The art of uninformed decisions
A primer to property testing

Eldar Fischer *

Abstract

Property testing is a new field in computational theory, that deals with the information that can be deduced from the input where the number of allowable queries (reads from the input) is significally smaller than its size.

Dedicated to the memory of the victims of terror, September 11, 2001.

1 Motivation and introduction

With the recent advances in technology, we are faced with the need to process increasingly larger amounts of data in faster times. Indeed, it was observed by Papadimitriou [37], that notions of efficiency of algorithms tend to gravitate towards stricter time restrictions as computer sciences progress. At first a problem was considered computable if there was an algorithm that can decide it in a finite time given any input instance. Afterwards came the notion of polynomial time computations, and later the possibility of making a computation faster for certain problems through use of parallel machines was investigated as well.

In all of the classes considered above, however, the algorithms involved still face the obvious obstacle of having to read the entire input prior to its assessment (in the parallel setting it is assumed that the input can be also read in parallel, which is not always a realistic assumption).

There are practical situations in which the input is so large, that even taking a linear time in its size to provide an answer is too much. Many modern databases have overwhelming sizes. There are also other instances where the input is not easily accessible – it could be stored for example on a server at the other end of a slow communication line. And it could also be the case that an

*NEC Research Institute, 4 Independence Way, Princeton NJ 08540, USA; and DIMACS. Current address: Technion, Computer Science Department, Technion City, Haifa 32000, Israel. Email: eldar@cs.technion.ac.il

explicit representation of the input in fact does not exist, and an "oracle" procedure that calculates its values in different locations is given instead.

This survey deals with algorithms that are designed to make a decision concerning the input after reading only a small portion thereof. In the general setting it is assumed that an algorithm has random access to the input, that is, the algorithm is told in advance what is the input size, and is supplied with an oracle that for a query 'i' yields the value of the input in this location. The goal is to minimize the number of calls to this oracle, and in the ideal case to make it a constant independent of the actual input size.

One necessary feature of such algorithms, that is immediately apparent, is that of their being probabilistic. The intuition is that even when not the whole input is actually read, all of the input must still be "considered" by the algorithm in order to give an output relevant to its entirety, and thus every component of the input must be read with some positive probability.

Even random algorithms cannot be expected to give accurate results. Consider for example the simple property of the input being all zeros. No algorithm reading only a small fraction of the input can distinguish with high probability between an input satisfying this property, and an input containing a single '1' in a randomly chosen location. In fact, there are some well known conjectures and results stating that most properties are "evasive" with respect to deciding them while reading only some of the input (evasiveness was studied extensively for graph properties; for the interested reader [35] provides an introduction to this subject).

It follows that in order to describe what algorithms can do without reading the whole input, it is required to formulate a new notion of approximate versions of the original decision problems. As a guide we look at what can be done with the above property of the input being all zeros – while even this property cannot be decided by such an algorithm, it is very easy, using a constant time randomized algorithm, to distinguish between the case that the input is all zeros, and the case that at least an ϵ fraction of the places in the input contain ones (where ϵ is any constant independent of the input size). The algorithm needs only to take $O(\epsilon^{-1})$ uniformly random samples of the input, and reject if it finds a '1'.

The following definition suffices for most instances of testing problems. It deals with a fixed input size – since testing deals primarily with informational bounds it is usually sufficient to analyze every fixed input size separately (and apart from rarely occurring uniformity issues, it is nearly equivalent to saying that the testing algorithm is given the size of the input in advance).

Definition 1 *An input, given as a function $f : \mathcal{D} \to F$, is said to be* ϵ-close to satisfying a property

P, if there exists a function $f' : \mathcal{D} \to F$ that differs from f in no more than $\epsilon|\mathcal{D}|$ places, and satisfies P. An input which is not ϵ-close to satisfying P is called ϵ-far from satisfying P.

There are some testability questions that deal with different notions of closeness, such as these defined using the edit distance between inputs when considered as strings; such a case appears in Section 6.

This leads to the following definition of what it means to ϵ-test a property. Remember that in this definition a read from the input, also called a *query*, is allowed from any location (in other words, the algorithm is granted random access to the input).

Definition 2 *Let P be a property, and let us fix the input size n. An* ϵ-test with $q = q(\epsilon, n)$ queries *for P is a probabilistic algorithm that reads the input in up to q places, and with probability at least $\frac{2}{3}$ distinguishes between the case that the input satisfies P, and the case that the input is ϵ-far from satisfying P.*

The '$\frac{2}{3}$' term in the definition is arbitrary of course, and can be amplified to any fixed probability smaller than 1, at the price of a linear increase in q, by running the algorithm a constant number of times and taking the majority vote.

Note that the above definition claims nothing about the running time of the algorithm, but deals only with the amount of data that is allowed to be read from the input. This makes property testing resemble the branches of computer science dealing with informational complexity, rather than those dealing with computational complexity. However, in most cases the algorithms making decisions based on a small portion of the input happen to have a small running time as well, making them also suitable for providing fast approximations of the input qualities.

A property that has an ϵ-test as above is called *ϵ-testable with q queries*. The best that can be hoped for with regards to large inputs is the situation where q depends only on ϵ and not on n. If this occurs for every fixed $\epsilon > 0$, then the property P is usually simply referred to as *testable*. Lowering the dependency on ϵ is also important, in which case the best that can be hoped for is a number of queries that is linear in ϵ^{-1}, as is the case with the property of the input being all zeros.

Sometimes lower bounds on the number of queries can be proven as well. There is no "official" standard on what makes a property "totally non-testable". In many cases involving inputs over a binary alphabet, any non-constant lower bound for a fixed ϵ is considered making the property non-testable. However, sometimes testing with $O(\log n)$ queries (for a fixed ϵ) is also considered a sufficient test, especially in cases, such as most problems dealing with inputs over infinite alphabets, where there are some known $\Omega(\log n)$ lower bounds.

A lower bound of the type n^c (for some positive constant c and ϵ) is sometimes considered the final say, but in several instances even testing algorithms that read this many queries are considered interesting, especially if they have a sublinear running time to match.

Sometimes, special types of testing algorithms are sought for. For example, by the definition above the algorithm is allowed to have a 2-sided error probability. A *1-sided* testing algorithm is an algorithm that in addition accepts with probability 1 any input that satisfies the property to be tested.

In other cases the adaptivity of the algorithm is also considered. A *non-adaptive* test for a property is a testing algorithm that specifies all queries in advance (according to some distribution), and only then is allowed to obtain the input values in the corresponding locations, based on which the algorithm accepts or rejects the input. In other words, a non-adaptive test cannot use the results of previous queries in the decision as to where to make the next query, but only in the decision as to whether to accept or reject the input in the end.

The rest of this survey is organized as follows. Section 2 deals with testing of a property defined in terms of polynomial functions, in the context of the works that have started this field. Section 3 deals with testing of languages, motivated by an attempt to correlate the notion of testing to other notions of low complexity (as it turns out there is little obvious correlation, and some surprising results come into play). Section 4 and Section 5 deal with one model of graph testing, and another angle of the attempt to give a logical characterization of testable languages, and Section 6 deals with another graph testing model. Section 7 deals with testing of properties motivated by the notion of monotonicity. Finally, Section 8 deals with some techniques for proving lower bounds on testability, and Section 9 deals with some future research directions and the relation of property testing to other old and new topics in computer science.

There is no hope to give complete proofs for all the results presented in a survey such as this. The strategy used in the following is to give proofs of some of the easier results, and to give proofs of special cases or proof outlines for some of the harder results. It is hoped that through this the reader can get a taste of the techniques used in this field.

Although in general older results tend to appear earlier in this survey, there is no serious attempt to fully reflect the chronological development of this field. For example, most results presented in Section 4 have appeared earlier than those presented in Section 3.

2 A first offering – testing for polynomiality

The first results in property testing were motivated by program testing (Blum, Luby and Rubinfeld [13], and Rubinfeld and Sudan [43]), and by probabilistically checkable proofs (Arora and Safra [6], and Arora, Lund, Motwani, Sudan and Szegedy [5]).

In the context of program testing the idea is that the "input" to be tested is not explicitly given. Instead, a program is given; this program is considered as a function from the set of its possible inputs to its outputs, and properties of this function have to be tested. The practical implication is clear – one cannot hope to explicitly write down the entire function, but a "query" into it can be answered in a reasonable time by running the given program on the input that corresponds to the queried "location". Thus testing can ensure that this program will perform within the expected parameters for most inputs.

