Notes for the first lecture in a course on Property Testing

Oded Goldreich[*]

March 13, 2016

Summary: This is the first lecture in an introductory course on property testing. In this lecture, we introduce and illustrate the basic notions of property testing, emphasizing the themes of *approximate decision* and *sub-linear complexity*. In particular, we discuss the key role of representation, point out the focus on properties that are not fully symmetric, present the definitions of (standard) testers and of proximity-oblivious testers (POTs), and make some general observations regarding POTs, testing, and learning. All is preceded by a discussion of the potential benefits of testing.

1 Introduction

Big data is a broad term for data sets so large or complex that traditional data processing applications are inadequate.

Wikipedia entry on *Big Data*, 17-Feb-2016.

Everybody talks of Big Data. The issue is, of course, making good use of the data, which requires analyzing it. But such an analysis may mean various things. At one extreme, it may mean locating tiny and possible rare (but valuable) pieces of information. On the other extreme, it may means detecting global structures or estimating global parameters of the data as a whole.

The field of *property testing* is related to the latter meaning. It is concerned with the analysis of global features of the data, like determining whether the data as a whole has some structural property or estimating some global parameter of its structure. The focus is on properties and parameters that go beyond a simple statistics that refers to the frequency of occurrence of various local patterns. This is not intended to say that such simple statistics are not of value, but rather that not everything of interest can be reduced to them.

In general, the data is a set of records that may be related in various ways. Its contents and meaning is reflected not only in the individual records, but also in the relations between them. The mere existence of relations between pairs of records can be modeled as a graph. Needless to say, this captures only one aspect of the data, and this aspect is an abstract structure. Nevertheless, when such a model is used, checking whether the graph that arises has certain structiral properties is of natural interest. Indeed, testing natural properties of huge graphs or estimating various parameters of such graphs is part of the agenda of *property testing.* More generally, *property testing* is concerned with testing structural properties of huge objects or estimating such structural parameters.

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

Important as it is, big data is not the only source of huge objects that are considered by *property testing*. Another type of huge objects are the functions that are computed by various programs or other computing devices. We stress that these objects do not appear in explicit form in reality; they are merely defined implicitly (and concisely) by these devices.

Our repeated reference to the huge size of the objects is meant to emphasize a salient feature of *property testing*. We refer to the fact that *property testing* seeks super-fast algorithms. In particular, it refers to algorithms that do not read the full description of the object, but rather inspect relatively small portions of the object.

The reader may justifiably wonder how it is possible to say anything meaningful about an object without looking at all of it. But on second thought, one may note that we are aware of such cases: All frequency statistics are of these forms. It is worthwhile to highlight two features of these statistics: They are *approximate* rather than exact, and they are generated based on *random choices*. Indeed, a notion of approximation and the use of randomness are pivotal to *property testing*. (Yet, we stress again that property testing goes beyond frequency statistics.)

1.1 Property testing at a glance

As will be detailed in this lecture, property testing is primarily concerned with super-fact approximate decisions, where the task is distinguishing between objects having a predetermined property and objects that are "far" from having this property. Related tasks such as estimating structural parameters of such objects or finding related substructures are also addressed. In all cases, the algorithms sought are such of sub-linear complexity, and in particular they only inspect realtively small portions of the object.

Typically, objects are modeled by functions, and distance between functions is measured as the fraction of the domain on which the functions differ. An object is considered far from having the property if its distance from any object that has the property exceeds a given proximity parameter. We consider (randomized) algorithms that may query the function at arguments of their choice, where this modeling allows for discussing algorithms that only inspect part of their input.

Cases in which such super-fact approximate decision is possible include testing properties of functions such as being a low degree polynomial, being monotone, and depending on a specified number of attributes; testing properties of graphs such as being bipartite and being triangle-free; and testing properties of geometric objects such as being well-clustered and being a convex body.

In the next section, we review the potential benefits of property testers. But before doing so, we wish to stress that, like with any theoretical research, the value of research in property testing is not confined to the suggested algorithms (i.e., resulting testers). The development and study of conceptual frameworks, let alone the development of algorithmic design and analysis techniques, is even more important for the theory of computation at large as well as for computer practice. While the impact on practice is typically hard to trace, the tight relations between property testing and the rest of the theory of computing are evident (and will be pointed out in adequate parts of this course).

1.2 On the benefits of property testers

Property testing is associated with approximate decision algorithms that run in sub-linear time or at least make a sub-linear number of queries to their input. The benefit of sub-linear complexity is significant when the input is huge, but this benefit comes at the cost of having an approximate decision rather than an exact one. The question addressed at this section is whether (or rather when can) this trading of accuracy for efficiency be worthwhile. The answer is application-dependent rather than universal: We discuss several different general settings is which such a trading is worthwhile.[1]

It is infeasible to fully recover the object. This may be the case either because linear time is infeasible for the huge objects being considered in the application or because probes to the object are too expensive to allows for inspecting all of it. In such settings, there is no choice but to use algorithms of sub-linear query complexity and settle for whatever they can provide (of course, the more – the better).

Objects either have the property or are far from having it. That is, we refer to applications in which we know a priori that the objects that we will encounter either have the property or are far from any object having the property. Intuitively, in such a case, objects are either perfect (i.e., have the property) or are very bad (i.e., far from it). In this case, we should not care about inputs that violate the promise (i.e., are neither in the set nor far from it), because such inputs correspond to objects that we are unlikely to encounter.

Objects that are close to having the property are good enough. That is, we refer to applications in which the utility of objects that are close to having the property is almost as good as the utility of objects that have the property. Alternatively, it may be possible to modify the object at a cost related to its distance from having the property. In such cases, we may not care too much about deciding that the object has the property while in reality the object may only be close to having this property.[2]

Testing as a preliminary step before deciding. That is, we refer to the possibility of using the approximate decision procedure as a preliminary step, and using the more costly exact decision procedure only if the preliminary step was completed successfully (i.e., the approximate decider accepted the input). This is advantageous provided that objects that are far from having the property are not very rare, since we definitely save resources when rejecting on such objects based on the preliminary step.

Testing as a preliminary step before reconstructing. This refers to settings in which we wish to fully recover the object, either by reading all of it or by running a learning algorithm, but we need to do so only if the object has the property. Hence, before invoking the reconstruction procedure, we want to (approximately) decide whether the object has the property. (In the case of reconstruction by a learning algorithm, this makes sense only if the approximate decision procedure is more efficient than the learning algorithm.) Again, using the approximate decision procedure is advantageous provided that objects that are far from having the property are not very rare.

[1]Recall "Paris is well worth a Mass" [*Paris vaut bien une messe*, King Henry IV, 1593].

[2]One may argue that in such cases, "tolerant testing" (see Section 3.2) is even more adequate. Yet, tolerant testing may be harder than standard testing (cf. [25]).

1.3 On the flavor of property testing research

Property testing seems to stand between algorithmic research and complexity theory. While the field's primary goal is the design of a certain type of algorithms (i.e., sublinear ones) for a certain type of problems (i.e., approximate decision), it often needs to determine the limits of such algorithms, which is a question of lower bounds having a complexity theoretic flavor. Furthermore, historically, property testing was associated with the study of Probabilistically Checkable Proofs (PCPs), and some connections do exist between the two, but property testing is not confined to PCPs (and/or to the study of "locally testable codes").

In addition to standing in between these two areas, the results of property testing have a color that makes the different from the maintream results in both areas. Its positive results are not perceived as mainstream algorithmic research and its negative results are not perceived as mainstream complexity theory. In both cases, the specific flavor of propery testing (i.e., sub-linear approximate decision) makes them stand out. But property testing is not the only research are that has this fate: The same can be said of Machine Learning and Distributed Computing, to mention just two examples.

One additional charateristic of property testing is that the positive results tend to be established by simple algorithms that are backed by a complex analysis. The simplicity of these algorithms has met the lack of respect of a few researchers, but this is a fundamental mistake on their side. The simplicity of algorithms is a virtue if one really considers using them, whereas the complexity of their analysis has no cost in terms of their applicability. Hence, simple algorithms that require a complex analysis are actually the greatest achievement that algorithmic research should strive at.

1.4 Organization and some notations

As stated above, we view property testing as primarily concerned with approximate decisions, a notion that is discussed in Section 2.2. (For perspective, we precede it with Section 2.1, which recall the notion of approximate search problems.) Next, in Section 2.3, we discuss the second key feature of property testing – its focus on sub-linear complexity. Then, in Section 2.4, we highlight yet another feature of property testing – its focus on properties that are not fully symmetric (i.e., are not invariant under arbitrary re-ordering of the sequence of values that represent the object). The general relation between objects and their reprwesentation is discussed in Section 2.5.

The core of this chapter is presented in Section 3. The basic notions, definitions, and goals of property testing will be presented in Section 3.1, and will be used extensively throughout the entire lecture series (with very few exceptions). In contrast, the ramifications discussed in Section 3.2 will be lightly used (if at all), and ditto for the general observations made in Sections 3.4 and 3.5 (which refer to the "algebra of property testing" and to its relation to learning, respectively). In Section 3.3, we shall present another notion that will be used quite a lot – that of a proximity-oblivious tester (POT).

Historical perspectives and suggestions for further reading are provided in Sections 4 and 5, respectively. Finally, in Section 6 we re-iterate some of issues discussed in the current lecture, in light of their importance to the rest of the course.

Some notation. We shall be using the following *standard notations*:

- For $n \in \mathbb{N}$, we let $[n] \overset{\text{def}}{=} \{1, ..., n\}$.

- For $x \in \{0,1\}^*$, we let $|x|$ denote the length of x and let x_i denote the i^{th} bit of x; that is, if $n = |x|$, then $x = x_1 \cdots x_n$ such that $x_i \in \{0,1\}$ for every $i \in [n]$.
- The Hamming weight of a string x, denoted $\mathrm{wt}(x)$, is the number of locations that hold the value one; that is,

$$\mathrm{wt}(x) = |\{i \in [|x|] : x_i = 1\}| = \sum_{i=1}^{|x|} x_i.$$

Suggested preliminaries for the course include basic familiarity with

1. the notions of decision, search and promise problems (see, e.g., [27, Sec. 1.2]);
2. probabilistic algorithms (see, e.g., [27, Sec. 6.1] or [49]); and
3. probabilistic inequalities such as the Union Bound and Chernoff Bound (see, e.g., [27, Apdx. D.1] or [49]).

Teaching note: It is quite likely that covering the material presented in these notes will require more than a single lecture.

2 Approximate decisions

The notion of approximation is well known in the context of optimization problems, which are a special type of search problems. We start by recalling these notions.

2.1 A detour: approximate search problems

Recall that search problems are defined in terms of binary relations, denoted $R \subseteq \{0,1\}^* \times \{0,1\}^*$, and consist of finding a "valid solution" y to a given instance x, where y is a valid solution to x if $(x,y) \in R$. The set of solutions for the instance x is denoted $R(x) \stackrel{\text{def}}{=} \{y : (x,y) \in R\}$, and the computation of function corresponds to the special case in which this set is a singleton.

In optimization problems, the valid solutions are assigned a value (or a cost), captured by a function $\nu : \{0,1\}^* \to \mathbb{R}$, and one is asked to find a solution of maximum value (resp., minimum cost); that is, given x, the task is to find $y \in R(x)$ such that $\nu(y) = \max_{z \in R(x)}\{\nu(z)\}$ (resp., $\nu(y) = \min_{z \in R(x)}\{\nu(z)\}$).[3]

A corresponding approximation problem is defined as finding a solution having value (resp., cost) close to the optimum; that is, given x the task is to find $y \in R(x)$ such that $\nu(y)$ is "close" to $\max_{z \in R(x)}\{\nu(z)\}$ (resp., to $\min_{z \in R(x)}\{\nu(z)\}$). One may also talk about the estimation problem, in which the task is to approximate the value of the optimal solution (rather than finding a solution that obtains that value).

The point we wish to make here is that, once a function ν and a proximity parameter are fixed, it is clear what one means by seeking an approximation solution for a search problem. But, what do we mean when we talk about approximate decision problems?

[3]Greater flexibility is achieved by allowing the value (resp., cost) to depend also on the instance; that is, use $\nu(x,y)$ rather than $\nu(y)$. Actually, this does not buy any additional generality, because we can always augment the solution y by the instance x and use $\nu'(\langle y,x \rangle) = \nu(x,y)$. On the other hand, using the more flexible formulation, one can get rid of the relation R by letting $\nu(x,y) = -\infty$ (resp., $\nu(x,y) = \infty$) if $(x,y) \notin R$.

2.2 Property testing: approximate decision problems

Indeed, *what can an approximate decision problem actually mean*?

Unfortunately, there is no *decisive answer* to such questions; one can only propose an answer and articulate its natural appeal. Indeed, we believe that a natural notion of approximate decision (or a relaxation of the decision problem) is obtained by ignoring "borderline" cases, which are captured by inputs that are close to the set but do not reside in it. That is, instead of asking whether an input x is in the set S, we consider the problem of distinguishing between the case that $x \in S$ and the case that x is "far" from S. Hence, we consider a promise problem (cf. [22] or [27, Sec. 2.4.1]), in which the YES-instances are the elements of S and the NO-instances are "far" from S.

