Ramifications and related topics

Oded Goldreich[*]

May 28, 2016

Warning: It seems that these lecture notes are headed towards becoming part of a book. So I allowed myself free citations to other chapters and sections, although these will generate somewhat cryptic labels.

In continuation to Section `[intro:rami]`, we review a few ramifications of the notion of property testers as well as related topics. Some of these were briefly mentioned in that section and others were not even mentioned there.

Summary: We briefly review a few ramifications of the notion of property testers as well as related topics. The list includes

1. Tolerant testing and distance approximation.
2. Testing under additional promises on the input.
3. Sample-based testers.
4. Other distance measures.
5. Local computation algorithms.
6. Non-interactive proofs of proximity (MAPs).

The different sections of this chapter can be read independently of one another.

1 Tolerant testing and distance approximation

In some setting objects that are close to having the property may be almost as good for these settings as objects that have the property (e.g., see some of the settings discussed in Section `[intro:benefits]`). But in such a case, when testing for the property, it may be desirable not to reject objects that are very close to having the property (or, put differently, "tolerate" a small deviation). This leads to a natural generalization of the testing task that calls for distinguishing between objects that are ϵ'-close to the property and objects that are ϵ-far from the property, for parameters $\epsilon' < \epsilon$. Indeed, standard property testing refers to the case of $\epsilon' = 0$, and tolerant testing may be viewed as "tolerating" a small deviation of the object from having the property, where typically ϵ' is a function of ϵ (and sometimes also of n).

[*]Department of Computer Science, Weizmann Institute of Science, Rehovot, Israel.

Definition 1 (tolerant testers): *Let $\Pi = \cup_{n\in\mathbb{N}}\Pi_n$ such that Π_n contains functions defined over $[n]$ and $\epsilon' : \mathbb{N} \times (0,1] \to (0,1]$. An ϵ'-*tolerant tester for Π *is a probabilistic oracle machine, denoted T, that satisfies the following two conditions.*

1. T accepts inputs that are ϵ'-close to Π: *For every $n \in \mathbb{N}$ and $\epsilon > 0$, and for every $f : [n] \to R_n$ such that $\delta_\Pi(f) \le \epsilon'(n, \epsilon)$, it holds that $\mathbf{Pr}[T^f(n, \epsilon) = 1] \ge 2/3$.*
2. T rejects inputs that are ϵ-far from Π: *For every $n \in \mathbb{N}$ and $\epsilon > 0$, and for every $f : [n] \to R_n$ such that $\delta_\Pi(f) > \epsilon$, it holds that $\mathbf{Pr}[T^f(n, \epsilon) = 0] \ge 2/3$.*

(Recall that $\delta_\Pi(f)$ denotes the distance of f from Π_n.)

We avoided defining a one-sided error probability version, because it is quite useless (see Exercise 1). Note that standard testers (as in Definition `[intro:tester:def]`) may be viewed as 0-tolerant testers. On the other hand, tolerant testing is related to distance approximation, where no *proximity parameter* is given and the tester is required to output an approximation (up to a given approximation parameter) of the distance of the object to the property.

Definition 2 (distance approximation for a property): *Let $\Pi = \cup_{n\in\mathbb{N}}\Pi_n$ and δ_Π be as in Definition 1. A* distance approximator for Π *is a probabilistic oracle machine, denoted M, that satisfies one of the following two conditions.*

Additive version**:** *For every $n \in \mathbb{N}$ and $f : [n] \to R_n$ and for every $\eta > 0$, it holds that*

$$\mathbf{Pr}\left[\left|M^f(n, \eta) - \delta_\Pi(f)\right| \le \eta\right] \ge 2/3.$$

Multiplicative version[1]**:** *For every $\eta, \eta' > 0$, every $n \in \mathbb{N}$ and $f : [n] \to R_n$ that is η'-far from Π_n, it holds that*

$$\mathbf{Pr}\left[\left|M^f(n, \eta, \eta') - \delta_\Pi(f)\right| \le \eta \cdot \delta_\Pi(f)\right] \ge 2/3.$$

Typically, the term "tolerant testing" is used when the parameter ϵ' is a fixed function of ϵ (e.g., $\epsilon' = \epsilon/2$ or $\epsilon' = \epsilon^2$), and "distance approximation" is used when one seeks an approximation scheme that is governed by an approximation parameter (which corresponds to $\epsilon - \epsilon'$ when the sought approximation is additive and to ϵ/ϵ' when it is multiplicative).

Note that *any property tester that makes queries that are each uniformly distributed in $[n]$ yields a mildly tolerant tester*. Specifically, if the tester has query complexity $q(n, \epsilon)$, then it accepts every function that is $(1/10q(n, \epsilon))$-close to the property with probability at least $2/3 - 0.1$, and using error reduction we can regain the original error bound of $2/3$ (see Exercise 2). Providing higher (i.e., higher than $1/q(n, \epsilon)$) levels of tolerance is typically harder than providing standard testers. Furthermore, in some cases the tolerant testers must have significantly higher complexity: In fact, a

[1]When defining multiplicative approximation, one should be careful in the case that the optimized value may be zero (as is the case here), since asking for a multiplicative approximation in such cases will force the approximator to distinguish the value zero from a value that is extremely close to zero. This problem is resolved by confining the approximation task to cases that the value is above some minimal threshold (which is denoted here by η'). Indeed, the problem does not arise in cases where the optimized value is always inside an interval that is sufficiently far from zero (e.g., the interval $[1, 2]$). A good practice in other cases is to combine the use of both approximators; that is, first invoke an additive approximator with deviation parameter set to $0.1\eta'$, and invoke a multiplicative approximator with parameters (η, η') only if the first estimate is at least $1.1\eta'$.

very dramatic gap may exist, in general, even for properties of Boolean functions [11]. Specifically, *there exists a property of functions $f : [n] \to \{0,1\}$ that is testable in a number of queries that only depends on the proximity parameter, but tolerantly testing it requires $n^{\Omega(1)}$ queries.* Nevertheless, in some cases, good tolerant testers can be obtained. A few examples follow.

In the context of testing graph properties. We first consider testing graph properties in the dense graph model (of Chapter `[graph-dense:chap]`). Recall that ϵ-testing whether a graph is bipartite can be done by inspecting the subgraph induced by a random set of $\widetilde{O}(1/\epsilon)$ vertices. In contrast, an ϵ'-tolerant ϵ-tester of this property is known only when inspecting a polynomially larger induced subgraph. Specifically, such tolerant testing can be performed by invoking a tester for a generalized partition problem (see Exercise 3). A much more general result follows.

Theorem 3 (tolerant testing in the dense graph model): *Let Π be a graph property that can be tested within query complexity that only depends on the proximity parameter. Then, for every constants $\epsilon > \epsilon' > 0$, the property Π has an ϵ'-tolerant ϵ-tester of query complexity that only depends on ϵ and ϵ'.*

We note that the query complexity of the tolerant tester provided by the known proof of Theorem 3 is significant higher than that of the corresponding standard tester. An interesting open problem is providing a functional relationship between the complexity of testing and the complexity of tolerant testing, even just for a natural subclass of graph properties. Specifically, consider the following version of this question.

Open Problem 4 (tolerant testing in the dense graph model): *Let $\mathcal{C}$ be a natural class of graph properties. Present a function $F : \mathbb{N} \to \mathbb{N}$ such that, for every $\Pi \in \mathcal{C}$ and every constant $\epsilon > 0$, it holds that if Π is ϵ-testable in query complexity $q(n,\epsilon)$, then Π has an 0.5ϵ-tolerant ϵ-tester of query complexity $F(q(n,\epsilon))$. Alternatively, show that no such function F may exist for $\mathcal{C}$.*

The same question can be posed with respect to testing graph properties in the bounded-degree graph model. We mention that some of the testers presented in Section `[bdg:local-search:sec]` are tolerant or can be easily modified to be tolerant (see Exercise 4).

In the context of testing distributions. Tolerant testing and distance approximation have been studied extensively in the context of testing distributions (see Chapter `[dist:chap]`). Recall that, in this context, the tester gets samples from the tested distribution (rather than oracle access to some function). Actually, Corollaries `[dist:vv-label-ub:cor]` and `[dist:vv-label-lb:cor]` are typically stated in terms of tolerant testing. We do so next, while referring to the notion of label-invariant properties of distributions (defined in Section `[dist:label-invariant]`).

Corollary 5 (tolerantly testing label-invariant properties of single distributions): *Let $\mathcal{D}$ be a label-invariant property of distributions over $[n]$. Then, for every $\epsilon' : (0,1] \to [0,1]$, the property $\mathcal{D}$ has an ϵ'-tolerant tester of sample complexity $s(n,\epsilon) = O((\epsilon - \epsilon'(\epsilon))^{-2} \cdot n/\log n)$.*

Corollary 6 (optimality of Corollary 5): *For all sufficiently small constants $\epsilon > \epsilon' > 0$, any ϵ'-tolerant ϵ-tester for each of the following properties of distributions over $[n]$ requires $\Omega(n/\log n)$ samples.*

1. *The uniform distribution over* $[n]$.
2. *The set of distributions that have support size* $n/2$ *and do not have any element in their support that has probability less than* $1/n$.
3. *The set of distributions that are* m*-grained, for any* $m \in [\Omega(n), O(n)]$*, where a distribution is* m-grained *if each element appears in it with probability that is an integer multiple of* $1/m$.

Recall that the lower bound does not necessarily hold for standard testing of the same properties (i.e., the case of $\epsilon' = 0$): Part 1 provides a striking example, since (as shown in Chapter `[dist:chap]`) the uniform distribution over $[n]$ is ϵ-testable by $O(\epsilon^{-2} \cdot \sqrt{n})$ samples.