Probabilistically checkable proofs deal with proving protocols that can be probabilistically verified by reading only a small portion of a proposed proof. Formulating the property of being an input that satisfies the protocol, in a way that allows testing for this, is the connection here. Both the context of program checking and the context of probabilistically checkable proofs place additional requirements on the procedure beyond testability (that happen to be achievable for the example below); their exact definition is outside the scope of a testing survey.

The first result regarding testability per-se is probably that of Blum, Luby and Rubinfeld [13] concerning linearity, and the first explicit definition of property testing was given by Rubinfeld and Sudan in [43] (in its turn it is based on some of the results of [44] and [26]). The first results dealt mainly with the algebraic notion of being a low degree polynomial.

Suppose that for some finite field $\mathcal{F}$ the input is given as a function f from $\mathcal{F}$ to $\mathcal{F}$, that queries consist of finding the value of f at a specified location $x \in \mathcal{F}$, and that the distance is measured by the number of values of f that have to be changed in order to make it satisfy the property to be tested, divided by the input size $|\mathcal{F}|$.

In this setting, it is easy to test using a few queries that f is a polynomial whose degree is bounded by some fixed constant k.

Proposition 2.1 *It takes $O(k+\epsilon^{-1})$ queries to ϵ-test that a function f from $\mathcal{F}$ to $\mathcal{F}$ (where $\mathcal{F}$ is a finite field of arbitrary size) is a polynomial of degree at most $k-1$*

Proof: The algorithm first queries f in k arbitrary different locations. If f is indeed a polynomial of degree at most $k-1$, then by the interpolation property of polynomials these k queried values determine all values of f.

The algorithm then queries f in $O(\epsilon^{-1})$ additional locations chosen uniformly and independently at random, and for each of these checks that the value of f indeed agrees with the value of the polynomial interpolated from the first k queried values. ■

In cases where functions from $\mathcal{F}^d$ to $\mathcal{F}$ are needed to be tested for the property of being low-degree (multivariate) polynomials, interpolation by itself is not satisfactory. The main results of [43] are another test for the 1-dimensional case that satisfies some additional properties mandated by the context of program checking, as well as an efficient test for the multidimensional case (see also [6] and [5] for related results and applications). The way to handle functions from $\mathcal{F}^d$ to $\mathcal{F}$ is by considering two random intersecting lines in $\mathcal{F}^d$, and testing that the restrictions of the input function to these give degree k polynomials that agree on the intersection point.

Theorem 2.2 ([43]) *The property of a function $f : \mathcal{F}^d \to \mathcal{F}$ being a polynomial of a total degree bounded by k can be ϵ-tested using $O(k^2 + k\epsilon^{-1})$ queries (independently of $|\mathcal{F}|$ and d).*

There was extensive research in the topic of testing a function for being a low degree polynomial, and related questions. The interested reader is referred to [7], that describes some additional results.

3 Testing of languages

As the possibility of testing an input for a property using few queries becomes apparent, the next natural step is to correlate the traditional categorization of properties into complexity classes with this new scheme.

An ultimate goal would be to try finding a logical characterization of the properties testable with a number of queries independent of the input size, in much the same manner as, say, the properties recognizable by a finite automaton have a characterization as all properties expressible in the appropriate second order monadic logic language.

This goal is far from achieved. A more humble goal is to identify whole classes, and not just individual properties, that are thus testable. Such classes, sometimes identified by the structure of their expression in an appropriate first order logic language, are given special attention in this survey.

Results are sometimes surprising. On one hand, there exist properties that are in AC_0 and are not testable even in a number of queries that is some small power of the input size. On the other hand, there exist NP-complete properties, such as graph 3-colorability, that are testable. A reason for this is that up to changing a small portion of the input, even simple first order logic expressions can capture surprisingly large portions of properties. For example, up to changing a small number of edges, 3-colorability is in fact indistinguishable from some very simple AC_0 property. More on this in Section 5.

For a potential source of classes of properties that are testable we turn first to the well-known classes of languages with low computational complexity, starting with the lowest – the languages recognizable by a finite automaton. These were investigated by Alon, Krivelevich, Newman and Szegedy [4] and were found to be testable. However, one cannot prove similar testability results for complexity classes much broader than this, as there are already context free languages that are not testable even with relatively many queries (in fact, the language of all possible concatenations of two palindromes is such a language).

Theorem 3.1 ([4]) *Any regular language is testable with a number of queries linear in $\epsilon^{-1}\log^3(\epsilon^{-1})$ and independent of the input size. On the other hand, there exist context-free languages that are not testable in a number of queries that is less than some fixed power of the input size.*

To give a taste of how such a result can be proven, a proof of the testability of a narrower class of languages is given here.

Proposition 3.2 *Let S be any fixed binary string. The property of a binary input string not containing S as a (not necessarily continuous) substring is ϵ-testable, in a number of queries linear in ϵ^{-1} and independent of the input size.*

Proof: The proof is by induction on the length of S. When S is of size 1 the test is trivial (it just performs a random sampling of the input and rejects if it finds an instance of the single character that comprises S).

Assume without loss of generality that the first character in S is '1', and let S' be the result of removing this first character of S. Let $v_1v_2\ldots v_n$ be an input of size n. The algorithm then chooses uniformly and independently $O(\epsilon^{-1})$ random locations in the input string, queries them, and remembers the smallest of them on which the input is '1'. Denote it by i_1.

In the case that no '1' was found in the previous stage, the algorithm accepts the input. This will happen only with a small probability (which can be made smaller than $2^{-|S|-2}$), unless the input is ϵ-close to being all '0', in which case it is clearly ϵ-close to not containing S as a substring.

With high probability, if a '1' was found in the preceding stage, then i_1 is such that $v_1 \dots v_{i_1-1}$ contains no more than $\frac{1}{2}\epsilon n$ instances '1'. To see this, denote by $\{j_1, \dots, j_{\epsilon n/2}\} \subset \{1, \dots, n\}$ the set of the first $\frac{1}{2}\epsilon n$ instances of '1' in the input (if such a set does not exist and yet i_1 was found then it clearly satisfies the above condition). It is easy to see that with high probability, $O(\epsilon^{-1})$ queries will be sufficient to hit a member of this set.

The algorithm continues by $\frac{1}{2}\epsilon$-testing the string $v_{i_1+1} \dots v_n$ for the property of not containing S' as a substring, and accepts the input if and only if the test in this stage accepts. By the induction hypothesis, this stage also takes a number of queries that is linear in ϵ^{-1}.

For the proof of correctness, it is quite clear that if the input does not contain S as a substring then the algorithm will accept with probability 1, because this necessarily implies that $v_{i_1+1} \dots v_n$ does not contain S' as a substring.

On the other hand, if the algorithm accepts $v_1 \dots v_n$, then we can define a string $w_1 \dots w_n$ that is ϵ-close to $v_1 \dots v_n$ and does not contain S as follows: $w_1, \dots, w_{i_1-1}$ are all set to '0' (this makes no more than $\frac{1}{2}\epsilon n$ changes from $v_1, \dots, v_n$), $w_{i_1} = v_{i_1} = 1$, and $w_{i_1+1}, \dots, w_n$ are set to a string that is $\frac{1}{2}\epsilon$-close to the string $v_{i_1+1} \dots v_n$ and contains no instance of S'.

It is clear that $w_1 \dots w_n$ is ϵ-close to $v_1 \dots v_n$. Also, $w_1 \dots w_n$ does not contain an instance of S that starts before i_1 as there are no occurrences of '1' in that range, and on the other hand does not contain an instance of S starting at i_1 or later because $w_{i_1+1} \dots w_n$ does not contain an instance of S'. This completes the proof of the properties of $v_1 \dots v_n$ in the case that the algorithm accepts, and thus the proof of the correctness of the test. ∎

It is still not known whether the testing of all context free languages can be bounded from *above* by a power of n that is smaller than 1. An investigation into this question is given in [39].

In [36], Newman found a generalization of the positive part of Theorem 3.1, by considering a non-uniform counterpart of the notion of a finite automaton.

Theorem 3.3 ([36]) *Every property identifiable by a read-once fixed-width oblivious branching program is testable with a number of queries polynomial in ϵ^{-1} (and independent of the input size).*

However, we close this section by noting that this result too is the best possible of its kind in some sense.