Of course, we need to clarify what "far" means. To this end, we fixed a metric, which will be the (relative) Hamming distance, and introduce a proximity parameter, denoted ϵ. Specifically, letting $\delta(x,z) = |\{i \in [|x|] : x_i \neq z_i\}|/|x|$ if $|x| = |z|$ and $\delta(x,z) = \infty$ otherwise, we define the distance of $x \in \{0,1\}^*$ from S as $\delta_S(x) \overset{\text{def}}{=} \min_{z \in S}\{\delta(x,z)\}$. Now, for a fixed value of $\epsilon > 0$, the foregoing promise problem consists of distinguishing S from $\{x : \delta_S(x) > \epsilon\}$, which means that inputs in $\{x : 0 < \delta(x) \leq \epsilon\}$ are ignored.

Notation. Throughout the text, unless explicitly said differently, ϵ will denote a proximity parameter, which determines what is considered far. We shall say that x is ϵ-far from S if $\delta_S(x) > \epsilon$, and otherwise (i.e., when $\delta_S(x) \leq \epsilon$) we shall say that x is ϵ-close to S. Recall that $\delta_S(x)$ denotes the relative Hamming distance of x from S; that is, $\delta_S(x)$ is the minimum, taken over all $z \in S \cap \{0,1\}^{|x|}$, of $|\{i \in [|x|] : x_i \neq z_i\}|/|x|$.

Lastly, we note that the set S will be associated with the property of being in it, which for simplicity will also be referred to as the property S. Approximate decision will be later called property testing; that is, *approximate decision for a set S corresponds to testing the property S.*

2.3 Property testing: sub-linear complexity

But *why did we relax standard decision problems into approximate decision problems*? The answer is, as in the case of approximate search problems, that this is done in order to allow for more efficient algorithms.

This answer is clear enough when the best known (or best possible) decision procedure requires more than linear time, let alone when the original decision problem is NP-Hard. But property testing deals also with properties that have linear-time algorithms. In these cases as well as in the former cases, the relaxation to approximate decision suggests the possibility of *sublinear-time algorithms*; that is, algorithms that do not even read their entire input. Such algorithms are particularly beneficial when the input is huge (see Section 1.2).

The latter suggestion requires a clarification. Talking about algorithms that do not read their entire input calls for a model of computation in which the algorithms have *direct access* to bits of the input. Unlike in complexity theory, such a model is quite common in algorithmic research: It is the standard RAM model. (For sake of abstraction, we will actually prefer to use the model of oracle machines, while viewing the oracle as the input device.)

Except in degenerate cases (in which the decision problem is essentially insensitive to many bits in the input), the relaxation to approximate decision seems necessary for avoiding the reading of the entire input. For example, if S is the set of strings having even parity, then an exact decision procedure must read all the bits of the input (since flipping a single bit will change the decision), but the approximate decision problem is trivial (since each n-bit string is $1/n$-close to S). A more interesting case is presented next.

The case of `majority`. Let $\mathtt{MAJ} = \{x : \sum_{i=1}^{|x|} x_i > |x|/2\}$. We shall show that the corresponding approximate decision problem can be solved by a (randomized) poly$(1/\epsilon)$-time algorithm, whereas no sublinear-time (randomized) algorithm can solve the corresponding (exact) decision problem. We shall also show that randomness is essential for the positive result.

Proposition 1 (a fast approximate decision procedure for MAJ): *There exists a randomized $O(1/\epsilon^2)$-time algorithm that decides whether x is in* MAJ *or is ϵ-far from* MAJ.

As usual in the context of rqandomized algorithms, deciding means outputting the correct answer with probability at least 2/3.

Proof: The algorithm queries the input x at $m = O(1/\epsilon^2)$ uniformly and independently distributed locations, denoted $i_1, ..., i_m$, and accepts if and only if the average value of these bits (i.e., $\sum_{j\in[m]} x_{i_j}/m$) exceeds $(1-\epsilon)/2$. In the analysis, we use the Chernoff Bound (or alternatively Chebyshev's Inequality)[4], which implies that, with probability at least 2/3, the average of the sample is within $\epsilon/2$ of the actual average; that is,

$$\mathbf{Pr}_{i_1,\ldots,i_m\in[|x|]}\left[\left|m^{-1}\cdot\sum_{j\in[m]} x_{i_j} - |x|^{-1}\cdot\sum_{i=1}^{|x|} x_i\right| \le \epsilon/2\right] \ \ge\ 2/3.$$

It follows that the algorithm accepts each $x \in \mathtt{MAJ}$ with probability at least 2/3, since $\sum_{i=1}^{|x|} x_i > |x|/2$. Likewise, it rejects each x that is ϵ-far from MAJ with probability at least 2/3, since $|x|^{-1} \cdot \sum_{i=1}^{|x|} x_i < 0.5 - \epsilon$. ■

Proposition 2 (lower bound on decision procedures for MAJ): *Any randomized algorithm that exactly decides membership in* MAJ *must make $\Omega(n)$ queries, where n is the length of the input.*

Teaching note: The following proof may be harder to follow than all other proofs in this lecture, with the exception of the proof of Proposition 11, which is also a lower bound. Some readers may prefer to skip these proof at the current time, and return to them at a latter time (e.g., after the lecture on lower bounds). I prefer to keep the proofs in place, but warn the readers not to stall at them.

Proof: For every $n \in \mathbb{N}$, we consider two probability distributions: A distribution X_n that is uniform over n-bit strings having Hamming weight $\lfloor n/2\rfloor + 1$, and a distribution Z_n that is uniform over n-bit strings having Hamming weight $\lfloor n/2\rfloor$. Hence, $\mathbf{Pr}[X_n \in \mathtt{MAJ}] = 1$ and $\mathbf{Pr}[Z_n \in \mathtt{MAJ}] = 0$. However, as shown in Claim 2.1, a randomized algorithm that queries either X_n or Z_n at $o(n)$ locations cannot distinguish these two cases with probabilistic gap that exceeds $o(1)$, and hence must be wrong on one of the two cases.

[4]Indeed, both inequalities are essentially equivalent when one seeks constant error probability. See discussion in [27, Apdx. D.1.2.4].

(Note that the randomized decision procedure must be correct on each input. The proof technique employed here shows that such a procedure fails even in the potentially simpler task of distinguishing between some distribution of YES-instances and some distribution of NO-instances. Failing to distinguish these two distributions implies that the procedure errs with too large probability on at least one of these two distributions, which in turn implies that there exist at least one input on which the procedure errs with too large probability.)

Claim 2.1 (indistinguishability claim): *Let A be an algorithm that queries its n-bit long input at q locations. Then, $|\mathbf{Pr}[A(X_n)=1] - \mathbf{Pr}[A(Z_n)=1]| \leq q/n$.*

We stress that the claim holds even if the algorithm is randomized and selects its queries adaptively (based on answers to prior queries).

Proof: It is instructive to view X_n as generated by the following random process: First $i \in [n]$ is selected uniformly, then $y \in \{0,1\}^n$ is selected uniformly among the strings of Hamming weight $\lfloor n/2 \rfloor$ that have zero in position i, and finally X_n is set to $y \oplus 0^{i-1}10^{n-i}$. Likewise, Z_n is generated by letting $Z_n \leftarrow y$. (This is indeed a complicated way to present these random variables, but is greatly facilitates the following analysis.)[5] Now, observe that, as long as A does not query location i, it behaves in exactly the same way on X_n and Z_n, since in both cases it effectively queries the same random y. The claim follows. ■

By the indistinguishability claim (Claim 2.1), if algorithm A queries its n-bit long input on less than $n/3$ locations, then $|\mathbf{Pr}[A(X_n)=1]-\mathbf{Pr}[A(Z_n)=1]| < 1/3$. Hence, either $\mathbf{Pr}[A(X_n)=1] < 2/3$, which means that A errs (w.p. greater than 1/3) on some $x \in \texttt{MAJ}$, or $\mathbf{Pr}[A(Z_n)=1] > 1/3$, which means that A errs (w.p. greater than 1/3) on some $z \notin \texttt{MAJ}$. The proposition follows. ■

Proposition 3 (randomization is essential for Proposition 1): *Any* deterministic *algorithm that distinguishes between inputs in* MAJ *and inputs that are* 0.5*-far from* MAJ *must make at least $n/2$ queries, where n is the length of the input.*

Proof: Considering an arbitrary *deterministic* algorithm that makes $q < n/2$ queries and accepts all inputs in MAJ, we shall show that this algorithm also accepts the all-zero string, which is 0.5-far from MAJ. Denoting this algorithm by A, consider the (unique) execution of A in which all queries of A are answered by zero, and denote the set of queried locations by Q. We now consider two different n-bit long strings that are consistent with these answers. The first string, denoted x, is defined such that $x_j = 1$ if and only if $j \notin Q$, and the second string is $z = 0^n$. Note that $x \in \texttt{MAJ}$ (since $\mathrm{wt}(x) = n - q > n/2$), which mandates $A(x) = 1$. It follows that $A(z) = 1$, since A behaves identically on x and z, whereas z is 0.5-far from MAJ. Hence, any *deterministic* that makes $q < n/2$ queries fails to distinguish between inputs in MAJ and inputs that are 0.5-far from MAJ. ■

Digest. We have seen that sub-linear time algorithms for approximate decision problems exist in cases in which exact decision requires linear time. The benefit of the former is significant when the input is huge, although this benefit comes at the cost of having an approximate decision rather than an exact one.

[5]See Exercise 2 for details.

2.4 Symmetries and invariants

The proof of Proposition 1 reflects the well known practice of using sampling in order to estimate the average value of a function defined over a huge population. The same practice applies to any problem that refers to the statistics of binary values, while totally ignoring the identity of the entities to which these values are assigned. In particular, this refers to symmetric properties (of binary sequences), which are defined as *sets S such that for every $x \in \{0,1\}^*$ and every permutation π over $[|x|]$ it holds that $x \in S$ if and only if $x_{\pi(1)} \cdots x_{\pi(|x|)} \in S$.*

Theorem 4 (testing symmetric properties of binary sequences): *For every symmetric property* (of binary sequences)*, S, there exists a randomized algorithm that makes $O(1/\epsilon^2)$ queries and decides whether x is in S or is ϵ-far from S.*

(The result can be generalized to symmetric properties of sequences over any finite alphabet.[6] The result does not generalize to sequences over infinite alphabet. In fact, there exist (natural) properties (see, e.g., [62, Sec. 5.5]) for which approximate decision requires an almost linear number of queries (i.e., almost linear in the number of different elements in the sequence).)

Proof: The key observation is for every n there exists a set (of weights) $W_n \subseteq \{0,1,...,n\}$ such that for every $x \in \{0,1\}^n$ it holds that $x \in S$ if and only if $\text{wt}(x) \in W_n$. (In the case of $\texttt{MAJ}$, the set W_n is $\{\lfloor n/2 \rfloor + 1, ..., n\}$.) Hence, deciding whether x is in S or is ϵ-far from S reduces to estimating $\text{wt}(x)$ and comparing it to $W_{|x|}$. Specifically, on input x, the algorithm proceeds as follows:

1. Queries the input x at $m = O(1/\epsilon^2)$ uniformly and independently distributed locations, denoted $i_1, ..., i_m$, and computes the value $v = \sum_{j\in[m]} x_{i_j}/m$.
2. Accepts if and only if there exists $w \in W_{|x|}$ such that $|v - (w/|x|)| \leq \epsilon/2$.

   Note that this step requires knowledge of $|x|$ (as well as of the set $W_{|x|}$) but no queries to x; its computational complexity depends on the "structure" of $W_{|x|}$ (or, equivalently, on the unary set $S \cap \{1\}^*$).

As in the proof of Proposition 1, the analysis of this algorithm reduces to (1) noting that $\mathbf{Pr}[|v - \text{wt}(x)/|x|| \leq \epsilon/2] \geq 2/3$, and (2) observing that the distance of x from S (i.e., $\delta_S(x) \cdot |x|$) equals to $\min_{w\in W_{|x|}}\{|w - \text{wt}(x)|\}$. ■

Beyond symmetric properties. The focus of property testing is on asymmetric properties (although there are exceptions, see for example [1]). By asymmetric properties we mean ones that are *not* invariant under *all* permutation of the bit locations, although they may be invariant under some (non-trivial) permutations. In the extreme case the property is not invariant under any non-trivial permutation of the bit locations; that is, $x \in S$ if and only if $x_{\pi(1)} \cdots x_{\pi(|x|)} \in S$ holds only when π is the identity permutation. This is the case for the "sorted-ness" property discussed next.

We say that a string $x \in \{0,1\}^*$ is sorted if $x_i \leq x_{i+1}$ for every $i \in [|x| - 1]$. Denote the set of sorted n-bit long strings by $\texttt{SORTED}_n$, and let $\texttt{SORTED} = \cup_{n\in\mathbb{N}}\texttt{SORTED}_n$. Although $\texttt{SORTED}_n$ is not invariant under any non-trivial permutation of $[n]$, we present an $O(1/\epsilon)$-time approximate decision procedure for it.

[6]When generalizing the result to the finite alphabet $\Sigma = \{0,1,..,t\}$, we consider the set (of frequency patterns) $F_n \subseteq (\{0,1,...,n\})^t$ such that for every $x \in \Sigma^n$ it holds that $x \in S$ if and only if $(\text{wt}_1(x), ..., \text{wt}_t(x)) \in F_n$, where $\text{wt}_j(x) = |\{i \in [n] : x_i = j\}$. The generalized tester will approximate each $\text{wt}_i(x)$ up to a deviation of $\epsilon/2t$.

Proposition 5 (a fast approximate decision procedure for SORTED): *There exists a randomized $O(\epsilon^{-1})$-time algorithm, that decides whether a given string is in* SORTED *or is ϵ-far from* SORTED*.*

(The set SORTED can be defined with respect to sequences over any set that is equipped with a total order. This generalization will be considered in a subsequent lecture. We note that the algorithm presented next does not extend to this general case; see [21, 24].)

