CHAPTER ONE

The Main Themes: Approximate Decision and Sublinear Complexity

Summary: In this chapter, we introduce, discuss, and illustrate the conceptual framework of property testing, emphasizing the themes of *approximate decision* and *sublinear 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. To begin, we consider the potential benefits of testing (i.e., approximate decisions of sublinear complexity).

Section 1.1 provides a very brief introduction to property testing, sketching its basic definition and providing an overview of its flavor and potential benefits. The pace here is fast and sketchy, unlike that in the rest of this chapter. The technical material is presented in Sections 1.2 and 1.3, which constitutes the main part of this chapter. A more detailed account of the organization of this part is provided in Section 1.1.4.

1.1. 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*, February 17, 2016

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

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

In general, the data are a set of records (or items) that may be interrelated in various ways. The contents and meaning of the data may be reflected not only in the individual

items (or records), but also in the relations between them. In such a case, important aspects of the data are reflected in the structural relations between their items. In particular, the indication of which pairs of items are related may be such an aspect, and it can be modeled as a graph. Needless to say, this captures only one aspect of the data, but this aspect could be very significant. When such a model is used, checking whether the graph that arises has certain structural 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.

Important as it is, big data is not the only source of huge objects that are considered by *property testing*. Other types 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 superfast algorithms that refrain from obtaining the full explicit description of the object. These algorithms inspect relatively small portions of the object and pass judgment based on such an inspection.

The reader may wonder how it is possible to say anything meaningful about an object without looking at all of it. On further thought, however, one may note that we are aware of such cases: All frequency statistics are of this form. 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.)

1.1.1. Property Testing at a Glance

As will be detailed in this chapter, property testing is primarily concerned with *superfact 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 certain huge substructures inside them are also addressed. In all cases, the algorithms sought are of sublinear complexity (i.e., complexity that is sublinear in the size of the object), and in particular they inspect only relatively 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 inspect only part of their input. In fact, our focus is on such algorithms, which make approximate decisions regarding their input (i.e., whether it has some property or is far from having it).

Cases in which such superfact approximate decisions are 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 triangle-free; and testing properties of visual images or 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, as with any theoretical research, the value of research in property testing is *not confined* to the actual use of the suggested algorithms (i.e., the resulting testers). The development and study of conceptual frameworks, let alone the development of algorithmic design and analysis techniques, is more important for the theory of computation at large as well as for computer practice. Although the impact on practice is typically hard to trace, the relations between property testing and the rest of the theory of computing are evident (and will be pointed out in relevant parts of this book).

1.1.2. On the Potential Benefits of Property Testers

Property testing is associated with approximate decision algorithms that run in sublinear time or at least make a sublinear number of queries to their input. The benefit of sublinear 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 in 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 in which such a trade-off is worthwhile.

It is infeasible to recover the object fully. 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 allow inspecting all of it. In such settings, there is no choice but to use algorithms of sublinear 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. Here 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 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. Here we refer to applications in which the utility of objects that are close to having the property is almost as valuable 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 ruling that the object has the property whereas in reality the object is only close to having this property.[1]

Testing as a preliminary step before deciding. Here 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.,

[1] **Advanced comment:** One may argue that in such cases, "tolerant testing" (see Section 1.3.2) is even more adequate. Yet, tolerant testing may be harder than standard testing (cf. [109]).

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 such objects based on the preliminary step.

Testing as a preliminary step before reconstructing. This refers to settings in which we wish to recover the object fully, either by reading all of it or by running a learning algorithm, but we wish 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.

1.1.3. 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 certain type of algorithms (i.e., ones of sublinear complexity) for 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 areas, but property testing is not confined to PCPs (and/or to the study of "locally testable codes" [see Chapter 13]).

In addition to standing in between algorithmic research and complexity theory, the results of property testing have a flavor that makes them different from the mainstream 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 property testing (i.e., approximate decision) makes its results stand out. But property testing is not the only research area that has this fate: The same can be said of machine learning and distributed computing, to mention just two examples.

One additional characteristic of property testing is that its positive results tend to be established by simple algorithms that are supported by a complex analysis. The simplicity of these algorithms has met a lack of respect among a few researchers, but this is a fundamental mistake on their part. 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 could hope for.

Like algorithmic research, property testing tends to split according to the "area" of the property and the "type" of objects being considered (i.e., the natural perception of the object). Indeed, the organization of the current book reflects this split, where Chapters 2–6 focus on (objects that are viewed as) functions and Chapters 8–10 focus on (objects that are viewed as) graphs. Furthermore, within the world of functions, one may distinguish types corresponding to the structure of the domain on which the function is defined (e.g., a group, a vector space, or a Boolean hypercube). The structure of the domain is often reflected by the invariances that are satisfied by the properties that one considers (e.g., affine invariance, closure under graph isomorphism, etc.). Still, conceptual frameworks,

techniques, ideas, and inspiration do cross the borders between the parts of the foregoing splits.

