Refining the Adaptivity Notion in the Huge Object Model

Tomer Adar
Technion – Israel Institute of Technology, Haifa, Israel

Eldar Fischer
Technion – Israel Institute of Technology, Haifa, Israel

Abstract

The Huge Object model for distribution testing, first defined by Goldreich and Ron in 2022, combines the features of classical string testing and distribution testing. In this model we are given access to independent samples from an unknown distribution P over the set of strings $\{0,1\}^n$, but are only allowed to query a few bits from the samples. The distinction between adaptive and non-adaptive algorithms, which occurs naturally in the realm of string testing (while being irrelevant for classical distribution testing), plays a substantial role also in the Huge Object model.

In this work we show that the full picture in the Huge Object model is much richer than just that of the adaptive vs. non-adaptive dichotomy. We define and investigate several models of adaptivity that lie between the fully-adaptive and the completely non-adaptive extremes. These models are naturally grounded by observing the querying process from each sample independently, and considering the "algorithmic flow" between them. For example, if we allow no information at all to cross over between samples (up to the final decision), then we obtain the *locally bounded* adaptive model, arguably the "least adaptive" one apart from being completely non-adaptive. A slightly stronger model allows only a "one-way" information flow. Even stronger (but still far from being fully adaptive) models follow by taking inspiration from the setting of streaming algorithms. To show that we indeed have a hierarchy, we prove a chain of exponential separations encompassing most of the models that we define.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases Huge-Object model, Property Testing

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2024.45

Category RANDOM

Related Version *Full Version*: `https://arxiv.org/abs/2306.16129`

Funding *Eldar Fischer*: Research supported by an Israel Science Foundation grant number 879/22.

1 Introduction

Property testing is the study of sublinear, query-based probabilistic decision-making algorithms. That is, algorithms that return ACCEPT or REJECT after reading only a small portion of their input. The study of (classical) property testing, starting with [6], [14] and [15], has seen an extensive body of work. See for example [10]. Usually, a property-testing algorithm with threshold parameter ε is required to accept an input that satisfies the property with high probability, and reject an input whose distance from any satisfying one is more than ε, with high probability as well. For string properties, which were the first to be studied (along with functions, matrices, etc. that can also be represented as strings), the distance measure is usually the normalized Hamming distance.

Distribution testing is a newer model, first defined implicitly in [11] (a version of which has already appeared in 2000 as a technical report). In [4] and [5] it was explicitly defined and researched. The algorithms in this model are much weaker, where instead of queries, the decision to accept or reject must be made based only on a sequence of independent samples drawn from an unknown distribution. In such a setting the distance metric is usually the variation distance. For a more comprehensive survey, see [7].

The study of a combination of string and distribution testing was initiated in [12]. Here the samples in themselves are considered to be very large objects, and hence after obtaining a sample (usually modeled as a string of size n), queries must be made to obtain some information about its contents. This requires an appropriate modification in the distance notion. This model is appropriately called the *Huge Object model.*

Contrast the above to the original "small object" distribution testing model, where it is assumed that every sample is immediately available to the algorithm in its entirety. In particular, in the original model, the algorithm does not have any choice of queries, as it just receives a sequence of independent samples from the distribution to be tested. Hence one might even call it a "formula" rather than an "algorithm". Grossly speaking, the only decision made is whether to accept or reject the provided sequence of sampled objects.

On the other hand, in the string testing model, an algorithm is provided with a (deterministic) input string, and may make query decisions based both on internal random coins and on answers to previous queries. An algorithm which makes use of the option of considering answers to previous queries when choosing the next query is called adaptive, while an algorithm that queries based only on coin tosses is called non-adaptive (the final decision on whether to accept or reject the input must, of course, depend on the actual answers).

Algorithms for the Huge Object model, due to their reliance on individual queries to the provided samples, can be adaptive or non-adaptive. This relationship with respect to the Huge Object model was first explored in [8].

However, as we shall demonstrate below, the complete picture here is richer than the standard adaptive/non-adaptive dichotomy used in classical string testing. As it turns out, several categories of adaptivity can be defined and investigated based on the consideration of the shared information between the different samples that are queried.

1.1 Adaptivity notions in the Huge Object model

For our purpose, unless we state otherwise, we assume that the sequence of samples is taken in advance (but is not directly disclosed to the algorithm), and is presented as a matrix from which the algorithm makes its queries. For a sequence of s samples from a distribution whose base set is $\{0,1\}^n$, this would be a binary $s \times n$ matrix.