Proof: On input $x \in \{0,1\}^n$, the algorithm proceeds as follows.

1. For $F = \{i\epsilon n/2 : i \in [2/\epsilon]\}$, query x at each $j \in F$.

   Let us denoted the retrieved $|F|$-bit long substring by y; that is, $y = x_{\epsilon n/2}x_{\epsilon n}x_{3\epsilon n/2}\cdots x_n$.
2. Queries x at $m = O(1/\epsilon)$ uniformly and independently distributed locations, denoted $i_1, ..., i_m$.
3. Accept if and only if the induced substring is sorted; that is, let $m' = |F| + m$ and $j_1 \leq j_2 \leq \cdots \leq j_{m'}$ such that $\{j_1, ..., j_{m'}\} = F \cup \{i_k : k \in [m]\}$ (as multi-sets), and accept if and only if $x_{j_k} \leq x_{j_{k+1}}$ for every $k \in [m' - 1]$.

This algorithm always accepts any $x \in$ SORTED, since any substring of such a sorted x is also sorted. Now, suppose that x is ϵ-far from SORTED$_n$. We consider two cases:

Case 1: the $|F|$-bit long substring y retrieved in Step 1 is not sorted. In this case, the algorithm rejects in Step 3, regardless of the values retrieved in Step 2.

Case 2: the $|F|$-bit long substring y retrieved in Step 1 is sorted. In this case, sorted strings that are "consistent" with the $|F|$-bit long substring $y = x_{\epsilon n/2}x_{\epsilon n}x_{3\epsilon n/2}\cdots x_n$ retrieved in Step 1 are determined up to the assignment in a single interval of the form $[t\epsilon n/2, (t+1)\epsilon n/2]$, where t is such that $y = 0^t1^{|F|-t}$. But since x is ϵ-far from SORTED, with high probability, Step 2 chooses a location on which x differs from the determined value (i.e., the value as determined for sorted strings). Details follow.

As stated above, in this case, there exists a $t \in \{0, 1, ..., |F|\}$ such that $y = 0^t1^{|F|-t}$; that is, $x_{i\epsilon/2} = 0$ if $i \in [t]$ and $x_{i\epsilon/2} = 1$ if $i \in [t+1, |F|]$. We say that a position $j \in [n]$ is violating if either $j \in [t\epsilon n/2]$ and $x_j = 1$ or $j \in [(t+1)\epsilon n/2, n]$ and $x_j = 0$. Note that any violating position j (which is not in $[t\epsilon n/2, (t+1)\epsilon n/2]$) causes rejection at Step 3 (if chosen in Step 2).[7] On the other hand, there must be at least $\epsilon n/2$ violating positions, since otherwise x is ϵ-close to SORTED$_n$ (because we can make x sorted by modifying it at positions $V \cup [t\epsilon n/2, (t+1)\epsilon n/2]$, where V denotes the set of violating positions). It follows that, in this case, the algorithm rejects with probability at least $1 - (1 - \epsilon/2)^m > 2/3$.

The proposition follows. ■

[7]In the first case (i.e., $j \in [t\epsilon n/2]$ and $x_j = 1$) rejection is caused since $x_{t\epsilon n/2} = 0$, whereas in the second case (i.e., $j \in [(t+1)\epsilon n/2, n]$ and $x_j = 0$) rejection is caused since $x_{(t+1)\epsilon n/2} = 1$.

Invariances (or symmetries) versus asymmetries. The properties `MAJ` and `SORTED` reside on opposite extremes of the invariance-vs-asymmetry axis: `MAJ` is invariant under each permutation of the domain $[n]$, which is totally symmetric w.r.t `MAJ`, whereas `SORTED` refers to the domain $[n]$ as a totally ordered set (and admit no invariance except the trivial one). In subsequent lectures, we shall see properties that are invariant under some non-trivial permutations but not under all permutations; that is, these properties are invariant under a non-trivial subgroup of the group of all permutations of $[n]$. Examples include low-degree multi-variate polynomials (which are invariant under affine transformations of the domain) and graph properties (which are invariant under any relabelling of vertex names).[8]

Indeed, while the role of symmetries in property testing is often highlighted (see, e.g., [60, 30]), here we call attention to the fact that the focus of property testing is on properties that are not fully symmetric (i.e., are somewhat asymmetric). In these cases, the testers and their analyses do not reduce to estimating average values via a uniformly distributed sample.

2.5 Objects and representation

So far we have referred to the instances of the (approximate) decision problems as abstract inputs, but it is time to recall that the instances are actually objects and that the inputs represent their description. The distinction between objects and their representation is typically blurred in computer science; nevertheless, this distinction is important. Indeed, reasonable and/or natural representations are always assumed either explicitly or implicitly (see, e.g., [27, Sec. 1.2.1]).[9] The specific choice of a reasonable and/or natural representation becomes crucial when one considers the exact complexity of algorithms (as is common in algorithmic research), rather than their general "ball park" (e.g., being in the complexity class $\mathcal{P}$ or not).

The representation is even more crucial in our context (i.e., in the study of property testing). This is the case for two reasons, which transcend the standard algorithmic concerns:[10]

1. We shall be interested in sub-linear time algorithms, which means that these algorithms query bits in the representation of the object. Needless to say, different representations mean different types of queries, and this difference is crucial when one does not fully recover the object by queries.

2. We shall be interested in the distance between objects (or, actually, in the distance between objects and sets of objects), whereas this distance will be measured in terms of the distance between their representations. In such a case, different representations of objects may yield vastly different distances between the same objects.

[8]For advanced studies of the role of invariances in property testing see [60] and [30], focusing on algebraic and combinatorial properties, respectively.

[9]For example, the computational problem that underlies the RSA cryptosystem is phrases as follows: Given integers N, e and y, find x such that $y \equiv x^e \pmod{N}$. This computational problem is believed to be hard when each of these integers is presented by its binary expansion, but it is easy when N is presented by its prime factorization. Likewise, factoring integers is believed to be hard when the integer is presented by its binary expansion, but it is easy when the integer is presented in unary (since this allows the solver to run in time that is exponential in the binary representation of the integer). Indeed, these two examples refer to unnatural representations of the inputs.

[10]One obvious concern, which is common to standard algorithmic research, is that one should either state complexities in terms of parameters of the object (e.g., the number of vertices and/or edges in a graph) or disallow overly redundant representations when complexities are stated in terms of the representation length.

To illustrate these concerns, suppose that the objects are represented by applying a good error correcting code to their standard representation, where a good error correcting code is one of constant relative distance and constant rate.[11] (This is indeed a contrived representation, and it is merely used to illustrate the above concerns.) Assuming that the code is efficiently decodable (even just in the case when no error occurs), the difference between this representation and the standard representation will have almost no impact on standard algorithmic research. In contrast, in such a case, the representation of every two objects will be far apart, which means that approximate decision (w.r.t this representation) will collapse to exact decision (w.r.t the objects). On the other hand, it may be impossible to recover single bits in the original representation by probing the codeword at a sub-linear number of locations.

In light of the above, when considering property testing, we always detail the exact representation of the objects. This representation will be presented either as a sequence, where the queries correspond to locations in the sequence, or as a function with queries corresponding to the elements in its domain. These two presentations are clearly equivalent via the obvious correspondence between sequences and functions (i.e., $x = (x_1, ..., x_n) \in \Sigma^n$ correspond to $f : [n] \rightarrow \Sigma$ such that $x_i = f(i)$ for every $i \in [n]$).

The choice of which presentation to use is determined either by the natural way we think of the corresponding objects or by mere technical convenience. For example, when discussing m-variate polynomials over a finite field $\mathcal{F}$, it is natural to present them as functions from $\mathcal{F}^m$ to $\mathcal{F}$. On the other hand, when discussing the set of strings that are accepted by a fixed finite-state automaton, it is natural to present them as sequences over $\{0, 1\}$.

The presentation of the object as a function is particularly appealing when the object has a concise implicit representation as a device (say a Boolean or Arithmetic circuit) that can be invoked on inputs of one's choice. Actually, in such a case, the function is the explicit representation of the object. On the other hand, when the object is a huge database that one can query at will, both presentation seem equally appealing.

3 Notions, definitions, goals, and basic observations

Following the property testing literature, we shall refer to approximate decision algorithms by the name "(property) testers". Likewise, we shall talk about "properties" rather than about sets. That is, a tester for property Π is an approximate decision algorithm for the set Π, where in both cases we refer to the same notion of distance and to the same proximity parameter ϵ.

The basic notions, definitions, and goals of property testing will be presented in Section 3.1, and will be used extensively throughout the entire lecture series (with very few exceptions). In contrast, the ramifications discussed in Section 3.2 will be lightly used (if at all), and ditto for the general observations made in Sections 3.4 and 3.5. In Section 3.3, we shall present another notion that will be used quite a lot – that of a proximity-oblivious tester (POT).

3.1 Basics

The properties that we shall focus on are properties of functions, which represent objects that we can probe by querying the function on the corresponding points. The size of the domain of these

[11] An error correcting code of relative distance $\gamma \in (0, 1]$ and rate $\rho \in (0, 1]$ is a mapping $C : \{0, 1\}^{\rho n} \rightarrow \{0, 1\}^n$ such that every two images of C are at relative Hamming distance at least γ.

functions is a parameter, denoted n. Without loss of generality, we consider the domain $[n]$. The range may also depend on n, and is denoted R_n. Hence, a property is a set Π_n of functions from $[n]$ to R_n.

The inputs. For sake of algorithmic uniformity and asymptotic analysis[12], we let n vary, and consider testers for $\Pi = \cup_{n\in\mathbb{N}}\Pi_n$, while providing these testers with the parameter n as explicit input (i.e., the tester can read this input at no cost). Hence, the same algorithm (tester) is used for all values of n. The *main input* of the tester is a function $f : [n] \to R_n$, which is viewed as an oracle to which the tester has a query access. Another explicit input that is given to the tester is the *proximity parameter*, denoted ϵ. Indeed, this means that the tester is uniform across all possible values of $\epsilon > 0$.

These conventions make positive results more useful; when presenting (query complexity) lower bounds, we will typically consider also non-uniform algorithms that may depend arbitrarily on n and ϵ. Indeed, in exceptional cases, typically in lower bounds, we may consider testers that operate only for some fixed value of the proximity parameter ϵ. We shall refer to such testers as ϵ-testers.

In some other cases, for the sake of usefulness, we may desire testers that are uniform across other parameters of the property at stake. For example, when considering properties of graphs of bounded degree, we should provide the tester with this degree bound. Likewise, when we consider properties of functions defined over a finite field, we should provide the tester with a representation of this finite field. (A more acute need for providing such an auxiliary input arises in the context of massively parametrized properties.)[13]

Distance. In accordance with the discussion in Section 2, we consider the relative distance between functions, denoted δ. Specifically, for $f, g : [n] \to R_n$, we let $\delta(f, g) = |\{i \in [n] : f(i) \neq g(i)\}|/n$; that is,

$$\delta(f, g) \stackrel{\rm def}{=} \mathbf{Pr}_{i\in[n]}[f(i) \neq g(i)], \tag{1}$$

where i is uniformly distributed in $[n]$. For $f : [n] \to R_n$ and $\Pi = \cup_{n\in\mathbb{N}}\Pi_n$ such that Π_n contains functions defined over $[n]$, we let $\delta_\Pi(f)$ denote the distance of f from Π_n; that is,

$$\delta_\Pi(f) \stackrel{\rm def}{=} \min_{g\in\Pi_n}\{\delta(f, g)\} \tag{2}$$

where $\delta_\Pi(f) \stackrel{\rm def}{=} \infty$ if $\Pi_n = \emptyset$.

Oracle machines. We model the testers as probabilistic oracle machines that access their main input via queries. Hence, the output of such a machine, denoted T, when given explicit inputs n and ϵ, and oracle access to $f : [n] \to R_n$, is a random variable, denoted $T^f(n, \epsilon)$. Indeed, for sake of simplicity, we consider the basic case in which no auxiliary inputs of the aforementioned type

[12]By *algorithmic uniformity* we mean presenting a single algorithm for all instances of the problem, rather than presenting a different algorithm per each value of the size parameter n (and/or the proximity parameter ϵ). By *asymptotic analysis* we mean a functional presentation of complexity measures in terms of the size and proximity parameters, while ignoring constant multiplicative factors (i.e., using the $O(\cdot)$ and $\Omega(\cdot)$ notation).

[13]For example, one may consider the property of being isomorphic to an explicitly given graph, which is viewed as a parameter. In this case, one graph is a parameter, while another graph (of the same size) is considered the main input: the tester is given free access to the parameter, and oracle access (for which it is charged) to the main input. See survey on massively parametrized problems [50].

are given. We shall associate the output 1 (resp., 0) with the decision to accept (resp., reject) the main input. We are now ready to present the main definition of property testing.

Definition 6 (a tester for property Π): *Let $\Pi = \cup_{n\in\mathbb{N}}\Pi_n$ such that Π_n contains functions defined over $[n]$. A* tester for Π *is a probabilistic oracle 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 $\mathbf{Pr}[T^f(n,\epsilon)=1] \geq 2/3$.*
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 $\mathbf{Pr}[T^f(n,\epsilon)=0] \geq 2/3$.*

If the first condition holds with probability 1 (i.e., $\mathbf{Pr}[T^f(n,\epsilon) = 1] = 1$)*, then we say that T has* one-sided error*; otherwise, we say that T has* two-sided error*.*

Indeed, the *error probability* of the tester is bounded by 1/3. As with the definition of $\mathcal{BPP}$ (and co$\mathcal{RP}$), the choice of the error bound is rather arbitrary as long as it is a constant smaller than 1/2; the error can be decreased by repeated application of the tester (while ruling by majority; see Exercise 3). Specifically, by using t repetitions, the error can be reduced to $\exp(-\Omega(t))$.