2 Additional promises on the input

As stated at the very beginning of this book, property testing refers to *promise problems* of a specific type. These problems consist of distinguishing inputs that have a predetermined property from inputs that are far from that property, where inputs that are neither in the property nor far from it are disregarded. (Tolerant testing follows this theme in distinguishing between inputs that are ϵ'-close to the property and inputs that are ϵ-far from it, where $\epsilon' < \epsilon$.)

We note that, as in all of computer science, some additional promises are typically made about the format in which the input is presented. For example, when the tester is given the size parameter n, it is guaranteed that the oracle (which represents the main input) is a function over $[n]$. Most conspicuously, in the study of testing graph properties, we have made such explicit assumption. Specifically, in the study of the dense graph model, we assumed that the adjacency predicate $g : [k] \times [k] \to \{0,1\}$ is symmetric (i.e., $g(u,v) = g(v,u)$ for every $u, v \in [k]$). Likewise, when studying of the bounded-degree graph model, we assumed that the incidence function $g : [k] \times [d] \to \{0,1,...,k\}$ is consistent in the sense that if $g(u,i) = v$ for some $u, v \in [k]$ and $i \in [d]$, then there exists $j \in [d]$ such that $g(v,j) = u$. These, however, are syntactic assumptions, which are easy to dispose of (or waive) by first testing whether the input oracle satisfies the corresponding syntactic condition.[2]

More essential promises refer to real properties of the objects, rather than to syntactic properties of their representation that only mandate that the representation is legal. Such promises are sometimes introduced for scholarly reasons, but in many cases they are justified by natural settings.[3] The point is that in many cases more efficient testers may be obtained under the promise. A few examples, all in the domain of graph properties, follow.

- Assuming a *degree bound in the dense graph model*: For any constant $\eta > 0$, the promise is that each vertex in the input k-vertex graph has degree at most $\eta \cdot k$. Under this promise, ϵ-testing `Bipartiteness` has query complexity $\mathrm{poly}(\lceil \eta/\epsilon \rceil) \cdot \widetilde{O}(\epsilon^{-3/2})$, whereas for general graphs only a bound of $\widetilde{O}(\epsilon^{-2})$ is known (see [23] and [2], respectively).

[2]In the dense graph model, one may just selects $O(1/\epsilon)$ pairs $(u,v) \in [k] \times [k]$ at random and checks whether $g(u,v) = g(v,u)$ holds. In addition, each subsequent query (u,v) is answered by the conjunction of $g(u,v)$ and $g(v,u)$, which means that we effectively replace (or correct) g by the $\epsilon/2$-close g' such that $g'(u,v) = g(u,v) \wedge g(v,u)$. In the bounded-degree graph model, for each random choice $(u,i) \in [k] \times [d]$, if $g(u,i) = v \in [k]$ then we may need to query $g(v,j)$ for all $j \in [d]$ (and apply a similar correction); see Exercise 5.

[3]A clear case where a promise is introduced for scholarly reasons appears in [19, Sec. 5.4], where a natural conjecture regarding complexity gaps between adaptive and nonadaptive testers is established in the context of promise problems. We mention that promise problems are sometimes introduced for scholarly reasons, but actually do have good conceptual justifications.

- Assuming *minor-freeness in the bounded-degree graph model*: For any constant graph H, the promise is that the input graph is H-minor free (and of bounded degree). Under this promise, ϵ-testing any graph property has query complexity that only depends on ϵ (see [29]). The result extends to any family of minor-closed graphs (and actually even to hyperfinite graphs).[4]

  In contrast, recall that testing `3-Colorability` of general k-vertex graphs, in this model, requires $\Omega(k)$ queries.

- Assuming *planarity in the general graph model*: The promise is that the input graph is planar (or, more generally, belongs to a family of minor-closed graphs). Under this promise, ϵ-testing `Bipartiteness` has query complexity that only depends on ϵ (see [7]).

  In contrast, recall that testing `Bipartiteness` of general k-vertex graphs, in this model, requires $\Omega(\sqrt{k})$ queries (even when the input graph is promised to be of bounded-degree).

3 Sample based testers

Throughout most of this book (i.e., with the exception of Chapter `[dist:chap]`), we studied testers that may freely *query the functions* that they test. In contrast, here we consider testers that only obtain "labeled samples"; that is, when testing a function $f : [n] \to R_n$, the tester is given a sequence of f-labeled samples, $((i_1, f(i_1)), ..., (i_s, f(i_s)))$, where $i_1, ..., i_s$ are drawn *independently and uniformly* in $[n]$. Such a tester is called sample-based, and it was already introduced (as a tool) in Section `[implicit:main-sec]`. We reproduce its definition (i.e., Definition `[implicit:tester:def]`) next.

Definition 7 (sample-based testers): *Let $\Pi = \cup_{n\in\mathbb{N}}\Pi_n$ such that Π_n contains functions defined over $[n]$, and $s : \mathbb{N}\times(0,1] \to \mathbb{N}$. A* sample-based tester of (sample) complexity s for Π *is a probabilistic machine, denoted T, that satisfies the following two conditions.*

1. T accepts inputs in Π: *For every $n \in \mathbb{N}$ and $\epsilon > 0$, and for every $f \in \Pi_n$, it holds that $\mathbf{Pr}[T(n, \epsilon; ((i_1, f(i_1))..., (i_s, f(i_s))) = 1] \geq 2/3$, where $s = s(n, \epsilon)$ and $i_1, ..., i_s$ are drawn independently and uniformly in $[n]$.*

2. T rejects inputs that are ϵ-far from Π: *For every $n \in \mathbb{N}$ and $\epsilon > 0$, and for every f with domain $[n]$ such that $\delta_\Pi(f) > \epsilon$, it holds that $\mathbf{Pr}[T(n, \epsilon; ((i_1, f(i_1))..., (i_s, f(i_s))) = 0] \geq 2/3$, where $i_1, ..., i_s$ are as in Item 1.*

If the first condition holds with probability 1, then we say that T has one-sided error.

The sequence $((i_1, f(i_1))..., (i_s, f(i_s)))$ is called an f-labeled sample of s points (in the domain of f). We mention that any class $\Pi = \cup_{n\in\mathbb{N}}\Pi_n$ can be tested by using a sample of size $O(\epsilon^{-1} \log |\Pi_n|)$, via reducing (sample-based) testing to (sample-based) proper learning (see Section `[intro:learning]`).

As we have seen in prior chapters, the ability to make queries is very powerful: even when the queries are selected non-adaptively, they may be selected to depend on one another. In contrast, a sample-based tester is quite restricted (i.e., it cannot obtain related samples). Nevertheless, a sample-based tester is desirable in many applications where obtaining samples is far more feasible

[4]See terminology in Sections `[bdg:partition-oracle:sec]` and `[bd-taste-poly]`.

than obtaining answers to queries of one's choice. The question, of course, is what can such sample-based testers achieve. One general answer is that sample-based testers of sub-linear complexity exist for any property that has a constant-query proximity-oblivious tester in which each query is uniformly distributed.[5]

Theorem 8 (from POTs to sample-based testers): *Let $\Pi = \cup_{n\in\mathbb{N}}\Pi_n$ such that Π_n contains functions defined over $[n]$ with range R_n, and suppose that Π has a q-query POT with threshold probability τ and detection probability ϱ.*

1. *If each query made by this POT is uniformly distributed in the function's domain, then Π has a sample-based tester of sample complexity $s(n,\epsilon) = \max(O(n^{(q-1)/q}/\varrho(\epsilon)^{b+(3/q)}), O(\varrho(\epsilon)^{-(3+b)}))$, where $b = 2$ in the general case and $b = 1$ if POT has one-sided error* (i.e., $\tau = 1$). *Furthermore, if the POT has one-sided error, then so does the sample-based tester.*

2. *If the POT is non-adaptive and has one-sided error* (i.e., $\tau = 1$), *then Π has a sample-based ϵ-tester that uses $O(\log|R_n|)\cdot n^\gamma$ samples, where γ depends on q and $\varrho(\epsilon)$, and ditto for the hidden constant in the O-notation. In the general case* (i.e., adaptive POT with $\tau < 1$), *the exponent γ may also depend on $|R_n|$.*

Both parts of Theorem 8 are actually more general, but some flavor of the conditions made in them is inherent (since there exists a property Π that has no sample-based tester of sub-linear complexity although it does have a one-sided non-adaptive two-query POT [20, Prop. 3.3]).[6] Applying Part 1 of Theorem 8 to any of the known 2-query POTs for monotonicity, which were presented in Chapter `[mono:chap]`, we obtain a sample-based tester of sample complexity $\widetilde{O}(\sqrt{n})\cdot\mathrm{poly}(1/\epsilon)$. It is actually easy to obtain such testers directly: Considering, for example, the "edge tester" presented in Section `[mono:edge-test:sec]`, we can emulate a selection of a random edge in the ℓ-dimensional hypercube by taking a sample of $O(\sqrt{2^\ell/\ell})$ vertices.

4 Other distance measures

Recall that distance between functions (having the same domain $[n]$) was defined in Section `[intro:basics]` as the fraction of the domain on which the functions disagree, which can be interpreted as the probability that the functions disagree on a *uniformly distributed point in their domain.* A more general definition may refer to the disagreement with respect to an arbitrary distribution $\mathcal{D}_n$ over $[n]$; that is, we may have

$$\delta_{\mathcal{D}_n}(f,g) \stackrel{\rm def}{=} \mathbf{Pr}_{i\sim\mathcal{D}_n}[f(i)\neq g(i)], \qquad (1)$$

where $i \sim \mathcal{D}_n$ means that i is distributed according to $\mathcal{D}_n$. In such a case, for a "distribution ensemble" $\mathcal{D} = \{\mathcal{D}_n\}$, we let $\delta_{\Pi,\mathcal{D}}(f) \stackrel{\rm def}{=} \min_{g\in\Pi_n}\{\delta_{\mathcal{D}_n}(f,g)\}$. This leads to a definition of *testing with respect to an arbitrary distribution ensemble $\mathcal{D}$*, viewing Definition `[intro:tester:def]` as a special case in which $\mathcal{D}_n$ is the uniform distribution over $[n]$.