Theorem 3.4 ([23]) *There exists a property identifiable by a read-twice oblivious branching program of width three, that is not testable with a number of queries independent of the input size.*

4 Graph testing in the dense context

Graph theory is a point of contact between pure mathematics and computer science. Because of the major role it plays in the theory of computational complexity, it is only natural that problems that arise from graph theory, such as (proper) vertex colorability, will also be investigated from the point of view of property testing.

The first investigation into testing of graph properties was performed by Goldreich, Goldwasser and Ron in [29], motivated by the notion of testing serving as a new notion of approximation, and by some related questions that arise in computational learning theory. Here the input is assumed to be an adjacency matrix of a graph G with n vertices. This means that a query consists of obtaining an answer as to whether a pair $v_i v_j$ of vertices is an edge of G or not, and that G is considered to be ϵ-far from satisfying a property P if one has to add and remove more than a total of $\epsilon\binom{n}{2}$ edges in G to make it satisfy P.

This also means that we restrict ourselves to properties that relate to the graph G underlying the input, such as vertex colorability or the existence of a large clique. More precisely, it means that we investigate only properties that are invariant with respect to graph isomorphisms – the permutations of the input that result from permuting the vertex set of the underlying graph.

This invariance of graph properties has some useful implications for testing. For example, it implies that if two graph properties P and Q satisfy that for every ϵ there exist only a finite number of graphs that satisfy one of them but are ϵ-far from satisfying the other, then either both of P and Q are testable with a number of queries independent of n, or neither of them is. In this setting P and Q are said to be *indistinguishable* from each other; the exact statement and proof of this observation are found in [3].

As it turns out, for every fixed given k the property of a graph being k-colorable can be ϵ-tested with a number of queries depending only on ϵ (and k). Informally, this means that the NP-hardness of k-colorability for $k \geq 3$ essentially follows from constructions that are not k-colorable but can be made k-colorable by removing only a small portion of the edges.

To illustrate the proof given in [29], we show it for the special case of $k = 2$.

Proposition 4.1 ([29]) *There exists an ϵ-test for the property of a graph being bipartite, that makes a number of queries which is polynomial in ϵ (and is independent of the input size).*

To prove this proposition, we first note that it is quite trivial to test for bipartiteness if one is already given the supposed proper 2-coloring along with the input graph.

Lemma 4.2 *There exists an algorithm, that given a graph G and a coloring of some of its vertices (with the possibility also to query for a given vertex whether it is colored and with what color), can distinguish (with high probability) using a number of queries which is linear in ϵ^{-1}, between the case that G contains no edges whose incident vertices are colored with the same color, and the case that G contains more than $\epsilon\binom{n}{2}$ such edges.*

Proof: There are a total of $\binom{n}{2}$ vertex pairs in G, some of which may have the property of being edges connecting identically colored vertices. A simple sampling procedure distinguishes between the case that no pairs have this property, and the case that more than $\epsilon\binom{n}{2}$ pairs do have it. ■

Proof sketch of Proposition 4.1: The test first picks a set S of $O(\epsilon^{-1}\log(\epsilon^{-1}))$ vertices of the input graph G independently and uniformly at random; with high probability it has the property that all but at most $\frac{1}{5}\epsilon n$ of the vertices with degree at least $\frac{1}{5}\epsilon n$ in G have a neighbor in S.

We now look at all possible partitions of S into S_1 and S_2, and define for each of these a partial 2-coloring of G as follows. Every vertex that has a neighbor in S_1 is colored by 2, and every vertex that has a neighbor in S_2 but not in S_1 is colored by 1. With this definition, finding out whether a given vertex is colored and with what color requires $|S| = O(\epsilon^{-1}\log(\epsilon^{-1}))$ queries of G.

For every possible partition of S, we can use the algorithm from Lemma 4.2 to find out whether the graph is $\frac{1}{5}\epsilon$-far from containing no monochromatic edges according to the corresponding coloring. However, the number of possible partitions is exponential in ϵ. In order to check all possible partitions of S at once, we first note that with a simple amplification technique, the error probability of the algorithm in Lemma 4.2 can be made exponentially small in $|S|$; the resulting algorithm still picks a number of vertices that is polynomial in ϵ, and queries only for edges between these vertices and between them and S. Thus we can pick a uniformly random set U consisting of a polynomial number of vertices of G, and then for every partition of S run the algorithm from Lemma 4.2 on this same set, reusing the same queries. With high probability the above algorithm will not make an error concerning any of the $2^{|S|}$ partitions of S.

To end the test, we accept the input if there exists a partition of S for which G is not far from having no monochromatic edges in the corresponding coloring, and otherwise reject. The algorithm will never reject an input which is in fact bipartite, because a proper 2-coloring of the input graph induces for every S a partition for which the corresponding coloring contains no monochromatic edges.

On the other hand, with high probability it is the case that if the algorithm accepts then the graph is ϵ-close to being bipartite. To prove this we look at the appropriate partition of S and its corresponding partial 2-coloring. For every vertex that is not colored by it, we just remove all of its incident edges and color it arbitrarily; we know that with high probability this will change no more than $\frac{4}{5}\binom{n}{2}$ edges (no more than $\frac{2}{5}\binom{n}{2}$ edges incident with vertices of degree no more than $\frac{1}{5}n$, and no more than another $\frac{2}{5}\binom{n}{2}$ edges for up to $\frac{1}{5}n$ additional vertices that were not colored on account of having no neighbors in S). We also know that the graph may contain up to $\frac{1}{5}\binom{n}{2}$ more edges that violate this coloring; by removing these as well we are done. ∎

The description of the calculation above is deceptively complex. In fact, one can easily show that instead of picking a set of vertices S, and then checking all its partitions against a second set of vertices U, one could just pick a set that functions as "$S \cup U$", and check whether the subgraph of G induced on this set is bipartite. Thus the above proof also yields a graph-theoretic result – a graph that is ϵ-far from being bipartite contains at least $\frac{2}{3}\binom{n}{k}$ induced subgraphs with k vertices that are not bipartite, where k depends only on ϵ^{-1} and is in fact polynomial in it.

For other partition-related graph properties, such as k-colorability (for a fixed k), having a given maximal cut, or having a large clique, the reader is referred to [29], that generalizes all of these into the following grand theorem.

Theorem 4.3 ([29]) *For a fixed k, let P be the property of a graph G with n vertices having a partition $V_1, \ldots, V_k$ of its vertex set, with $\alpha_i n \le |V_i| \le \alpha'_i n$ for every i (for fixed, given $\alpha_i < \alpha'_i$), and such that for every $1 \le i \le j \le n$ the number of edges between V_i and V_j (or within V_i if $i = j$) is between $\beta_{i,j} n^2$ and $\beta'_{i,j} n^2$ (for fixed, given $\beta_{i,j} < \beta'_{i,j}$).*

Property P is testable, with a number of queries that is polynomial in ϵ.

5 Dense graphs continued – use of the Regularity Lemma

This section is about the use of Szemerédi's Regularity Lemma for property testing, and its application to proving testability for a class of graph properties that are defined by certain first order expressions.

To formulate logical expressions for properties, we shall use a language that includes variables that range over vertices, first order quantifiers over these variables, the adjacency relation "u is a neighbor of v", the equality relation "$u = v$", and boolean connectives. All properties thus expressible are called *first order graph properties.* We further classify them by the first order quantifiers that they use. The best way to illustrate this is by giving some examples:

- "The graph G contains a triangle". This is a first order graph property of type '$\exists$', as it has an expression using only this quantifier: $\exists u_1 u_2 u_3 (u_1 \sim u_2 \wedge u_1 \sim u_3 \wedge u_2 \sim u_3)$, where $u \sim v$ is the relation stating that u and v are neighbors (and in particular are not the same vertex).

- "The graph G does not contain an induced square". This is a first order graph property of type '$\forall$', as it states that for every set of four distinct vertices, certain adjacency configurations do not occur.

- "The graph G contains three distinct vertices u_1, u_2, u_3 such that every other vertex has exactly one of them as a neighbor, and furthermore there exists no triangle containing any of these three vertices". This is a first order graph property of type '$\exists\forall$'.