Focus: query complexity. Our main focus will be on the *query complexity* of the tester, when considered as a function of n and ϵ: We say that the tester 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 tester makes at most $q(n,\epsilon)$ queries. Clearly, any property can be tested in query complexity $q(n,\epsilon) = n$. The first priority is to have the query complexity be sub-linear in n, and the slower it grows with n, the better. The ultimate goal, which is not always achievablele, is to have the query complexity be independent of n. We shall also care about the dependence of the query complexity on ϵ, and in particular whether it is $O(1/\epsilon)$, $\mathrm{poly}(1/\epsilon)$ or worse.

The *time complexity* of the tester will be our secondary focus, although it is obviously important. We shall say that a tester is efficient if its time complexity is almost linear in its query complexity. As in algorithmic research, we allow standard manipulation of the symbols (i.e., elements of R_n) and addresses (i.e., elements of $[n]$) at unit costs; for example, uniformly selecting $i \in [n]$ or comparing $f(i)$ to $v \in R_n$ are considered as performed in a single computational step.

Illustrating the foregoing terminology. Let us rephrase some of the results presented in Section 2 using the forgoing terminology.

- Proposition 1: There exists a $O(1/\epsilon^2)$-time tester for `MAJ`.
- Proposition 3: Every deterministic tester (or even a deterministic 0.5-tester) for `MAJ` has query complexity $\Omega(n)$.
- Theorem 4: Every symmetric property of Boolean functions can be tested in query complexity $O(1/\epsilon^2)$.
- Proposition 5: There exists a $O(1/\epsilon)$-time one-sided error tester for `SORTED`.

Non-adaptivity. We also distinguish between adaptive and non-adaptive testers, where a tester is called non-adaptive if it determines all its queries based on its explicit inputs and internal coin tosses, independently of the specific function to which it is given oracle access. In contrast, an adaptive tester may determine it $i+1^{\rm st}$ query based on the answers it has received to the prior i queries. Note the all the aforementioned testers are non-adaptive.

A non-adaptive tester T can be de-composed into two modules, denoted Q and D, such that Q uses the randomness r of T in order to generate queries $i_1,...,i_q$, whereas D decides according to r and the answers obtained; that is, $T^f(n,\epsilon;r) = D(r,f(i_1),...,f(i_q))$, where $(i_1,...,i_q) \leftarrow Q(n,\epsilon;r)$.

Global versus local. Property testers of "low" query complexity give rise to a global-versus-local phenomenon. In this case, a global property of the function $f:[n]\to R_n$ (i.e., its belonging to Π_n) is reflected by its "local behavior" (i.e., the pattern seen in the portion of f that is inspected by the tester (as determined by the random outcome of its coin tosses)).[14] This perspective is more appealing when the tester is non-adaptive. In this case, and using the foregoing de-coupling, the "local behavior" refers to the values of f on the points in the sequence $(i_1,...,i_q) \leftarrow Q(n,\epsilon;r)$, where each choice of r corresponds to a different portion of $[n]$. This perspective is even more appealing when the tester has one-sided error and its final decision (i.e., $D(r,f(i_1),...,f(i_q))$) only depends on the answers obtained (i.e., is independent of r).

Common abuses. When describing specific testers, we often neglect to mention their explicit inputs (i.e., n and ϵ), which are always clear from the context. Likewise, even when we discuss auxiliary parameters (like degree bounds for graphs or descriptions of finite fields), we neglect to write them as explicit inputs of the tester. Finally, for simplicity, we often use Π rather than Π_n, even when we refer to inputs of length n. Likewise, we may define Π as a finite set consisting of functions over a fixed domain $[n]$ and a fixed range. In any case, unless explicitly stated differently, the value of the size parameter n has to be thought of as generic (and ditto for the value of the proximity parameter ϵ).

3.2 Ramifications

Recall that distance between functions (having the same domain $[n]$) was defined in Eq. (1) as the probability that they 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

$$\delta_{\mathcal{D}_n}(f,g) \stackrel{\rm def}{=} \mathbf{Pr}_{i\sim\mathcal{D}_n}[f(i)\neq g(i)], \tag{3}$$

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 6 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 provides it with samples drawn according to the distribution in question (i.e., the distribution $\mathcal{D}_n$

[14]Indeed, the portions of $[n]$ observed when considering all possible outcomes of the tester's randomness are likely to cover $[n]$, but the tester's decision on a specific outcome depends only on the function's values on the corresponding portion of $[n]$.

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 $\mathcal{D}_n$).[15] In such a case, one may consider both the tester's query complexity and its sample complexity.[16]

The aforementioned use of samples raises the question of what can be done, even with respect to the uniform distribution, when the tester only obtains samples; that is, when it cannot query the function on points of its choice. We call such a tester sample-based, and clarify that, when testing a function $f : [n] \to R_n$, this 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 $[n]$. As we shall see in subsequent lectures, 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 it is desirable in many applications where obtaining samples is far more feasible than obtaining answers to queries of one's choice. (An extensive study of sample-based testers was initiated by Goldreich and Ron [36].)

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. Note that the formalism presented so far does not apply to this notion, and a different formalism will be used. (The study of testing properties of distributions was initiated by Batu *et al.* [8], and we shall devote a subsequent lecture to it).

Tolerant testing. Getting back to the basic framework of Definition 6 and recalling some of the settings discussed in Section 1.2, we note that a natural generalization of testing refers to 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" small deviation of the object from having the property.[17] (The study of tolerant testing properties of functions was initiated by Parnas, Ron, and Rubinfeld [51].)

[15]Specifically, a distribution-free tester for Π is a probabilistic oracle machine, denoted T, such that for every $n \in \mathbb{N}$ 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 access to samples from $\mathcal{D}_n$ (and explicit inputs n and ϵ).

[16]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 [7].

[17]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 parameter) of the distance of the object to the property. Typically, the term "tolerant testing" is used when the parameter ϵ' is a fixed function of ϵ (e.g., $\epsilon' = \epsilon/2$), 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).

Other distance measures. As stated above, almost all research in property testing focuses on the distance measure defined in Eq. (1), which corresponds to the relative Hamming distance between sequences. We already saw a deviation from this archetypical case in Eq. (3). Different distance measures, which are natural in some settings, include the edit distance and the $\mathcal{L}_1$-distance. (The study of property testing with respect to the edit distance was initiated by Batu *et al.* [9], and a study of testing with respect to the $\mathcal{L}_p$-distance was initiated by Berman, Raskhodnikova, and Yaroslavtsev [13].)

3.3 Proximity-oblivious testers (POTs)

How does the tester use the proximity parameter?

In the examples presented in Section 2, the proximity parameter was used to determine the number of queries. These testers were quite simple, and it was not clear how to decompose them to repetitions of even simpler testers. Nevertheless, as we shall see in subsequent lectures, in many cases such a decomposing is conceptually helpful. Furthermore, in many cases the basic testers do not use the proximity parameter at all (see Proposition 8), but are rather repeated for a number of times that depend on this parameter. This leads to the notion of a proximity-oblivious tester (POT), which is defined next.

Below, we present a general notion of a POT, which allows two-sided error probability. The notion of a one-sided error POT, obtained as a special case by setting the threshold $\tau = 1$, emerges much more naturally in applications in the sense that one may actually first think of a POT and then get to a standard tester (as per Definition 6) from this POT. In contrast, two-sided error POTs seem quite contrived, and may be viewed as an afterthought or as an exercise. Indeed, the reader may want to focus on the one-sided error version (and just substitute τ by 1), at least in first reading.

We stress that the POT does not obtain a proximity parameter as input, but its rejection probability is a function of the distance of the tested object from the property. In the case of the one-sided error version, the rejection probability is zero when the object has the property, and it increases with the distance of the object from the property. In the case of the two-sided error version, the rejection probability is at most $1-\tau$ when the object has the property, and it increases above $1-\tau$ with the distance of the object from the property. In the following definition, we refer to the acceptance probability of the POT, and state that it decreases below τ as a function of the distance of the object from the property. The latter function is denoted ϱ.

Definition 7 (Proximity Oblivious Testers): *Let* $\Pi = \cup_{n\in\mathbb{N}}\Pi_n$ *such that* Π_n *contains functions defined over* $[n]$ (as in Definition 6). *Let* $\tau \in (0,1]$ *be a constant and* $\varrho : (0,1] \to (0,1]$ *be monotonically non-decreasing. A* proximity-oblivious tester (POT) with threshold probability τ and detection probability ϱ for Π *is a probabilistic oracle machine, denoted T, that satisfies the following two conditions.*

1. T accepts inputs in Π with probability at least τ: *For every $n \in \mathbb{N}$ and every $f \in \Pi_n$, it holds that* $\mathbf{Pr}[T^f(n)=1] \geq \tau$.

2. T accepts inputs that not in Π with probability that decreases below τ as a function of their distance from Π: *For every $n \in \mathbb{N}$ and every $f : [n] \to R_n$ that is not in Π, it holds that* $\mathbf{Pr}[T^f(n)=1] \leq \tau - \varrho(\delta_\Pi(f))$.

When $\tau = 1$, we say that T has one-sided error.

Hence, $f \notin \Pi$ is rejected with probability at least $(1-\tau) + \varrho(\delta_\Pi(f))$. The postulate that ϱ is monotonically non-decreasing means that *any function that is ϵ-far from Π is rejected with probability at least* $(1-\tau)+\varrho(\epsilon)$; that is, if $\delta_\Pi(f) > \epsilon$ (and not only if $\delta_\Pi(f) = \epsilon$), then f is rejected with probability at least $(1-\tau)+\varrho(\epsilon)$.

Note that if $\tau < 1$, then it is not necessarily the case that all $f \in \Pi$ are accepted with the same probability (e.g., τ); it may be that some are accepted with probability τ and others are accepted with probability greater than τ (see [38]). We also mentioned that Definition 7 can be generalized such that both the detection probability function ϱ and the threshold τ depend on n, but we shall not use this generalization. Before making any additional comments, let us see an example of a proximity-oblivious tester.

Proposition 8 (a two-query POT for `SORTED`): *There exists a* (one-sided error) *proximity oblivious tester for* `SORTED` *using two queries and having quadratic detection probability, where* `SORTED` *is as in Proposition 5.*

Proof: (Here it will be more convenient to view the input as a bit string.) On input n and oracle access to $x \in \{0,1\}^n$, the tester selects uniformly $i,j \in [n]$ such that $i < j$ and accepts if and only if $x_i \le x_j$. To lower-bound the detection probability of this tester, we consider an arbitrary $x \in \{0,1\}^n$ at distance $\delta_S(x) > 0$ from $S \stackrel{\rm def}{=} \texttt{SORTED}$.

Observing that $x' = 0^{n-{\rm wt}(x)}1^{{\rm wt}(x)}$ is the sorted version of x, consider the disagreements between x and x'. Specifically, let D_1 denote the set of locations in the $(n-{\rm wt}(x))$-bit long prefix of x that hold 1's (i.e., $D_1 = \{i \in [n-{\rm wt}(x)] : x_i = 1\}$). Likewise, denote the set of locations in the ${\rm wt}(x)$-bit long suffix of x that hold 0's by D_0 (i.e., $D_0 = \{i \in [n-{\rm wt}(x)+1, n] : x_i = 0\}$), and note that $|D_0| = |D_1|$. Now, on the one hand, $\delta_S(x) \le \delta(x,x') = (|D_1|+|D_0|)/n$, which implies that $d \stackrel{\rm def}{=} |D_1| \ge \delta_S(x) \cdot n/2$.[18] On the other hand, for each pair $(i,j) \in D_1 \times D_0$ it holds that $1 \le i < j \le n$ and $x_i = 1 > 0 = x_j$. Hence,

$$\begin{aligned}
\mathbf{Pr}_{i,j\in[n]:i<j}[x_i > x_j] &\ge \frac{|D_1|\cdot|D_0|}{\binom{n}{2}} \\
&= \frac{d^2}{n(n-1)/2} \\
&> \frac{(\delta_S(x)\cdot n/2)^2}{n^2/2} \\
&= 0.5\cdot\delta(x)^2.
\end{aligned}$$

The proposition follows. ■

General observations about POTs. The query complexity of POTs is stated as a function of n, and at times – as in Proposition 8 – their query complexity will be a constant (independent

[18]The fact that $\delta_S(x) \ge d/n$ is not used here. This fact can be established by considering a matching between D_1 and D_0, and observing that any string in S must differ from x on at least one endpoint of each pair in the mattching. We also mention that this lower bound is tight (e.g., consider the case of $x = 0^{n-w-d}1^d0^d1^{w-d}$).

of n). All POTs we shall see in these lectures have one-sided error.[19] Nevertheless, we observe that *the threshold constant in two-sided error POTs is immaterial*, as long as it is not one: See Exercise 4. More importantly, we show how to derive standard testers out of POTs, while noting that for one-sided testers (resp., two-sided error) the resulting tester has query complexity that is linear (resp., quadratic) in $1/\varrho$:

Theorem 9 (deriving standard testers from POTs):

1. *If* Π *has a one-sided error POT of query complexity* q *with detection probability* ϱ*, then* Π *has a one-sided error tester of query complexity* $q' = O(q/\varrho)$*.*

2. *If* Π *has a POT of query complexity* q *with threshold probability* $\tau \in (0,1)$ *and detection probability* ϱ*, then* Π *has a tester of query complexity* $q' = O(q/\varrho^2)$*.*

The time complexity of the derived tester relates to that of the POT in a similar manner. If the POT is non-adaptive, then so is the derived tester.