One step farther is to consider *distribution-free testers.* Such a tester should satisfy the foregoing requirement for *all* possible distributions $\mathcal{D}$, and it is typically equipped with a special device that

[5]We stress that different queries are allowed to depend on one another; we only postulate that each query, by itself, is uniformly distributed in the function's domain.

[6]The range of the corresponding functions is exponential in n.

provides it with samples drawn according to the distribution in question (i.e., the distribution $\mathcal{D}_n$ used in the definition of distance). That is, a distribution-free tester for Π is an oracle machine that can query the function $f:[n]\to R_n$ as well as obtain samples drawn from *any* distribution $\mathcal{D}_n$, and its performance should refer to $\delta_{\Pi,\mathcal{D}}(f)$ (i.e., the distance of f from Π_n as measured according to the distribution $\mathcal{D}_n$).[7]

Definition 9 (distribution-free testing): *A* distribution-free tester for Π *is a probabilistic oracle machine, denoted T, such that for every $\epsilon > 0$ and $n \in \mathbb{N}$ and every distribution $\mathcal{D}_n$ over $[n]$, the following two conditions hold:*

1. *For every $f \in \Pi_n$, it holds that* $\mathbf{Pr}[T^{f,\mathcal{D}_n}(n,\epsilon)=1] \geq 2/3$.
2. *For every $f:[n]\to R_n$ such that $\delta_{\Pi,\mathcal{D}}(f) > \epsilon$, it holds that* $\mathbf{Pr}[T^{f,\mathcal{D}_n}(n,\epsilon)=0] \geq 2/3$.

In both items, $T^{f,\mathcal{D}_n}(n,\epsilon)$ denotes the output of T when given oracle access to $f:[n]\to R_n$ as well as samples that are drawn independently from $\mathcal{D}_n$ (and explicit inputs n and ϵ).

Note that, unlike in Section 3, the tester has oracle access to the function f. It is provided with samples drawn according to $\mathcal{D}_n$, but there is no need to provide it with the corresponding f-labels (since it can obtain these by itself by querying f). In such a case, one may consider both the tester's query complexity and its sample complexity.[8]

In order to justify the foregoing definition, let us spell out the type of settings that it is supposed to capture. In such settings the function f represents some measurement (or numerical parameter) of possible situations, which are represented by f's domain. In other words, f represents an assignment of values to possible situations. The distribution $\mathcal{D}_n$ represents an (a priori) unknown distribution on the possible situations, and testing is defined with respect to this distribution because this is the distribution in which situations occur. But in such a case, it stands to reason that samples from this distribution are available to the tester.

Taking a more technical attitude, one may say that if the tester is being evaluated with respect to an arbitrary distribution, then it is fair to provide it with some information regarding this distribution. Failure to do so, will deem the testing task infeasible; that is, for any non-degenerate property, a distribution-free tester that obtains no samples of the distribution will have to query all locations in the function's domain (since the distribution may be totally concentrated on any of them).

Few efficient distribution-free testers are known (see, e.g., [25, 8]); in general, it seems that distribution-free testing is quite hard (see, e.g., [**?**]). We wonder whether this deem picture may change if one restricts the class of distributions; for example, by considering only distributions that have a minimal amount of min-entropy (e.g., consider distributions over $[n]$ in which each element appears with probability at most $n^{-0.5}$). Indeed, a model of lower-bounded randomness seems reasonable here like in other cases in which one seeks a probabilistic model of a natural reality.

[7]We stress that $\mathcal{D}_n$ is *a priori* unknown to the tester, which may gain partial information about it from the samples.

[8]In particular, one may also consider the case that the tester does not query the function on each sample obtained from $\mathcal{D}_n$; see [4].

Testing distributions. A seemingly related, but actually different, notion is that of testing properties of distributions. Here we do not test properties of functions (with respect to a distance defined according to some distribution), but rather test properties of distributions, when given samples drawn independently from a target distribution. For details, see Chapter `[dist:chap]`.

Beyond (weighted) Hamming distance. Almost all research in property testing focuses on the distance measure defined in Section `[intro:basics]`, which corresponds to the relative Hamming distance between sequences. We already saw a deviation from this archetypical case in Eq. (1), but the measure defined in Eq. (1) is merely a weighted Hamming distance. That is, the distance between functions is defined in terms of the set of domain elements on which they disagree. In the archetypical definition (presented in Section `[intro:basics]`) one just considers the density of this set in the domain, and in Eq. (1) one considers is weighted density. But measures of the set of points of disagreements do not exhaust the natural measures of distance between functions (or objects).

Different distance measures, which are natural in some settings, include the *edit distance* and the $\mathcal{L}_1$*-distance*. The edit distance is most natural when the object is viewed as a sequence. In such a case, *deleting* or *inserting* a symbol has unit cost, regardless of the position in which the operation takes place (whereas the Hamming distance may charge such an operation according to the length of the corresponding suffix, which is being shifted by the insertion).[9] As in the case of Hamming distance, a relative notion of edit distance is obtained by dividing the absolute distance (or edit cost) by the total length of the sequences (which may be of different length). Variants of "edit distance" refer to the set of operations that are counted at unit cost (e.g., in addition to symbol deletion and insertion, one sometimes includes also the cut-and-paste operation).

We stress that the Hamming distance (and also the edit distance) counts each disagreement alike (i.e., for every two symbols σ and τ, it only distinguishes $\sigma = \tau$ from $\sigma \neq \tau$). But when the function's values are non-binary and its range is equipped with a natural notion of distance, it makes sense to "charge" disagreements according to their magnitude. This is particularly appealing for function that range over the reals. Specifically, for $f, g : [n] \to [0, 1]$, it makes sense to consider the norm of the function $f - g$, and the $\mathcal{L}_1$-norm is indeed most appealing. In contrast, Hamming distance corresponds to the $\mathcal{L}_0$-norm. The reader may easily conceive of situations in which a tiny $\mathcal{L}_1$-distance is insignificant (e.g., it may be due to "measurement error"), and in such cases considering the Hamming distance is inappropriate.

5 Local computation algorithms

Recall that property testing is a *decision problem* (of the promise type); that is, the desired output is a bit, which indicates whether the (huge) object has the tested property or is far from having it. Distance approximation (see Definition 2) generalizes property testing in the sense that the desired output is supposed to approximate a quantity in $[0, 1]$. More generally, we can consider super-fast algorithms that approximate other parameters of the (huge) object such as the average degree of a graph, its girth, its diameter, its expansion coefficient, etc (where we confined ourselves to the domain of graphs). In all cases, we refer to a relatively short output that approximates a numerical parameter of the huge object.

[9]For example, the string $(01)^t$ is at Hamming distance $2t$ from the string $(10)^t$, but the edit distance between them is only two, since $(10)^t = 1(01)^{t-1}0$.

A more general task refers to *solving search problems* concerning these huge objects, where the desired output is also a huge object. Clearly, we cannot expect a super-fast algorithm to output an object that is larger than its running time. Hence, the notion of outputting an object is to be modified. The algorithm will not explicitly output the desired object, but will rather provide oracle access to it; that is, it will answer queries regarding the output by making queries to its own input oracle.

5.1 Definitions

The following definition generalizes the definition of a local (deterministic) reduction, presented in the context of lower bound techniques (i.e., Definition `[local-reduction:def]`). Actually, we extend the definition by allowing the algorithm to be adaptive and make a non-constant number of queries. Furthermore, we consider any search problem $\mathtt{R}_{\overline{\epsilon}}$ that is parameterized by a tuple[10] of approximation parameters $\overline{\epsilon} > 0$; that is, on input f, the task is to find a (solution) f' such that $(f, f') \in \mathtt{R}_{\overline{\epsilon}}$. We denote the set of solutions (for f) by $\mathtt{R}_{\overline{\epsilon}}(f) = \{f' : (f, f') \in \mathtt{R}_{\overline{\epsilon}}\}$.

Definition 10 (local computation algorithms, deterministic version): *For every $\overline{\epsilon} > 0$, let $\mathtt{R}_{\overline{\epsilon}}$ be a binary relation containing pairs of finite functions, and let $q : \mathbb{N} \times (0,1]^* \to \mathbb{N}$. A deterministic oracle machine M is said to q-*locally solve *the search problem $\{\mathtt{R}_{\overline{\epsilon}}\}_{\overline{\epsilon}>0}$ if for every $n \in \mathbb{N}$ there exists $n' \in \mathbb{N}$ such that for every $\overline{\epsilon} > 0$ and for every function f over $[n]$ there exists a function f' over $[n']$ such that the following conditions hold.*

1. Locality (emulating oracle access to f'): *On input $n, \overline{\epsilon}$ and $i \in [n']$, and oracle access to f, the machine M outputs $f'(i)$ after making at most $q(n, \overline{\epsilon})$ queries.*

2. Validity (i.e., f' is a valid solution to f): *If $\mathtt{R}_{\overline{\epsilon}}(f) \neq \emptyset$, then the pair (f, f') is in $\mathtt{R}_{\overline{\epsilon}}$.*

In addition, we require that n' be computable based on n.

Note that for functions f such that $\mathtt{R}_{\overline{\epsilon}}(f) = \emptyset$, it is only required that the machine answers consistently with some function f'. Indeed, Definition `[local-reduction:def]` (a local (ϵ, ϵ')-reduction of Π to Π') is a special case of Definition 10: It corresponds to the case that the local computation algorithm defines a mapping of functions over $[n]$ to functions o0ver $[n']$ such that any $f \in \Pi_n$ is mapped to $f' \in \Pi'_{n'}$ and any f that is ϵ-far from Π_n is mapped to f' that is ϵ'-far from $\Pi'_{n'}$; that is, $(f, f') \in \mathtt{R}_{\epsilon,\epsilon'}$ if either $f \in \Pi_n$ and $f' \in \Pi'_{n'}$ or f is ϵ-far from Π_n and f' is ϵ'-far from $\Pi'_{n'}$.