It is not hard to show that the third example above is related to the property of 3-colorability – every graph satisfying this property is not far from being 3-colorable, as the vertices u_1, u_2, u_3 determine a proper 3-coloring of the rest of the graph, and conversely every proper 3-coloring of a graph can be used to find a small modification of the graph that satisfies the property above. In other words, the above first order property and the property of being 3-colorable are indistinguishable, as per the definition outlined in Section 4. When it comes to property testing, first order expressions have more expressive power than they do in the context of traditional complexity theory.

Alon, Fischer, Krivelevich and Szegedy have shown in [3] what classes of first order expressions are wholly testable.

Theorem 5.1 ([3]) *All first order graph properties of type '$\exists\forall$' are testable with a number of queries independent of n. On the other hand, there exists a '$\forall\exists$' property that is not thus testable.*

The proof of the positive part of this theorem relies heavily on the notion of regular set pairs as defined by Szemerédi in [45] (as for the negative part, the main idea of its proof is given in Section 8).

Definition 3 (density and regularity of set pairs) *For two nonempty disjoint vertex sets A and B of a graph G, we define $e(A, B)$ to be the number of edges of G between A and B. The* edge density *of the pair is defined by $d(A, B) = \frac{e(A,B)}{|A||B|}$. We say that the pair A, B is γ*-regular*, if for any two subsets A' of A and B' of B, satisfying $|A'| \geq \gamma|A|$ and $|B'| \geq \gamma|B|$, their edge density satisfies $|d(A', B') - d(A, B)| < \gamma$.*

For a full survey regarding the many combinatorial applications of regular pairs and Szemerédi's Regularity Lemma (given below), the reader is referred to [34]. The main strength of the definition of regularity is in the fact that regular pairs have much in common with random bipartite graphs of the same density. Consider for example the following.

Lemma 5.2 *For every $\eta > 0$ there exist $\gamma = \gamma(\eta)$ and $\delta = \delta(\eta)$ such that if A, B and C are disjoint vertex sets of a graph G where each pair is γ-regular and has density at least η, then G contains at least $\delta|A||B||C|$ distinct triangles with a vertex from A, from B and from C respectively.*

The proof is not very hard, but we shall omit it here and refer the reader to [34], or to [3] for proofs that are more specific to the topic of testing.

The following lemma is a strong tool in graph theory, as it states that in essence most of the vertex-pairs of any given graph can be neatly fitted into a structure of regular pairs. This structure is given by an *equipartition* of G – a partition of its vertex set into sets whose sizes differ from each other by no more than 1.

Lemma 5.3 (Szemerédi's Regularity Lemma [45]) *For every m and $\epsilon > 0$ there exists some $T = T(m, \epsilon)$ with the following property.*

If G is a graph with $n \geq T$ vertices, and $\mathcal{A}$ is an equipartition of the vertex set of G into no more than m sets, then there exists an equipartition $\mathcal{B}$ that is a refinement of $\mathcal{A}$ with k sets, where $m \leq k \leq T$, for which all set pairs but at most $\epsilon\binom{k}{2}$ of them are ϵ-regular.

By using the Regularity Lemma, one could re-prove for example that proper k-colorability is testable – in fact, a combinatorial statement to this effect was implicitly given in [40], before the concept of algorithmic testing came into being.

Another handy use is when testing for the non-existence of fixed size cliques; see for example the following combinatorial proposition.

Proposition 5.4 (N. Alon, private communication) *For every ϵ there exists $\delta = \delta(\epsilon)$ such that any graph with n vertices, that is ϵ-far from being triangle-free, contains at least $\delta\binom{n}{3}$ distinct triangles.*

Proof sketch: To prove this we consider an equipartition $\{V_1, \ldots, V_k\}$ of the vertices of the graph G according to Lemma 5.3, where $5\epsilon^{-1} \leq k \leq T(5\epsilon^{-1}, \epsilon')$ (we make it a refinement of an arbitrary equipartition into $5\epsilon^{-1}$ sets), in which all but at most $\epsilon'\binom{k}{2}$ of the pairs are ϵ'-regular. The parameter ϵ' is set to $\min\{\frac{1}{5}\epsilon, \gamma(\frac{1}{5}\epsilon)\}$, where γ is the function appearing in Lemma 5.2.

We now consider a subgraph G' of G, obtained by deleting all edges of G that are internal to V_i for any i, or are between V_i and V_j for any pair V_i, V_j which is not ϵ'-regular, or whose density is less than $\frac{1}{5}\epsilon$.

Noting that G' was obtained from G by deleting less than $\epsilon\binom{n}{2}$ edges, it follows that G' contains a triangle, since G is ϵ-far from being triangle-free. By the construction of G', this is necessarily a triangle with a vertex from V_i, a vertex from V_j and a vertex V_k, for some V_i, V_j, V_k which are all distinct and for which all pairs are in particular $\gamma(\frac{1}{5}\epsilon)$-regular (with respect to G) and with density at least $\frac{1}{5}\epsilon$. Thus the existence of such V_i, V_j, V_k implies by Lemma 5.2 the existence of at least $\delta'\binom{n}{3}$ distinct triangles in G for $\delta' = 6\delta(\frac{1}{5}\epsilon)(T(5\epsilon^{-1}, \epsilon'))^{-3}$, where δ denotes the corresponding function from Lemma 5.2. ■

From Proposition 5.4 it follows that the following is a 1-sided ϵ-test for being triangle-free: Choose independently and uniformly $O(\delta(\epsilon)^{-1})$ random vertex triplets (where δ is the function of the proposition), and check whether any of them is a triangle. If a triangle was found then reject the input; otherwise accept.

There is one serious drawback to results proven using the Regularity Lemma: The dependency of $T(m, \epsilon)$ on its parameters is very severe, a tower in a polynomial of $m\epsilon^{-1}$ in fact, and by [32] this dependency cannot be significantly improved as long as the full strength of the Regularity Lemma is employed. This makes the tests thus derived of a theoretical importance rather than a practical one.

Unlike the situation with proper k-colorability, currently there is no known test (with a constant number of queries) for the property of being triangle-free that has a better dependency on ϵ. In fact, it is known that the number of queries of a 1-sided test for this property cannot be polynomial in ϵ, using a bound from [12].

When one tries to generalize the above result to properties defined as not containing a fixed given induced subgraph, other than a clique, another problem arises (an induced subgraph is a subgraph obtained from the original graph by deleting vertices and their incident edges, but deleting no other edges). In the above proof implicit use was made of the fact that removing edges cannot *add* new triangles to the graph; but in the case of induced squares, for example, removing an edge can actually make its pair part of a new specimen of the forbidden subgraph.

In proving the Regularity Lemma one cannot do away completely with the existence of a small fraction of non-regular pairs, which are the core of the problem in proving similar results for induced subgraphs, because of the above mentioned inability to safely remove edges. The workaround in [3] to this problem is in proving a variant of the Regularity Lemma, that enables one to find a rather large induced subgraph of the original graph, for which there exist an equipartition with all the pairs being regular, and that has the additional property of being able to "model" the original graph in the sense that if the graph G is far from satisfying the property then so is this subgraph.

The full details are outside the scope of this survey; the reader is referred to [3], where this method is used for proving that all '$\exists\forall$' first order graph properties are testable, and to [20], where testability is proven for some additional classes of properties that further generalize the '$\exists\forall$' class.

As a final note, we mention some instances in which using the full version of the Regularity Lemma can be avoided, making for a better dependency of the number of queries on ϵ. It still remains to be seen whether such a use can be avoided in the general case, or even for the property of a graph being triangle-free.

In the case where the property is that of not containing a fixed bipartite graph as a not necessarily induced subgraph, this property can be tested with a number of queries polynomial in ϵ. The proof uses a Zarankiewicz type theorem, that states in essence that the only graphs satisfying such a property are almost edge-less. On the other hand, there exists no 1-sided test making a number of queries that is polynomial in ϵ^{-1}, for any property defined as not containing a fixed non-bipartite graph; this was proven in [1].

Suppose that a property is defined by not containing a fixed *induced* subgraph, and that we wish to test only bipartite graphs for satisfying it. The following is achieved using a rather restricted variant of the Regularity Lemma, proven for this purpose.

Theorem 5.5 ([22] with an improvement by N. Alon) *Any property of bipartite graphs, that is defined as not containing a given fixed induced subgraph, has a 2-sided test using a number of queries exponential in ϵ, and a 1-sided test using a number of queries doubly-exponential in ϵ.*

It is not known at present whether testing of such properties of bipartite graphs can be made polynomial in ϵ^{-1} or not.