Property testing has a clear potential for practical applications, although it seems not to have materialized so far. The most begging applications are to the practice in areas such as machine learning, compressed sensing, computer vision, statistics, and privacy preserving data analysis. To provide some illustration to this potential, we mention the experimental search-and-cluster engine [77], which is based on [177], which in turn uses [123, 98], which are informed by [121, 140]. (Indeed, a nondirect line of influence should be expected in the transportation of theoretical research to practice.) Applications that are more directly inspired by [140] are reported in [209, 163]. We also mention the connection between the study of testing visual images [230, 243] and finding matches between images [193, 194]. Lastly, we mention that research in the somewhat related area of "streaming algorithms" [15] has witnessed more interaction with practice (including computer networks and databases [213], compressed sensing (e.g., [80]), and numerical linear algebra [270]).

1.1.4. Organization and Some Notations

As stated previously, we view property testing as concerned primarily with approximate decisions, a notion discussed in Section 1.2.2. (For perspective, we recall in Section 1.2.1 the notion of approximate search problems.) Next, in Section 1.2.3, we discuss the second key feature of property testing – its focus on sublinear complexity. Then, in Section 1.2.4, we highlight yet another feature of property testing – its focus on properties that are not fully symmetric (i.e., are not invariant under arbitrary reordering of the sequence of values that represent the object). In general, the relation between objects and their representation is crucial in the context of property testing, and this issue is discussed in Section 1.2.5.

The core of this chapter is presented in Section 1.3. The basic notions, definitions, and goals of property testing are presented in Section 1.3.1 and used extensively throughout the entire book (with very few exceptions). In contrast, the ramifications discussed in Section 1.3.2 are used lightly (if at all), and ditto for the general observations made in Sections 1.3.4 and 1.3.5 (which refer to the "algebra of property testing" and to the testing-by-learning connection, respectively). In Section 1.3.3, we present another notion that is used extensively – that of a proximity-oblivious tester (POT).

Historical perspectives, suggestions for further reading, and exercises are provided in Sections 1.4 and 1.5. Finally, in Section 1.6 we reiterate some of issues discussed in the current chapter, in light of their importance to the rest of the book.

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

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

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

Teaching Note: Section 1.2 provides a paced presentation of the mindframe that underlies property testing, illustrating key issues such as approximate decision and sublinear complexity. In case of time constraints, one can skip this section and go directly to Sections 1.3.1 and 1.3.3. Still, we recommend taking the slower pace and covering also Sections 1.3.4 and 1.3.5, although this may mean spending more than a single lecture on the current chapter. The ramifications discussed in Section 1.3.2 are discussed in greater detail in Chapter 12, but we believe that an early detour into these variants provides a good perspective on the main definition presented in Section 1.3.1 (while acknowledging that this may be too much for some readers).

1.2. 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, for the purpose of providing a wide perspective.

1.2.1. A Detour: Approximate Search Problems

Recall that search problems are defined in terms of binary relations, and consist of finding a "valid solution" y to a given instance x, where y is a valid solution to x if (x, y) satisfies the binary relation associated with the problem. Letting $R \subseteq \{0,1\}^* \times \{0,1\}^*$ denote such a relation, we say that y is a solution to x if $(x, y) \in R$, and the set of solutions for the instance x is denoted $R(x) \stackrel{\text{def}}{=} \{y : (x, y) \in R\}$. Hence, given x, the task is to find $y \in R(x)$, provided that $R(x) \neq \emptyset$. (The computation of a function corresponds to the special case in which all these sets are singletons.)

In optimization problems, the valid solutions are assigned a value (or a cost), captured by a function $v : \{0,1\}^* \to \mathbb{R}$, 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 $v(y) = \max_{z \in R(x)}\{v(z)\}$ (resp., $v(y) = \min_{z \in R(x)}\{v(z)\}$).[2]

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 $v(y)$ is "close" to $\max_{z \in R(x)}\{v(z)\}$ (resp., to $\min_{z \in R(x)}\{v(z)\}$). One may also talk about the estimation problem, in which the task is to approximate the value of the optimal solution (rather than actually finding a solution that obtains that value).

The point we wish to make here is that, once a function v 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?

1.2.2. Property Testing: Approximate Decision Problems

Indeed, *what can an approximate decision problem possibly 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

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

approximate decision (or a natural 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 that of x being "far" from S. Hence, we consider a promise problem (cf. [105, 129] or [131, 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 fix 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 $\delta_S(x) \stackrel{\text{def}}{=} \min_{z \in S}\{\delta(x,z)\}$. 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 $\{x : 0 < \delta_S(x) \leq \epsilon\}$ 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*.

1.2.3. Property Testing: Sublinear Complexity

But *why did we relax standard decision problems into approximate decision problems*? The answer is that, as in the case of approximate search problems, 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 1.1.2).

