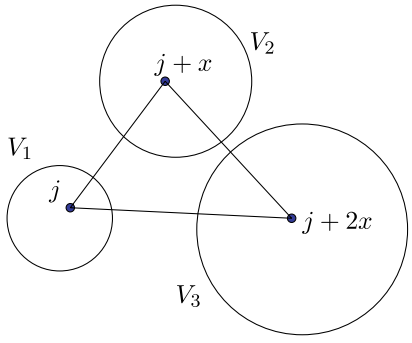

Fig. 9.1 An illustration of the structure of Behrend graphs.

between $j \in V_2$ and $j' \in V_3$ is *labeled* by x, if $j' = (j + x)$, and we shall say that an edge between $j \in V_1$ and $j' \in V_3$ is *labeled* by x, if $j' = (j + 2x)$.

The graph BG_X contains $3|X|m$ edges. Since $|X| = o(m)$ we don't yet have a dense graph (or, more precisely, if ϵ is a constant, then the graph is not ϵ-far from being triangle-free according to the dense-graphs model), but we shall attend to that shortly. For every $j \in \{1, \ldots, m\}$ and $x \in X$, the graph contains a triangle $(j, (j + x), (j + 2x))$ where $j \in V_1$, $(j + x) \in V_2$ and $(j + 2x) \in V_3$. There are $m \cdot |X|$ such edge-disjoint triangles and every edge is part of one such triangle. That is, in order to make the graph triangle-free it is necessary to remove a constant fraction of the edges.

On the other hand, we next show that, based on the assumption that X is 3AP-free, there are no other triangles in the graph. To verify this consider any three vertices j_1, j_2, j_3 where $j_i \in V_i$ and such that there is a triangle between the three vertices. By definition of the graph, $j_2 = (j_1 + x_1)$, for some $x_1 \in X$, $j_3 = (j_2 + x_2)$, for some $x_2 \in X$, and $j_3 = (j_1 + 2x_3)$, for some $x_3 \in X$. Therefore, $(j_1 + x_1 + x_2) = (j_1 + 2x_3)$. That is, we get that $x_1 + x_2 = 2x_3$. Since X contains no three-term arithmetic progression, the last implies that $x_1 = x_2 = x_3$, meaning that the triangle (j_1, j_2, j_3) is of the form $(j, (j + x), (j + 2x))$.

To get a dense graph that is ϵ-far from triangle-free for any given ϵ, we "blow up" the graph BG_X. In "blowing-up" we mean that we replace each vertex in BG_X by an independent set of s vertices (where s will be determined shortly), and we put a complete bipartite graph between every two such "super-vertices". We make the following observations:

- The number of vertices in the resulting graph is $6m \cdot s$, and the number of edges is $3|X|m \cdot s^2$.
- It is necessary to remove $|X|m \cdot s^2$ edges in order to make the graph triangle-free. This follows from the fact that there are $|X|m$ edge-disjoint triangles in BG_X, and when turning them into "super-triangles" it is necessary to remove at least s^2 edges from each super-triangle.
- There are $|X|m \cdot s^3$ triangles in the graph (this follows from the construction of $B_G(x)$, and the blow-up, which replaces each triangle in the original graph with s^3 triangles).

Given ϵ and n, we select m to be the largest integer satisfying $\epsilon \leq m^{-g(m)}/36$. This ensures that m is a super-polynomial function of $1/\epsilon$ (for $g(m) = \Theta(1/\sqrt{\log m})$ we get that $m \geq (c/\epsilon)^{c\log(c/\epsilon)}$ for a constant $c > 0$.) Next we set $s = n/(6m)$ so that $6m \cdot s = n$, and the number of edges that should be removed is

$$|X|m \cdot s^2 \geq m^{2-g(m)} \cdot s^2 = (6ms)^2 \cdot m^{-g(m)}/36 = \epsilon n^2.$$

Finally, if the algorithm takes a sample of size q, then the expected number of triangles in the subgraph induced by the sample is at most

$$q^3 \cdot \frac{|X|ms^3}{n^3} = q^3 \cdot \frac{m^{2-g(m)}}{6^3 m^3} < q^3 \cdot \frac{1}{cm}.$$

If $q < m^{1/3}$ then this is much smaller than 1, implying that w.h.p. the algorithm won't see a triangle. But since m is super-polynomial in $1/\epsilon$, q must be super-polynomial as well.

As noted previously, this lower bound can be extended to hold for two-sided error algorithms [15].

9.2 A Lower Bound for Testing Bipartiteness of Constant Degree Graphs

In this subsection we give a high-level description of the lower bound of $\Omega(\sqrt{n})$ (for constant ϵ) for testing bipartiteness in the bounded-degree model [76]. The lower bound holds for adaptive two-sided error algorithms. We use here some notions (e.g., violating edges) that were introduced in Section 8.1 (where we described an algorithm for testing bipartiteness in the bounded-degree model whose complexity is $O(\sqrt{n} \cdot \text{poly}(\log n/\epsilon))$).

To obtain such a lower bound we define two families of graphs. In one family all graphs are bipartite, and in the other family (almost all) graphs are ϵ-far from bipartite, for some constant ϵ (e.g., $\epsilon = 0.01$). We then show that no algorithm that performs less than $\sqrt{n}/c$ queries (for some constant c) can distinguish with sufficiently high success probability between a graph selected randomly from the first family and a graph selected randomly from the second family. This implies that there is no testing algorithm whose query complexity is at most $\sqrt{n}/c$. We explain how this is proved (on a high level) momentarily, but first we describe the two families.

We assume that the number of vertices, n, is even. Otherwise, the graphs constructed have one isolated vertex, and the constructions are over the (even number of) $n-1$ remaining vertices. In both families all vertices reside on a cycle (since n is assumed to be even, the cycle is of even length). The ordering of the vertices on the cycle is selected randomly among all $n!$ permutations. In both families, in addition to the cycle, we put a random matching between the vertices (thus bringing the degree to 3). The only difference between the families is that in one family the matching is totally random, while in the other it is constrained so that only pairs of vertices whose orderings on the cycle have different parity (e.g., the 2nd vertex and the 5th vertex) are allowed to be matched. In other words, it is a random *bipartite* matching.

Step 1. The first claim that needs to be established is that indeed almost all graphs in the first family are far from being bipartite. This involves a basic counting argument, but needs to be done carefully. Namely, we need to show that with high probability over the choice of the random matching, *every* two-way partition has many (a constant fraction of) violating edges. We would have liked to simply show that for each fixed partition, since we select the matching edges randomly, with very high probability there are many violating edges among them, and then to take a union bound over all two-way partitions. This doesn't quite work, since the number of two-way partitions is too large compared to the probability we get for each partition (that there are many violating edges with respect to the partition). Instead, the counting argument is slightly refined, and in particular, uses the cycle edges as well. The main observation is that we don't actually need to count in the union bound those partitions that already have many violating edges among the cycle edges. The benefit is that the union bound now needs to be over a smaller number of partitions, and the proof of the claim follows.

Step 2. Given an algorithm, we want to say something about the distribution induced on "query–answer" transcripts, when the probability is taken both over the coin flips of the algorithm and over the random choice of a graph (in either one of the two families). We want to

show that if the algorithm asks too few queries, then these transcripts are distributed very similarly. How is such a transcript constructed? At each step the algorithm asks a query (based on the past, with possible randomness) and is given an answer according to the randomly selected graph. The main observation is that for the sake of the analysis, instead of generating the graph randomly and then answering queries, it is possible to generate the graph (according to the correct distribution) *during* the process of answering the algorithm's queries.

To first illustrate this in an easier case (in the dense-graphs model, where the queries are vertex-pair queries), think of selecting a graph randomly by independently letting each pair (u, v) be an edge with probability p. In this case, whenever the algorithm asks a query, the process that generates the graph flips a coin with bias p and answers. Clearly, the distribution over query–answer transcripts is the same if we first construct the graph and then let the algorithm run and perform its queries, or if we construct the graph while answering the algorithm's queries.

Going back to our problem, let's think how this can be done for our graph distributions. In the first query (v, i), both in case the query concerns a cycle edge or a matching edge (we can assume that the algorithm knows the labeling of the three types of edges (e.g., 1 and 2 for cycle edges and 3 for matching edge)), the answer is a uniformly selected vertex u, with the only constraint that $u \neq v$. In general, at any point in time we can define the *knowledge graph* that the algorithm has. As long as the algorithm didn't close a cycle (this will be an important event), the knowledge graph consists of trees (for an illustration, see Figure 9.2). Both processes will attribute to each new vertex that is added to the graph its parity on the cycle. The only difference between the processes is that in the process that constructs a bipartite graph, for each matching-edge query $(v, 3)$, the parity of the matched vertex u is determined by the parity of v, while in the other family there is some probability for each parity (depending on the number of vertices that already have a certain parity).

The crucial point is that for each query (v, i), the probability that the query is answered by a vertex that already appears in the knowledge graph is $O(n'/n)$, where n' is the number of vertices in the knowledge

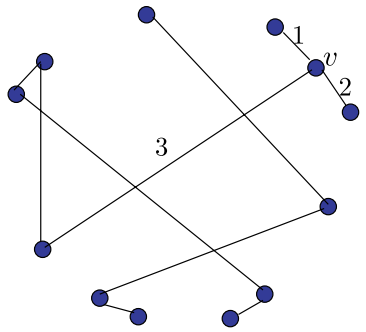

Fig. 9.2 An illustration of the knowledge graph, which consists of trees. The longer lines in the figure correspond to matching edges (labeled by 3), and the shorter lines to cycle edges (labeled by 1 and 2).

graph (and it can be at most twice the number of queries already performed). On the other hand, if the vertex in the answer is not in the knowledge graph, then in both cases it is a uniformly selected vertex. Now, if the total number of queries performed is less than $\sqrt{n}/4$, then the probability that the algorithm gets as an answer a vertex in the knowledge graph, is less than $(\sqrt{n}/4) \cdot (\sqrt{n}/4)/n = 1/16$. Otherwise, the distributions on the query–answer transcripts are identical.

10

Other Results

In this section we describe several families of results that did not fall naturally into the previous sections. The list of results is clearly not comprehensive. In particular we note that one area that was not covered is testing geometric properties (e.g., [45, 49, 55]). See [47] for some works in this area (in the broader context of sublinear-time algorithms).

10.1 Testing Monotonicity

Let X be a partially ordered set (poset) and let R be a fully ordered set. We say that a function $f : X \to R$ is monotone if for every $x, y \in X$ such that x is smaller than y (according to the partial order defined over X) it holds that $f(x)$ is smaller or equal to $f(y)$ (according to the full order defined over R). In what follows we discuss several special case, as well as the general case.

10.1.1 Testing Monotonicity in One Dimension

We start by considering the following problem of testing monotonicity in one dimension, or testing "sortedness" (first studied by Ergun et al. [55]). Let $f : [n] \to R$, where the range R is some fully ordered

set. The function f can also be viewed as a string of length n over R. We say that f is *monotone* (or *sorted*) if $f(i) \leq f(j)$ for all $1 \leq i < j \leq n$. There are actually several algorithms for testing this property [55, 71]. Their query complexity and running time are $O(\log n/\epsilon)$, and $\Omega(\log n)$ queries are necessary for constant ϵ (by combining the non-adaptive lower bound of Ergun et al. [55] with a result of Fischer [58]).