Proof: On input proximity parameter $\epsilon > 0$, the standard tester invokes the POT for a number of times that depends on $\varrho(\epsilon)$. Specifically, if the POT has one-sided error, then the tester invokes it for $O(1/\varrho(\epsilon))$ times and accepts if and only if all invocations accepted. If the POT has threshold probability $\tau \in (0,1)$, then the tester invokes it for $O(1/\varrho(\epsilon)^2)$ times and accepts if and only if at least a $\tau - 0.5 \cdot \varrho(\epsilon)$ fraction of the invocations accepted. The analysis of this tester reduces to observing that a function that is ϵ-far from Π is accepted by the POT with probability at most $\tau - \varrho(\epsilon)$ (whereas any function in Π is accepted with probability at least τ).[20] $\blacksquare$

Discussion. Note that the standard tester for `SORTED` obtained from the POT of Proposition 8 (by applying Theorem 9) is inferior to the tester of Proposition 5. This is not a singular case (see [35]). Furthermore, some properties have good testers, but do not have good POTs; that is, there exist natural properties that have poly$(1/\epsilon)$-time testers but have no constant-query POTs.[21]

3.4 The algebra of property testing

In this section we show that natural classes of testable properties are closed under union but not under intersection (and complementation). That is, if Π' and Π'' are testable within some complexity bounds, then so is $\Pi' \cup \Pi''$ (up to a constant factor), but $\Pi' \cap \Pi''$ may be much harder to test.[22]

[19] Rather contrived exceptions are presented in Exercises 5 and 6.

[20] The first assertion relies on the postulation that ϱ is monotonically non-decreasing. This postulate implies that any function that is ϵ-far from Π (rather than only functions that are at distance exactly ϵ from Π) is rejected with probability at least $(1-\tau) + \varrho(\epsilon)$.

[21] One such example is of testing bipartitness in the dense graph model: See [29] and [35], respectively. We would certainly welcome a simpler example.

[22] This is a general result that refers to all possible Π' and Π''. In contrast, in some cases, both $\Pi' \cup \Pi''$ and $\Pi' \cap \Pi''$ may be much easier to test than Π' and Π''.

Unions. The basic idea is that if Π' and Π'' are testable by algorithms T' and T'', respectively, then one may test $\Pi' \cup \Pi''$ by invoking both testers and accepting if and only if at least one of these invocations accepted. This procedure doubles the error probability of ordinary testers and squares the detection probability of one-sided error POTs, so in the former case we should first apply error reduction.

Theorem 10 (testing the union of properties):

1. *If Π' and Π'' are each testable within query complexity q, then $\Pi' \cup \Pi''$ is testable within query complexity $O(q)$. Furthermore, one-sided error testing is preserved.*
2. *Suppose that Π' has a q-query one-sided error POT with detection probability $\varrho : (0,1] \to (0,1]$, and ditto for Π''. Then, $\Pi' \cup \Pi''$ has a $2q$-query one-sided error POT with detection probability ϱ^2.*

Furthermore, the time complexity is preserved up to a constant factor.

(Indeed, it is unclear how to handle the case of two-sided error POTs. The issue is that, in this case, inputs in $\Pi' \setminus \Pi''$ may be rejected by the foregoing procedure with probability $1 - \tau$, where τ is the threshold probability, whereas inputs that are at distance δ from $\Pi' \cup \Pi''$ may be rejected with probability $(1 - \tau + \varrho(\delta))^2$. But for $\tau < 1$ and sufficiently small $\delta > 0$, it holds that $1-\tau > (1-\tau+\varrho(\delta))^2$. In contrast, for $\tau = 1$ and any $\delta > 0$, it holds that $1-\tau = 0 < (1-\tau+\varrho(\delta))^2$.)

Proof: By the hypothesis of Part 1, each of the two properties is testable within query complexity q, but these testers may have error probability as large as $1/3$. Hence, we first obtain corresponding testers of error probability at most $1/6$, by invoking each of the original testers for a constant number of times. Combining the two resulting testers, as described above, we establish Part 1.

Turning to Part 2, we note that no error reduction is needed here; the claimed result follows by merely lower-bounding the probability that both testers reject when the input is at distance δ from $\Pi' \cup \Pi''$. ■

Hardness for testing. The negative results regarding intersection and complementation rely on properties that are hard to test. For concreteness we start by presenting one such hardness result.

Proposition 11 (hardness of testing membership in a linear code): *Let G be a $0.5n$-by-n Boolean matrix in which every $0.05n$ columns are linearly independent and every non-empty linear combination of the rows has Hamming weight at least $0.1n > 1$. Let $\Pi = \{xG : x \in \{0,1\}^{0.5n}\}$ be the linear code generated by G. Then, 0.1-testing Π requires more than $0.05n$ queries.*

The existence of a matrix G that satisfies the hypothesis can be proved using the probabilistic method (see Exercise 8). The proof of Proposition 11 only uses the fact that every $0.05n$ columns are linearly independent (and that there are $0.5n$ rows). The fact that the code has distance greater than one will be used later (when using Π to show the negative result regarding complementation).

Teaching note: See teaching note following proposition 2. In short, some readers may prefer to skip the following proof, and return to it at a latter time (e.g., after the lecture on lower bounds).

Proof: We shall use the following two observations.

Observation 1: An algorithm that makes $0.05n$ queries cannot distinguish the case that its input is uniformly distributed in Π from the case that its input is uniformly distributed in $\{0,1\}^n$.

This is the case since, for a uniformly distributed $x \in \{0,1\}^{0.5n}$, each $0.05n$-bit long subsequence of xG is uniformly distributed in $\{0,1\}^{0.05}$ (see Exercise 9).

Observation 2: All but $2^{0.99n}$ of the strings in $\{0,1\}^n$ are 0.1-far from Π (see Exercise 10).

This follows by straightforward counting.

Observation 2 implies that a 0.1-tester must reject a uniformly distributed n-bit string with probability at least $(1-2^{-0.01n}) \cdot 2/3 > 0.6$. On the other hand, such a tester must acceptt any string in Π with probability at least $2/3$ (and so reject with probability at most $1/3$). But Observation 1 asserts that no gap in the rejection probability is possible when making at most $0.05n$ queries. ■

Intersection (and complementation). Proposition 11 yields an example in which the complexity of testing a property is vastly different from the complexity of testing its complement. Specifically, consider the property $\Pi' = \{0,1\}^n \setminus \Pi$, where Π is as in Proposition 11. Note that every n-bit string is $1/n$-close to Π' (since each string in Π is surrounded by strings in Π'); hence, testing Π' is trivial (i.e., if $\epsilon \geq 1/n$, then we may accept the input without examining it at all, and otherwise reading the entire input means making at most $1/\epsilon$ queries). But, testing the complement of Π' (i.e., Π) is utmost hard (i.e., requires a linear number of queries)!

A small twist on the foregoing argument allows proving the following result.

Theorem 12 (testing the intersection of properties): *There exist Π' and Π'' such that the following holds:*

1. *Each of these two properties is testable within query complexity $1/\epsilon$; actually, they are each testable with query complexity q such that $q(n,\epsilon) = 0$ if $\epsilon \geq 1/n$ and $q(n,\epsilon) = n$ otherwise.*
2. *Testing $\Pi' \cap \Pi''$ requires a linear number of queries; actually, 0.1-testing $\Pi' \cap \Pi''$ requires more than $0.05n$ queries.*

Proof: Starting with Π as in Proposition 11, we consider $\Pi' = \Pi \cup \{0x' : x' \in \{0,1\}^{n-1}\}$ and $\Pi'' = \Pi \cup \{1x' : x' \in \{0,1\}^{n-1}\}$. Part 1 follows from the fact that every n-bit string is $1/n$-close to Π' (resp., Π''). Part 2 follows from the fact that $\Pi' \cap \Pi'' = \Pi$. ■

The case of monotone properties. We say that $\Pi \subseteq \{0,1\}^*$ is monotone if for every $x \in \Pi$ and $w \in \{0,1\}^*$ it holds that $x \vee w = (x_1 \vee w_1, ..., x_n \vee w_n)$ is in Π; that is, Π is preserved under resetting some bits to 1. We first note that the discrepancy between the complexity of testing a property and the complexity of testing its complement is maintained also for monotone properties (see Exercise 11). More importantly, in contrast to Theorem 12, we have

Theorem 13 (testing the intersection of monotone properties): *Let Π' and Π'' be monotone properties.*

1. *If Π' and Π'' are testable within query complexity q' and q'', respectively, then, for every $\epsilon' \in (0,\epsilon)$, the property $\Pi' \cap \Pi''$ is ϵ-testable within query complexity $q(n,\epsilon) = O(q'(n,\epsilon') + q''(n,\epsilon-\epsilon'))$. Furthermore, one-sided error testing is preserved.*

2. *Suppose that Π' has a q-query one-sided error POT with detection probability $\varrho : (0,1] \to (0,1]$, and ditto for Π''. Then, $\Pi' \cap \Pi''$ has a $2q$-query one-sided error POT with detection probability $\varrho'(\delta) = \varrho(\delta/2)$.*

Furthermore, the time complexity is preserved up to a constant factor.

Theorem 13 is generalized and abstracted in Exercise 12. A totally different case in which testability is preserved under intersection is presented in the notes on testing dictatorship.

Proof: The basic idea is that if Π' and Π'' are testable by algorithms T' and T'', respectively, then one may test $\Pi' \cap \Pi''$ by invoking both testers and accepting if and only if both these invocations accepted.[23] While this procedure fails for general properties, we show that it works for monotone ones. The key observation is that if x is ϵ'-close to Π' and ϵ''-close to Π'', then it is $(\epsilon' + \epsilon'')$-close to $\Pi' \cap \Pi''$. This is shown next.

We first show that *if $\Pi \subseteq \{0,1\}^n$ is monotone, then, for every $x \in \{0,1\}^n$ there exists $w \in \{0,1\}^n$ such that $x \vee w \in \Pi$ and $\mathrm{wt}(w) \leq \delta_\Pi(x) \cdot n$.* This is shown by observing that (by definition) there exist $w_1, w_0 \in \{0,1\}^n$ such that $\mathrm{wt}(w_1) + \mathrm{wt}(w_0) = \delta_\Pi(x) \cdot n$ and $(x \vee w_1) \oplus w_0 \in \Pi$. Now, by monotonicity, $((x \vee w_1) \oplus w_0) \vee w_0) \in \Pi$, whereas $((x \vee w_1) \oplus w_0) \vee w_0) = x \vee w_1 \vee w_0$ (since $(\sigma \oplus \tau) \vee \tau = \sigma \vee \tau$ for any $\sigma, \tau \in \{0,1\}$). The proof is completed by letting $w = w_1 \vee w_0$.

We then show that, for every $x \in \{0,1\}^n$ it holds $\delta_{\Pi' \cap \Pi''}(x) \leq \delta_{\Pi'}(x) + \delta_{\Pi''}(x)$. Let w' (resp., w'') be such that $x \vee w' \in \Pi'$ and $\mathrm{wt}(w') \leq \delta_{\Pi'}(x) \cdot n$ (resp., $x \vee w'' \in \Pi''$ and $\mathrm{wt}(w'') \leq \delta_{\Pi''}(x) \cdot n$). Now, by using monotonicity again, we have $x \vee w' \vee w'' \in \Pi'$ and $x \vee w' \vee w'' \in \Pi''$, whereas $\mathrm{wt}(w' \vee w'') \leq \mathrm{wt}(w') + \mathrm{wt}(w'')$. The theorem follows. ■

Warning: subsets and supersets. Recall that the complexity of decision problems does not always decrease or increase when considering decision problems that correspond to subsets of the original set.[24] The same holds in the context of property testing: A trivial example can be obtained by starting with Π as in Proposition 11, and considering $\emptyset \subset \Pi \subset \{0,1\}^n$ (or $\{0^n\} \subset \Pi \subset \{0,1\}^n \setminus \{x\}$ for any $x \in \{0,1\}^n \setminus \Pi$).

3.5 Testing via learning

A general observation, which is rarely used, is that property testing reduces to learning. The reason we shall not use this observation is that we typically seek testers that are more efficient than the corresponding learners. Still, for sake of perspective, we detail the said connection.

To streamline the presentation, we use the terminology of Definition 6 in our definition of learning. (In the learning literature the set Π is called a concept class, the functions $f : [n] \to R_n$

[23] In the case of two-sided error tester, this is done after reducing the error probability to 1/6. In any case, for Part 1, the tester selects an appropriate ϵ', and invokes the ϵ'-tester for Π' and the $(\epsilon - \epsilon')$-tester for Π''.