We say that an algorithm is *non-adaptive* if it chooses its entire set of queries before making them, which means that it cannot choose later queries based on the answers to earlier ones. This is identical to the definition of a non-adaptive algorithm for string properties.

A *fully adaptive* algorithm is allowed to choose every query based on answers to all queries made before it. This is quite similar to the definition of an adaptive algorithm for string properties, but restricting ourselves to this dichotomy does not give the full picture. We refine the notion of adaptivity by considering more subtle restrictions on the way that the algorithms plan their queries, leading to query models that are not as expressive as those of fully adaptive algorithms, but are still more expressive than those of non-adaptive ones. In this introduction we only introduce the rationale of every model; the formal definitions appear in the preliminaries section.

One interesting restriction, which is surprisingly difficult to analyze, is "being adaptive for every individual sample, without sharing adaptivity between different samples" (the results of random coin tosses are still allowed to be shared). We say that an algorithm is *locally-bounded* if it obeys this restriction. This model captures the concept of distributed execution, in a way that every node has a limited scope of a single sample, and only when all nodes are done, their individual outcomes are combined to facilitate a decision.

A more natural restriction is "being able to query only the most recent sample". We say that an algorithm is *forward-only* if it cannot query a sample after querying a later one. This can be viewed (if we abandon the above-mentioned matrix representation) as the algorithm being provided with oracle access to only one sample at a time, not being able to "go back in time" once a new sample was taken. An example for the usage of the model is an anonymous survey. As long as the survey session is alive, we can present new questions based on past interactions and on the current one, but once the session ends, we are not able to recall the same participant for further questioning.

A natural generalization of forward-only adaptiveness is having a bounded memory for holding samples (rather than only having one accessible sample at a time). Once the memory is full, the algorithm must drop one of these samples (making it inaccessible) in order to free up space for a new sample. An additional motivation for this model is the concept of stream processing, whose goal is computing using sublinear memory. Relevant to our work is [2], where the input stream is determined by an unknown distribution, in contrast to the usual streaming setting where the order of the stream is arbitrary. Within the notion of having memory of a fixed size, we actually distinguish two models. In the weak model, when the memory is full, the oldest sample is dropped. In the strong model, the algorithm decides (possibly adaptively) which sample to drop.

We show that every two consecutive models in the above hierarchy have an exponential separation, which means that there is a property that requires $\Omega(\text{poly}(n))$ queries for an ε-test in the first model (for some fixed ε), but is also ε-testable using $O\left(\text{poly}\left(\varepsilon^{-1}\right)\log n\right)$ queries in the second model (for every $\varepsilon > 0$). Moreover, our upper bounds always have one-sided error, while the lower bounds apply for both one-sided and two-sided error algorithms. The exact relationship between the weak and the strong limited memory models remains open, however.

We believe that investigating limited adaptiveness models can apply to other areas where there are two "query scales". That is, when investigating a model takes into account collections of objects that are restricted both in the way that whole objects are obtained and in the access model *inside* each obtained object. For example, one could think of a distributed computing scenario where the communication between the nodes follows a LOCAL or a CONGEST scheme (see [13]), but additionally each node holds a "large" input from which it may only perform sub-linear time computation *between* the communication rounds.

1.2 Organization of the paper

We start with formal definitions of the models which are required to state our results, followed by an overview of the results themselves and a description of the main ideas of their proofs. This review includes the definitions of the properties showing our separation results, along with a sketch of the lower bounds and the algorithms for the upper bounds. The proofs themselves are deferred to the full version of this paper.

2 Preliminaries

The following are the core definitions and lemmas used throughout this paper, including the model definitions used in the overview in Section 3. Here, all distributions are defined over finite sets.

▶ **Definition 1** (Common notations). *For a set A,* the power set of A *is denoted by $\mathcal{P}(A)$. For two sets A and B, the set of all functions $f : A \to B$ is denoted by B^A. For a finite set A, the* set of all permutations over A *is denoted by $\pi(A)$.*

▶ **Definition 2** (Set of distributions). *Let Ω be a finite set. The* set of all distributions that are defined over Ω *is denoted by $\mathcal{D}(\Omega)$.*

While parts of this section are generalizable to distributions over non-finite sets Ω with compact topologies, we restrict ourselves to distributions over finite sets, which suffice for our application.