We first show that the "naive" algorithm, which simply takes a uniform sample of indices in $[n]$ and checks whether non-monotonicity is violated (i.e., whether in the sample there are $i < j$ such that $f(i) > f(j)$), requires $\Omega(\sqrt{n})$ queries for constant ϵ. To see why this is true, consider the function $f(i) = i + 1$ for odd i, $1 \leq i \leq n-1$, and $f(i) = i - 1$ for even i, $2 \leq i \leq n$. That is, the string corresponding to f is (assuming for simplicity that n is even): $2, 1, 4, 3, \ldots, n, n-1$. We call each pair $i, i+1$ where $f(i) = i + 1$ and $f(i+1) = i$ a *matched pair*. Note that the algorithm rejects only if it gets a matched pair in the sample. On one hand, this function is $1/2$-far from being monotone, because in order to make it monotone it is necessary to modify its value on at least one member of each matched pair. On the other hand, by the (lower bound part of the) birthday paradox, the probability that a uniform sample of size $s \leq \sqrt{n}/2$ contains a matched pair is less than $1/3$.

In Figure 10.1 we give the "binary-search-based" algorithm of [55], where we assume without loss of generality that all function values are distinct. This assumption can be made without loss of generality because if this is not the case then we can replace each value $f(i)$ with

Algorithm 10.1: Testing Monotonicity for $f : [n] \to R$

- *Uniformly and independently at random select $s = 2/\epsilon$ indices $i_1, \ldots, i_s$.*
- *For each index i_r selected, query $f(i_r)$, and perform a binary search on f for $f(i_r)$ (recall that f can be viewed as a string or array of length n).*
- *If the binary search failed for any i_r, then output* reject. *Otherwise output* accept.

Fig. 10.1 Monotonicity testing algorithm for $f : [n] \to R$.

$f'(i) = (f(i), i)$, where $(f(i), i) < (f(j), j)$ if and only if either $f(i) < f(j)$ or $f(i) = f(j)$ but $i < j$. The distance to monotonicity of the new function f' is the same as the distance to monotonicity of the original function f.

Clearly, if the function f is monotone, then the algorithm accepts with probability 1 since every possible binary search must succeed. Assume from this point on that f is ϵ-far from being monotone. We show that the algorithm rejects with probability at least 2/3.

We say that an index $j \in [n]$ is a *witness* (to the non-monotonicity of f), if a binary search for $f(j)$ fails.

Lemma 10.1. If f is ϵ-far from being monotone, then there are at least ϵn witnesses.

Proof. Assume, contrary to the claim, that there are less than ϵn witnesses. We shall show that f is ϵ-close to being monotone, in contradiction to the premise of the lemma. Specifically, we shall show that if we consider all non-witnesses, then they constitute a monotone sequence. For each pair of non-witnesses, j, j' where $j < j'$, consider the steps of the binary search for $f(j)$ and $f(j')$, respectively. Let u be the first index for which the two searches diverge. Namely, $j \leq u \leq j'$ (where at least one of the inequalities must be strict because $j < j'$) and $f(j) \leq f(u) \leq f(j')$ (where again at least one of the inequalities must be strict since the function values are distinct). But then $f(j) < f(j')$, as required. Now we are done since by modifying each witness to obtain the value of the nearest non-witness, we can make f into a monotone function. The total number of modifications equals the number of witnesses, which is at most ϵn. □

Corollary 10.2. If f is ϵ-far from monotone then Algorithm 10.1.1 rejects f with probability at least 2/3.

Proof. The probability that the algorithm does not reject (i.e., accepts) equals the probability that no witness is selected in the sample.

This probability is upper bounded by $(1-\epsilon)^s < e^{-\epsilon s} = e^{-2} < 1/3$, as required. □

10.1.2 Testing Monotonicity in Higher Dimensions

For a function $f : [n]^m \to R$ (where, as before, the range R is a fully ordered set), we say that f is monotone if $f(x) \leq f(y)$ for every x, y such that $x \prec y$, where $\prec$ denotes the natural partial order over strings (that is, $x_1 \dots x_m \prec y_1 \dots y_m$ if $x_i \leq y_i$ for every $i \in [m]$, and $x_i < y_i$ for at least one $i \in [m]$). Let $\epsilon_{\mathrm{M}}(f)$ denote the distance of f from the closest monotone function (with the same domain and range).

Batu et al. [28] extended the algorithm of [55] to higher dimensions, at an exponential cost in the dimension m. The complexity of their algorithm is $O((2\log n)^m \epsilon^{-1})$. Halevy and Kushilevitz [81] reduced the complexity (for sufficiently large n) to $O(m4^m \log n \epsilon^{-1})$, and Ailon and Chazelle [1] further improved this bound to $O(m2^m \log n \epsilon^{-1})$.

Dodis et al. [54] showed that it is possible to obtain a linear dependence on m at a cost of a logarithmic dependence on $|R|$, where the dependence on $\log|R|$ can be replaced by $m \log n$. An outline of the algorithm (Algorithm 10.2) is given in Figure 10.2. The complexity of Algorithm 10.2 depends on the complexity of the test performed in Step 1b and on the probability that it rejects a uniformly selected $f_{i,\alpha,\beta}$, which is then used to set $t(m, \epsilon, n)$.

Algorithm 10.2: (Outline of) Testing Monotonicity for $f : [n]^m \to R$

1. *Repeat the following $t(m, \epsilon, n)$ times:*
   (a) *Uniformly select $i \in [m]$, $\alpha \in [n]^{i-1}$ and $\beta \in [n]^{m-i}$.*
   (b) *Perform a test on the one-dimensional function $f_{i,\alpha,\beta} : [n] \to R$ that is defined by $f_{i,\alpha,\beta}(x) = f(\alpha x \beta)$.*
2. *If no test caused rejection then accept.*

Fig. 10.2 Monotonicity testing algorithm (outline) for $f : [n]^m \to R$.

We note that the high-level structure of the algorithm is reminiscent of the algorithm of [123] for testing multivariate polynomials over large fields (shortly discussed in Section 3.2). Recall that their algorithm considers restrictions of the tested function f to random lines, and checks that each restriction is a univariate polynomial of bounded degree.

We first consider in more detail the case of testing monotonicity of Boolean functions over m bit strings (that is, over the m-dimensional Boolean hypercube), and later talk about the general case. That is, we consider testing a function $f : \{0,1\}^m \to \{0,1\}$ (which is equivalent to testing $f : [n]^m \to R$ for $n = 2$ and any R such that $|R| = 2$). Observe that in this case, since the size of the domain of each $f_{i,\alpha,\beta}$ is 2, the one-dimensional test in Step 1b of Algorithm 10.2 simply checks whether $f(i,\alpha,\beta)(0) \leq f(i,\alpha,\beta)(1)$, or equivalently, whether $f(\alpha 0 \beta) \leq f(\alpha 1 \beta)$ (as must be the case if the function is monotone).

Thus, similarly to the tests for linearity and low-degree polynomials, we consider a characterization of monotone functions and show that it is *robust* (though less robust than the characterization we had for linear functions). The characterization is that a function f is monotone if and only if for every pair $x, y \in \{0,1\}^m$ that differ only on the i-th bit for some $1 \leq i \leq m$, where $x_i = 0$ and $y_i = 1$ (so that $x \prec y$), it holds that $f(x) \leq f(y)$. Clearly, if f is monotone, then the above holds, and the other direction is also easy to verify.[1] Algorithm 10.2 for the special case of $f : \{0,1\}^m \to \{0,1\}$ thus becomes the algorithm given in Figure 10.3.

As noted before, if f is monotone then the algorithm accepts with probability 1. We would like to show that if $\epsilon_{\mathrm{M}}(f) > \epsilon$ then the algorithm rejects with probability at least 2/3. To this end we define:

$$U \stackrel{\text{def}}{=} \{(x,y) : x \text{ and } y \text{ differ on a single bit and } x \prec y\}, \tag{10.1}$$

as the set of all *neighboring* pairs on the m-dimensional hypercube (where $|U| = 2^{m-1} \cdot m$),

$$V(f) \stackrel{\text{def}}{=} \{(x,y) \in U : f(x) > f(y)\} \tag{10.2}$$

[1] Given $x \prec y$ that differ on more than one bit, consider a sequence of intermediate points between x and y (according to the partial order) where every two consecutive points in this sequence differ on a single bit.

Algorithm 10.3: Testing Monotonicity for $f : \{0,1\}^m \to \{0,1\}$

1. *Repeat the following* $\Theta(m/\epsilon)$ *times:*
   (a) *Uniformly select* $i \in [m]$ *and* $x \in \{0,1\}^m$ *such that* $x_i = 0$.
   (b) *Let* $y = x_1 \dots x_{i-1} 1 x_{i+1} \dots x_m$ *(that is,* y *is obtained by flipping the* i*-th bit of* x*).*
   (c) *If* $f(x) > f(y)$ *then* reject *(and exit).*
2. *If no test caused rejection then* accept.

Fig. 10.3 Monotonicity testing algorithm for $f : \{0,1\}^m \to \{0,1\}$.

as the set of all *violating* neighboring pairs, and

$$\eta(f) \stackrel{\text{def}}{=} \frac{|V(f)|}{|U|} = \Pr_{(x,y)\in U}[f(y) < f(x)] \tag{10.3}$$

as the probability that a neighboring pair is violating. The main lemma is:

Lemma 10.3. For every $f : \{0,1\}^m \to \{0,1\}$, $\eta(f) \geq \frac{\epsilon_{\mathrm{M}}(f)}{m}$.

The correctness of the algorithm directly follows from Lemma 10.3, since the algorithm uniformly selects $\Theta(n/\epsilon) = \Omega(1/\eta(f))$ (assuming $\epsilon_{\mathrm{M}}(f) > \epsilon$) pairs $(x,y) \in U$ and checks whether $(x,y) \in V(f)$. We will sketch how f can be turned into a monotone function by performing at most $m \cdot \eta(f) \cdot 2^m$ modifications. Since $\epsilon_{\mathrm{M}}(f) \cdot 2^m$ is the minimum number of modifications required to make f monotone, Lemma 10.3 follows. We first add a few more notations and definitions.

Definition 10.1. For any $i \in [m]$, we say that a function $h : \{0,1\}^m \to \{0,1\}$ is monotone in dimension i, if for every $\alpha \in \{0,1\}^{i-1}$ and $\beta \in \{0,1\}^{m-i}$, $h(\alpha 0 \beta) \leq h(\alpha 1 \beta)$. For a set of indices $T \subseteq [m]$, we say that h is monotone in dimensions T, if for every $i \in T$, the function h is monotone in dimension i.

We next define a *switch* operator, S_i that transforms any function h to a function $S_i(h)$ that is monotone in dimension i.