[24] In contrast, a promise problem, $(S_{\mathrm{yes}}, S_{\mathrm{no}})$, never becomes harder (resp., easier) when taking subsets (resp., supersets) of both S_{yes} and S_{no}. The point is for $S' \subset S$, moving from the promise problem $(S, \{0,1\}^* \setminus S)$ to the promise problem $(S', \{0,1\}^* \setminus S')$ means moving the border between YES-instances and NO-instances, rather than omitting instances from the promise set (which equals $\{0,1\}^*$ in both cases).

are called concepts, one usually focuses on $R_n = \{0,1\}$ and $n = 2^\ell$, and views ℓ as the main parameter. More importantly, as in Definition 6 we provide the learner with oracle access to the function (rather than with labeled examples as is standard in the learning literature), focus on the uniform distribution (rather than on the distribution-free case), and fix the error probability to equal 1/3 (rather than using an additional parameter).)[25]

Definition 14 (learning and proper learning for Π, following [63] and [52]): *Let $\Pi = \cup_{n\in\mathbb{N}}\Pi_n$ such that Π_n contains functions defined over $[n]$. A* learner for Π *is a probabilistic oracle machine, denoted L, such that for every $n \in \mathbb{N}$ and every $f \in \Pi_n$, with probability at least $2/3$, it holds that $L^f(n,\epsilon)$ is a description of a function that is ϵ-close to f. If the output function always belongs to Π, then we say that L performs* proper learning.

Note that, in contrast to testing, nothing is required in case f is ϵ-far from Π (let alone when it only holds that $f \notin \Pi$). On the other hand, and again in contrast to testing, when $f \in \Pi$, the learner is required to output a function (called a hypothesis) that is ϵ-close to the target function f (and not only say "yes"). When considering the computational complexity of the learner, one typically requires that the learner outputs a concise representation of the function, and in case of proper learning this representation should fit the prescribed representation of functions in Π.

We note that every Π can be properly learned within query complexity $q(n,\epsilon) = \min(n, O(\epsilon^{-1}\log|\Pi_n|))$, where the second bound follows by an algorithm that scans all possible $h \in \Pi_n$ and uses the same sample of $O(\epsilon^{-1}\log|\Pi_n|)$ random points to estimate the distance between h and the target function f (see Exercise 13). Such an estimation procedure is pivotal for establishing the following result.

Theorem 15 (learning implies testing): *Let $\Pi = \cup_{n\in\mathbb{N}}\Pi_n$ be as in Definition 14, and suppose that Π can be learned within query complexity $q(n,\epsilon)$. Then, Π can be tested within query complexity $q'(n,\epsilon) = q(n, 0.3\epsilon) + O(1/\epsilon)$. Furthermore, if the learning algorithm is proper, runs in time $t(n,\epsilon)$ and outputs descriptions of functions such that, with respect to that representation, evaluating these functions and checking their membership in Π can be done in time $T(n)$, then Π can be tested within query complexity $q'(n,\epsilon) = q(n,0.7\epsilon)+O(1/\epsilon)$ and time complexity $t'(n,\epsilon) = t(n,0.7\epsilon)+O(T(n)/\epsilon)$.*

We mention that similar results hold with respect to a variety of models including sample-based learning and testing and distribution-free learning and testing. Note that in the case of non-proper learning we invoke the learner with a proximity parameter that is strictly smaller than $\epsilon/2$, whereas in the case of proper learning we may use a proximity parameter that is larger than $\epsilon/2$ (as long as it is strictly smaller than ϵ). More importantly, the stated bound on time complexity (i.e., $t'(n,\epsilon) = t(n,0.7\epsilon) + O(T(n)/\epsilon)$) does not hold in the case of non-proper learning (see [29, Sec. 3.2]).

Proof: On input f and proximity parameter ϵ, the tester proceeds as follows:

1. The tester invokes the non-proper (resp., proper) learner on f with proximity parameter 0.3ϵ (resp., 0.7ϵ), obtaining a description of a hypothesis $h : [n] \to R_n$.

[25] The last deviation from the standard presentation of learning algorithm weakens the definition, since error-reduction is not available in this context (akin the situation with randomized algorithms for search problems; cf. [27, Sec. 6.1.2]). Specifically, if we invoke the learner t times and obtain hypotheses $h_1,...,h_t$, then it is not clear how to combine them in order to obtain (with probability $1-\exp(-|Omega(t)))$ a function that is ϵ-close to the target function f.

2. The tester checks whether h is 0.3ϵ-close to Π_n (resp., is in Π_n):

   Case of non-proper learning: The tester checks whether h is 0.3ϵ-close to Π_n, and if the answer is negative it rejects.

   This step requires no access to f, but it may require going over all functions in Π_n and comparing each of them to h.

   (If $f \in \Pi_n$, then, with probability at least 2/3, it holds that h is 0.3ϵ-close to Π_n, but otherwise noting is guaranteed.)

   Case of proper learning: The tester checks whether $h \in \Pi_n$, and if the answer is negative it rejects.

   This step can be implemented in time $T(n)$, and requires no access to f.

   (If $f \in \Pi_n$, then, with probability at least 2/3, it holds that $h \in \Pi_n$ (since the learner is proper), but otherwise noting is guaranteed.)

   (If the tester did not reject, then it proceeds to the next step.)

3. The tester uses an auxiliary sample of $O(1/\epsilon)$ in order to estimate the distance between h and f up to an additive term of 0.1ϵ, with error probability 0.1.

   Case of non-proper learning: The tester accepts if and only if according to this estimate, the distance between h and f is at most 0.5ϵ.

   Case of proper learning: The tester accepts if and only if according to this estimate, the distance between h and f is at most 0.85ϵ.

   (In the case of proper learning, the estimate is performed in time $O(1/\epsilon) \cdot T(n)$.)

We start with the case of non-proper learning. If $f \in \Pi_n$, then, with probability at least 2/3, the hypothesis h is 0.3ϵ-close to f, and in this case, with probability at least 0.9, the tester will accept (since h is 0.3ϵ-close to Π_n and with high probability the estimated distance between h and f is at most 0.4ϵ). On the other hand, if f is ϵ-far from Π_n, then either h is 0.3ϵ-far from Π_n or h is 0.7ϵ-far from f, and in the latter case, with probability at least 0.9, the tester will reject (since in the later case and h is estimated to be 0.6ϵ-far from f).

We now turn to the case of proper learning. If $f \in \Pi_n$, then, with probability at least 2/3, the hypothesis $h \in \Pi_n$ is 0.7ϵ-close to f, and in this case, with probability at least 0.9, the tester will accept (since h is estimated to be 0.8ϵ-close to f). On the other hand, if f is ϵ-far from Π_n, then either $h \notin \Pi_n$ or $h \in \Pi_n$ is ϵ-far from f, and in the latter case, with probability at least 0.9, the tester will reject (since in the later case and h is estimated to be 0.9ϵ-far from f).

Hence, in both cases, the tester accepts any $f \in \Pi_n$ with probability at least $(2/3) \cdot 0.9 = 0.6$, and rejects any f that is ϵ-far from π with probability at least 0.9. By modifying the tester such that it accepts obliviously from the input with probability 0.2, we obtain a tester than accepts functions in Π with probability at least $0.2+0.8\cdot0.6 > 2/3$ and rejects ϵ-far functions with probability greater than $0.9 - 0.2 > 2/3$. ■

4 Historical notes

Property testing emerged, implicitly, in the work of Blum, Luby and Rubinfeld [15], which presents, among other things, a tester for linearity (or rather group homomorphism). This line of research was pursued in [26, 57], culminating in the work of Rubinfeld and Sudan [58], where the approach was abstracted and captured by the notion of *robust characterization*.

The starting point of Rubinfeld and Sudan [58] is the observation that the (algebraic) properties considered in [15, 26, 57] have a *local characterization*; that is, a function f has the property Π if and only if the values assigned by f to every "admissible local neighborhood" satisfy some local property. Hence, the definition of a local characterization specifies a set of local neighborhoods (i.e., $O(1)$-long sequences of elements in the function's domain) as well as a local property (i.e., a set of corresponding $O(1)$-long sequences of values that are admissible for each local neighborhood)[26], and $f \in \Pi$ if and only if the values assigned by f to each local neighborhood satisfy the local property.

A robust characterization is then defined as a local characterization in which the distance of a function from the property is reflected by the number of local conditions that it violates. That is, an (ϵ, ρ)-robust characterization of Π is a local characterization of Π such that every function that is ϵ-far from Π violates at least a ρ fraction of the local conditions (i.e., the values assigned by f to at least a ρ fraction of the local neighborhoods violate the local property).

As noted by Rubinfeld and Sudan [58], the existence of a (ϵ, ρ)-robust characterization of Π implies an ϵ-tester for Π, which samples $2/\rho$ random neighborhoods, queries the function values at the corresponding points, and accepts if and only if the corresponding local conditions are all satisfied. Note that the resulting tester is non-adaptive and has one-sided error.

A general and systematic study of property testers was initiated by Goldreich, Goldwasser, and Ron [29]. Their notion of a tester allows for adaptive queries and two-sided error probability, while viewing non-adaptivity and one-sided error as special cases. The bulk of [29] focuses on testing graph properties (see [29, Sec. 5-10]), but the paper also contains general results (see [29, Sec. 3-4]). It is worthy mentioning that their main point of reference was the model of PAC learning.

The work of Goldreich, Goldwasser, and Ron [29] advocated viewing property testing as a new type of computational problems, rather than as a tool towards program checking [14] (as viewed in [15]) or towards the construction of PCP systems (see below). The instances of these problems were viewed as descriptions of objects and their representation as functions became a non-obvious step, which required justification. For example, in the case of testing graph properties, the starting point is the graph itself, and its representation as a function is an auxiliary conceptual step.[27]

The distinction between objects and their representations became more clear when alternative representations of graphs were studied in [32, 33, 44]. At this point, query complexity that is polynomially related to the size of the object (e.g., its square root) was no longer considered inhibiting. This shift in scale is discussed next.

Initially, property testing was viewed as referring to functions that are implicitly defined by some succinct programs (as in the context of program checking) or by "transcendental" entities (as in the context of PAC learning). From this perspective the yardstick for efficiency is being polynomial in the length of the query, which means being polylogarithmic in the size of the object. However, when viewing property testing as being applied to (huge) objects that may exist in explicit form in reality, it is evident that any sub-linear complexity may be beneficial.

[26] Alternatively, the local property may be defined as a set of pairs, where each pair consists of a local neighborhood and a corresponding sequence of values.

[27] In [29] graphs are represented by their adjacency relation (or matrix), which is not overly redundant when dense graphs are concerned. In contrast, in [32] bounded-degree graphs are considered and they are represented by their sequence of incidence lists.

Proximity Oblivious Testers. The notion of (one-sided error) proximity oblivious testing is implicit in many works, starting with [15]. Its systematic study was initiated by Goldreich and Ron [35]. The notion of two-sided error proximity oblivious testing was defined and studied in [38].

Ramifications. Property testing with respect to general distributions as well as distribution-free testing, sample-based testing, and tolerant testing were all mentioned in [29, Sec. 2]. However, the focus of [29] as well as of almost all subsequent works was on the basic framework of Definition 6 (i.e., using queries in testing w.r.t the uniform distribution). An explicit study of the various ramifications started (later) in [41, 36, 51], respectively.

The PCP connection. All known PCP constructions rely on testing codewords of some code. In the "first generation" of PCP constructions (i.e., [5, 6, 23, 4, 3]), the relevant codes were the Hadamard code and (generalized) Reed-Muller codes, which led to the use of linearity testers and low-degree tests, respectively. In the "second generation" of PCP constructions (e.g., [10, 42, 43]), the use and testing of the long-code (suggested for these applications by [10]) became pivotal.[28] Some works that belong to the "third generation" of PCP constructions return to (generalized) Reed-Muller codes (e.g., [39, 11, 48, 18]), whereas for some (e.g., [17]) any "basic PCPP" will do.[29] In all cases, codeword testing is performed by using a constant number of queries, and the corresponding codes are called *locally testable*. A systematic study of such codes has been initiated in [39], and the subject will be treated is a subsequent lecture.

5 Suggested reading and exercises

Needless to say, this lecture series will only cover a tiny fraction of the research in the area of property testing, let alone research in areas that are closely related to it. Several surveys of property testing and sub-areas of it have appeared in the past. A collection of such surveys appears in [28], which contains also some examples of contemporary research (dated 2010). Two more extensive surveys were written by Ron [55, 56]: The first offers a computational learning theoretic perspective [55], and the second is organized according to techniques [56].

Most current research in property testing is listed and annotated in the *Property Testing Review* (`http://ptreview.sublinear.info/`). (This is a good opportunity to thank the moderators, Eric Blais, Sourav Chakraborty, and C. Seshadhri, for their service to the property testing community.) Some works of complexity theoretic flavor are also posted on *ECCC* (`http://eccc-preview.hpi-web.de/`).

The benefit of adaptivity. One natural question regarding property testing refers to the benefit of adaptive queries over non-adaptive ones. Indeed, the same question arises in any query-based

[28]Our periodicity scheme defines the first generation of constructions as those culminating in the (original proof of the) PCP Theorem [4, 3], and the second generation of constructions as focused on optimizing parameters of the (binary) query complexity (e.g., [10, 42, 43, 59]). The works of the "third generation" tend to focus on other considerations such as proof length (e.g., [39, 11]), combinatorial constructions (e.g., [19, 17]), and lower error via few multi-valued queries (e.g., [48, 18]). Actually, the second generation should be characterized as focusing on the optimization of the "inner verifier" (while relying on an "outer verifier" derived by applying the Parallel Repetition Theorem [54] to a two-prover system derived from the PCP Theorem), whereas works of the third generation also pay attention to the construction of the "outer verifier" (placing works such as [20, 45, 47, 46] in the third generation).