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 almost all the bits in the input), the relaxation to an approximate decision seems necessary to avoid 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 (see Proposition 1.1), whereas no sublinear-time (randomized) algorithm can solve the corresponding (exact) decision problem (see Proposition 1.2). We shall also show that randomness is essential for the positive result (see Proposition 1.3).

Proposition 1.1 (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 randomized 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 $i_1, \ldots, i_m$, 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),[3] which implies that, with probability at least $2/3$, the average of the sample is within $\epsilon/2$ of the actual average; that is,

$$\mathbf{Pr}_{i_1,\ldots,i_m\in[|x|]}\left[\left|\frac{\sum_{j\in[m]} x_{i_j}}{m} - \frac{\sum_{i=1}^{|x|} x_i}{|x|}\right| \leq \epsilon/2\right] \geq 2/3. \tag{1.1}$$

We stress that Eq. (1.1) holds since $m = \Omega(1/\epsilon^2)$. It follows that the algorithm accepts each $x \in \mathtt{MAJ}$ with probability at least $2/3$, since in this case $\sum_{i=1}^{|x|} x_i > |x|/2$. Likewise, it rejects each x that is ϵ-far from `MAJ` with probability at least $2/3$, since in this case $\sum_{i=1}^{|x|} x_i \leq (0.5-\epsilon)\cdot|x|$. ■

Teaching Note: We assume that the reader is comfortable with the assertion captured by Eq. (1.1); that is, the reader should find Exercise 1.1 easy to solve. If this is not the case, then we advise the reader to become comfortable with such assertions and arguments before continuing reading. Appendix A should suffice for readers who have basic familiarity with probability theory. Likewise, we assume that the reader is comfortable with the notion of a randomized algorithm; basic familiarity based on [131, Sec. 6.1] or any part of [212] should suffice.

Proposition 1.2 (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 chapter, with the exception of the proof of Proposition 1.11, which is also a lower bound. Some readers may prefer to skip these proofs at the current time, and return to them at a later time (e.g., after reading Chapter 7). We prefer to keep the proofs in place, but warn readers not to stall at them.

[3] **Advanced comment:** Indeed, both inequalities are essentially equivalent when one seeks constant error probability. See discussion in Appendix A.4.

***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 1.2.1, a randomized algorithm that queries either X_n or Z_n at $o(n)$ locations cannot distinguish these two cases with a probabilistic gap that exceeds $o(1)$, and hence must be wrong in one of the two cases.

(Note that the randomized decision procedure must be correct on each input. The proof technique employed here proceeds by showing that any "low-complexity" 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 exists at least one input on which the procedure errs with too large probability.)

Claim 1.2.1 (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 0 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 it greatly facilitates the following analysis.)[4] 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. (Furthermore, conditioned on not having queried i so far, the distribution of i is uniform over all unqueried locations.) The claim follows. ■

By the indistinguishability claim (Claim 1.2.1), 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 implies that A errs (w.p. greater than $1/3$) on some $x \in \mathtt{MAJ}$, or $\mathbf{Pr}[A(Z_n) = 1] > 1/3$, which implies that A errs (w.p. greater than $1/3$) on some $z \notin \mathtt{MAJ}$. The proposition follows. ■

Proposition 1.3 (Randomization is essential for Proposition 1.1): *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*:** Fixing an arbitrary *deterministic* algorithm A that makes $q < n/2$ queries, we shall show that if A accepts each input in MAJ, then it also accepts the all-zero

[4] See Exercise 1.2 for details regarding the equivalence of the alternative and original definitions of X_n (resp., of Z_n).

string, which is 0.5-far from MAJ. It will follow that A fails to distinguish between some inputs in MAJ and some inputs that are 0.5-far from MAJ.

Relying on the hypothesis that A is deterministic, we consider the unique execution of A in which all queries of A are answered with 0, 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 $x \in$ MAJ (since $\mathrm{wt}(x) = n - q > n/2$), whereas z is 0.5-far from MAJ. However, A behaves identically on x and z, since in both cases it obtains the answer 0 to each of its queries, which means that $A(x) = 1$ if and only if $A(z) = 1$. Hence, A errs either on x (which is in MAJ) or on z (which is 0.5-far from MAJ). The proposition follows. ■

Digest. We have seen that sublinear time (in fact constant-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 (and using randomized algorithms rather than deterministic ones).

1.2.4. Symmetries and Invariants

The proof of Proposition 1.1 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 other words, 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 1.4 (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 fixed alphabet.[5] The result does not generalize to sequences over unbounded alphabet. In fact, there exist symmetric properties over unbounded alphabet for which the approximate decision problem requires a linear number of queries (see Exercise 1.3).)

***Proof*:** The key observation is for every n there exists a set (of weights) $W_n \subseteq \{0, 1, \ldots, n\}$ such that for every $x \in \{0,1\}^n$ it holds that $x \in S$ if and only if $\mathrm{wt}(x) \in W_n$, where $\mathrm{wt}(x) = |\{i \in [n] : x_i \neq 0\}|$. (In the case of MAJ, the set W_n is

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