Definition 10.2. Let $h : \{0,1\}^m \to \{0,1\}$. For every $i \in [m]$, the function $S_i(h) : \{0,1\}^m \to \{0,1\}$ is defined as follows: For every $\alpha \in \{0,1\}^{i-1}$ and every $\beta \in \{0,1\}^{m-i}$, if $h(\alpha\,0\,\beta) > h(\alpha\,1\,\beta)$ then $S_i(h)(\alpha\,0\,\beta) = h(\alpha\,1\,\beta)$, and $S_i(h)(\alpha\,1\,\beta) = h(\alpha\,0\,\beta)$. Otherwise, $S_i(h)$ is defined as equal to h on the strings $\alpha\,0\,\beta$ and $\alpha\,1\,\beta$.

Let

$$D_i(f) \stackrel{\text{def}}{=} |\{x : S_i(f)(x) \neq f(x)\}| \tag{10.4}$$

so that $D_i(f)$ is twice the number of pairs in $V(f)$ that differ on the i-th bit (and $\sum_{i=1}^{n} D_i(f) = 2 \cdot |V(f)|$).

Lemma 10.4. For every $h : \{0,1\}^m \to \{0,1\}$ and for every $i, j \in [m]$, $D_i(S_j(h)) \leq D_i(h)$.

As a direct corollary of Lemma 10.4 (applying it to the special case that $D_i(h) = 0$ for every i in a subset T) we get:

Corollary 10.5. For every $h : \{0,1\}^m \to \{0,1\}$ and $j \in [m]$, if h is monotone in dimensions $T \subseteq [m]$, then $S_j(h)$ is monotone in dimensions $T \cup \{j\}$.

We won't prove Lemma 10.4 here but we shall show how Lemma 10.3 (and hence the correctness of the algorithm) follows. Let $g = S_n(S_{n-1}(\cdots(S_1(f))\cdots)$. By the definition of g,

$$\text{dist}(f,g) \;\leq\; \frac{1}{2^m} \cdot \sum_{i=1}^{m} D_i(S_{i-1}(\cdots(S_1(f))\cdots). \tag{10.5}$$

By successive applications of Lemma 10.4,

$$D_i(S_{i-1}(\cdots(S_1(f))\cdots) \;\leq\; D_i(S_{i-2}(\cdots(S_1(f))\cdots) \;\leq\; \cdots \;\leq\; D_i(f), \tag{10.6}$$

and so (by combining Equations (10.5) and (10.6)),

$$\text{dist}(f,g) \;\leq\; \frac{1}{2^m} \cdot \sum_{i=1}^{m} D_i(f). \tag{10.7}$$

By successive application of Corollary 10.5, the function g is monotone, and hence $\text{dist}(f,g) \geq \epsilon_{\text{M}}(f)$. Therefore,

$$\sum_{i=1}^{m} D_i(f) \geq \text{dist}(f,g) \cdot 2^m \geq \epsilon_{\text{M}}(f) \cdot 2^m. \tag{10.8}$$

On the other hand, by definition of $D_i(f)$,

$$\sum_{i=1}^{m} D_i(f) \;=\; 2 \cdot |V(f)| \;=\; 2 \cdot \eta(f) \cdot |U| \;=\; \eta(f) \cdot 2^m \cdot m. \tag{10.9}$$

Lemma 10.3 follows by combining Equations (10.8) and (10.9).

Extending the Result to $f : [n]^m \to \{0,1\}$ for $n > 2$. In this case it is possible to define one-dimensional tests (for Step 1b in Algorithm 10.2) that select a pair of points $\sigma < \tau$ in $[n]$ according to particular distributions and check whether $f_{i,\alpha,\beta}(\sigma) \leq f_{i,\alpha,\beta}(\tau)$. By extending the analysis from the case $n = 2$, and in particular modifying the *switching* operator $S_i(\cdot)$ to a *sorting* operator, it can be shown that for some distributions, $\Theta(m(\log n)\epsilon^{-1})$ such tests suffice (and for another distribution, $\Theta((m/\epsilon)^2)$ tests suffice (the latter is for the uniform distribution on pairs)).

Extending the Result to $f : [n]^m \to R$ for $n \geq 2$ and $R > 2$. By performing a *range reduction*, it is shown in [54] that $O(m \log n \log |R| \epsilon^{-1})$ queries suffice.

General Posets. Fischer et al. [63] consider the more general case in which the domain is any poset (and not necessary $[n]^m$). They show that testing monotonicity of Boolean functions over general posets is equivalent to the problem of testing 2CNF assignments (namely, testing whether a given assignment satisfies a fixed 2CNF formula or is far from any such assignment). They also show that for every poset it is possible

to test monotonicity over the poset with a number of queries that is sublinear in the size of the domain poset; specifically, the complexity grows like a square root of the size of the poset. Finally, they give some efficient algorithms for several special classes of posets (e.g., posets that are defined by trees).

10.2 Testing in the General-Graphs Model

Recall that in the general-graphs model the distance measure between graphs (and hence to a property) is defined with respect to the number of edges in the graph (or an upper bound on this number), as in the sparse-graphs model. Since the algorithm may have to deal with graphs of varying density, it is allowed to perform both neighbor queries and vertex queries (as well as degree queries).

Testing Bipartiteness. The general-graphs model was first studied by Kaufman et al. [96]. Their focus was on the property of bipartiteness, which exhibits the following interesting phenomenon. As shown in Section 4.2, for dense graphs there is an algorithm whose query complexity is $\text{poly}(1/\epsilon)$ [10, 72]. In contrast, as sketched in Section 9.2, for bounded-degree graphs there is a lower bound of $\Omega(\sqrt{n})$ [76] (and, as described in Section 8.1, there is an almost matching upper bound [74]). The question Krivelevich et al. asked is: what is the complexity of testing bipartiteness in *general* graphs (using the general model)?

They answer this question by describing and analyzing an algorithm for testing bipartiteness in general graphs whose query complexity (and running time) is $O(\min(\sqrt{n}, n^2/m) \cdot \text{poly}(\log n/\epsilon))$. Thus, as long as the average degree of the graph is $O(\sqrt{n})$, the running time (in terms of the dependence on n) is $\tilde{O}(\sqrt{n})$, and once the average degree goes above this threshold, the running time starts decreasing.

Krivelevich et al. first consider the case that the graph is *almost regular*. That is, the maximum degree d and the average degree d_{avg} are of the same order. They later showed how to reduce the problem of testing bipartiteness of general graphs (where d may be much larger than d_{avg}) to bipartiteness of almost-regular graphs. This reduction involves emulating the execution of the algorithm on an "imaginary"

almost-regular graph where the queries to this imaginary graph can be answered by performing queries to the "real" graph G.

The algorithm for almost-regular graphs builds on the testing algorithm for bipartiteness of bounded-degree graphs [74] (which is described in Section 8.1 and whose query complexity is $O(\sqrt{n} \cdot \text{poly}(\log n/\epsilon))$). In fact, as long as $d \leq \sqrt{n}$, the algorithm is equivalent to the algorithm in [74]. In particular, as in [74], the algorithm selects $\Theta(1/\epsilon)$ *starting vertices* and from each it performs several random walks (using neighbor queries), each walk of length $\text{poly}(\log n/\epsilon)$. If $d \leq \sqrt{n}$ then the number of these walks is $O(\sqrt{n} \cdot \text{poly}(\log n/\epsilon))$, and the algorithm simply checks whether an odd-length cycle was detected in the course of these random walks.

If $d > \sqrt{n}$ then there are two important modifications: (1) the number of random walks performed from each vertex is reduced to $O(\sqrt{n/d} \cdot \text{poly}(\log n/\epsilon))$; and (2) for each pair of end vertices that are reached by walks that correspond to paths whose lengths have the same parity, the algorithm performs a vertex-pair query. Similarly to the $d \leq \sqrt{n}$ case, the graph is rejected if an odd-length cycle is found in the subgraph induced by all queries performed.

Krivelevich et al. also present an almost matching lower bound of $\Omega(\min(\sqrt{n}, n^2/m))$ (for a constant ϵ). This bound holds for all testing algorithms (that is, for those which are allowed a two-sided error and are adaptive). Furthermore, the bound holds for regular graphs.

Testing Triangle-Freeness. Another property that was studied in the general-graphs model is testing triangle-freeness (and more generally, subgraph-freeness) [9]. Recall that for this property there is an algorithm in the dense-graphs model whose complexity depends only on $1/\epsilon$ [7] (see Section 6.3), and the same is true for constant degree graphs [76]. Here too the question is what is the complexity of testing the property in general graphs. In particular this includes graphs that are sparse (that is, $m = O(n)$), but do not have constant degree.

The main finding of Alon et al. [9] is a lower bound of $\Omega(n^{1/3})$ on the necessary number of queries for testing triangle-freeness that holds whenever the average degree d_{avg} is upper bounded by $n^{1-\nu(n)}$, where $\nu(n) = o(1)$. Since when $d = \Theta(n)$ the number of queries sufficient for

testing is independent of n [7], we observe an abrupt, *threshold-like* behavior of the complexity of testing around n. Additionally, they provide sublinear upper bounds for testing triangle-freeness that are at most quadratic in the corresponding lower bounds (which vary as a function of the graph density).

Testing k-colorability. Finally, a study of the complexity of testing k-colorability (for $k \geq 3$) is conducted by Ben-Eliezer et al. [31]. For this property there is an algorithm with query complexity $\text{poly}(1/\epsilon)$ in the dense-graphs model [72, 10] (where the algorithm uses only vertex-pair queries), and there is a very strong lower bound of $\Omega(n)$ for testing in the bounded-degree model [36] (where the algorithm uses neighbor queries). Ben-Eliezer et al. consider the complexity of testing k-colorability as a function of the average degree d_{avg} in models that allow different types of queries (and in particular may allow only one type of query). In particular they show that while for vertex-pair queries, testing k-colorability requires a number of queries that is a monotone decreasing function in the average degree d_{avg}, the query complexity in the case of neighbor queries remains roughly the same for every density and for large values of k. They also study a new, stronger, query model, which is related to the field of Group Testing.

10.3 Testing Membership in Regular Languages and Other Languages

Alon et al. [12] consider the following problem of testing membership in a regular language. For a predetermined regular language $L \subseteq \{0,1\}^*$, the tester for membership in L should accept every word $w \in L$ with probability at least 2/3, and should reject with probability at least 2/3 every word w that differs from any $w' \in L$ on more than $\epsilon|w|$ bits. We stress that the task is not to decide whether a language is regular, but rather the language is predetermined, and the test is for membership in the language.

The query complexity and running time of the testing algorithm for membership in a regular language is $\tilde{O}(1/\epsilon)$, that is, independent of the length n of w. (The running time is dependent on the size of the

(smallest) finite automaton accepting L, but this size is considered to be a fixed constant with respect to n.) Alon et al. [12] also show that a very simple context-free language (of all strings of the form $vv^R uu^R$, where w^R denotes the reversal of a string w) cannot be tested using $o(\sqrt{n})$ queries.