Definition 10 is restricted to deterministic computation, whereas we may gain more applications by allowing also randomized machines (let alone that our entire mindset in this book views randomized algorithms as the norm). It is crucial, however, that all invocations of the local computation algorithm refer to the same function f'. Towards this end, we provide the machine with global randomness, denoted ω, which should be used in all invocations. In addition, the machine may use auxiliary ("local") randomness, which is implicit in our notation. We distinguish the global randomness from the local randomness, because we wish to use different error probability parameters with respect to each of them (see further discussion following Definition 11).

[10] A tuple of approximation parameters (rather than a single parameter) is used for greater expressibility. In fact, this is essential in some cases (e.g., for capturing the notion of a local (ϵ, ϵ')-reduction presented in Definition `[local-reduction:def]`).

The following definition generalizes the definition of a randomized local reduction (i.e., Definition `[rnd-local-reduction:def]`, although the presentation here is different).[11]

Definition 11 (local computation algorithms, randomized version): *Let q and $\{\mathtt{R}_{\overline{\epsilon}}\}_{\overline{\epsilon}>0}$ be as in Definition 10, and let $r : \mathbb{N} \times (0,1]^* \to \mathbb{N}$ and $\eta : \mathbb{N} \to [0,1]$. A randomized oracle machine M is said to q-*locally solve* the search problem $\{\mathtt{R}_{\overline{\epsilon}}\}_{\overline{\epsilon}>0}$* with error probability *η if for every $n \in \mathbb{N}$ there exists $n' \in \mathbb{N}$ such that for every $\overline{\epsilon} > 0$ and for every function f over $[n]$ the following conditions hold.*

1. Locality (emulating oracle access to f'_ω): *For every $\omega \in \{0,1\}^{r(n,\overline{\epsilon})}$ there exists a function f'_ω over $[n']$ such that, on input $n, \overline{\epsilon}, \omega$ and $i \in [n']$, and oracle access to f, with probability at least $2/3$, the machine M outputs $f'_\omega(i)$ after making at most $q(n,\overline{\epsilon})$ queries. In other words, M always makes at most $q(n,\overline{\epsilon})$ queries and $\mathbf{Pr}[M^f(n,\overline{\epsilon},\omega;i) = f'_\omega(i)] \geq 2/3$, where the probability is over the internal coin tosses of M* (whereas ω is fixed).

2. Validity (i.e., f'_ω is a valid solution to f): *If $\mathtt{R}_{\overline{\epsilon}}(f) \neq \emptyset$, then, with probability at least $1 - \eta(n)$ over the choice of ω, the pair (f, f'_ω) is in $\mathtt{R}_{\overline{\epsilon}}$, where f'_ω is as in the locality condition. That is,*
$$\mathbf{Pr}_{\omega \in \{0,1\}^{r(n,\overline{\epsilon})}}[(f, f'_\omega) \in \mathtt{R}_{\overline{\epsilon}}] \geq 1 - \eta(n).$$

In addition, we require that n' be computable based on n. The string ω is called the global randomness*, and the internal coin tosses of M are called the* local randomness.

Note that Definition 11 refers to two types of error probability. The first type is the probability that the choice of the global randomness ω yields an invalid solution f'_ω. Note that this error probability can not be generically reduced by repetitions, since different ω's may yield different functions f'_ω. This is a general phenomenon that refers to any randomized algorithms for solving search problems, and it is the reason that we introduced an explicit parameter, η, for bounding this error probability. In contrast, the second type of error probability, which refers to the (implicit) local randomness of M, can be reduced by repetitions (and therefore we felt no need to introduce a parameter that governs it but rather set this error probability to 1/3).[12]

A richer formalism. In Definition 11, the oracle machine M is provided with the global randomness ω. A richer formalism allows to provide it (or rather its query-serving module) with arbitrary information that is computed based on ω and the oracle f during a *preprocessing stage*. In such a case, the query complexity of the preprocessing stage (or rather module) is stated separately. Needless to say, an oracle machine as in Definition 11 can be obtained by invoking the preprocessing module each time a query (to f'_ω) needs to be served, but this is wasteful (especially since typically the preprocessing module has higher complexity than the query-serving module). This motivates the following definition, where the output of the preprocessing module, denoted z, may (but need not) explicitly include the global randomness ω.

[11]Specifically, as in Definition 10, we explicitly refer to the (possibly adaptive) computing machine. In addition, here we explicitly refer to the global randomness and to the error probability.

[12]In particular, incorporating the local randomness in the global randomness would have set the error probability for each computation of f'_ω to η, whereas the current formalism allows greater flexibility (which is particularly important in the case that the expected number of local computations is larger than $1/\eta(n)$).

Definition 12 (local computation algorithms, two-stage version): *Let q, r, η and $\{\mathsf{R}_{\overline{\epsilon}}\}_{\overline{\epsilon}>0}$ be as in Definition 11. A pair of oracle machines, (M_1, M_2), is said to (q_1, q_2)-*locally solve *the search problem $\{\mathsf{R}_{\overline{\epsilon}}\}_{\overline{\epsilon}>0}$* with error probability *η if for every $n \in \mathbb{N}$ there exists $n' \in \mathbb{N}$ such that for every $\overline{\epsilon} > 0$ and for every function f over $[n]$ the following conditions hold.*

1. Preprocessing: *The oracle machine M_1 is deterministic, and for every $\omega \in \{0,1\}^{r(n,\overline{\epsilon})}$, on input $(n, \overline{\epsilon})$ and ω, it makes at most $q_1(n, \overline{\epsilon})$ queries.*

2. Locality (emulating oracle access to f'_z): *For every $z \in \{0,1\}^*$ there exists a function f'_z over $[n']$ such that, on input $n, \overline{\epsilon}, \omega$ and $i \in [n']$, and oracle access to f, with probability at least $2/3$, the machine M_2 outputs $f'_z(i)$ after making at most $q_2(n, \overline{\epsilon})$ queries. In other words, M_2 always makes at most $q_2(n, \overline{\epsilon})$ queries and $\mathbf{Pr}[M_2^f(n, \overline{\epsilon}, z; i) = f'_z(i)] \geq 2/3$, where the probability is over the internal coin tosses of M_2.*

3. Validity (i.e., f'_z is a valid solution to f): *If $\mathsf{R}_{\overline{\epsilon}}(f) \neq \emptyset$, then, with probability at least $1 - \eta(n)$ over the choice of $\omega \in \{0,1\}^{r(n,\overline{\epsilon})}$, the pair (f, f'_z) is in $\mathsf{R}_{\overline{\epsilon}}$, where $z \leftarrow M_1^f(n, \overline{\epsilon}, \omega)$ and f'_z is as in the locality condition.*

Again, we require that n' be computable based on n. The string ω is called the global randomness, *and the internal coin tosses of M_2 are called the* local randomness.

As noted above, every (q_1, q_2)-local solver yields a $(q_1 + q_2)$-local solver, but it is beneficial to use the former formulation when $q_1 \gg q_2$. (In conrast, the fact that any q-local solver yields a $(0, q)$-local solver, where the fictitious preprocessing just maps ω to itself, is quite useless.)

5.2 Finding huge structures

As mentioned in Section [dense:gpp:sec], local computations of huge structures are implicit in the testers for the various graph partition problems (in the dense graph model). For concreteness, we consider the case of Bipartiteness. In this case, the local computation algorithm finds a 2-partition of the vertices of a given bipartite graph such that there are relatively few edges with both endpoints in the same part.

Proposition 13 (finding approximate 2-colorings in bipartite graphs, in the dense graph model): *There exists a $(\text{poly}(1/\epsilon) \cdot \log(1/\eta), \widetilde{O}(1/\epsilon))$-local algorithm that, on input k, ϵ, η and oracle access to an adjacency predicate of a k-vertex Bipartite graph, $G = ([k], E)$, finds, with probability at least $1 - \eta$, a 2-partition $\chi : [k] \to \{1, 2\}$ such that $|\{\{u,v\} \in E : \chi(u) = \chi(v)\}| < \epsilon k^2/2$.*

In other words, the algorithm solve the search problem R_ϵ with error probability 2^{-m}, where R_ϵ contains all pairs (G, χ) such that $G = ([k], E)$ is a bipartite graph and $|\{\{u,v\} \in E : \chi(u) = \chi(v)\}| < \epsilon k^2/2$, where $k^2/2$ is used as an approximation to the total number of pairs over $[k]$ (see discussion in Section [dense:model.sec]).

Proof Sketch: Our starting point is the analysis of Algorithm [dense:bip.alg], which views the sample of $\widetilde{O}(1/\epsilon^2)$ vertices selected by the algorithm as consisting of two parts, U and S, such that $|U| = \widetilde{O}(1/\epsilon)$ and $|S| = \widetilde{O}(1/\epsilon^2)$. The analysis establishes that for 3/4 of the possible U's, there exists a 2-partition of U, denoted (U_1, U_2), such that *letting $\chi(v) = 1$ if and only if v has no neighbor in U_1* yields a 2-partition $\chi : [k] \to \{1, 2\}$ as desired (i.e., $|\{\{u,v\} \in E : \chi(u) = \chi(v)\}| <$

$\epsilon k^2/2$). The set S is used to (implicitly) estimate the suitability of each of the 2-partitions of U. Specifically, if χ has at least $\epsilon k^2/4$ monochromatic (w.r.t χ) edges, then, with probability at least $1 - \exp(-\Omega(\epsilon \cdot |S|)) = 1 - 2^{-2|U|}$, the subgraph induced by S will contain a monochromatic edge.[13] Hence, we can find, in a preprocessing stage, a partition of U that determines a suitable χ.