6 Testing of sparse graphs

As far as inputs given in terms of the adjacency matrix of a graph are concerned, all graphs with $o(n^2)$ edges are indistinguishable from null (edge-less) graphs. This makes testing of properties such as not containing a cycle trivial – the algorithm just needs to sample and distinguish between the case that G has more than $\epsilon\binom{n}{2}$ edges, and the case that G has less than $\frac{1}{2}\epsilon\binom{n}{2}$ edges. In the first case the algorithm rejects G, and in the second accepts.

However, there is still the interesting (and non-trivial) question of whether one could, for an input graph G with n vertices and m edges, distinguish between the case that G is acyclic, and the case that no removal of $\epsilon(n+m)$ edges from G (rather than $\epsilon\binom{n}{2}$) can make it so. In [31] Goldreich and Ron have defined a different model for encoding input graphs, in which such a testing notion makes sense ($\epsilon(n+m)$ is used, and not ϵm, because of technical complications arising in graphs with $o(n)$ edges).

Under the model discussed here, a graph is not encoded by its adjacency matrix, but instead the list of neighbors is given for each vertex (it is assumed that the input is indeed an encoding of a graph, so if u is in the list of neighbors of v then v is also in the list of u). Note that the input is not a sequence of bits, but a sequence of integers, some of which serve as 'pointers' to the different lists. Thus a query in this model consists of either finding out the number of neighbors of a vertex v_i, or finding the identity of the j'th neighbor of v_i according to its adjacency list (in the bounded degree model defined below we shall only need queries of the second type).

In the case where inputs are limited to graphs with a fixed maximum degree Δ, and the allowable modifications of the graphs are thus restricted as well, one may assume that all the adjacency lists are of this fixed size, with the possibility of some members in the lists being 'null' for vertices with a degree less than Δ. In this model the difference between graphs in relation to $n+m$ is linearly proportional to the hamming distance between inputs, so we can just use the latter.

When the maximum degree is not bounded, the notion of the distance is more complex because the sizes of the adjacency lists can vary. The distance between inputs is actually a function of the edit distances between the corresponding adjacency lists; there are in addition models in which it is also allowed to add and remove vertices from the graph (in removing a vertex it is natural to take into account also the removal of its incident edges), but we shall not discuss all models in detail here.

In a work by Goldreich and Ron [31] several properties (some of which, such as connectivity and acyclicity, are trivial to test in the dense graph model but are not trivial in this one) are proven to be testable. On the other hand, some other properties that are easily testable in the dense context are not testable in this one. The following is a partial summary of their results.

Theorem 6.1 ([31]) *The following properties are testable with a number of queries not depending* *on n, for the model of testing graphs with a bounded degree Δ given by their adjacency lists: k-edge connectivity for any fixed given k, not having a given fixed graph H as a (not necessarily induced) subgraph, and having no cycles.*

On the other hand, ϵ-testing of bipartiteness for some fixed ϵ, and $\Delta = 3$, requires at least $\frac{1}{3}\sqrt{n}$ queries.

In this context a testing algorithm typically consists of performing a breadth first search from randomly chosen vertices, and then accepting or rejecting the input based on the resulting subgraphs.

For example, testing for connectedness can be done using the following simple lemma (it is also not hard to see that this lemma is translatable to the context where the input graph and its modification are subject to a bound on their maximum degree).

Lemma 6.2 *If a graph G with n vertices requires the addition of at least ϵn edges to make it connected, then at least $\frac{1}{2}\epsilon n$ of the vertices of G are in connected components of size at most $2\epsilon^{-1}$.*

Proof: It is easy to see that such a graph G contains more than ϵn connected components. Each of them contains at least one vertex, and on the other hand it is clear that less than half of them can be of size more than $2\epsilon^{-1}$. ■

The test now consists of picking $O(\epsilon^{-1})$ random vertices, and performing a breadth first search from each to them to check whether it is part of a connected component with less than $2\epsilon^{-1}$ vertices (this takes $O(\epsilon^{-1})$ queries per vertex). If one of the chosen vertices is in such a component then the algorithm rejects, and otherwise it accepts.

The above test takes $O(\epsilon^{-2})$ queries, and works for both the bounded degree model and the unbounded degree one. Note however that [31] contains an improved analysis, that in particular requires only $O(\epsilon^{-1})$ queries for the unbounded degree model, and slightly more than that for the bounded degree one.

7 Testing posets for monotonicity

Keeping lists in a sorted order is a common practice, and for good reasons. It is then natural to ask how many queries it takes to ensure that a sequence of n integers is mostly sorted, that is, to distinguish between the case that it is monotone nondecreasing, and the case that no subsequence of $(1-\epsilon)n$ integers from it is monotone nondecreasing.

This is equivalent to the question of ϵ-testing a sequence of n integers for this property, where the distance is given by the fraction of integers that need to be altered in order to make the sequence monotone. Ergün, Kannan, Kumar, Rubinfeld and Viswanathan have shown in [17] that for a fixed ϵ it is enough to make $O(\log n)$ queries. Although this is dependent on n, it is still a much better alternative to reading the whole input.

Theorem 7.1 ([17]) *There exists an ϵ-test that makes $O(\epsilon^{-1}\log n)$ queries, for the property of a sequence of n integers being monotone nondecreasing.*

Proof sketch: For any monotone increasing sequence of size n, one can check using $O(\log n)$ queries whether a given number is in that sequence, by the well known binary search method.

We now look in our input, denote it by $v_1 v_2 \ldots v_n$. We can safely assume that all values are distinct, as otherwise we can consider the sequence $v'_1 v'_2 \ldots v'_n$ defined by $v'_i = nv_i + i$ instead. We call v_i *well positioned* if the binary search procedure above, when applied to v_i over $v_1 v_2 \ldots v_n$, indeed finds the position i.

It takes $O(\log n)$ queries to check that a given v_i is well positioned. Our algorithm makes $O(\epsilon^{-1})$ iterations of this procedure for random i's chosen uniformly and independently, to distinguish between the case that all values are well positioned, and the case that at least ϵn of the values are not well positioned. In the first case the algorithm accepts the input, and in the second case rejects.

If the input is in fact monotone, then clearly all its values are well positioned, and so the algorithm accepts. On the other hand, if the algorithm accepts, then it means that more than $(1-\epsilon)n$ of the values are well positioned. To end the proof, we show that the subsequence of all well positioned values is monotone; such a subsequence can easily be completed to a monotone nondecreasing sequence of size n by putting new values in the places where the original values are not well positioned.

Assuming that $i < j$, and that v_i and v_j are both well positioned, we conclude by showing that $v_i \le v_j$. For this it is sufficient to find k satisfying $i \le k \le j$ such that v_k was visited by the binary searches for both values, because this would imply $v_i \le v_k$ and $v_k \le v_j$. To find it, we just pick k as the last location that was queried in both binary searches for i and j. Common locations to the two binary searches exist because in particular the location $\lfloor \frac{n}{2} \rfloor$ is common to all binary searches.

■

One should note here that the above algorithm only looks adaptive. In fact, the algorithm can supply all the query locations in advance – to check that a value is well positioned, it is enough to unconditionally query the binary search sequence that would arise if it were well positioned, since to learn that a value is not well positioned it is enough to know that the actual binary search sequence diverges at some point from these queried locations.

As it turns out, the above property cannot be tested using a constant number of queries. An $\Omega(\log n)$ lower bound (for some constant ϵ) is given in [17] for non-adaptive algorithms that are restricted to making decisions based on comparisons. In [19] it is proven that for this particular property better general algorithms do not exist, by showing first that for a certain class of properties, including monotonicity, there are comparison-based algorithms that are optimal.

It is also of interest to investigate testing for natural notions of monotonicity that differ from the simple notion of a sequence being nondecreasing. The next step is to consider functions whose domain is not endowed with a linear order as $\{1, \ldots, n\}$ is, but with other types of partial orders instead. The first work in this direction is by Goldreich, Goldwasser, Lehman, Ron and Samorodnitsky [28], that have proven the following result about the monotonicity of functions from $\{0,1\}^d$ (with the usual product order over the hypercube) to $\{0,1\}$.