One important subclass of the context-free languages is the Dyck language, which includes strings of properly balanced parentheses. Strings such as "(( )( ))" belong to this class, whereas strings such as "(( )" or ") (" do not. If we allow more than one type of parenthesis then "([ ])" is a balanced string but "([ )]" is not. Formally, the Dyck language D_m contains all balanced strings that contain at most m types of parentheses. Thus, for example "(( )( ))" belongs to D_1 and "([ ])" belongs to D_2. Alon et al. [12] show that membership in D_1 can be tested by performing $\tilde{O}(1/\epsilon)$ queries, whereas membership in D_2 cannot be tested by performing $o(\log n)$ queries.

Parnas et al. [114] present an algorithm that tests whether a string w belongs to D_m. The query complexity and running time of the algorithm are $\tilde{O}\left(n^{2/3}/\epsilon^3\right)$, where n is the length of w. The complexity does not depend on m, the number of different types of parentheses. They also prove a lower bound of $\Omega(n^{1/11}/\log n)$ on the query complexity of any algorithm for testing D_m for $m > 1$. Finally, they consider the context-free language for which Alon et al. [12] gave a lower bound of $\Omega(\sqrt{n})$: $L_{\text{REV}} = \{uu^r vv^r : u, v \in \Sigma^*\}$. They show that L_{REV} can be tested in $\tilde{O}(\frac{1}{\epsilon}\sqrt{n})$ time, thus almost matching the lower bound.

Newman [110] extend the result of Alon et al. [12] for regular languages and give an algorithm that has query complexity $\text{poly}(1/\epsilon)$ for testing whether a word w is accepted by a given constant-width oblivious read-once branching program. (It is noted in [38] that the result can be extended to the non-oblivious case.) On the other hand, Fischer et al. [65] show that testing constant-width oblivious read-*twice* branching programs requires $\Omega(n^\delta)$ queries, and Bollig [38] shows that testing read-once branching programs of quadratic size (with no bound on the width) requires $\Omega(n^{1/2})$ queries (improving on [39]).

In both [65] and [38] lower bounds for membership in sets defined by CNF formulas are also obtained, but the strongest result is in [32]: an $\Omega(n)$ lower bound for 3CNF (over n variables). This should be

contrasted with an $O(\sqrt{n})$ upper bound that holds for 2CNF [63]. More generally, Ben-Sasoon et al. [32] provide sufficient conditions for linear properties to be hard to test, where a property is linear if its elements form a linear space.

11

Extensions, Generalizations, and Related Problems

11.1 Distribution-Free Testing

The notion of distribution-free testing was already introduced and discussed in Section 3.3.1 (in the context of applications of self-correcting). Here we mention a few other results in this model.

In addition to the result described in Section 3.3.1, Halevy and Kushilevitz [81, 84] describe a distribution-free monotonicity testing algorithm for functions $f : [n]^m \to R$ with query complexity $O((2\log n)^m/\epsilon)$. Note that the complexity of the algorithm has exponential dependence on the dimension m of the input. This is in contrast to some of the standard testing algorithms [54, 71] where the dependence on m is linear (to be precise, the complexity is $O(m \log n \log |R|/\epsilon)$, where $|R|$ is the effective size of the range of the function, that is, the number of distinct values of the function). In a further investigation of distribution-free testing of monotonicity [83, 84], Halevy and Kushilevitz showed that the exponential dependence on m is unavoidable even in the case of Boolean functions over the Boolean hypercube (that is, $f : \{0,1\}^m \to \{0,1\}$).

Motivated by positive results for standard testing of several classes of Boolean functions (as described in Section 5) Glasner and Servedio [68] ask whether these results can be extended to the distribution-free model of testing. Specifically, they consider monotone and general monomials (conjunction), decisions lists, and linear threshold functions. They prove that for these classes, in contrast to standard testing, where the query complexity does not depend on the number of variables n, every distribution-free testing algorithm must make $\Omega((n/\log n)^{1/5})$ queries (for constant ϵ). While there is still a gap between this lower bound and the upper bound implied by learning these classes, a strong dependence on n is unavoidable in the distribution-free case.

Finally we note that Halevy and Kushilevitz [82] also study distribution-free testing of graph properties in sparse graphs, and give an algorithm for distribution-free testing of connectivity, with similar complexity to the standard testing algorithm for this property.

11.2 Testing in the Orientation Model

In the *orientation model*, introduced by Halevy et al. [85], there is a fixed and known underlying undirected graph G. For an unknown orientation of the edges of G (that is, each edge has a direction), the goal is to determine whether the resulting directed graph has a prespecified property or is far from having it. Here distance is measured as the fraction of edges whose orientation should be flipped (edges cannot be removed or added). To this end, the algorithm may query the direction of edges of its choice. Note that since the underlying undirected graph G is known in advance, the model allows to perform any preprocessing on G with no cost in terms of the query complexity of the algorithm.

Halevy et al. [85] first show the following relation between the orientation model and the dense-graphs model: for every graph property $\mathcal{P}$ there is a property of orientations $\vec{\mathcal{P}}$ together with an underlying graph G, such that $\mathcal{P}$ is testable in the dense-graphs model if and only if $\vec{\mathcal{P}}$ is testable in the orientation model (with respect to G). They also study the following properties in orientation model: being *drain-source-free*, being H-free for a fixed forbidden digraph H, and being strongly connected.

In follow-up work, Halevy et al. [86] study testing properties of constraint graphs. Here, each of the two orientations of an edge is thought of as an assignment of 0 or 1 to a variable associated with the edge. A property is defined by the underlying graph and a function on each vertex, where the arity of the function is the degree of the vertex. An assignment to the variables (an orientation of the graph) has the property if the function at every vertex is satisfied by the assignment to its incident edges. The main result in [86] is that for a certain family of such constraint graphs it is possible to test whether an assignment to the edges satisfies all constraints or is ϵ-far from any satisfying assignment by performing $2^{\tilde{O}(1/\epsilon)}$ queries. This result has several implications, among them that for every read-twice CNF formula ϕ it is possible to test assignments for the property of satisfying ϕ by performing $2^{\tilde{O}(1/\epsilon)}$ queries to the assignment. This positive result stands in contrast to the negative results of [63] and [32] for testing satisfiability of slightly more general CNF formula.

Chakroborty et al. [41] consider a property of orientations that was proposed in [85]: testing *st*-connectivity. They give a one-sided error algorithm for testing *st*-connectivity in the orientation model whose query complexity is double-exponential in $1/\epsilon^2$. Interestingly, the algorithm works by reducing the *st*-connectivity testing problem to the problem of testing languages that are decidable by branching problems, where this problem was solved by Newman [110] (as mentioned in Section 10.3).

Another natural property of orientations, which was suggested in [86], is testing whether an orientation is Eulerian. As mentioned briefly in Section 7.5, it is possible to test whether an undirected graph is Eulerian by performing $\text{poly}(1/\epsilon)$ queries, both in the bounded-degree model [76] and in the sparse (unbounded-degree) model [112]. These results can be extended to directed graphs [111]. Unfortunately, in the orientation model there is no algorithm for testing whether an orientation is Eulerian whose query complexity is $\text{poly}(1/\epsilon)$ in general. Fischer et al. [62] show that for general graphs there is a lower bound of $\Omega(m)$ (where m is the number of graph edges) for one-sided error testing. For bounded-degree graphs they give a lower bound of $\Omega(m^{1/4})$ for non-adaptive one-sided error testing, and an $\Omega(\log m)$ lower bound

for one-sided error adaptive testing. For two-sided error testing the lower bounds are roughly logarithmic functions of the corresponding one-sided error lower bounds (in the case of bounded-degree graphs).

Their upper bound for general graphs is $O((dm \log m)^{2/3}\epsilon^{-4/3})$ for one-sided error testing, and $\min\{\tilde{O}(d^{1/3}m^{2/3}\epsilon^{-4/3}), \tilde{O}(d^{3/16}m^{3/4}\epsilon^{-5/4})\}$ for two-sided error testing (where d is the maximum degree in the graph). They also give more efficient algorithms for special cases. In particular, if the graph is an α-expander, then the complexity depends only on d and $1/(\epsilon\alpha)$, where the dependence is linear.

11.3 Tolerant Testing and Distance Approximation

Two natural extensions of property testing, first explicitly studied in [115], are *tolerant testing* and *distance approximation*. A tolerant property testing algorithm is required to accept objects that are ϵ_1-close to having a given property $\mathcal{P}$ and reject objects that are ϵ_2-far from having property $\mathcal{P}$, for $0 \leq \epsilon_1 < \epsilon_2 \leq 1$. Standard property testing refers to the special case of $\epsilon_1 = 0$. Ideally, a tolerant testing algorithm should work for any given $\epsilon_1 < \epsilon_2$, and have complexity that depends on $\epsilon_2 - \epsilon_1$. However, in some cases the relation between ϵ_1 and ϵ_2 may be more restricted (e.g., $\epsilon_1 = \epsilon_2/2$). A closely related notion is that of *distance approximation* where the goal is to obtain an estimate of the distance that the object has to a property. In particular, we would like the estimate to have an additive error of at most δ for a given error parameter δ, or we may also allow a multiplicative error.[1]

In [115] it was first observed that some earlier works imply results in these models. In particular this is true for coloring and other partition problems on dense graphs [72], connectivity of sparse graphs [42], edit distance between strings [24] and L_1 distance between distributions [26] (which will be discussed in Section 11.4). The new results obtained in [115] are for monotonicity of functions $f : [n] \to R$, and clusterability of a set of points. The first result was later improved in [2] and extended to higher dimensions in [56].

[1] We note that if one does not allow an additive error (that is, $\delta = 0$), but only allows a multiplicative error, then a dependence on the distance that the object has to the property must be allowed.

In [60] it is shown that there exist properties of Boolean functions for which there exists a test that makes a constant number of queries, yet there is no such tolerant test. In contrast, in [64] it is shown that *every* property that has a testing algorithm in the dense-graphs model whose complexity is only a function of the distance parameter ϵ, has a distance approximation algorithm with an additive error δ in this model, whose complexity is only a function of δ.[2] Distance approximation in sparse graphs is studied in [105]. Guruswami and Rudra [80] present tolerant testing algorithms for several constructions of locally testable codes, and Kopparty and Saraf [103] study tolerant linearity testing under general distributions and its connection to locally testable codes.

11.4 Testing and Estimating Properties of Distributions

In this subsection we discuss a research direction that is closely related to property testing (where some of the problems can be viewed as actually falling into the property testing framework).

Given access to samples drawn from an unknown distribution $\boldsymbol{p}$ (or several unknown distributions, $\boldsymbol{p}^1,\ldots,\boldsymbol{p}^m$) and some measure over distributions (respectively, m-tuples of distributions), the goal is to approximate the value of this measure for the distribution $\boldsymbol{p}$ (respectively, the distributions $\boldsymbol{p}^1,\ldots,\boldsymbol{p}^m$), or to determine whether the value of the measure is below some threshold α or above some threshold β. In either case, the algorithm is allowed a constant failure probability.[3] For example, given access to samples drawn according to two distributions $\boldsymbol{p}$ and $\boldsymbol{q}$, we may want to decide whether $|\boldsymbol{p}-\boldsymbol{q}| \leq \alpha$ or $|\boldsymbol{p}-\boldsymbol{q}| > \beta$ (for certain settings of α and β). The goal is to perform the task by observing a number of samples that is sublinear in the size of the domain over which the distribution(s) is (are) defined. In what follows, the running times of the algorithms mentioned are linear (or almost linear) in their respective sample complexities. We shortly review the known results

[2] The dependence on δ may be quite high (a tower of height polynomial in $1/\delta$), but there is *no* dependence on the size of the graph.