Let us spell out the local computation algorithm that emerges. The preprocessing stage of this algorithm uses the global randomness in order to select random sets of vertices U and S such that $|U| = \widetilde{O}(1/\epsilon)$ and $|S| = \widetilde{O}(1/\epsilon^2)$. It then determines a 2-partition of U, denoted (U_1, U_2), such that, with high probability (over the choice of U and S), the corresponding 2-partition χ is a good solution (where $\chi(v) = 1$ if and only if v has no neighbor in U_1). *The 2-partition of U is determined by finding a legal 2-coloring of the subgraph of G that is induced by $U \cup S$*, and using the 2-partition that it induces on U. The query-serving stage is deterministic: It answers the query v by querying the graph G on the pairs $\{v, u\}$ for all $u \in U_1$, and answers accordingly. (Hence, this algorithm uses no local randomness.)

As shown in the proof of Lemma [dense:bipartite:lem], with high probability, the set U neighbors both endpoints of almost all edges in G (i.e., all but at most $\epsilon k^2/4$ of the edges). Whenever this event occurs, a 2-partition of U that yields a bad solution $\chi : [k] \to \{1, 2\}$ will be detected as bad (by S), with overwhelmingly high probability. Hence, with high probability, the 2-partition of U selected in the preprocessing stage is good. Finally, in order to obtain error probability of η, we perform the foregoing procedure using $t = O(\log(1/\eta)$ candidate sets U and a set S that is t times larger.[14] ■

Finding other huge structures. In general, in the context of graphs, two notable types of desired output are (1) a partition of the vertices of the (input) graph that satisfies some properties, and (2) a subgraph (of the input graph) that satisfies some properties. We briefly review these types of problems next.

As mentioned in the beginning of this section, in the dense graph model, for any fixed t and any sequence of desired vertex and edge densities, a t-partition of the vertices that approximately satisfies these densities can be found by a poly$(1/\epsilon)$-local algorithm (with error probability poly(ϵ), whenever the exact t-partition exists), where ϵ is the proximity parameter. This follows from the methods used to construct testers for the corresponding graph partition problems (discussed in Section [dense:gpp:sec]).

Turning to the bounded-degree graph model, we mention that Section [bdg:partition-oracle:sec] evolves around the local construction and utilization of *partition oracles*. Recall that for given parameters $\epsilon > 0$ and $t \in \mathbb{N}$, such a partition oracle of a graph $G = ([k], E)$ is a function $P : [k] \to \cup_{i \in [t]} \binom{[k]}{i}$ such that (1) for every vertex $v \in [k]$ it holds that $v \in P(v)$; (2) the subgraph of G induced by $P(v)$ is connected (and has at most t vertices); and (3) different $P(v)$'s are disjoint and the total number of edges among them is at most ϵk.

The problem of *finding huge subgraphs* that satisfy various properties was also considered in several works. In particular, the related problems of finding a *maximal matching* and *approximate*

[13] Actually, letting $S = \{s_1, ..., s_m\}$, it suffices to check the pairs (s_{2i-1}, s_{2i}) for $i = 1, ..., m/2$.

[14] That is, in the preprocessing stage, we select t random sets $U^{(1)}, ..., U^{(t)}$, each of size $\widetilde{O}(1/\epsilon)$ and a single set S of size $t \cdot \widetilde{O}(1/\epsilon^2)$, find an i such that $U^{(i)}$ is good (per Definition [dense:bipartite-good-set:def]), determine the 2-partition of $U^{(i)}$ by considering the subgraph of G that is induced by $U^{(i)} \cup S$, and output this 2-partition (for use by the query-serving module). Recall that a set is good if it neighbors all but $\epsilon k/6$ of the vertices that have degree at least $\epsilon k/6$, and note that we can use S (or an auxiliary set of $t \cdot \widetilde{O}(1/\epsilon^2)$ random vertices) in order to test this condition.

maximum matching in bounded-degree graphs were considered in several works (see [10] and the references therein).[15] The problem of finding a *sparse spanning subgraph* was studied in [27].

5.3 Local reconstruction

A natural computational problem related to property testing is designing a "filter" that returns the input function if it has the designed property and returns an arbitrary function having the property otherwise. In other words, the filter must always return an output in Π, but if the input f is in Π then the filter must return f itself. That is, for a property of functions Π, we consider the search problem $\mathtt{R}$ such that for every f it holds that $\emptyset \neq \mathtt{R}(f) \subseteq \Pi$, and $\mathtt{R}(f) = \{f\}$ if $f \in \Pi$. A more smooth definition requires that, for some monotonically non-decreasing function $\rho : [0,1] \to [0,1]$ such that $\rho(0) = 0$, it holds that *if f is δ-close to Π, then $\mathtt{R}(f) \subseteq \Pi$ contains only functions that are $\rho(\delta)$-close to Π* (and $\mathtt{R}(f) \neq \emptyset$).

Such filters, also called *local reconstructors*, are extremely beneficial in settings that are complimentary to ones that motivate tolerant testers. Specifically, here we envision a gap between the utility of functions having a certain property and functions lacking it. Testing the function for the property eliminates the danger of functions that are far from the property. The foregoing filters guarantee that also functions that are close to the property will not cause problems; they will be transformed into functions that have the property and are close to the original functions (and so they may preserve other features of the original functions).

A concrete case where this issue arises is the one of mechanisms that preserve the "privacy" of the data when answering questions that correspond to functions of the data, provided that these functions satisfy certain properties (e.g., Lipschitz properties, cf. [26]). The point is that the security guarantee (i.e., "privacy") may be compromised even if the function deviates only a little from the property, but the utility of the function does not change by much if the function is slightly modified.

In any case, properties that consist of functions that are randomly self-reducible by few queries (see Definition `[junta:self-reduction:def]`) have trivial local reconstruction algorithms, which use no global randomness (but do use local randomness). Recall, however, that such properties must have distance (i.e., every two distinct functions in the property are far apart). Hence, the focus of the study of local reconstruction algorithms is on properties that have no distance, like monotonicity (and Lipschitz) properties.

6 Non-interactive proofs of proximity (MAPs)

The $\mathcal{P} \neq \mathcal{NP}$ conjecture means that, in general, *verifying the validity of a proof for a given assertion is easier than deciding the validity of the given assertion, when it is not accompanied by a proof.* A natural question is whether the same holds also with respect to approximate decisions; that is, can short proofs assist property testers, which in this case may be called property verifiers.

This question leads to the following model of probabilistic proof systems, which is a hybrid of MA ("randomized NP") and property testing. These proof systems, called MAPs (for Merlin-Arthur proofs of proximity), consist of a (relatively short) proof given to a verifier that has (randomized) query access to the main (tested) object. We stress that the verifier has free access to the alleged proof, and that our focus is on short proofs (i.e., proofs that are significantly shorter than the

[15]These works also study finding *d-coloring* in graphs of maximal degree d.

input).[16] Insisting on short proofs is natural in the current context, since our final goal is to obtain super-fast algorithms; recall that, in general, property testing is concerned with sub-linear complexity (i.e., sub-linear in input length).

The choice of considering a randomized proof system, rather than a deterministic one, also fits the context of property testing, since property testers are inherently randomized.[17] Hence, we view the MAP model as the natural "NP analogue" of property testing.

The following definition augments the property testing framework by replacing the tester with a *verifier* that receives, in addition to oracle access to the input (denoted f), also free access to a short proof (denoted π). The guarantee is that, for any input that has the property, there exists a proof that makes the verifier accept (with high probability), whereas for any input that is far from the property the verifier will reject any alleged proof (with high probability).

Definition 14 (non-interactive proofs of proximity (MAPs)): *Let* $\Pi = \cup_{n\in\mathbb{N}}\Pi_n$ *such that* Π_n *contains functions defined over* $[n]$. *A system of* non-interactive proofs of proximity (MAP) for Π *is a probabilistic oracle machine, called a* verifier *and denoted* V, *that satisfies the following two conditions, for every* $n \in \mathbb{N}$ *and* $\epsilon > 0$.

1. Completeness: V accepts inputs in Π when given an adequate proof. *Specifically, for every* $f \in \Pi_n$ *there exists* $\pi \in \{0,1\}^*$ *such that* $\mathbf{Pr}[V^f(n,\epsilon,\pi)=1] \geq 2/3$.

   If $\mathbf{Pr}[V^f(n,\epsilon,\pi)=1] = 1$, *then we say that* V *has* perfect completeness.

2. Soundness: V rejects inputs that are far from Π. *Specifically, for every* $f : [n] \to R_n$ *that is* ϵ*-far from* Π_n *and every* $\pi \in \{0,1\}^*$ *it holds that* $\mathbf{Pr}[V^f(n,\epsilon,\pi)=0] \geq 2/3$.

We say that V *has* proof complexity $p : \mathbb{N} \to \mathbb{N}$ *if the completeness condition holds with a proof* π *of length at most* $p(n)$. *We say that* V *has* query complexity $q : \mathbb{N} \times (0,1] \to \mathbb{N}$ *if, on input* n, ϵ, π *and oracle access to any* $f : [n] \to R_n$, *the verifier makes at most* $q(n,\epsilon)$ *queries.*[18]

In addition, one may limit the computational complexity of the verifier; for example, require that its running time is sub-linear in the length of the input (or almost linear in its proof and query complexities). We note that property testers may be viewed as MAPs of proof complexity zero, and in that case perfect completeness corresponds to one-sided error probability.

As mentioned upfront, our focus is on MAPs of sub-linear proof complexity; indeed, any property has a "trivial" MAP with linear proof complexity and query complexity $O(1/\epsilon)$ (see Exercise 6). Interestingly, even very short proofs can reduce the query complexity in a significant fashion: For example, $\Pi = \{uuvv : u, v \in \{0,1\}^*\}$ has a MAP of logarithmic proof complexity and query complexity $O(1/\epsilon)$, whereas testing Π requires $\Omega(\sqrt{n})$ queries (see Exercise 9).