Theorem 7.2 ([28]) *ϵ-testing that a function from $\{0,1\}^d$ to $\{0,1\}$ is monotone nondecreasing can be done using $O(\epsilon^{-1}d)$ queries.*

The testing algorithm for this consists of choosing a random pair of points in $\{0,1\}^d$ that differ in exactly one coordinate, comparing the values of the input function over these, and repeating this procedure $O(\epsilon^{-1}d)$ times. For the proof of the correctness of this the reader is referred to [28].

A generalization of the above result to d-dimensional matrices of any size (and not only $\{0,1\}^d$) is given in [16].

Theorem 7.3 ([16]) *A function from $\{1, \ldots, n\}^d$ to $\{1, \ldots, m\}$ can be ϵ-tested for monotonicity using $O(\epsilon^{-1}d \log n \log m)$ queries.*

Dealing with d-dimensional matrices with integer values (not restricted to some $\{1, \ldots, m\}$) is also covered by [16] – a simple analysis of the algorithm that is given there implies that the 'm' term in the expression for the number of queries can be replaced by the actual size of the image of the input function, which is bounded by n^d. This gives an $O(\epsilon^{-1}d^2 \log^2 n)$ bound on the number of required queries in this case.

As for binary functions over $\{1, \ldots, n\}^d$, in [16] it is shown that in this case the number of queries does not need to depend on n at all.

Other partially ordered sets can also serve as domains for functions to be tested for monotonicity. For example, trees. There is currently a work in progress [21] dealing with generalized monotonicity testing, showing in particular the possibility of testing monotonicity of a binary function over a tree. In [21] there are also non-trivial lower bounds for the number of queries required to test for monotonicity, including the case of a function from $\{0,1\}^d$ to $\{0,1\}$; in particular, this number has to depend on d.

8 Proving non-testability results

The methods presented thus far deal with proving upper bounds for property testing, through means of presenting efficient testing algorithms. With the emergence of property testing, lower bounds naturally become interesting as well, and the first example of a property that is hard to test (one that belongs to NP in the traditional complexity framework) was given in [29].

This section presents what is currently the most efficient tool in proving lower bounds for property testing, Yao's method [46]. The idea is the following. Instead of trying to find a worst case input to every possible (randomized) ϵ-testing algorithm, we find a probability distribution over the inputs, such that every *deterministic* algorithm will fail with probability close to $\frac{1}{2}$ on an input taken according to this distribution. From this it follows that every randomized algorithm will also fail with at least the same probability over this input distribution, and so for every fixed randomized algorithm there will be at least one ("non-randomized") input that achieves at least the same average failure rate.

In our context lower bounds are proven by finding two distributions, one on inputs satisfying the property and one on inputs that are ϵ-far (for some constant ϵ) from satisfying it, such that for every set of q possible queries, the probability distribution of the values of the input in these locations will be nearly the same in both cases. This means that q queries are insufficient for testing for this property by a non-adaptive algorithm, even with a 2-sided error probability. In cases where binary inputs are involved, a lower bound of q queries on non-adaptive algorithms implies a lower bound of $\log q$ queries on adaptive ones (see the end of Section 9 for more information on this), which suffices if it is only required to prove that a constant number of queries is insufficient for testing a particular property. Sometimes the gap between adaptive and non-adaptive properties can be narrowed further, by using some feature of the particular hard to test property or by getting the input distributions to satisfy certain additional conditions, thus achieving stronger lower bounds for adaptive algorithms as well.

For lower bounds on non-adaptive 1-sided algorithms, it is enough to find a distribution over inputs that are ϵ-far from satisfying the property, but in which for any fixed set of q locations, with high probability the values of the input on these location is extensible to some input that satisfies the property. A concrete example can help explain this – in the case of testing for a graph being triangle free, a lower bound that is super-polynomial in ϵ^{-1} (but in this case is of course independent of the input size) is given using a graph that is far from being triangle free, but still does not contain too many distinct triangles, so q queries will not capture a triangle with high probability. Such a graph can be constructed using the number-theoretic construction in [12]; the details and a generalization thereof to other properties defined in terms of not containing a fixed (not necessarily induced) subgraph are found in [1].

As the main example here for using Yao's method (and one that concerns 2-sided algorithms as well), we consider the following problem: Given as input the adjacency matrices of two graphs with n vertices, test for the property of these two graphs being isomorphic.

This property was shown in [3] to be non-testable, and this was then used to show that there exist non-testable '$\forall\exists$' type first order graph properties. The original exposition of the proof in [3] uses counting instead of Yao's method; here we give a proof using Yao's method that can serve as an example for proving lower bounds.

Proposition 8.1 ([3]) *The property of two graphs (given by their adjacency matrices) being isomorphic is not $\frac{1}{8}$-testable with any constant number of queries.*

Proof: We shall show that for any q there exists n, such that q edge queries are insufficient for distinguishing between the case that the two graphs are isomorphic, and the case that all vertex permutations of the first graph differ from the second graph in more than $\frac{1}{4}\binom{n}{2}$ places (remember that the total size of the input is $2\binom{n}{2}$).

For this we construct distributions over inputs. The distribution D_P is constructed by letting the input consist of a random graph G, with each edge being independently chosen with probability $\frac{1}{2}$, and a second graph that is a uniformly random permutation of G. The distribution D_N is constructed by letting the input consist of a random graph G as before, and another, independently chosen, random graph G'.

It is clear that an input chosen according to D_P satisfies the property with probability 1. On the other hand, by a standard large deviation argument, in an input chosen according to D_N any fixed vertex permutation of G will with probability more than $1 - 2^{-n^2/50}$ differ in more than $\frac{1}{4}\binom{n}{2}$ places from G'. Thus, with probability $1 - o(1)$, all vertex permutations of G will have at least this difference from G'.

We now show that no non-adaptive algorithm will distinguish between an input selected according to D_P, and an input selected according to D_N (that is, it will have nearly the same acceptance probability for both); we recall that it is sufficient to show it for deterministic algorithms.

Suppose that the algorithm has queried the vertex pairs $p_1, \ldots, p_q$, some inside the first graph, some inside the second one. For an input chosen according to D_N, the values of these pairs will be q uniformly and independently chosen random values in $\{0, 1\}$. We shall show that the distribution of these values is also nearly uniform and independent when choosing an input according to D_P.

For this, let $u_1, \ldots, u_k$ be all vertices of the first graph that are contained in any pairs queried by the algorithm, and let $v_1, \ldots, v_l$ be all such vertices of the second graph. It is clear that $k \leq 2q$ and $l \leq 2q$. Now let $\sigma : \{1, \ldots, n\} \to \{1, \ldots, n\}$ be the permutation used in D_P to choose the second graph according to the first. Let E denote the event that, for this permutation, $\sigma(u_i) \neq v_j$ for every $1 \leq i \leq k$ and $1 \leq j \leq l$. It is clear that, conditioned on the case that E occurs, $p_1, \ldots, p_q$ will be again q uniformly and independently chosen values of $\{0, 1\}$. But since for n large enough (as a function of q) the event E occurs with probability close to 1, it means that the (unconditioned) distribution of $p_1, \ldots, p_q$ will be close to the uniform independent distribution also when an input is chosen according to D_P.

The above shows that the queries $p_1, \ldots, p_q$ cannot distinguish between an input chosen according to D_P and one chosen according to D_N. This means that if we define a distribution D as choosing with probability $\frac{1}{2}$ an input according to D_P and with probability $\frac{1}{2}$ an input according to D_N, then any deterministic nonadaptive algorithm making q queries will fail on it with probability close to $\frac{1}{2}$. By applying Yao's method we are done. ∎

The proof of the negative part of Theorem 5.1 in [3] was done by proving that there exists a '$\forall\exists$' first order graph property that is indistinguishable from some graph isomorphism property similar to the one above.

As a final remark, it can be shown that the distributions of $p_1, \ldots, p_k$ in the two cases become close enough for our purposes when $n = \Theta(q^2)$. Since it can also be shown that in testing of graph properties there can only be a quadratic gap between adaptive and non-adaptive algorithms (see [3]), this means that in fact the above property cannot be tested with a number of queries that is less than some power of $2\binom{n}{2}$, the input size.

9 Related research and future directions

Property testing is a young field, that currently enjoys rapid growth. This section's purpose is to give a glimpse into some topics not covered by the rest of this survey. Some of them were left uncovered due to size or scope considerations, while others were left out simply because they are still waiting to be investigated. The interested reader is also referred to other surveys of this and related fields, such as [41], [33] and [27].