[3] An alternative model may allow the algorithm to obtain the probability that the distribution assigns to any element of its choice. We shall not discuss this model.

and then give details for one of the results: approximating the entropy of a distribution.

11.4.1 Summary of Results

Testing that distributions are close. Batu et al. [26] consider the problem of determining whether the distance between a pair of distributions over n elements is small (less than $\max\left\{\frac{\epsilon}{4\sqrt{n}}, \frac{\epsilon^2}{32n^{1/3}}\right\}$), or large (more than ϵ) according to the L_1 distance. They give an algorithm for this problem that takes $O(n^{2/3}\log n/\epsilon^4)$ independent samples from each distribution. This result is based on testing closeness according to the L_2 distance, which can be performed using $O(1/\epsilon^4)$ samples. This in turn is based on estimating the deviation of a distribution from uniform (which we mentioned in Section 8.2 in the context of testing expansion) [75].

In recent work (discussed in more detail below), Valiant [126] shows that $\Omega(n^{2/3})$ samples are also necessarily for this testing problem (with respect to the L_1 distance). For the more general problem of distinguishing between the case that the two distributions are ϵ_1-close and the case that they are ϵ_2-far, where ϵ_1 and ϵ_2 are both constants, Valiant [126] proves an almost linear (in n) lower bound.

One can also consider the problem of testing whether a distribution $\boldsymbol{p}$ is close to a fixed and known distribution $\boldsymbol{q}$, or is far from it (letting $\boldsymbol{q}$ be the uniform distribution is a special case of this problem). Batu et al. [25] show that it is possible to distinguish between the case that the distance in L_2 norm is $O\left(\frac{\epsilon^3}{\sqrt{n}\log n}\right)$ and the case that the distance is greater than ϵ using $\tilde{O}(\sqrt{n}\text{poly}(1/\epsilon))$ samples from $\boldsymbol{p}$.

Testing random variables for independence. Batu et al. [25] also show that it is possible to test whether a distribution over $[n] \times [m]$ is independent or is ϵ-far from any independent joint distribution, using a sample of size $\tilde{O}(n^{2/3}m^{1/3}\text{poly}(1/\epsilon))$.

Approximating the entropy. A very basic and important measure of distributions is their (binary) entropy. The main result of Batu et al. [23] is an algorithm that computes a γ-multiplicative

approximation of the entropy using a sample of size $O(n^{(1+\eta)/\gamma^2} \log n)$ for distributions with entropy $\Omega(\gamma/\eta)$ where n is the size of the domain of the distribution and η is an arbitrarily small positive constant. They also show that $\Omega(n^{1/(2\gamma^2)})$ samples are necessary. A lower bound that matches the upper bound of Batu et al. [23] is proved in [126].

Approximating the support size. Another natural measure for distributions is their support size. To be precise, consider the problem of approximating the support size of a distribution when each element in the distribution appears with probability at least $\frac{1}{n}$. This problem is closely related to the problem of approximating the number of distinct elements in a sequence of length n. For both problems, there is a nearly linear in n lower bound on the sample complexity, applicable even for approximation with *additive* error [117].

A unifying approach to testing symmetric properties of distributions. Valiant [126] obtains the lower bounds mentioned in the foregoing discussion as part of a general study of estimating symmetric measures over distributions (or pairs of distributions). That is, he considers measures of distributions that are preserved under renaming of the elements in the domain of the distributions. Roughly speaking, his main finding is that for every such property, there exists a threshold such that elements whose probability weight is below the threshold "do not matter" in terms of the task of estimating the measure (with a small additive error). This implies that such properties have a "canonical estimator" that computes its output based on its estimate of the probability weight of elements that appear sufficiently often in the sample ("heavy elements"), and essentially ignores those elements that do not appear sufficiently often.[4] In the other direction, lower bounds can

[4] Valiant talks about testing, and refers to his algorithm as a "canonical tester". We have chosen to use the terms "estimation" and "canonical estimator" for the following reason. When one discusses "testing" properties of distributions then the task may be to distinguish between the case that the measure in question is 0 (close to 0) and the case that it is above some threshold ϵ, rather than distinguishing between the case that the measure is below ϵ_1 (for ϵ_1 that is not necessarily close to 0) and above ϵ_2, which is essentially an additive estimation task. This is true for example in the case of testing closeness of distributions. The two tasks just described are different types of tasks, and, in particular, for

be derived by constructing pairs of distributions on which the value of the estimated measure is significantly different, but that give the same probability weight to the heavy elements (and may completely differ on all light elements).

Other results. Other works on testing/estimating properties of distributions include [27, 4, 122].

11.4.2 Estimating the Entropy

In this subsection we give the details for the algorithm that estimates the entropy of a distribution [23]. Consider a distribution $\boldsymbol{p}$ over the set $[n]$ where the probability of element i is denoted by $\boldsymbol{p}_i$. Recall that the entropy of the distribution $\boldsymbol{p}$ is defined as follows:

$$H(\boldsymbol{p}) \stackrel{\text{def}}{=} -\sum_{i=1}^{n} p_i \log p_i = \sum_{i=1}^{n} p_i \log(1/p_i). \tag{11.1}$$

Given access to samples $i \in [n]$ distributed according to $\boldsymbol{p}$, we would like to estimate $H(\boldsymbol{p})$ to within a multiplicative factor γ. That is, we seek an algorithm that obtains an estimate $\widehat{H}$ such that $H(\boldsymbol{p})/\gamma \leq \widehat{H} \leq \gamma \cdot H(\boldsymbol{p})$ with probability at least $2/3$ (as usual, we can increase the success probability to $1-\delta$ by running the algorithm $O(\log(1/\delta))$ times and outputting the median value).

We next describe the algorithm of [23] whose sample complexity and running time are $O\left(n^{\frac{1+\eta}{\gamma^2}} \log n\right)$ conditioned on $H(\boldsymbol{p}) = \Omega(\gamma/\eta)$. If there is no lower bound on the entropy, then it is impossible to obtain any multiplicative factor [23], and even if the entropy is quite high (i.e., at least $\log n/\gamma^2 - 2$), then $n^{\frac{1}{\gamma^2}-o(1)}$ samples are necessary.

The main result of [23] is:

Theorem 11.1. For any $\gamma > 1$ and $0 < \epsilon_0 < 1/2$, there exists an algorithm that can approximate the entropy of a distribution over $[n]$

---

the former task, low-frequency elements may play a role (as is the case in testing closeness of distributions where the collision counts of low-frequency elements play a role). Thus, saying that low-frequency elements may be ignored when testing properties distributions is not precise. The statement is true for *estimating* (symmetric) measures of distributions.

whose entropy is at least $\frac{4\gamma}{\epsilon_0(1-2\epsilon_0)}$ to within a multiplicative factor of $(1+2\epsilon_0)\gamma$ with probability at least 2/3 in time $O(n^{1/\gamma^2}\log n\epsilon_0^{-2})$.

The main idea behind the algorithm is the following. Elements in $[n]$ are classified as either *heavy* or *light* depending on their probability mass. Specifically, for any choice of $\alpha > 0$,

$$B_\alpha(\boldsymbol{p}) \stackrel{\text{def}}{=} \left\{ i \in [n] \,:\, p_i \geq n^{-\alpha} \right\}. \tag{11.2}$$

The algorithm separately approximates the contribution to the entropy of the heavy elements and of the light elements, and then combines the two.

In order to describe the algorithm and analyze it, we shall need the following notation. For a distribution $\boldsymbol{p}$ and a set T,

$$w_{\boldsymbol{p}}(T) = \sum_{i \in T} p_i \quad \text{and} \quad H_T(\boldsymbol{p}) = -\sum_{i \in T} p_i \log(p_i). \tag{11.3}$$

Note that if T_1, T_2 are disjoint sets such that $T_1 \cup T_2 = [n]$ then $H(\boldsymbol{p}) = H_{T_1}(\boldsymbol{p}) + H_{T_2}(\boldsymbol{p})$. The algorithm (Algorithm 11.1) is given in Figure 11.1.

In the next two subsections we analyze separately the contribution of the heavy elements and the contribution of the light elements to the estimate computed by Algorithm 11.1.

Algorithm 11.1: Algorithm Approximate-Entropy(γ, ϵ_0)

1. *Set* $\alpha = 1/\gamma^2$.
2. *Get* $m = \Theta(n^\alpha \log n \epsilon_0^{-2})$ *samples from* $\boldsymbol{p}$.
3. *Let* $\boldsymbol{q}$ *be the empirical probability vector of the* n *elements. That is,* q_i *is the number of times* i *appears in the sample divided by* m.
4. *Let* $\widehat{B}_\alpha = \{i \,:\, q_i > (1-\epsilon_0)n^{-\alpha}\}$.
5. *Take an additional sample of size* $m = \Theta(n^\alpha \log n/\epsilon_0^2)$ *from* $\boldsymbol{p}$ *and let* $\hat{w}(S)$ *be the total empirical weight of elements in* $S = [n] \setminus \widehat{B}_\alpha$ *in the sample.*
6. *Output* $\hat{H} = H_{\widehat{B}_\alpha}(\boldsymbol{q}) + \frac{\hat{w}(S)\log n}{\gamma}$.

Fig. 11.1 The algorithm for approximating the entropy of a distribution.

Approximating the Contribution of Heavy Elements

The next lemma follows by applying a multiplicative Chernoff bound.

Lemma 11.2. For $m = 20n^{\alpha}\log n/\epsilon_0^2$ and $\boldsymbol{q}$ as defined in the algorithm, with probability at least $1 - \frac{1}{n}$ the following two conditions hold for every $i \in [n]$:

1. If $p_i \geq \frac{1-\epsilon_0}{1+\epsilon_0}n^{-\alpha}$ (in particular this is true of $i \in B_\alpha(\boldsymbol{p})$) then $|p_i - q_i| \leq \epsilon_0 p_i$; and
2. If $p_i < \frac{1-\epsilon_0}{1+\epsilon_0}n^{-\alpha}$ then $q_i < (1-\epsilon_0)n^{-\alpha}$.

By Lemma 11.2, we get that with high probability, $B_\alpha(\boldsymbol{p}) \subseteq \widehat{B}_\alpha$, and for every $i \in \widehat{B}_\alpha$ (even if $i \notin B_\alpha(\boldsymbol{p})$), $|q_i - p_i| \leq \epsilon_0 p_i$. The next lemma bounds the deviation of $H_T(\boldsymbol{q})$ from $H_T(\boldsymbol{p})$ conditioned on q_i being close to p_i for every $i \in T$.