General results. A few of the known results regarding MAPs include:

[16]Jumping ahead, we mention that any property can be ϵ-verified by making $O(1/\epsilon)$ queries, when given access to a proof of linear length (see Exercise 6). On the other hand, if the proof is shorter than the query complexity, then having free access (rather than query access) to it is immaterial; however, this is not necessarily the case.

[17]Indeed, deterministic MAPs are quite restricted; see Exercise 7.

[18]Indeed, as in the definition of property testers, the query complexity of the verifier depends on ϵ. A concious decision is made here not to allow the proof complexity to depend on ϵ (and not to allow the query complexity to depend on π), since this choice seems more natural.

- *MAPs versus property testers.* For every $\alpha \in (0,1)$, there exists a property Π that has a MAP of logarithmic proof complexity and query complexity $\text{poly}(1/\epsilon)$, whereas testing Π requires $\Omega(n^{\alpha})$ queries (see [17], building on [24]).
- *Proof length versus query complexity trade-off in MAPs.* There exists a property Π such that, for every $p \geq 1$, the property Π has a MAP of proof complexity p and query complexity n/p, whereas every MAP of proof complexity p for Π requires $\Omega(n^{0.999}/p)$ queries [24].
- *Two-sided versus one-sided error MAPs.* Any MAP can be transformed into a MAP with perfect completeness at the cost of increasing its proof and query complexities by at most a polylogarithmic (in n) factor [24].
- *Properties that are extremely hard for MAPs.* There exists a property Π such that every MAP of proof complexity $n/100$ for Π requires $\Omega(n)$ queries [24].

In addition, we mention that, for every $p < n$, every context-free language has a MAP of proof complexity p and query complexity $\epsilon^{-1} \cdot \widetilde{O}(n)/p$; the same holds for sets recognized by read-once branching programs of polynomial size [18].

Relation to other forms of non-interactive proof systems. Being a type of probabilistic but non-interactive proof systems, MAPs are related to but different from PCPPs (PCPs of Proximity), PCPs, and "randomized NP" proof systems (captured by the complexity class $\mathcal{MA}$). [19] These four non-interactive proof systems differ in the way they refer to the (purported) assertion and to the (alleged) proof. In MAPs and PCPPs, the verifier only gets oracle access to the assertion, its queries are accounted for (in the query complexity), but the soundness condition only refers to inputs that are far from the set of valid assertions. In contrast, in (randomized) NP-proof systems and in PCPs the verifier gets free access to the assertion (and the soundness condition refers to all invalid assertions). Turning to the alleged proof, in (randomized) NP-proofs and in MAPs, the verifier gets free access to it, whereas in PCPs and PCPPs the verifier only gets oracle access to the alleged proof. A crucial difference between MAPs and all the other three types of proof systems is that in MAPs the focus is on short proofs (whereas in NP, as well as in PCPs and PCPPs, the proof is typically longer than the assertion).[20] The taxonomy that arises is captured in Figure 1. To summarize: In MAPs, the input is presented via an oracle (like in PCPP and in property testers), whereas the proof is presented explicitly (like in NP/MA). Hence, as stated upfront, MAPs are an NP version of property testers.

Relation to interactive proofs of proximity. The MAP model can be viewed as a special case of a more general model of *interactive proofs of proximity* (IPP), which is an interactive proof (IP) version of property testing. In an IPP, the prover and verifier exchange messages, but the verifier can only access the main input via oracle queries. In this case the verifier V is an interactive strategy, which also has oracle access to the main input (denoted f). In addition, we also consider interactive strategies for the prover. For a pair of interactive strategies, (A, B), we denote by

[19]PCPs and PCPPs are discussed in Chapter `[ltc:chap]`; see, in particular, Sections `[ltp:sec]` and `[pcpp.sec]`. For a wider perspective on probabilistic proof systems, the interested reader is referred to [15, Chap. 9].

[20]**Advanced comment:** The focus on short proofs is reminicent of the study of *laconic interactive proof systems*, but that study focused on systems in which at least two messages are sent [22].

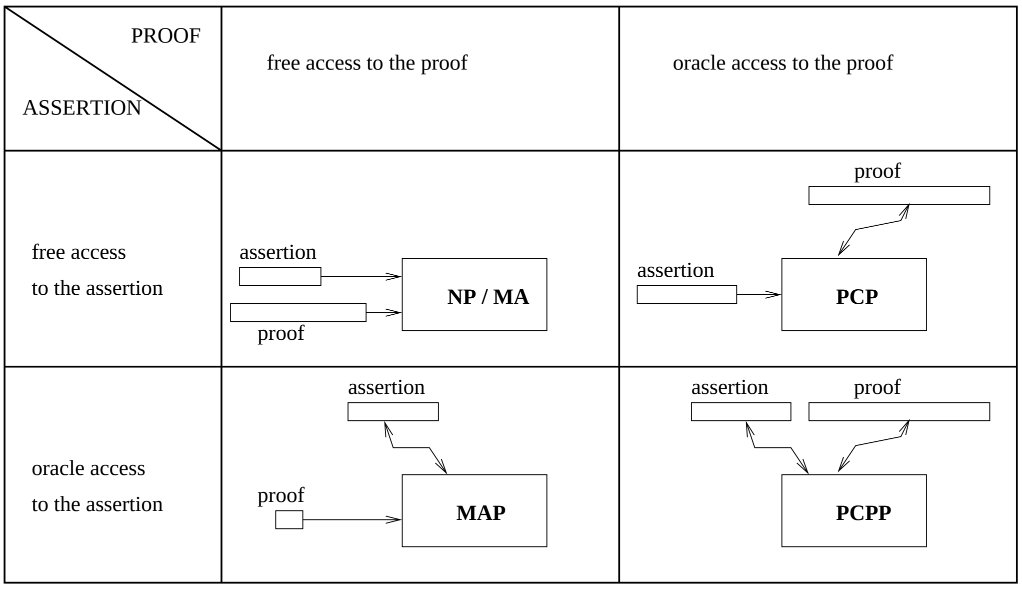
PROOF
ASSERTION
free access to the proof
oracle access to the proof
free access to the assertion
oracle access to the assertion
assertion
proof
NP / MA
PCP
MAP
PCPP

Figure 1: A taxonomy of four non-interactive proof systems.

$\langle A, B^f \rangle(z)$ the output of strategy B after interacting with A on explicit input z, while having oracle access to f.[21]

Definition 15 (interactive proofs of proximity (IPPs): *Let* $\Pi = \cup_{n\in\mathbb{N}}\Pi_n$ *such that* Π_n *contains functions defined over* $[n]$*. A system of* interactive proofs of proximity (IPP) for Π *is a probabilistic strategy, called a* verifier *and denoted* V*, that satisfies the following two conditions, for every* $n \in \mathbb{N}$ *and* $\epsilon > 0$*.*

1. Completeness: V accepts inputs in Π when interacting with an adequate prover. *Specifically, for every* $f \in \Pi_n$ *there exists a prover strategy* P *such that* $\mathbf{Pr}[\langle P, V^f \rangle(n, \epsilon) = 1] \geq 2/3$.
2. Soundness: V rejects inputs that are far from Π. *Specifically, for every* $f : [n] \to R_n$ *that is* ϵ*-far from* Π_n *and every strategy* $\widetilde{P}$ *it holds that* $\mathbf{Pr}[\langle \widetilde{P}, V^f \rangle(n, \epsilon) = 0] \geq 2/3$.

We say that V *has* query complexity $q : \mathbb{N} \times (0, 1] \to \mathbb{N}$ *if, on input* n, ϵ *and oracle access to any* $f : [n] \to R_n$*, the verifier makes at most* $q(n, \epsilon)$ *queries. The* communication complexity *of the IPP is defined as the total length of messages sent by the prover to the verifier.*

As with MAPs, the focus is on IPPs of sub-linear query and communication complexities. In addition, we may also bound the length of the messages sent by the verifier, and the total number of messages exchanges (a.k.a the number of communication rounds). And, again, we may also limit the computational complexity of the verifier.

Indeed, MAPs may be viewed as IPPs with uni-directional communication going from the prover to the verifier, and in this case the prover may just send a single message. We mention that IPPs are more powerful than MAPs: *there exists a property* Π *that has an IPP of polylogarithmic*

[21] In the following definition the first strategy (i.e., A) may depend arbitrarily on f, and so there is no point in providing it with oracle access to f.

communication complexity such that every MAP of proof complexity $n^{0.499}$ *for* Π *requires more than* $n^{0.499}$ *queries* [24]. Interestingly, *IPPs of sub-linear complexity* (i.e., total running time $n^{0.5+o(1)}$) are known *for all sets in the complexity class* $\mathcal{NC}$ [31].

A complexity theoretic perspective. The foregoing results illustrate complexity gaps between property testing, non-interactive proofs of proximity, and interactive proofs of proximity. Tentatively denoting by $\mathcal{PT}^{\rm pl}$, $\mathcal{MAP}^{\rm pl}$ and $\mathcal{IPP}^{\rm pl}$ the classes of properties that admit testers and verifiers of polylogarithmic query and communication complexity,[22] we get a separation between these classes; that is, $\mathcal{PT}^{\rm pl} \subset \mathcal{MAP}^{\rm pl} \subset \mathcal{IPP}^{\rm pl}$. In some sense, this means that (very natural) approximate decision versions of the complexity classes $\mathcal{RP}$, $\mathcal{MA}$ and $\mathcal{IP}$ are separated.