[29]In her alphabet reduction, Dinur [17] can use any "PCP of Proximity" (as defined in [11, 19]) for membership in a code of constant relative distance.

model. Adaptive queries can always be emulated at exponential cost; that is, q adaptive queries to a function $f : [n] \to R$ can be emulated by less than $|R|^q$ non-adaptive queries.[30] The question is whether a cheaper emulation is possible. Within the context of property testing, the answer seems to vary according to the type of properties. Types of properties for which the emulation has no overhead are shown in [12, 24]. In the context of graph properties, the answer varies according to the specific model: See [40, 34] versus [53].

One-sided versus two-sided error. Another natural question regarding property testing refers to the difference between one-sided and two-sided error probability. Needless to say, the same question arises in any model of probabilistic computation (see, for example, $\mathcal{BPP}$-versus-$\mathcal{RP}$). Interestingly, in the context of property testing, a huge gap may exist between these two versions. For example, Proposition 1 asserts an $O(1/\epsilon^2)$-time tester of two-sided error probability for `MAJ`, whereas any one-sided error tester for `MAJ` must make a linear number of queries (see Exercise 7). Gaps exist also in models of testing graphs properties (see the results regarding ρ-Clique in [29] and the results regarding cycle-freeness in [32, 16]).

We mention that the "reverse type of one-sided error" testing, where the tester is required to always reject (i.e., reject with probability 1) objects that are far from the property, has not been studied for a good reason (see [61, Prop. 5.6]).

Hierarchy. Complexity hierarchies are known in many computational models (see, for example, the classical computational complexity hierarchies [27, Chap. 4]). It turns out that such hierarchies (i.e., query hierarchies) exist also in property testing [31].

Basic Exercises

The following exercises detail some claims that were made in the main text.

Exercise 1 (details for the proof of Proposition 1): *Use the Chernoff Bound (or alternatively Chebyshev's Inequality) to prove that the average value of the sample points approximates the average value of all x_i's. The details involve defining m random variables, and using the aforementioned inequality.*

Exercise 2 (details for the proof of Claim 2.1): *Prove that X_n* (resp., *Z_n*) *as redefined in the proof of Claim 2.1 is uniformly distributed over n-bit strings of Hamming weight $\lfloor n/2 \rfloor + 1$* (resp., $\lfloor n/2 \rfloor$).

Guideline: A crude solution amounts to computing the probability mass given to each string according to each of the definitions. A nicer solution is to show that the redefined processes yield output distributions that are symmetric with respect to the indices.

Exercise 3 (error reduction for testers): *Show that the error probability of a property tester can be reduced to 2^{-k} at the cost of increasing its query (and time) complexity by a factor of $O(k)$, and while preserving one-sided error.*

[30] This is done by trying all possible answers to the previous queries, which defines a full $|R|$-ary tree of depth $q-1$ with vertices at distance i from the root representing possible sequences of i answers to the first i queries. Since each such sequence of answers determines the $i+1^{\text{st}}$ query in the corresponding execution, the number of queries equals the number of vertices in this tree (i.e., $\sum_{i=0}^{q-1} |R|^i$). Alternatively, the number of queries can be preserved at the cost of decreasing the distinguishing gap of the tester by the said factor.

Guideline: Invoke the original tester t times, while using independent randomness in each invocations, and accept if and only if the majority of these invocations accepted. The analysis reduces to showing that if we repeat an experiment that succeeds with probability $2/3$ for t times, then, with probability $1-\exp(-\Omega(t))$, the majority of the trials succeed.

Exercise 4 (on the threshold probability of POTs): *Show that, for every $\tau \in (0,1]$ and $\tau' \in (0,1)$, and for every $\rho : (0,1] \to (0,1]$ and $q : \mathbb{N} \to \mathbb{N}$, if Π has a q-query POT with threshold probability τ and detection probability ϱ, then Π has a q-query POT with threshold probability τ' and detection probability $\varrho' = \Omega(\varrho)$.*

Guideline: Consider a POT that invokes the given POT with probability p (to be determined), and accepts (or rejects) otherwise.

Exercise 5 (a two-sided error POT for MAJ): *Show that the following algorithm constitutes a two-sided error POT with linear detection probability for* MAJ*. For odd n, on input $x \in \{0,1\}^n$, the algorithm selects uniformly $i \in [n]$, and outputs x_i. For even n, a small modification is required so that we can still use the threshold probability $\tau = 1/2$.*

Guideline: The probability that x is accepted is $\mathrm{wt}(x)/|x|$. For an even n, reducing the acceptance probability by a factor of $1-n^{-1}$ will do, since $(1-n^{-1}) \cdot (0.5+n^{-1}) \geq 0.5$.

Exercise 6 (a less trivial two-sided error POT): *Since the example provided by Exercise 5 is quite disappointing, we consider the following set* BAL $= \{x : \mathrm{wt}(x) = |x|/2\}$ (of "balanced" strings). *Show that the following algorithm constitutes a two-sided error POT with threshold probability* 0.5 *and quadratic detection probability for* BAL*. On input $x \in \{0,1\}^*$, the algorithm selects uniformly $i,j \in [n]$, and accepts if and only if $x_i \neq x_j$. Note that the same algorithm constitutes a POT also for $S_c = \{x : c \cdot |x| \leq \mathrm{wt}(x) \leq (1-c) \cdot |x|\}$, for every constant $c \in (0, 0.5)$, but the threshold probability and the detection probability are different in this case.*

Guideline: The probability that x is accepted equals $2\mathrm{wt}(x) \cdot (|x| - \mathrm{wt}(x))/|x|^2$, which is smaller than $1/2$ if and only if $\mathrm{wt}(x) \neq |x|/2$. In the case of S_c, the threshold probability is $2c(1-c)$, and the detection probability function is linear.

Exercise 7 (on one-sided error testers for MAJ): *Prove that* MAJ *has no one-sided error tester of sub-linear query complexity.*

Guideline: Consider an arbitrary sub-linear algorithm T and an execution (i.e., selection of randomness for T) in which T rejects the string 0^n, which is 0.5-far from MAJ. (Such an execution exists since $\mathbf{Pr}[T^{0^n}(n,0.5)=0] \geq 2/3 > 0$.) Denoting by Q the set of locations queried in this execution, consider the string x such that $x_i = 0$ if and only if $i \in Q$, and note that $\mathbf{Pr}[T^x(n,\epsilon)=1] < 1$. On the other hand, if $|Q| < n/2$, then $x \in$ MAJ.

Exercise 8 (the existence of good linear codes): *Show that there exists a $0.5n$-by-n Boolean matrix in which every $0.05n$ columns are linearly independent and every non-empty linear combination of the rows has Hamming weight at least $0.1n$.*

Guideline: Using the probabilistic method (see [2]), upper-bound the probability that a random matrix does not satisfy the foregoing conditions.

Exercise 9 (detail for the proof of Proposition 11): *Let G be an m-by-n Boolean matrix in which every t columns are linearly independent. Prove that for a uniformly distributed $x \in \{0,1\}^m$, each t-bit long subsequence of xG is uniformly distributed in $\{0,1\}^t$.*

Guideline: Consider the corresponding m-by-t matrix G', and note that each image of the map $x \mapsto xG'$ has 2^{m-t} preimages.

Exercise 10 (another detail for the proof of Proposition 11): *Let G be an $0.5n$-by-n Boolean matrix. Prove that, for sufficiently large n, the number of n-bit strings that are 0.1-close to $\{xG : x \in \{0,1\}^{0.5n}\}$ is at most $2^{0.99n}$.*

Exercise 11 (the testability of monotone properties is not closed under complementation): *Show that there is a monotone property Π such that testing Π is trivial* (any n-bit string is $1/n$-close to Π), *but 0.001-testing $\{0,1\}^* \setminus \Pi$ requires a linear number of queries.*

Guideline: Let G' be a $0.05n$-by-$(n-1)$ Boolean matrix in which every $0.001n$ columns are linearly independent and every non-empty linear combination of the rows has Hamming weight at least $0.1n$. Let $\Pi' = \{xG' : x \in \{0,1\}^{0.05n} \setminus \{0^{0.05n}\}$ and $\Pi'' = \{w' \vee w'' : w' \in \Pi' \wedge w'' \in \{0,1\}^{n-1}\}$; that is, Π' is the linear code generated by G' with the exception of the all-zero string, and Π'' is its "monotone closure". Letting $\Pi = \{0,1\}^n \setminus \{1w : w \in \Pi''\}$, note that each string is $1/n$-close to Π, whereas $\{0,1\}^n \setminus \Pi = \{0w : w \in \Pi''\}$ is hard to test.[31] The latter claim is proved by showing (analogously to the proof of Proposition 11) that the uniform distribution on Π' is perfectly indistringuishable (by $0.05n$ queries) from the uniform distribution on $\{0,1\}^{n-1}$, and observing that $|\Pi''| \leq 2^{0.9n} \cdot |\Pi'|$.

Exercise 12 (generalization of Theorem 13): *Let Π' and Π'' be properties of functions defined over the same domain, D.*

1. *Prove that if, for every $f : D \to \{0,1\}^*$, it holds that $\delta_{\Pi' \cap \Pi''}(f) \leq \delta_{\Pi'}(f) + \delta_{\Pi''}(f)$, then, for every $\epsilon' \in (0, \epsilon)$, the property $\Pi' \cap \Pi''$ is ϵ-testable within query complexity $q(n,\epsilon) = O(q'(n,\epsilon') + q''(n, \epsilon - \epsilon'))$, where q' and q'' denote the query complexities of testing the properties Π' and Π'', respectively.*

2. *Show that if Π' and Π'' are monotone properties, then $\delta_{\Pi' \cap \Pi''}(f) \leq \delta_{\Pi'}(f) + \delta_{\Pi''}(f)$ holds for any function f.*

3. *Generalizing Part 1, supopose that for $F : (0,1] \times (0,1] \to (0,1]$ it holds that $\delta_{\Pi' \cap \Pi''}(f) \leq F(\delta_{\Pi'}(f), \delta_{\Pi''}(f))$, for every f. Show that, for every $\epsilon > 0$, if $F(\epsilon', \epsilon'') \leq \epsilon$, then the property $\Pi' \cap \Pi''$ is ϵ-testable within query complexity $O(q'(n,\epsilon') + q''(n,\epsilon''))$.*

4. *Show that a function as in Part 3 does not exist for the properties used in the proof of Theorem 12.*

Guideline: Parts 1–3 are implicit in the proof of Theorem 13. Part 4 can be proved either by direct inspection of these properties or by arguing that the contrary hypothesis contradicts Theorem 12.

Exercise 13 (a generic learning algorithm): *Show that every $\Pi = \cup_{n \in \mathbb{N}} \Pi_n$ can be properly learned within query complexity $q(n,\epsilon) = \min(n, O(\epsilon^{-1} \log |\Pi_n|))$.*

[31]Indeed, Π is anti-monotone, so one may consider $\{1^n \oplus w : w \in \Pi\}$ instead.

Guideline: The key observation that a sample of $O(t/\epsilon)$ random points allows for approximating the distance between two functions up to an additive term of $\epsilon/2$ with error probability 2^{-t}. The bound of $O(\epsilon^{-1}\log|\Pi_n|)$ follows by observing that the same sample can be used to estimate the distance of each $h \in \Pi_n$ to the target function f, and applying a union bound.

Additional Exercises

The following exercises present a few useful observation regarding oracle machines in general.

Exercise 14 (straightforward emulation of adaptive queries): *Show that the execution of any oracle machine that makes q* (possibly adaptive) *queries to an unknown function $f : [n] \to R$, can be emulated by a non-adaptive machine that makes at most $\sum_{i=0}^{q-1} |R|^i$ queries.*

Guideline: Fixing the internal coin tosses of the machine, consider a tree that describes all its possible q-long sequences of queries, where the vertices correspond to queries and the edges correspond to possible answers.

Exercise 15 (upper bound on the randomness complexity of oracle machines):[32] *Let Π be a promise problem regarding functions from $[n]$ to R, where ϵ-testing a property of such functions is a special case* (in which the YES-instances are function having the property and the NO-instances are function that are ϵ-far from the property). *Suppose that M is a randomized oracle machines that solves the problem Π with error probability at most $1/4$, while making q queries. Assuming that n is sufficiently large, show that Π can be solved by a randomized oracle machines that makes at most q queries, tosses at most $\log n + \log\log|R|$ coins, and has error probability at most $1/3$. Note that the randomness-efficient machine derived here is not necessarily computationally-efficient.*

Guideline: Suppose that M tosses r coins, and observe that the number of possible functions that M is required to decide about is at most $|R|^n$. Using the probabilistic method, show that there exists a $\log_2 O(|R|^n)$-set $S \subseteq \{0,1\}^r$, such that for every function $f : [n] \to R$ it holds that

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

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 16 (on the tightness of the bound provided in Exercise 15):[33] *Let Π be a promise problem regarding functions from $[n]$ to R. We say that Π is ρ-*evasive *if there exists a function $f : [n] \to R$ such that for every $Q \subset [n]$ of density ρ, there exists a* YES*-instance* (of Π) *denoted f_1 and a* NO*-instance denoted f_0 such that for every $x \in Q$ it holds that $f_1(x) = f_0(x) = f(x)$. Show that if a ρ-evasive Π can be decided* (say, with error probability 1/3) *by an oracle machine M that makes q queries, then this machine must toss at least $\log_2(\rho n/q)$ coins. Note that for many natural properties and for sufficiently small constant $\epsilon > 0$, the problem of ϵ-testing the property is $\Omega(1)$-evasive.*[34]

[32]Based on [37].

[33]Based on [37].