Lemma 11.3. For every set T such that for every $i \in T$, $|q_i - p_i| \leq \epsilon_0 p_i$,

$$|H_T(\boldsymbol{q}) - H_T(\boldsymbol{p})| \leq \epsilon_0 H_T(\boldsymbol{p}) + 2\epsilon_0 w_{\boldsymbol{p}}(T)\ .$$

Proof. For $i \in T$, let ϵ_i be defined by $q_i = (1+\epsilon_i)p_i$ where by the premise of the lemma, $|\epsilon_i| \leq \epsilon_0$.

$$\begin{aligned} H_T(\boldsymbol{q}) - H_T(\boldsymbol{p}) &= -\sum_{i\in T}(1+\epsilon_i)p_i\log((1+\epsilon_i)p_i) \\ &\quad + \sum_{i\in T} p_i \log p_i \qquad (11.4) \\ &= -\sum_{i\in T}(1+\epsilon_i)p_i\log p_i - \sum_{i\in T}(1+\epsilon_i)p_i\log(1+\epsilon_i) \\ &\quad + \sum_{i\in T} p_i \log p_i \qquad (11.5) \end{aligned}$$

$$= -\sum_{i\in T} \epsilon_i p_i \log(1/p_i)$$
$$-\sum_{i\in T}(1+\epsilon_i)p_i\log(1+\epsilon_i). \tag{11.6}$$

If we now consider the absolute value of this difference:

$$\begin{aligned}
&|H_T(\boldsymbol{q}) - H_T(\boldsymbol{p})| \\
&\quad\le \left|\sum_{i\in T}\epsilon_i p_i\log(1/p_i)\right| + \left|\sum_{i\in T}(1+\epsilon_i)p_i\log(1+\epsilon_i)\right| && (11.7)\\
&\quad\le \sum_{i\in T}|\epsilon_i| p_i\log(1/p_i) + \sum_{i\in T}(1+\epsilon_i)p_i\log(1+\epsilon_i) && (11.8)\\
&\quad\le \epsilon_0 H_T(\boldsymbol{p}) + 2\epsilon_0 w_T(\boldsymbol{p}). && (11.9)
\end{aligned}$$

□

Approximating the Contribution of Light Elements

Recall that $S = [n] \setminus \widehat{B}_\alpha$ so that, By Lemma 11.2, with high probability $S \subseteq [n] \setminus B_\alpha(\boldsymbol{p})$.

Claim 11.4. Let $\hat{w}(S)$ be the fraction of samples, among $m = \Theta((n^\alpha/\epsilon_0^2)\log n)$ that belong to S (as defined in Algorithm 11.1). If $w_{\boldsymbol{p}}(S) \ge n^{-\alpha}$ then with probability $1 - 1/n$,

$$(1-\epsilon_0)w_{\boldsymbol{p}}(S) \le \hat{w}(S) \le (1+\epsilon_0)w_{\boldsymbol{p}}(S).$$

The claim directly follows by a multiplicative Chernoff bound.

Lemma 11.5. If $p_i \le n^{-\alpha}$ for every $i \in S$ then

$$\alpha \cdot \log n \cdot w_{\boldsymbol{p}}(S) \;\le\; H_S(\boldsymbol{p}) \;\le\; \log n \cdot w_{\boldsymbol{p}}(S) + 1.$$

Proof. Conditioned on a particular weight $w_{\boldsymbol{p}}(S)$, the entropy $H_S(\boldsymbol{p})$ is maximized when $p_i = w_{\boldsymbol{p}}(S)/|S|$ for all i. In this case

$$\begin{aligned} H_S(\boldsymbol{p}) &= w_{\boldsymbol{p}}(S)\log(|S|/w_{\boldsymbol{p}}(S)) && (11.10)\\ &= w_{\boldsymbol{p}}(S)\log|S| + w_{\boldsymbol{p}}(S)\log(1/w_{\boldsymbol{p}}(S)) && (11.11)\\ &\leq w_{\boldsymbol{p}}(S)\log n + 1. && (11.12) \end{aligned}$$

On the other hand, $H_S(\boldsymbol{p})$ is minimized when its support is minimized. Since $p_i \leq n^{-\alpha}$ for every $i \in S$, this means that $n^{\alpha} w_{\boldsymbol{p}}(S)$ of the elements have the maximum probability $p_i = n^{\alpha}$, and all others have 0 probability. In this case $H_S(\boldsymbol{p}) = \alpha w_{\boldsymbol{p}}(S)\log n$. □

Putting it Together

We now prove Theorem 11.1 based on Algorithm 11.1. By Lemma 11.2 we have that with high probability:

1. If $i \in B_\alpha(\boldsymbol{p})$ then $i \in \widehat{B}_\alpha$. That is, $p_i \leq n^{-\alpha}$ for every $i \in S = [n] \setminus \widehat{B}_\alpha$.
2. Every $i \in \widehat{B}_\alpha$ satisfies $|q_i - p_i| \leq \epsilon_0 p_i$.

Assume from this point on that the above two properties hold. Let B be a shorthand for $\widehat{B}_\alpha$ and let $S = [n] \setminus B$ be as defined in Algorithm 11.1. Assume first that $w_{\boldsymbol{p}}(S) \geq n^{-\alpha}$. In this case, Lemma 11.5 tells us that (since $\alpha = 1/\gamma^2$)

$$\frac{1}{\gamma^2} \cdot w_{\boldsymbol{p}}(S) \cdot \log n \;\leq\; H_S(\boldsymbol{p}) \;\leq\; w_{\boldsymbol{p}}(S)\log n + 1 \tag{11.13}$$

or equivalently:

$$\frac{1}{\log n} \cdot (H_S(\boldsymbol{p}) - 1) \leq w_{\boldsymbol{p}}(S) \leq \frac{1}{\log n} \cdot \gamma^2 \cdot H_S(\boldsymbol{p}). \tag{11.14}$$

By Claim 11.4,

$$(1 - \epsilon_0) w_{\boldsymbol{p}}(S) \leq \hat{w}(S) \leq (1 + \epsilon_0) w_{\boldsymbol{p}}(S). \tag{11.15}$$

If we now use Equations (11.14) and (11.15) and apply Lemma 11.3 (using $|q_i - p_i| \leq \epsilon_0 p_i$ for every $i \in B$), we get:

$$\begin{aligned}
&H_B(\boldsymbol{q}) + \frac{\hat{w}(S)\log n}{\gamma} \\
&\quad \leq (1+\epsilon_0)H_B(\boldsymbol{p}) + 2\epsilon_0 + \frac{(1+\epsilon_0)w_{\boldsymbol{p}}(S)\log n}{\gamma} && (11.16)\\
&\quad \leq (1+\epsilon_0)\cdot(H_B(\boldsymbol{p}) + \gamma H_S(\boldsymbol{p})) + 2\epsilon_0 && (11.17)\\
&\quad \leq (1+\epsilon_0)\gamma H(\boldsymbol{p}) + 2\epsilon_0 && (11.18)\\
&\quad \leq (1+2\epsilon_0)\gamma H(\boldsymbol{p}), && (11.19)
\end{aligned}$$

where in the last inequality we used the fact that $\gamma > 1$, and $H(\boldsymbol{p}) \geq \frac{4\gamma}{\epsilon_0(1-2\epsilon_0)} > 4$ so that $2\epsilon_0 < \epsilon_0 \cdot \gamma \cdot H(\boldsymbol{p})$. Similarly,

$$\begin{aligned}
&H_B(\boldsymbol{q}) + \frac{w_q(S)\log n}{\gamma} \\
&\quad \geq (1-\epsilon_0)H_B(\boldsymbol{p}) - 2\epsilon_0 + \frac{(1-\epsilon_0)w_{\boldsymbol{p}}(S)\log n}{\gamma} && (11.20)\\
&\quad \geq (1-\epsilon_0)\cdot\left(H_B(\boldsymbol{p}) + \frac{H_S(\boldsymbol{p})-1}{\gamma}\right) - 2\epsilon_0 && (11.21)\\
&\quad \geq \frac{H(\boldsymbol{p})}{\gamma(1+2\epsilon_0)} && (11.22)
\end{aligned}$$

(the last inequality follows from the lower bound on $H(\boldsymbol{p})$ by tedious (though elementary) manipulations). Finally, if $w_{\boldsymbol{p}}(S) < n^{-\alpha}$ then by Claim 11.4 $w_{\boldsymbol{q}}(S) \leq (1+\epsilon_0)n^{-\alpha}$ with high probability. Therefore, $w_q(S)\log n/\gamma$ is at most $(1+\epsilon_0)n^{-\alpha}\log n/\gamma$ (and at least 0). It is not hard to verify that the contribution to the error is negligible, assuming γ is bounded away from 1.

We note that if $\boldsymbol{p}$ is monotone, that is $p_i \geq p_{i+1}$ for all i, then there is a more sophisticated algorithm that uses $\mathrm{poly}(\log n, \log\gamma)$ samples [122].

Acknowledgments

We would like to thank an anonymous reviewer for many helpful comments.

References

[1] N. Ailon and B. Chazelle, "Information theory in property testing and monotonicity testing in higher dimensions," *Information and Computation*, vol. 204, pp. 1704–1717, 2006.

[2] N. Ailon, B. Chazelle, S. Comandur, and D. Liue, "Estimating the distance to a monotone function," in *Proceedings of the Eight International Workshop on Randomization and Computation (RANDOM)*, pp. 229–236, 2004.

[3] N. Alon, "Testing subgraphs of large graphs," *Random Structures and Algorithms*, vol. 21, pp. 359–370, 2002.

[4] N. Alon, A. Andoni, T. Kaufman, K. Matulef, R. Rubinfeld, and N. Xie, "Testing k-wise and almost k-wise independence," in *Proceedings of the Thirty-Ninth Annual ACM Symposium on the Theory of Computing*, pp. 496–505, 2007.

[5] N. Alon, S. Dar, M. Parnas, and D. Ron, "Testing of clustering," *SIAM Journal on Discrete Math*, vol. 16, no. 3, pp. 393–417, 2003.

[6] N. Alon, R. A. Duke, H. Lefmann, V. Rodl, and R. Yuster, "The algorithmic aspects of the regularity lemma," *Journal of Algorithms*, vol. 16, pp. 80–109, 1994.

[7] N. Alon, E. Fischer, M. Krivelevich, and M. Szegedy, "Efficient testing of large graphs," *Combinatorica*, vol. 20, pp. 451–476, 2000.

[8] N. Alon, E. Fischer, I. Newman, and A. Shapira, "A combinatorial characterization of the testable graph properties: It's all about regularity," in *Proceedings of the Thirty-Eighth Annual ACM Symposium on the Theory of Computing*, pp. 251–260, 2006.

[9] N. Alon, T. Kaufman, M. Krivelevich, and D. Ron, "Testing triangle freeness in general graphs," in *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 279–288, 2006.

[10] N. Alon and M. Krivelevich, "Testing k-colorability," *SIAM Journal on Discrete Math*, vol. 15, no. 2, pp. 211–227, 2002.