9.1 Other works concerning testability

As is the case with all surveys, this one could not cover anything and everything in the field. The following (also partial) list is given in acknowledgement of this fact.

Other algebraically motivated properties: The question of whether an input function over a vector space forms a bounded degree polynomial is just one of several algebraically motivated properties. For example, one could ask whether a 'multiplication table' given as input forms a (finite) group. In [17] some upper and lower bounds are given for this. For a supposed group with n elements there is an ϵ-test using $O(n^{3/2}\log^c n)$ queries, for some global c; this is a rather large quantity, but it is still much smaller then the size of the input, n^2. One should also note that in the context of prover assisted testing (see below), this quantity can be further reduced.

Another topic, somewhat related to testing polynomiality, is testing whether a given function satisfies a property defined by a functional equation, for example "$f(x+y)+f(x-y) = 2f(x)f(y)$". Some properties of this type are investigated for functions defined over certain subsets of $\mathcal{R}$, in [42].

Testing of matrix poset properties: Another way to generalize the result about testing binary matrices for monotonicity is in extending it to other properties of matrices, defined in terms of first order expressions using the partial order that underlies a matrix (that is, the product order of each location's coordinates). Such properties are investigated in [22].

Geometrically motivated properties: In this context the input is not a binary sequence or a sequence of integers, but a sequence of real numbers. As a first example, it is shown in [17] how to test, using $O(\log n)$ queries, a sequence of n points from $\mathcal{R}^2$ for the property of being the nodes of a convex polygon given in a clock-wise order. In [15] testing for several geometric properties of points from $\mathcal{R}^d$, such as being the nodes of a convex polygon given in *any* order, is considered. For most of these matching upper and lower bounds are given; these are rather large (most are fixed powers of n), but are still much smaller than the input size.

Another example, given a fixed d, is testing for the property of a given set of vectors in $\mathcal{R}^d$ being (k, b)-clusterable. Usually in such a context a more relaxed notion of testing needs to be used; here it is only possible to efficiently distinguish between the case that the input is (k, b)-clusterable, and the case that even removing an ϵ fraction of the points will not make the rest $(k, (1+\epsilon)b)$-clusterable. The testing procedure is given in [2].

A geometrically motivated problem in a different direction is that of testing whether a given matrix of real numbers has certain properties, that are expected from a distance function of a discrete metric space. Properties of this type are investigated in [38].

9.2 Some topics related to testability

Property testing is already making itself felt in other areas of computer science. The following are some topics that received some inspiration from this field.

Low-rank approximations: There are many applications to finding a low-rank matrix that approximates a given input matrix. In [25] it is shown how to construct a low-rank approximation for a given matrix (in the form of a fast oracle) in time that depends only on the approximation parameters and not on the size of the approximated matrix. While strictly speaking this is not a testability result, it does share some of the methods. Other results regarding fast approximation of matrices are given in [24].

Testing with a prover: This topic deals with accepting or rejecting an input based on communicating with a computationally unbounded prover. It is somewhat like interactive proofs, only the operating requirements of the verifier here are the ones used in the field of property testing.

The general problem is to distinguish between the case that the input satisfies a property, and the case that the input is ϵ-far from satisfying the property. In general the prover is allowed to supply a long proof, but the verifier is still allowed only a small number of queries from the input as well as from the supplied proof.

Testing with a prover is investigated in [18] and [9]; in these the running time of the verifier is restricted as well, and this poses some restrictions on the size of the proof through the size of the allowable address-space. As expected, there are problems that are hard to test in the usual context but are easy to test with a prover. For example, it takes $\Omega(\sqrt{n})$ queries to test that a function $f : \{1, \ldots, n\} \to \{1, \ldots, n\}$ is a bijection, but if a prover provides a function g that is supposedly its inverse, then it takes a constant number of queries to test that f and g are indeed close to being inverses of each other.

Statistical deductions: In this context one would like to check certain properties of a distribution (say, the distribution of the heights of NBA basketball players), where closeness is defined by the L_1 norm $|A - B| = \sum_{i \in R} |A(i) - B(i)|$. In addition, instead of querying, one can only obtain a sample of values that were independently chosen using this distribution (in the above example, one can only measure the heights of randomly selected NBA players).

When the distribution is defined over a set of n elements, it takes many samples to 'ϵ-test' (under this new definition) for a property, but in many cases it still takes much less than the $\Omega(n \log n)$ samples needed to actually write down an approximation of the whole distribution. For example, based on the methods of [30] one can prove that $O(\sqrt{n} \log^c n)$ queries (for some global constant c) are sufficient to ϵ-test for the property of being the uniform distribution over the whole set. On the other hand, it was proven that at least $\Omega(\sqrt{n})$ samples are required for this. Further results in this direction include those about testing that two distributions (also given by samples) are close, and that a joint distribution (with two 'coordinates') is independent. These are given in [11] and [10] respectively.

Quantum testability: Just as other notions of computational and informational complexity have their quantum counterparts, a quantum counterpart of the notion of property testing is being developed as well. There are already results in [14] showing that in some cases quantum computers can efficiently test properties that classical computer cannot, while some other properties are equally hard to test in the quantum context.

9.3 A few more questions

Let us conclude with a few possible future directions in property testing, that some of the people in the field find interesting. The following list is not meant to be exhaustive or representative, and was chosen based mainly on personal preference.

Weighted testing: In most of the cases described above, the distance used in the definition of being ϵ-far from satisfying a property is the hamming distance between inputs. It is interesting to investigate also counterparts of the above testability questions in which closeness is measured by a weighted hamming distance.

In the most lenient model it is assumed that the algorithm is given all the weights in advance – thus the weights are not considered as part of the input, but as part of the description of the parameters within which the algorithm is supposed to operate. For some of the results, such as the result in [36] about testing of languages recognizable by a read-once bounded width oblivious branching program, the proof extends naturally to the weighted case. In other instances, such as the testing that a graph is triangle free, the question of the weighted case seems very hard (though some special cases, such as a weight function generated by giving weights to the vertices and using their multiples for the edges, can be deduced from the unweighted case). It seems very probable that for many properties testability in fact does not extend to the weighted case.

Connections to traditional complexity: There are properties that are hard to decide but are easy to test (such as graph 3-colorability), and properties that are easy to decide (even AC_0 ones) but are hard to test. It would be interesting to take the approximating feature of testing into account, and formulate some connections between testable properties (subject to some uniformity assumptions on the testing algorithms) and easy to decide properties that approximate them, apart from the nearly obvious connection to Promise-BPP (for graph properties it can also be made into a connection with P, using the Regularity Lemma).

Another somewhat related question is this: What is the expressive power of first order graph expressions when approximations are taken into account? The results of [3] suggest that it may be stronger then expected.

Adaptive versus non-adaptive testing: A reminder – a non-adaptive testing algorithm is an algorithm that has to supply all the queries in advance, and only then is given the corresponding input values, based on which it accepts or rejects the input. In general, an adaptive ϵ-test for a property of binary sequences using q queries can be converted to a non-adaptive one using up to $2^q - 1$ queries – one just goes over the whole decision tree of the adaptive algorithm, makes all queries appearing there in advance, and then navigates the tree using the query results in order to accept or reject the input. There are known properties for which this gap is essential, for example properties derived from the sparse graph model investigated in [31]; the discussion in [19] contains some examples and details.

For graph properties in the dense model, it is proven in [3] that an adaptive algorithm making q queries can be converted to a non-adaptive one making only $\binom{2q}{2}$ queries. A look at the simple proof reveals that it is based on the fact that graph properties are invariant with respect to graph isomorphisms. Examining other properties, such as counting properties that are invariant with respect to any permutation of the input (in [8] arguments related to such invariance are used for proving lower bounds for certain approximation problems), one gets the impression that a larger invariance group of a property implies a smaller gap between its adaptive and non-adaptive testing capabilities. It would be interesting to formalize and prove this observation.

References

[1] N. Alon, Testing subgraphs in large graphs, *Proceedings of the* 42^{nd} *IEEE FOCS* (2001), to appear.

[2] N. Alon, S. Dar, M. Parnas and D. Ron, Testing of clustering, *Proceedings of the* 41^{st} *IEEE FOCS* (2000), 240–250.