[34]A partial list includes sets of low degree polynomials, any code of linear distance, monotonicity, juntas, and various graph properties. Indeed, this list is confined to examples that will appear in subsequent lectures.

Guideline: Suppose that M tosses r coins, and let f be a function as in the ρ-evasive condition. Consider all 2^r possible executions of M^f, and let Q denote the set of queries made in these executions. Then, $|Q| \leq 2^r \cdot q$. On the other hand, $|Q| > \rho \cdot n$, since otherwise these executions cannot distinguish f_0 from f_1 (since all these executions will only query locations in Q).

6 Digest: The most important points

Given that the notes for this lecture are quite long, it seems good to list some of the points that will be instrumental for the subsequent lectures. Needless to say, the definition of a property tester (i.e., Definition 6) is pivotal for all that follows. The notion of a proximity oblivious tester (see Definition 7) will also be used a lot (sometimes only implicitly). Two important points that underly the study of such testers are:

- Representation. The tested objects will be (typically) represented in a natural and concise manner, and n will denote their size. They will be presented either as sequences over an alphabet Σ (e.g., $x \in \Sigma^n$) or as functions from $[n]$ to Σ (i.e., in such a case we consider $x : [n] \to \Sigma$).

  Indeed, here we seized the opportunity to present these notions while referring to an arbitrary alphabet rather than only to the binary alphabet (i.e., $\Sigma = \{0,1\}$), as done in previous sections.

- The (standard) notion of distance. For $x, y \in \Sigma^n$, we consider their relative Hamming distance, denoted $\delta(x,y) \overset{\text{def}}{=} |\{i \in [n] : x_i \neq y_i\}|/n$. For $x \in \Sigma^n$ and $S \subseteq \Sigma^n$, we denote by $\delta_S(x)$ the relative Hamming distance of x from S; that is, $\delta_S(x)$ is the minimum, taken over all $z \in S \cap \{0,1\}^{|x|}$, of $\delta(x,z)$.

  We shall say that x is ϵ-far from S if $\delta_S(x) > \epsilon$, and otherwise (i.e., when $\delta_S(x) \leq \epsilon$) we shall say that x is ϵ-close to S. Indeed, typically, ϵ will denote a proximity parameter, which determines what is considered far.

Our main focus will be on the *query complexity* of standard testers (i.e., as in Definition 6), measured in terms of the size of the tested object, denoted n, and the proximity parameter, denoted ϵ. The first priority is to have *query complexity that is sub-linear in* n, and the slower this complexity grows with n, the better. At times, especially when discussing lower bounds, we may fix the value of the proximity parameter (i.e., set ϵ to be a small positive constant), and consider the complexity of the residual tester, called an ϵ-tester, as a function of n only. The ultimate goal, which is not always achievable, is to have the query complexity be independent of n. We shall also care about the dependence of the query complexity on ϵ, and in particular whether it is $O(1/\epsilon)$, $\text{poly}(1/\epsilon)$, or worse.

The *time complexity* of the tester will be our secondary focus, although it is obviously important. We shall say that a tester is efficient if its time complexity is almost linear in its query complexity.

The forgoing refers to standard testers (i.e., as in Definition 6). In contrast, the complexity of proximity-oblivious testers (as in Definition 7) will only depend on n, and their rejection probability is related to the distance of the object from the property. The latter relation is captured by a monotonically non-decreasing function, typically denoted ϱ. As shown in Theorem 9, a proximity-oblivious tester yields an ordinary tester by repeating the former for $O(1/\varrho^c(\epsilon))$ times, where $c = 1$

in case of one-sided error testing (and $c = 2$ otherwise), and ϵ is the proximity parameter. This brings us to the last point in this section.

- One-sided error probability refers to the case that the tester always accepts any object that has the property, but may accept with bounded probability objects that are far from the property (which means that it errs with some bounded probability).[35] General testers (a.k.a two-sided error testers) may err, with bounded probability, both in the case the object has the property and in the case it is far from the property.

References

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

[2] N. Alon and J.H. Spencer. *The Probabilistic Method.* John Wiley & Sons, Inc., 1992. Second edition, 2000.

[3] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy. Proof Verification and Intractability of Approximation Problems. *Journal of the ACM*, Vol. 45, pages 501–555, 1998. Preliminary version in *33rd FOCS*, 1992.

[4] S. Arora and S. Safra. Probabilistic Checkable Proofs: A New Characterization of NP. *Journal of the ACM*, Vol. 45, pages 70–122, 1998. Preliminary version in *33rd FOCS*, 1992.

[5] L. Babai, L. Fortnow, and C. Lund. Non-Deterministic Exponential Time has Two-Prover Interactive Protocols. *Computational Complexity*, Vol. 1, No. 1, pages 3–40, 1991. Preliminary version in *31st FOCS*, 1990.

[6] L. Babai, L. Fortnow, L. Levin, and M. Szegedy. Checking Computations in Polylogarithmic Time. In *23rd ACM Symposium on the Theory of Computing*, pages 21–31, 1991.

[7] 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.

[8] T. Batu, L. Fortnow, R. Rubinfeld, W.D. Smith, P. White. Testing that distributions are close. In *41st IEEE Symposium on Foundations of Computer Science*, pages 259–269, 2000.

[9] 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.

[10] M. Bellare, O. Goldreich and M. Sudan. Free Bits, PCPs and Non-Approximability – Towards Tight Results. *SIAM Journal on Computing*, Vol. 27, No. 3, pages 804–915, 1998. Extended abstract in *36th FOCS*, 1995.

[35] Recall that a property is associated with the set of objects having the property.

[11] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan. Robust PCPs of Proximity, Shorter PCPs, and Applications to Coding. *SIAM Journal on Computing*, Vol. 36 (4), pages 889–974, 2006. Extended abstract in *36th STOC*, 2004.

[12] E. Ben-Sasson, P. Harsha, and S. Raskhodnikova. 3CNF Properties are Hard to Test. *SIAM Journal on Computing*, Vol. 35(1), pages 1–21, 2005.

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

[14] M. Blum and S. Kannan. Designing Programs that Check their Work. In *21st ACM Symposium on the Theory of Computing*, pages 86–97, 1989.

[15] M. Blum, M. Luby and R. Rubinfeld. Self-Testing/Correcting with Applications to Numerical Problems. *Journal of Computer and System Science*, Vol. 47, No. 3, pages 549–595, 1993.

[16] A. Czumaj, O. Goldreich, D. Ron, C. Seshadhri, A. Shapira, and C. Sohler. Finding cycles and trees in sublinear time. *Random Structures and Algorithms*, Vol. 45(2), pages 139–184, 2014.

[17] I. Dinur. The PCP Theorem by Gap Amplification. *Journal of the ACM*, Vol. 54 (3), Art. 12, 2007. Extended abstract in *38th STOC*, 2006.

[18] I. Dinur and P. Harsha. Composition of Low-Error 2-Query PCPs Using Decodable PCPs. *SIAM Journal on Computing*, Vol. 42 (6), pages 2452–2486, 2013. Extended abstract in *50th FOCS*, 2009.

[19] I. Dinur and O. Reingold. Assignment-testers: Towards a combinatorial proof of the PCP-Theorem. *SIAM Journal on Computing*, Vol. 36 (4), pages 975–1024, 2006. Extended abstract in *45th FOCS*, 2004.

[20] I. Dinur and S. Safra. The importance of being biased. In *34th ACM Symposium on the Theory of Computing*, pages 33–42, 2002.

[21] F. Ergun, S. Kannan, R. Kumar, R. Rubinfeld, and M. Viswanathan. Spot checkers. *Journal of Computer and System Science*, Vol. 60, pages 717–751, 2000.

[22] S. Even, A.L. Selman, and Y. Yacobi. The Complexity of Promise Problems with Applications to Public-Key Cryptography. *Information and Control*, Vol. 61, pages 159–173, 1984.

[23] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Approximating Clique is almost NP-complete. *Journal of the ACM*, Vol. 43, pages 268–292, 1996. Preliminary version in *32nd FOCS*, 1991.

[24] E. Fischer. On the strength of comparisons in property testing. *Information and Computation*, Vol. 189(1), pages 107–116, 2004.

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

[26] P. Gemmell, R.J. Lipton, R. Rubinfeld, M. Sudan, and A. Wigderson. Self-Testing/Correcting for Polynomials and for Approximate Functions . In the proceedings of *ACM Symposium on the Theory of Computing*, pages 32–42, 1991.

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

[28] O. Goldreich (ed.). *Property Testing: Current Research and Surveys.* Springer, LNCS, Vol. 6390, 2010.

[29] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, pages 653–750, July 1998. Extended abstract in *37th FOCS*, 1996.

[30] O. Goldreich and T. Kaufman. Proximity Oblivious Testing and the Role of Invariances. In *15th RANDOM*, pages 579–592, 2011.

[31] O. Goldreich, M. Krivelevich, I. Newman, and E. Rozenberg. Hierarchy Theorems for Property Testing. *Computational Complexity*, Vol. 21(1), pages 129–192, 2012.

[32] O. Goldreich and D. Ron. Property testing in bounded degree graphs. *Algorithmica*, pages 302–343, 2002. Extended abstract in *29th STOC*, 1997.

[33] O. Goldreich and D. Ron. A sublinear bipartite tester for bounded degree graphs. *Combinatorica*, Vol. 19 (3), pages 335–373, 1999. Extended abstract in *30th STOC*, 1998.

[34] 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.

[35] O. Goldreich and D. Ron. On Proximity Oblivious Testing. *SIAM Journal on Computing*, Vol. 40, No. 2, pages 534–566, 2011. Extended abstract in *41st STOC*, 2009.

[36] O. Goldreich and D. Ron. On Sample-Based Testers. In *6th ITCS*, pages 337–345, 2015.

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

[38] O. Goldreich and I. Shinkar. Two-Sided Error Proximity Oblivious Testing. *ECCC*, TR12-021, 2012. (See Revision 4, 2014.)

[39] O. Goldreich and M. Sudan. Locally testable codes and PCPs of almost-linear length. *Journal of the ACM*, Vol. 53 (4), pages 558–655, 2006. Extended abstract in *43rd FOCS*, 2002.

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

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

[42] J. Hastad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, Vol. 182, pages 105–142, 1999. Preliminary versions in *28th STOC* (1996) and *37th FOCS* (1996).

[43] J. Hastad. Getting optimal in-approximability results. *Journal of the ACM*, Vol. 48, pages 798–859, 2001. Extended abstract in *29th STOC*, 1997.

[44] T. Kaufman, M. Krivelevich, and D. Ron. Tight Bounds for Testing Bipartiteness in General Graphs. *SIAM Journal on Computing*, Vol. 33 (6), pages 1441–1483, 2004.

[45] S. Khot. On the power of unique 2-prover 1-round games. In *34th ACM Symposium on the Theory of Computing*, pages 767–775, 2002.

[46] S. Khot, G. Kindler, E. Mossel, and R. O'Donnell. Optimal Inapproximability Results for MAX-CUT and Other 2-Variable CSPs? *SIAM Journal on Computing*, Vol. 37 (1), pages 319–357, 2007. Extended abstract in *44th FOCS*, 2004.

[47] S. Khot and O. Regev. Vertex cover might be hard to approximate to within 2-epsilon. *Journal of Computer and System Science*, Vol. 74 (3), pages 335–349, 2008. Extended abstract in *18th CCC*, 2003.

[48] D. Moshkovitz and R. Raz. Two-query PCP with subconstant error. *Journal of the ACM*, Vol. 57 (5), 2010. Extended abstract in *49th FOCS*, 2008.

[49] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[50] I. Newman. Property Testing of Massively Parametrized Problems – A Survey. In [28].

[51] 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.

[52] L. Pitt and L.G. Valiant. Computational limitations on learning from examples. *Journal of the ACM*, Vol. 35 (4), pages 965–984, 1988.

[53] S. Raskhodnikova and A. Smith. A note on adaptivity in testing properties of bounded degree graphs. *ECCC*, TR06-089, 2006.

[54] R. Raz. A Parallel Repetition Theorem. *SIAM Journal on Computing*, Vol. 27 (3), pages 763–803, 1998. Extended abstract in *27th STOC*, 1995.

[55] D. Ron. Property testing: A learning theory perspective. *Foundations and Trends in Machine Learning*, Vol. 1(3), pages 307–402, 2008.

[56] D. Ron. Algorithmic and analysis techniques in property testing. *Foundations and Trends in Theoretical Computer Science*, Vol. 5, pages 73–205, 2010.

[57] R. Rubinfeld and M. Sudan. Self-Testing Polynomial Functions Efficiently and Over Rational Domains. In the proceedings of *SODA*, pages 23–32, 1992.

[58] R. Rubinfeld and M. Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal on Computing*, Vol. 25(2), pages 252–271, 1996. Unifies and extends part of the results contained in [26] and [57].

[59] A. Samorodnitsky and L. Trevisan. A PCP Characterization of NP with Optimal Amortized Query Complexity. In *32nd ACM Symposium on the Theory of Computing*, pages 191–199, 2000.

[60] M. Sudan. Invariances in Property Testing. In [28].

[61] R. Tell. On Being Far from Far and on Dual Problems in Property Testing. *ECCC*, TR15-072, 2015. (Revision 1, Aug. 2015.)

[62] G.J. Valiant. Algorithmic Approaches to Statistical Questions. PhD Thesis, University of California at Berkeley, 2012.

[63] L.G. Valiant. A Theory of the Learnable. *Communications of the ACM*, Vl. 27 (11), pages 1134–1142, 1984.