7 Chapter notes

7.1 Historical notes

Property testing with respect to general distributions as well as distribution-free testing, sample-based testing, and tolerant testing were all mentioned in [16, Sec. 2]. However, the focus of [16] as well as of almost all subsequent works was on the basic framework of Definition [intro:tester:def] (i.e., using queries in testing w.r.t the uniform distribution). An explicit study of the various ramifications started (later) in [25, 20, 30], respectively.

The study of tolerant testing of properties was initiated by Parnas, Ron, and Rubinfeld [30]. Theorem 3 was proved by Fischer and Newman [14]. Distance approximation in the context of bounded-degree and general graphs were first studied in [28]. An extensive study of sample-based testers was initiated by Goldreich and Ron [20]. Part 1 of Theorem 8 is due to [20], whereas Part 2 is due to [13]. The study of distribution-free testers was effectively initiated by Halevy and Kushilevitz [25], and a few works followed. We mention the work of Glasner and Servedio, which proves that distribution-free testing is significantly harder than standard testing also for very basic properties of Boolean functions [**?**].

The study of property testing with respect to the edit distance was initiated by Batu *et al.* [5], and a study of testing with respect to the $\mathcal{L}_p$-distance was initiated by Berman, Raskhodnikova, and Yaroslavtsev [6].

Local computation algorithms were defined, in full generality, by Rubinfeld *et al.* [32]. This notion generalizes the notion of finding huge structures, which is implicit in [16], and the notion of local reconstruction proposed by Ailon *et al.* [1] (and pursued in [33, 26]). An analogue of the notion of local reconstruction for the context in which the object is a distribution, called *sampling correctors*, was recently proposed in [**?**].

The notion of non-interactive proofs of proximity (MAPs) was introduced and studied by Gur and Rothblum [24], subsequent to the introduction and study of interactive proofs of proximity (IPPs) by Rothblum, Vadhan, and Wigderson [31].[23] We mention that IPPs are a special case of a general framework suggested before by Ergün, Kumar, and Rubinfeld [9]. We also mention that the notion of MAPs is implicit in the work of Fischer, Goldhirsh, and Lachish [12], who (concurrently

[22]More generally, we may denote by $\mathcal{PT}[q]$ the class of properties that admit testers of query complexity q, and by $\mathcal{MAP}[p,q]$ (resp., $\mathcal{IPP}[c,q]$) the classe of properties that admit verifiers of query complexity q and proof complexity p (resp., communication complexity c). The foregoing results assert that $\mathcal{PT}[n^{0.999}]$ does not contain $\mathcal{MA}[O(\log n), \mathrm{poly}(1/\epsilon)]$, and that $\mathcal{MA}[n^{0.499}, n^{0.499}]$ does not contain $\mathcal{IPP}[\mathrm{poly}(\log n), \mathrm{poly}(\epsilon^{-1}\log n)]$.

[23]We mention that Rothblum in [24], is different from Rothblum in [31].

and independently of [24]) referred to the existence of MAPs as an obstacle to proving lower bounds on a robust notion of testing (which they call "partial tests").

7.2 Exercises

Exercise 1 (tolerant testers with one-sided error probability):[24] *We say that a tolerant tester* (as in Definition 1) *has* one-sided error *if Condition 1 holds with probability 1* (i.e., T accepts with probability 1 any $f : [n] \rightarrow R_n$ that is ϵ'-close to Π). *Show that if for some $\epsilon > 0$ and $n \in \mathbb{N}$ it holds that Π_n is non-trivial w.r.t ϵ-testing* (i.e., $\Pi_n \neq \emptyset$ and there exists a function that is ϵ-far from Π), *then any ϵ'-tolerant ϵ-tester for Π_n makes more than $\epsilon' n$ queries, for any $\epsilon' \in (0, \epsilon)$.*

Guideline: Suppose that an ϵ'-tolerant ϵ-tester T makes q queries, and consider its execution when given access $f : [n] \rightarrow R_n$ that is ϵ-far from Π. Then, with positive probability, T rejects. Fix such a rejecting sequence of coins for T, and consider the set of locations Q that were queried. Finally, for an arbitrary $g \in \Pi_n$, consider a hybrid h of g and f such that $h(j) = f(j)$ if $j \in Q$ and $h(j) = g(j)$ otherwise. Since, h is rejected with positive probability, it must hold that it is ϵ'-far from $g \in \Pi_n$, which implies $|Q| > \epsilon' n$.

Exercise 2 (generic derivation of a weak tolerant tester): *Show that any ϵ-tester for Π that makes q queries that are each uniformly distributed,*[25] *yields and $(1/10q)$-tolerant ϵ-tester for Π of query complexity $O(q)$.*

Guideline: Observe that such a tester accepts any function that is ϵ'-close to Π with probability at least $2/3 - q \cdot \epsilon'$. Use error reduction to regain the original error bound of 2/3.

Exercise 3 (tolerant tester for t-colorability in the dense graph model): *For any $t \geq 2$, present an ϵ'-tolerant ϵ-tester for t-*`Colorability` *by using a reduction to several graph partition problems. Specifically, referring to the framework presented in Definition* `[dense:general-gpp:def]`, *use t-partition problems in which the absolute upper bounds on the edge density inside parts* (i.e., the $H_{i,i}^{\mathsf{abs}}$'s) *sum-up to ϵ', and test these properties with a proximity parameter set to* $(\epsilon - \epsilon')/2$.

Guideline: In case of $t = 2$, let $m = \lceil 4\epsilon'/(\epsilon - \epsilon') \rceil$, and, for every $i \in [m]$, consider the property associated with the non-trivial bounds $H_{1,1}^{\mathsf{abs}} = i \cdot (\epsilon - \epsilon')/4$ and $H_{2,2}^{\mathsf{abs}} = (m - i) \cdot (\epsilon - \epsilon')/4$. Essentially, run each of the m corresponding testers with proximity parameter $(\epsilon - \epsilon')/2$, and accept if and only if at least one of them accepts.

Exercise 4 (tolerant testers in the bounded-degree graph model): *Present tolerant testers for degree regularity, connectivity, and cycle-freeness. Specifically, present ϵ'-tolerant ϵ-testers of query complexity* $\text{poly}(1/\epsilon')$ *for some $\epsilon' = \Omega(\epsilon)$.*

Guideline: Use Claim `[bdg:deg-regularity:clm]`, Proposition `[bdg:connectivity:prop]`, and Proposition `[bdg:cycle-freeness:prop]`, respectively. Note that algorithmic steps that reject based on evidence for violation of the property should not be used here.

[24]Based on [34].

[25]We stress that different queries are allowed to depend on one another; we only postulate that each query, by itself, is uniformly distributed in the function's domain.

Exercise 5 (disposing of a promise underlying the bounded-degree graph model): *Recall that, in the bounded-degree graph model, the tester is given oracle access to a purported incidence function $g : [k] \times [d] \to \{0, 1, ..., k\}$ of a k-vertex graph of maximal degree d such that if $g(u,i) = v$ for some $u, v \in [k]$ and $i \in [d]$, then there exists $j \in [d]$ such that $g(v, j) = u$. Show that we can waive the latter assumption by increasing the query complexity of the tester by a factor of d and an additive term of $O(d/\epsilon)$.*

Guideline: First, test the condition by selecting $O(1/\epsilon)$ random pairs $(u,i) \in [k] \times [d]$. Assuming that this test passed, invoke the original tester while answering its queries according to the function $g' : [k] \times [d] \to \{0, 1, ..., k\}$ such that $g'(u,i) = v$ if $g(u,i) = v \in [k]$ and $u \in \{g(v,j) : j \in [d]\}$ (and $g'(u,i) = 0$ otherwise). Provide a detailed analysis of the performance of this tester.

Exercise 6 (MAPs with linear length proofs): *Show that every property Π has a MAP of linear proof complexity and query complexity $O(1/\epsilon)$.*

Guideline: The proof π consists of a copy of the input function f; the verifier checks whether $\pi \in \Pi$, and then checks that $\pi = f$ by quering f at $O(1/\epsilon)$ random locations.

Exercise 7 (a lower bound on the randomness complexity of MAPs):[26] *Consider a promise problem $(\Pi_{\rm YES}, \Pi_{\rm NO})$ regarding functions from $[n]$ to R, and say that Π is* strongly ρ-evasive *if there exists a function $f_1 : [n] \to R$ in $\Pi_{\rm YES}$ such that for every $Q \subset [n]$ of density ρ, there exists $f_0 \in \Pi_{\rm NO}$ such that for every $x \in Q$ it holds that $f_1(x) = f_0(x)$. Suppose that $(\Pi_{\rm YES}, \Pi_{\rm NO})$ is strongly ρ-evasive and that membership in $\Pi_{\rm YES}$ can be verified* (say, with error probability 1/3) *by an oracle machine M that makes q queries, while getting a proof of arbitrary length, and being guaranteed that the input is in $\Pi_{\rm YES} \cup \Pi_{\rm NO}$. Show that M must toss at least $\log_2(\rho n/q)$ coins. Note that this means that if $(\Pi, \{f : \delta_\Pi(f) > \epsilon\})$ is strongly ρ-evasive, then an ϵ-MAP for Π* (i.e., a MAP that works when the proximity parameter is set to ϵ) *must toss at least $\log_2(\rho n/q)$ coins.*[27]

Guideline: Suppose that M tosses r coins, and let f_1 be a function as in the strong ρ-evasive condition, and π be a suitable proof for f_1. Consider all 2^r possible executions of $M^{f_1}(\pi)$, and let Q denote the set of queries made in these executions. Then, $|Q| \le 2^r \cdot q$. On the other hand, $|Q| > \rho \cdot n$, since otherwise these executions cannot distinguish f_1 from the corresponding function f_0 that is guaranteed by the strong ρ-evasive condition.