[11] N. Alon, M. Krivelevich, T. Kaufman, S. Litsyn, and D. Ron, "Testing Reed-Muller codes," *IEEE Transactions on Information Theory*, vol. 51, no. 11, pp. 4032–4038, 2005. An extended abstract of this paper appeared under the title: Testing Low-Degree Polynomials over GF(2), in the proceedings of RANDOM 2003.

[12] N. Alon, M. Krivelevich, I. Newman, and M. Szegedy, "Regular languages are testable with a constant number of queries," *SIAM Journal on Computing*, pp. 1842–1862, 2001.

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

[14] N. Alon and A. Shapira, "Testing satisfiability," *Journal of Algorithms*, vol. 47, pp. 87–103, 2003.

[15] N. Alon and A. Shapira, "Testing subgraphs in directed graphs," *Journal of Computer and System Sciences*, vol. 69, pp. 354–482, 2004.

[16] N. Alon and A. Shapira, "A characterization of the (natural) graph properties testable with one-sided error," in *Proceedings of the Forty-Sixth Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 429–438, 2005.

[17] N. Alon and A. Shapira, "Every monotone graph property is testable," in *Proceedings of the Thirty-Seventh Annual ACM Symposium on the Theory of Computing (STOC)*, pp. 129–137, 2005. To appear in SICOMP.

[18] N. Alon and A. Shapira, "A characterization of easily testable induced subgraphs," *Combinatorics Probability and Computing*, vol. 15, pp. 791–805, 2006.

[19] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, "Proof verification and the hardness of approximation problems," *Journal of the ACM*, vol. 45, no. 1, pp. 501–555, 1998. a preliminary version appeard in Proc. 33^{rd} FOCS, 1992.

[20] S. Arora and M. Sudan, "Improved low-degree testing and its applications," in *Proceedings of the Thirty-Second Annual ACM Symposium on the Theory of Computing (STOC)*, pp. 485–495, 1997.

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

[22] L. Babai, L. Fortnow, and C. Lund, "Non-deterministic exponential time has two-prover interactive protocols," *Computational Complexity*, vol. 1, no. 1, pp. 3–40, 1991.

[23] T. Batu, S. Dasgupta, R. Kumar, and R. Rubinfeld, "The complexity of approximating the entropy," *SIAM Journal on Computing*, vol. 35, no. 1, pp. 132–150, 2005.

[24] T. Batu, F. Ergun, J. Kilian, A. Magen, S. Raskhodnikova, R. Rubinfeld, and R. Sami, "A sublinear algorithm for weakly approximating edit distance," in *Proceedings of the Thirty-Fifth Annual ACM Symposium on the Theory of Computing (STOC)*, pp. 316–324, 2003.

[25] T. Batu, E. Fischer, L. Fortnow, R. Kumar, and R. Rubinfeld, "Testing random variables for independence and identity," in *Proceedings of the Forty-Second Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 442–451, 2001.

[26] T. Batu, L. Fortnow, R. Rubinfeld, W. Smith, and P. White, "Testing that distributions are close," in *Proceedings of the Forty-First Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 259–269, 2000.

[27] T. Batu, R. Kumar, and R. Rubinfeld, "Sublinear algorithms for testing monotone and unimodal distributions," in *Proceedings of the Thirty-Sixth Annual ACM Symposium on the Theory of Computing (STOC)*, pp. 381–390, 2004.

[28] T. Batu, R. Rubinfeld, and P. White, "Fast approximate PCPs for multidimensional bin-packing problems," *Information and Computation*, vol. 196, no. 1, pp. 42–56, 2005.

[29] M. Bellare, D. Coppersmith, J. Håstad, M. Kiwi, and M. Sudan, "Linearity testing over characteristic two," *IEEE Transactions on Information Theory*, vol. 42, no. 6, pp. 1781–1795, 1996.

[30] M. Bellare, O. Goldreich, and M. Sudan, "Free bits, PCPs and nonapproximability – towards tight results," *SIAM Journal on Computing*, vol. 27, no. 3, pp. 804–915, 1998.

[31] I. Ben-Eliezer, T. Kaufman, M. Krivelevich, and D. Ron, "Comparing the strength of query types in property testing: the case of testing k-colorability," in *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2008.

[32] E. Ben-Sasson, P. Harsha, and S. Raskhodnikova, "3CNF properties are hard to test," *SIAM Journal on Computing*, vol. 35, no. 1, pp. 1–21, 2005.

[33] I. Benjamini, O. Schramm, and A. Shapira, "Every minor-closed property of sparse graphs is testable," in *Proceedings of the Fourtieth Annual ACM Symposium on the Theory of Computing*, pp. 393–402, 2008.

[34] E. Blais, "Testing juntas almost optimally," in *Proceedings of the Fourty-First Annual ACM Symposium on the Theory of Computing*, pp. 151–158, 2009.

[35] M. Blum, M. Luby, and R. Rubinfeld, "Self-testing/correcting with applications to numerical problems," *Journal of the ACM*, vol. 47, pp. 549–595, 1993.

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

[37] A. Bogdanov and L. Trevisan, "Lower bounds for testing bipartiteness in dense graphs," in *Proceedings of the Nineteenth Computationa Complexity Conference (CCC)*, 2004.

[38] B. Bollig, "Property testing and the branching program size," in *Proceedings of FCT, Lecture notes in Computer Science 3623*, pp. 258–269, 2005.

[39] B. Bollig and I. Wegener, "Functions that have read-once branching programs of quadratic size are not necessarily testable," *Information Processing Letters*, vol. 87, no. 1, pp. 25–29, 2003.

[40] C. Borgs, J. Chayes, L. Lovász, V. T. Sós, B. Szegedy, and K. Vesztergombi, "Graph limits and parameter testing," in *Proceedings of the Thirty-Eighth Annual ACM Symposium on the Theory of Computing*, pp. 261–270, 2006.

[41] S. Chakraborty, E. Fischer, O. Lachish, A. Matsliah, and I. Newman, "Testing *st*-connectivity," in *Proceedings of the Eleventh International Workshop on Randomization and Computation (RANDOM)*, pp. 380–394, 2007.

[42] B. Chazelle, R. Rubinfeld, and L. Trevisan, "Approximating the minimum spanning tree weight in sublinear time," in *Automata, Languages and Programming: Twenty-Eighth International Colloquium (ICALP)*, pp. 190–200, 2001.

[43] H. Chockler and D. Gutfreund, "A lower bound for testing juntas," *Information Processing Letters*, vol. 90, no. 6, pp. 301–305, 2006.

[44] A. Czumaj, A. Shapira, and C. Sohler, "Testing hereditary properties of non-expanding bounded-degree graphs," Technical Report TR07-083, Electronic Colloquium on Computational Complexity (ECCC), 2007.

[45] A. Czumaj and C. Sohler, "Property testing with geometric queries," in *Proceedings of the Ninth Annual European Symposium on Algorithms (ESA)*, pp. 266–277, 2001.

[46] A. Czumaj and C. Sohler, "Abstract combinatorial programs and efficient property testers," *SIAM Journal on Computing*, vol. 34, no. 3, pp. 580–615, 2005.

[47] A. Czumaj and C. Sohler, "Sublinear-time algorithms," *Bulletin of the EATCS*, vol. 89, pp. 23–47, 2006.

[48] A. Czumaj and C. Sohler, "Testing expansion in bounded-degree graphs," in *Proceedings of the Forty-Eighth Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 570–578, 2007.

[49] A. Czumaj, C. Sohler, and M. Ziegler, "Property testing in computation geometry," in *Proceedings of the Eighth Annual European Symposium on Algorithms (ESA)*, pp. 155–166, 2000.

[50] I. Diakonikolas, H. K. Lee, K. Matulef, K. Onak, R. Rubinfeld, R. A. Servedio, and A. Wan, "Testing for concise representations," in *Proceedings of the Forty-Eighth Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 549–557, 2007.

[51] I. Diakonikolas, H. K. Lee, K. Matulef, R. A. Servedio, and A. Wan, "Efficient testing of sparse GF(2) polynomials," in *Automata, Languages and Programming: Thirty-Fifth International Colloquium (ICALP)*, pp. 502–514, 2008.

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

[53] Y. Dinitz and J. Westbrook, "Maintaining the classes of 4-edge-connectivity in a graph on-line," *Algorithmica*, vol. 20, no. 3, pp. 242–276, 1998.

[54] Y. Dodis, O. Goldreich, E. Lehman, S. Raskhodnikova, D. Ron, and A. Samorodnitsky, "Improved testing algorithms for monotonocity," in

Proceedings of the Third International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM), pp. 97–108, 1999.

[55] F. Ergun, S. Kannan, S. R. Kumar, R. Rubinfeld, and M. Viswanathan, "Spot-checkers," *Journal of Computer and System Sciences*, vol. 60, no. 3, pp. 717–751, 2000.

[56] S. Fattal and D. Ron, "Approximating the distance to monotonicity in high dimensions," To appear in *Transactions on Algorithms*, 2009.

[57] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy, "Approximating clique is almost NP-complete," *Journal of the ACM*, pp. 268–292, 1996.

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

[59] E. Fischer, "Testing graphs for colorability properties," *Random Structures and Algorithms*, vol. 26, no. 3, pp. 289–309, 2005.

[60] E. Fischer and L. Fortnow, "Tolerant versus intolerant testing for boolean properties," *Theory of Computing*, vol. 2, pp. 173–183, 2006.

[61] E. Fischer, G. Kindler, D. Ron, S. Safra, and S. Samorodnitsky, "Testing juntas," *Journal of Computer and System Sciences*, vol. 68, no. 4, pp. 753–787, 2004.

[62] E. Fischer, O. Lachish, A. Matsliah, I. Newman, and O. Yahalom, "On the query complexity of testing orientations for being Eulerian," in *Proceedings of the Twelveth International Workshop on Randomization and Computation (RANDOM)*, pp. 402–415, 2008.

[63] E. Fischer, E. Lehman, I. Newman, S. Raskhodnikova, R. Rubinfeld, and A. Samrodnitsky, "Monotonicity testing over general poset domains," in *Proceedings of the Thirty-Fourth Annual ACM Symposium on the Theory of Computing (STOC)*, pp. 474–483, 2002.

[64] E. Fischer and I. Newman, "Testing versus estimation of graph properties," *SIAM Journal on Computing*, vol. 37, no. 2, pp. 482–501, 2007.

[65] E. Fischer, I. Newman, and J. Sgall, "Functions that have read-twice constant width branching programs are not necessarily testable," *Random Structures and Algorithms*, vol. 24, no. 2, pp. 175–193, 2004.

[66] K. Friedl and M. Sudan, "Some improvements to total degree tests," in *Proceedings of the 3rd Annual Israel Symposium on Theory of Computing and Systems*, pp. 190–198, 1995. Corrected version available online at http://theory.lcs.mit.edu/~madhu/papers/friedl.ps.

[67] P. Gemmell, R. Lipton, R. Rubinfeld, M. Sudan, and A. Wigderson, "Self testing/correcting for polynomials and for approximate functions," in *Proceedings of the Thirty-Second Annual ACM Symposium on the Theory of Computing (STOC)*, pp. 32–42, 1991.