[3] N. Alon, E. Fischer, M. Krivelevich and M. Szegedy, Efficient testing of large graphs, *Combinatorica* 20 (2000), 451–476.

[4] N. Alon, M. Krivelevich, I. Newman and M. Szegedy, Regular languages are testable with a constant number of queries, *SIAM Journal on Computing* 30 (2001), 1842–1862.

[5] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy, Proof verification and the hardness of approximation problems, *Journal of the ACM* 45 (1998), 501–555 (a preliminary version appeared in Proc. 33^{rd} FOCS, 1992).

[6] S. Arora and S. Safra, Probabilistic checking of proofs: A new characterization of NP, *Journal of the ACM* 45 (1998), 70–122 (a preliminary version appeared in Proc. 33^{rd} FOCS, 1992).

[7] S. Arora and M. Sudan, Improved low-degree testing and its applications, *Proceedings of the* 29^{th} *ACM STOC* (1997), 485–495.

[8] Z. Bar-Yossef, R. Kumar and D. Sivakumar, Sampling algorithms: Lower bounds and applications, *Proceedings of the* 33^{rd} *ACM STOC* (2001), 266–275.

[9] T. Batu, R. Rubinfeld and P. White, Fast approximation PCPs for multidimensional bin-packing problems, *Proceedings of the* 3^{rd} *International Workshop on Randomization and Approximation Techniques in Computer Science* (1999), 246–256.

[10] T. Batu, E. Fischer, L. Fortnow, R. Kumar, R. Rubinfeld and P. White, Testing random variables for independence and identity, *Proceedings of the* 42^{nd} *IEEE FOCS* (2001), to appear.

[11] T. Batu, L. Fortnow, R. Rubinfeld, W. Smith and P. White, Testing that distributions are close, *Proceedings of the* 41^{st} *IEEE FOCS* (2000), 259–269.

[12] F. A. Behrend, On sets of integers which contain no three terms in arithmetic progression, *Proceedings of the National Academy of Sciences of the United States of America* 32 (1946), 331–332.

[13] M. Blum, M. Luby and R. Rubinfeld, Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences* 47 (1993), 549–595 (a preliminary version appeared in Proc. 22^{nd} STOC, 1990).

[14] H. Buhrman, L. Fortnow, I. Newman and H. Röhrig, Quantum property testing, manuscript.

[15] A. Czumaj, C. Sohler and M. Ziegler, Property testing in computational geometry, *Proceedings of the 8th Annual European Symposium on Algorithms* (2000), 155–166.

[16] Y. Dodis, O. Goldreich, E. Lehman, S. Raskhodnikova, D. Ron and A. Samorodnitsky, Improved testing algorithms for monotonicity, *Proceedings of the* 3^{rd} *International Workshop on Randomization and Approximation Techniques in Computer Science* (1999), 97–108.

[17] F. Ergün, S. Kannan, R. Kumar, R. Rubinfeld and M. Viswanathan, Spot checkers, *Journal of Computer and System Science* 60 (2000), 717–751 (a preliminary version appeared in Proc. 30^{th} STOC, 1998).

[18] F. Ergün, R. Kumar and R. Rubinfeld, Fast approximate PCPs, *Proceedings of the* 31^{st} *ACM STOC* (1999), 41–50.

[19] E. Fischer, On the strength of comparisons in property testing, manuscript (available as ECCC TR00-083).

[20] E. Fischer, Testing graphs for colorability properties, *Proceedings of the* 12^{th} *ACM-SIAM SODA* (2001), 873–882.

[21] E. Fischer, E. Lehman, I. Newman, S. Raskhodnikova, R. Rubinfeld, A. Samorodnitsky and M. Sipser, in preparation.

[22] E. Fischer and I. Newman, Testing of matrix properties, *Proceedings of the* 33^{rd} *ACM STOC* (2001), 286–295.

[23] E. Fischer and I. Newman, A function that has read-twice small width branching program but is not ϵ-testable, in preparation.

[24] A. Frieze and R. Kannan, Quick approximation to matrices and applications, *Combinatorica* 19 (1999), 175-220.

[25] A. Frieze, R. Kannan and S. Vempala, Fast Monte-Carlo algorithms for finding low-rank approximations, *Proceedings of the* 39^{th} *IEEE FOCS* (1998), 370–378.

[26] P. Gemmell, R. Lipton, R. Rubinfeld, M. Sudan and A. Wigderson, Self-testing/correcting for polynomials and for approximate functions, *Proceedings of the* 23^{rd} *ACM STOC* (1991), 32–42.

[27] O. Goldreich, Combinatorial property testing – a survey, In: *Randomization Methods in Algorithm Design* (P. Pardalos, S. Rajasekaran and J. Rolim eds.), AMS-DIMACS (1998), 45–60.

[28] O. Goldreich, S. Goldwasser, E. Lehman, D. Ron and A. Samorodnitsky, Testing monotonicity, *Combinatorica*, to appear (a preliminary version written by the first four authors appeared in Proc. 39^{th} FOCS, 1998).

[29] O. Goldreich, S. Goldwasser and D. Ron, Property testing and its connection to learning and approximation, *Journal of the ACM* 45 (1998), 653–750 (a preliminary version appeared in Proc. 37^{th} FOCS, 1996).

[30] O. Goldreich and D. Ron, On testing expansion in bounded-degree graphs, manuscript (available as ECCC TR00-20).

[31] O. Goldreich and D. Ron, Property testing in bounded degree graphs, *Proceedings of the* 29^{th} *ACM STOC* (1997), 406–415.

[32] W. T. Gowers, Lower bounds of tower type for Szemerédi's Uniformity Lemma, *Geometric and Functional Analysis* 7 (1997), 322–337.

[33] M. Kiwi, F. Magniez and M. Santha, Exact and approximate testing/correcting of algebraic functions: A survey, manuscript (available as ECCC TR01-014).

[34] J. Komlós and M. Simonovits, Szemerédi's Regularity Lemma and its applications in graph theory, In: *Combinatorics, Paul Erdös is Eighty*, Vol II (D. Miklós, V. T. Sós, T. Szönyi eds.), János Bolyai Math. Soc., Budapest (1996), 295–352.

[35] L. Lovász and N. Young, Lecture notes on evasiveness of graph properties, Technical report, Princeton University, 1994. Available at:
`http://www.uni-paderborn.de/fachbereich/AG/agmadh/WWW/english/scripts.html`

[36] I. Newman, Testing of functions that have small width branching programs, *Proceedings of the* 41^{st} *IEEE FOCS* (2000), 251–258.

[37] C. H. Papadimitriou, *Computational complexity*, Addison-Wesley (1994).

[38] M. Parnas and D. Ron, Testing metric properties, *Proceedings of the* 33^{rd} *ACM STOC* (2001), 276–285.

[39] M. Parnas, D. Ron and R. Rubinfeld, Testing parenthesis languages, *Proceedings of the* 5^{th} *International Workshop on Randomization and Approximation Techniques in Computer Science* (2001), 261–272.

[40] V. Rödl and R. Duke, On graphs with small subgraphs of large chromatic number, *Graphs and Combinatorics* 1 (1985), 91–96.

[41] D. Ron, Property testing (a tutorial), In: *Handbook of Randomized Computing* (S. Rajasekaran, P. M. Pardalos, J. H. Reif and J. D. P. Rolim eds), Kluwer Press (2001).

[42] R. Rubinfeld, On the robustness of functional equations, *SIAM Journal on Computing* 28 (1999), 1972–1997.

[43] R. Rubinfeld and M. Sudan, Robust characterization of polynomials with applications to program testing, *SIAM Journal of Computing* 25 (1996), 252–271 (first appeared as a technical report, Cornell University, 1993).

[44] R. Rubinfeld and M. Sudan, Testing polynomial functions efficiently and over rational domains, *Proceedings of the* 3^{rd} *Annual ACM-SIAM Symposium on Discrete Algorithms* (1992), 23–43.

[45] E. Szemerédi, Regular partitions of graphs, In: *Proc. Colloque Inter. CNRS* No. 260 (J. C. Bermond, J. C. Fournier, M. Las Vergnas and D. Sotteau eds.), 1978, 399–401.

[46] A. C. Yao, Probabilistic computation, towards a unified measure of complexity. *Proceedings of the* 18^{th} *IEEE FOCS* (1977), 222–227.