Exercise 8 (upper bound on the randomness complexity of MAPs):[28] *Let $\Pi = \cup_n \Pi_n$, where Π_n is a subset of $\{f : [n] \to R_n\}$. Suppose that Π has a MAP of query complexity q and proof complexity p such that $p(n) \le n$. Show that Π has a MAP of query complexity q, proof complexity p, and randomness complexity at most $\log n + \log\log(2|R_n|)$ coins. Note that the randomness-efficient verifier derived here is not necessarily computationally-efficient.*

[26]This exercise is related to Exercise `[intro:randomness-lb:exer]`, which refers to the case of deciding rather than verifying. In Exercise `[intro:randomness-lb:exer]`, ρ-evasive meant that there exists a function $f : [n] \to R$ such that for every $Q \subset [n]$ of density ρ, there exist $f_1 \in \Pi_{\rm YES}$ and $f_0 \in \Pi_{\rm NO}$ such that for every $x \in Q$ it holds that $f_1(x) = f_0(x) = f(x)$. Here, we mandate that $f = f_1$.

[27]Note that for many natural properties and for sufficiently small constant $\epsilon > 0$, the problem of ϵ-testing the property is strongly $\Omega(1)$-evasive. A partial list includes sets of low degree polynomials, any code of linear distance, monotonicity, juntas, and various graph properties.

[28]Based on [21, 24]. This exercise extends Exercise `[intro:randomness-ub:exer]`, which refers to the case of deciding rather than verifying.

Guideline: Suppose that the MAP V tosses $r = r(n)$ coins, and observe that the number of possible functions that V is required to rule about is at most $|R_n|^n$, whereas each such function has $2^{p(n)}$ possible proofs. Using the probabilistic method, show that there exists a $\log_2 O(|R_n|^n \cdot 2^{p(n)})$-set $S \subseteq \{0,1\}^r$, such that for every function $f : [n] \to R$ and every $\pi \in \{0,1\}^{p(n)}$ it holds that

$$|\mathbf{Pr}_{\omega\in S}[M^f(\omega;\pi) = 1] - \mathbf{Pr}_{\omega\in\{0,1\}^r}[M^f(\omega;\pi) = 1]| < 1/12.$$

Then, a randomness-efficient machine may select ω uniformly in S, and emulate M while providing it with ω (as the outcome of the internal coin tosses used by M).

Exercise 9 (MAPs are stronger than property testers):[29] *Show that the property* $\Pi = \{uuvv : u, v \in \{0,1\}^*\}$ *has a MAP of logarithmic proof complexity and query complexity* $O(1/\epsilon)$*, whereas it is not testable with* $o(\sqrt{n})$ *queries.*

Guideline: Regarding the MAP, for $uuvv \in \Pi$, consider the proof $\pi = |u| \in \{0,1\}^{\log_2 |u|}$. As for the lower bound consider the uniform distribution over $\Pi_n = \Pi \cap \{0,1\}^n$ versus the uniform distribution over all n-bit long strings. Actually, consider a minor variation on the first distribution obtained by picking $i \in [n/2]$ uniformly at random and selecting $u \in \{0,1\}^i$ and $v \in \{0,1\}^{0.5n-i}$ uniformly at random. Note that a machine that makes $o(\sqrt{n})$ queries cannot distinguish these two distributions, since in the first distribution only locations that are at distance either i or $0.5n - i$ apart are correlated.

References

[1] N. Ailon, B. Chazelle, S. Comandur, and D. Liu. Property-preserving data reconstruction. *Algorithmica*, Vol. 51 (2), pages 160–182, 2008.

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

[3] N. Alon, M. Krivelevich, I. Newman, and M. Szegedy. Regular Languages are Testable with a Constant Number of Queries. *SIAM Journal on Computing*, Vol. 30 (6), pages 1842–1862, 2001.

[4] M. Balcan, E. Blais, A. Blum, and L. Yang. Active property testing. In *53rd IEEE Symposium on Foundations of Computer Science*, pages 21–30, 2012.

[5] T. Batu, F. Ergun, J. Kilian, A. Magen, S. Raskhodnikova, R. Rubinfeld, and R. Sami. A sublinear algorithm for weakly approximating edit distance. In *35th ACM Symposium on the Theory of Computing*, pages 316–324, 2003.

[6] P. Berman, S. Raskhodnikova, and G. Yaroslavtsev. Lp-testing. In *46th ACM Symposium on the Theory of Computing*, pages 164–173, 2014.

[7] A. Czumaj, M. Monemizadeh, K. Onak, and C. Sohler. Planar Graphs: Random Walks and Bipartiteness Testing. In *52nd IEEE Symposium on Foundations of Computer Science*, pages 423–432, 2011.

[29]Based on [3, 12]: The lower bound is based on [3], which actually considered the context-free language $\{uu^R vv^R : u, v \in \{0,1\}^*\}$, where $u^R = u_m \cdots u_1$ is the "reverse" of $u = u_1 \cdots u_m$. The upper bound was first mentioned in [12].

[8] E. Dolev and D. Ron. Distribution-Free Testing for Monomials with a Sublinear Number of Queries. *Theory of Computing*, Vol. 7(1), pages 155–176, 2011.

[9] F. Ergün, R. Kumar, and R. Rubinfeld. Fast approximate PCPs. In *31st ACM Symposium on the Theory of Computing*, pages 41–50, 1999.

[10] G. Even, M. Medina, and D. Ron. Best of Two Local Models: Local Centralized and Local Distributed Algorithms. CoRR abs/1402.3796, 2014.

[11] E. Fischer and L. Fortnow. Tolerant Versus Intolerant Testing for Boolean Properties. *Theory of Computing*, Vol. 2 (9), pages 173–183, 2006.

[12] E. Fischer, Y. Goldhirsh, and O. Lachish. Partial tests, universal tests and decomposability. In *5th Innovations in Theoretical Computer Science*, pages 483–500, 2014. Full version posted on *ECCC*, TR13-082, 2013.

[13] E. Fischer, O. Lachish, and Y. Vasudev. Trading Query Complexity for Sample-Based Testing and Multi-testing Scalability. In *56th IEEE Symposium on Foundations of Computer Science*, pages 1163–1182, 2015.

[14] E. Fischer and I. Newman. Testing versus estimation of graph properties. In *37th STOC*, pages 138–146, 2005.

[15] O. Goldreich. *Computational Complexity: A Conceptual Perspective.* Cambridge University Press, 2008.

[16] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM 45*, 4 (July 1998), 653–750. (Preliminary Version in *37th FOCS*, 1996).

[17] O. Goldreich, T. Gur, and I. Komargodski. Strong Locally Testable Codes with Relaxed Local Decoders. In *30th* IEEE Conference on Computational Complexity, pages 1–41, 2015.

[18] O. Goldreich, T. Gur, and R. Rothblum. Proofs of Proximity for Context-Free Languages and Read-Once Branching Programs. In *42nd Int. Colloquium on Automata, Languages and Programming*, pages 666–677, 2015.

[19] O. Goldreich and D. Ron. Algorithmic Aspects of Property Testing in the Dense Graphs Model. *SIAM Journal on Computing*, Vol. 40, No. 2, pages 376–445, 2011.

[20] O. Goldreich and D. Ron. On Sample-Based Testers. In *6th Innovations in Theoretical Computer Science*, pages 337–345, 2015.

[21] O. Goldreich and O. Sheffet. On The Randomness Complexity of Property Testing. *Computational Complexity*, Vol. 19 (1), pages 99–133, 2010.

[22] O. Goldreich, S. Vadhan, and A. Wigderson. On interactive proofs with a laconic prover. *Computational Complexity*, Vol. 11 (1-2), pages 1–53, 2002.

[23] M. Gonen and D. Ron. On the Benefit of Adaptivity in Property Testing of Dense Graphs. In *Proc. of RANDOM'07*, LNCS Vol. 4627, pages 525–539, 2007. *Algorithmica* (special issue for RANDOM and APPROX 2007), Vol. 58 (4), pages 811–830, 2010.

[24] T. Gur and R. Rothblum. Non-interactive proofs of proximity. *ECCC*, TR13-078, 2013.

[25] S. Halevy and E. Kushilevitz. Distribution-free property testing. In *7th RANDOM*, pages 341–353, 2003.

[26] M. Jha and S. Raskhodnikova. Testing and Reconstruction of Lipschitz Functions with Applications to Data Privacy. *SIAM Journal on Computing*, Vol. 42(2), pages 700–731, 2013. Extended abstract in *52nd FOCS*, 2011.

[27] R. Levi, D. Ron, and R. Rubinfeld. Local Algorithms for Sparse Spanning Graphs. In *18th RANDOM*, LIPIcs, Vol. 28, Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, pages 826–842, 2014.

[28] S. Marko and D. Ron. Distance Approximation in Bounded-Degree and General Sparse Graphs. *Transactions on Algorithms*, Vol. 5 (2), Art. 22, 2009.

[29] I. Newman and C. Sohler. Every Property of Hyperfinite Graphs Is Testable. *SIAM Journal on Computing*, Vol. 42 (3), pages 1095–1112, 2013.

[30] M. Parnas, D. Ron, and R. Rubinfeld. Tolerant property testing and distance approximation. *Journal of Computer and System Science*, Vol. 72(6), pages 1012–1042, 2006.

[31] G. Rothblum, S. Vadhan, and A. Wigderson. Interactive Proofs of Proximity: Delegating Computation in Sublinear Time. In *45th ACM Symposium on the Theory of Computing*, pages 793–802, 2013.

[32] R. Rubinfeld, G. Tamir, S. Vardi, and M. Xie. Fast Local Computation Algorithms. In *2nd ICS*, pages 223–238, 2011.

[33] M. Saks and C. Seshadhri. Local monotonicity reconstruction. *SIAM Journal on Computing*, Vol. 39 (7), pages 2897–2926, 2010.

[34] R. Tell. A Note on Tolerant Testing with One-Sided Error. *ECCC*, TR16-032, 2016.