[68] D. Glasner and R. A. Servedio, "Distribution-free testing lower bounds for basic Boolean functions," in *Proceedings of the Eleventh International Workshop on Randomization and Computation (RANDOM)*, pp. 494–508, 2007.

[69] O. Goldreich, "Combinatorial property testing - a survey," in *Randomization Methods in Algorithm Design*, pp. 45–60, 1998.

[70] O. Goldreich, "Short locally testable codes and proofs (a survey)," Technical Report TR05-014, Electronic Colloquium on Computational Complexity (ECCC), 2005.

[71] O. Goldreich, S. Goldwasser, E. Lehman, D. Ron, and A. Samordinsky, "Testing monotonicity," *Combinatorica*, vol. 20, no. 3, pp. 301–337, 2000.

[72] O. Goldreich, S. Goldwasser, and D. Ron, "Property testing and its connection to learning and approximation," *Journal of the ACM*, vol. 45, no. 4, pp. 653–750, 1998.

[73] O. Goldreich and D. Ron, "Property testing in bounded degree graphs," in *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing (STOC)*, pp. 406–415, 1997. This is an extended abstract of [76] which includes algorithms for testing 2-vertex connectivity and 3-vertex connectivity.

[74] O. Goldreich and D. Ron, "A sublinear bipartite tester for bounded degree graphs," *Combinatorica*, vol. 19, no. 3, pp. 335–373, 1999.

[75] O. Goldreich and D. Ron, "On testing expansion in bounded-degree graphs," *Electronic Colloqium on Computational Complexity*, vol. 7, no. 20, 2000.

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

[77] O. Goldreich and D. Ron, "Algorithmic aspects of property testing in the dense graphs model," in *Proceedings of the Thirteenth International Workshop on Randomization and Computation (RANDOM)*, pp. 520–533, 2009.

[78] O. Goldreich and L. Trevisan, "Three theorems regarding testing graph properties," *Random Structures and Algorithms*, vol. 23, no. 1, pp. 23–57, 2003.

[79] M. Gonen and D. Ron, "On the benefits of adaptivity in property testing of dense graphs," in *Proceedings of the Eleventh International Workshop on Randomization and Computation (RANDOM)*, pp. 525–537, 2007. To appear in *Algorithmica*.

[80] V. Guruswami and A. Rudra, "Tolerant locally testable codes," in *Proceedings of the Ninth International Workshop on Randomization and Computation (RANDOM)*, pp. 306–317, 2005.

[81] S. Halevy and E. Kushilevitz, "Distribution-free property testing," in *Proceedings of the Seventh International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*, pp. 341–353, 2003.

[82] S. Halevy and E. Kushilevitz, "Distribution-free connectivity testing," in *Proceedings of the Eight International Workshop on Randomization and Computation (RANDOM)*, pp. 393–404, 2004.

[83] S. Halevy and E. Kushilevitz, "A lower bound for distribution-free monotonicity testing," in *Proceedings of the Ninth International Workshop on Randomization and Computation (RANDOM)*, pp. 330–341, 2005.

[84] S. Halevy and E. Kushilevitz, "Distribution-free property testing," *SIAM Journal on Computing*, vol. 37, no. 4, pp. 1107–1138, 2007.

[85] S. Halevy, O. Lachish, I. Newman, and D. Tsur, "Testing orientation properties," Technical Report TR05-153, Electronic Colloquium on Computational Complexity (ECCC), 2005.

[86] S. Halevy, O. Lachish, I. Newman, and D. Tsur, "Testing properties of constraint-graphs," in *Proceedings of the Twenty-Second IEEE Annual Conference on Computational Complexity (CCC)*, pp. 264–277, 2007.

[87] A. Hassidim, J. Kelner, H. Nguyen, and K. Onak, "Local graph partitions for approximation and testing," in *Proceedings of the Fiftieth Annual Symposium on Foundations of Computer Science (FOCS)*, 2009.

[88] J. Håstad, "Clique is hard to approximate within $n^{1-\epsilon}$," *Acta Mathematica*, vol. 182, pp. 105–142, 1999.

[89] D. S. Hochbaum and D. B. Shmoys, "Using dual approximation algorithms for scheduling problems: Theoretical and practical results," *Journal of the ACM*, vol. 34, pp. 144–162, January 1987.

[90] D. S. Hochbaum and D. B. Shmoys, "A polynomial approximation scheme for machine scheduling on uniform processors: Using the dual approximation approach," *SIAM Journal on Computing*, vol. 17, no. 3, pp. 539–551, 1988.

[91] H. Ito and Y. Yoshida, "Property testing on k-vertex connectivity of graphs," in *Automata, Languages and Programming: Thirty-Fifth International Colloquium (ICALP)*, pp. 539–550, 2008.

[92] C. S. Jutla, A. C. Patthak, A. Rudra, and D. Zuckerman, "Testing low-degree polynomials over prime fields," in *Proceedings of the Forty-Fifth Annual Symposium on Foundations of Computer Science (FOCS)*, 2004.

[93] S. Kale and C. Seshadhri, "Testing expansion in bounded degree graphs," Technical Report TR07-076, Electronic Colloquium on Computational Complexity (ECCC), 2007.

[94] S. Kale and C. Seshadhri, "Testing expansion in bounded degree graphs," in *Automata, Languages and Programming: Thirty-Fifth International Colloquium (ICALP)*, pp. 527–538, 2008.

[95] D. Karger, "Global min-cuts in $\mathcal{RNC}$ and other ramifications of a simple min-cut algorithm," in *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 21–30, 1993.

[96] T. Kaufman, M. Krivelevich, and D. Ron, "Tight bounds for testing bipartiteness in general graphs," *SIAM Journal on Computing*, vol. 33, no. 6, pp. 1441–1483, 2004.

[97] T. Kaufman and S. Litsyn, "Almost orthogonal linear codes are locally testable," in *Proceedings of the Forty-Sixth Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 317–326, 2005.

[98] T. Kaufman and D. Ron, "Testing polynomials over general fields," *SIAM Journal on Computing*, vol. 35, no. 3, pp. 779–802, 2006.

[99] T. Kaufman and M. Sudan, "Algebraic property testing: The role of invariance," in *Proceedings of the Fourtieth Annual ACM Symposium on the Theory of Computing*, pp. 403–412, 2008. For a full version see the ECCC technical report TR07–111.

[100] M. Kearns and D. Ron, "Testing problems with sub-learning sample complexity," *Journal of Computer and System Sciences*, vol. 61, no. 3, pp. 428–456, 2000.

[101] J. C. Kiefer, *Introduction to Statistical Inference*. Springer Verlag, 1987.

[102] Y. Kohayakawa, B. Nagle, and V. Rödl, "Efficient testing of hypergraphs," in *Automata, Languages and Programming: Twenty-Ninth International Colloquium (ICALP)*, pp. 1017–1028, 2002.

[103] S. Kopparty and S. Saraf, "Tolerant linearity testing and locally testable codes," in *Proceedings of the Thirteenth International Workshop on Randomization and Computation (RANDOM)*, pp. 601–614, 2009.

[104] R. Kumar and R. Rubinfeld, "Sublinear time algorithms," Samir Khuller's SIGACT News column, 2003.

[105] S. Marko and D. Ron, "Distance approximation in bounded-degree and general sparse graphs," *Transactions on Algorithms*, vol. 5, no. 2, 2009. Article number 22.

[106] M. Mihail, "Conductance and convergence of Markov chains — A combinatorial treatment of expanders," in *Proceedings of the Thirtieth Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 526–531, 1989.

[107] S. Muthukrishnan, "Data streams: Algorithms and applications," *Foundations and Trends in Theoretical Computer Science*, vol. 1, no. 2, 2005.

[108] A. Nachmias and A. Shapira, "Testing the expansion of a graph," Technical Report TR07-118, Electronic Colloquium on Computational Complexity (ECCC), 2007.

[109] D. Naor, D. Gusfield, and C. Martel, "A fast algorithm for optimally increasing the edge-connectivity," *SIAM Journal on Computing*, vol. 26, no. 4, pp. 1139–1165, 1997.

[110] I. Newman, "Testing membership in languages that have small width branching programs," *SIAM Journal on Computing*, vol. 31, no. 5, pp. 1557–1570, 2002.

[111] Y. Orenstein, "Private communication," To appear in Y. Orenstein's MSC thesis, 2009.

[112] M. Parnas and D. Ron, "Testing the diameter of graphs," *Random Structures and Algorithms*, vol. 20, no. 2, pp. 165–183, 2002.

[113] M. Parnas and D. Ron, "Testing metric properties," *Information and Computation*, vol. 187, no. 2, pp. 155–195, 2003.

[114] M. Parnas, D. Ron, and R. Rubinfeld, "Testing membership in parenthesis languages," *Random Structures and Algorithms*, vol. 22, no. 1, pp. 98–138, 2003.

[115] M. Parnas, D. Ron, and R. Rubinfeld, "Tolerant property testing and distance approximation," *Journal of Computer and System Sciences*, vol. 72, no. 6, pp. 1012–1042, 2006.

[116] M. Parnas, D. Ron, and A. Samorodnitsky, "Testing basic boolean formulae," *SIAM Journal on Discrete Math*, vol. 16, no. 1, pp. 20–46, 2002.

[117] S. Raskhodnikova, D. Ron, R. Rubinfeld, and A. Smith, "Strong lower bonds for approximating distributions support size and the distinct elements problem," in *Proceedings of the Forty-Eighth Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 559–568, 2007. To appear in *SIAM Journal on Computing*.

[118] N. Robertson and P. D. Seymour, "Graph minors. XIII. The disjoint paths problem," *Journal of Combinatorial Theory Series B*, vol. 63, no. 1, pp. 65–110, 1995.

[119] N. Robertson and P. D. Seymour, "Graph minors. XX. Wagner's conjecture," *Journal of Combinatorial Theory Series B*, vol. 92, no. 2, pp. 325–357, 2004.

[120] D. Ron, "Property testing," in *Handbook on Randomization, Volume II*, (S. Rajasekaran, P. M. Pardalos, J. H. Reif, and J. D. P. Rolim, eds.), pp. 597–649, 2001.

[121] D. Ron, "Property testing: A learning theory perspective," *Foundations and Trends in Machine Learning*, vol. 1, no. 3, pp. 307–402, 2008.

[122] R. Rubinfeld and R. Servedio, "Testing monotone high-dimensional distributions," in *Proceedings of the Thirty-Seventh Annual ACM Symposium on the Theory of Computing (STOC)*, pp. 147–156, 2005.

[123] R. Rubinfeld and M. Sudan, "Robust characterization of polynomials with applications to program testing," *SIAM Journal on Computing*, vol. 25, no. 2, pp. 252–271, 1996.

[124] E. Szemerédi, "Regular partitions of graphs," in *Proceedings, Colloque Inter. CNRS*, pp. 399–401, 1978.

[125] L. G. Valiant, "A theory of the learnable," *CACM*, vol. 27, no. 11, pp. 1134–1142, 1984.

[126] P. Valiant, "Testing symmetric properties of distributions," in *Proceedings of the Fourtieth Annual ACM Symposium on the Theory of Computing*, pp. 383–392, 2008.