▶ **Definition 3** (Property). *A* property $\mathcal{P}$ *over a finite alphabet Σ is defined as a sequence of compact sets $\mathcal{P}_n \subseteq \mathcal{D}(\Sigma^n)$. Here* compactness *refers to the one defined with respect to the natural topology inherited from $\mathbb{R}^{|\Sigma|^n}$.*

All properties are defined over $\Sigma = \{0, 1\}$ unless we state otherwise.

2.1 Distances

The following are the distance measures that we use. In the sequel, we will omit the subscript (e.g. use "$d(x, y)$" instead of "$d_{\mathrm{H}}(x, y)$") whenever the measure that we use is clear from the context.

▶ **Definition 4** (Normalized Hamming distance). *For two strings $s_1, s_2 \in \Sigma^n$, we use $d_{\mathrm{H}}(s_1, s_2)$ to denote their* normalized Hamming distance, $\frac{1}{n} |\{1 \le i \le n | s_1[i] \neq s_2[i]\}|$.

For all our distance measures we also use the standard extension to distances between sets, using the corresponding infimum (which in all our relevant cases will be a minimum). For example, For a string $s \in \{0, 1\}^n$ and a set $A \subseteq \{0, 1\}^n$, we define $d_{\mathrm{H}}(s, A) = \min\limits_{s' \in A} d_{\mathrm{H}}(s, s')$.

▶ **Definition 5** (Variation distance). *For two distributions P and Q over a common set Ω, we use $d_{\mathrm{var}}(P, Q)$ to denote their* variation distance, $\max_{E \subseteq \Omega} |\Pr_P[E] - \Pr_Q[E]|$. *Since Ω is finite there is an equivalent definition of $d_{\mathrm{var}}(P, Q) = \frac{1}{2} \sum_{s \in \Omega} |P(s) - Q(s)|$.*

▶ **Definition 6** (Transfer distribution). *For two distributions P over Ω_1 and Q over Ω_2, we say that a distribution T over $\Omega_1 \times \Omega_2$ is a* transfer distribution *between P and Q if for every $x_0 \in \Omega_1$, $\Pr_{(x,y)\sim T}[x = x_0] = \Pr_P[x_0]$, and for every $y_0 \in \Omega_2$, $\Pr_{(x,y)\sim T}[y = y_0] = \Pr_Q[y_0]$. We use $\mathcal{T}(P, Q)$ to denote the set of all transfer distributions between P and Q.*

We note that for finite Ω_1 and Ω_2 the set $\mathcal{T}(P, Q)$ is compact as a subset of $\mathcal{D}(\Omega_1 \times \Omega_2)$.

▶ **Definition 7** (Earth Mover's Distance). *For two distributions P and Q over a common set Ω with a metric d_Ω, we use $d_{\mathrm{EMD}}(P, Q)$ to denote their* earth mover's distance, *defined by the infimum of the "average distance" demonstrated by a transfer distribution,* $\inf_{T \in \mathcal{T}(P,Q)} \mathrm{E}_{(x,y)\sim T} [d_\Omega(x, y)]$.

In the sequel, the above "inf" can and will be replaced by "min", by the compactness of $\mathcal{T}(P, Q)$ for finite Ω. Most papers (including the original [12]) use an equivalent definition that is based on linear programming, whose solution is the optimal transfer distribution.

In our theorems, Ω is always $\{0, 1\}^n$ for some n and the metric is the Hamming distance. Sometimes, as an intermediate phase, we may use a different Ω (usually $\{1, \dots, k\}^n$ for some k), and then show a reduction back to the binary case.

▶ **Definition 8** (Distance from a property). *The* distance *of a distribution P from a property $\mathcal{P} = \langle \mathcal{P}_n \rangle$ is loosely noted as $d(P, \mathcal{P})$ and is defined to be $d_{\mathrm{EMD}}(P, \mathcal{P}_n) = \inf_{Q \in \mathcal{P}_n} d_{\mathrm{EMD}}(P, Q)$.*

It is very easy to show that for any two distributions $P, Q \in \mathcal{D}(\Sigma^n)$ we have $d_{\mathrm{EMD}}(P, Q) \leq d_{\mathrm{var}}(P, Q)$. This means that the topology induced by the variation distance is richer than that induced by the earth mover's distance (actually for finite sets these two topologies are identical). In particular it means that all considered properties form compact sets with respect to the earth mover's distance. We obtain the following lemma.

▶ **Lemma 9.** *For a property $\mathcal{P}$ of distributions over strings, and any distribution $P \in \mathcal{D}(\Sigma^n)$, there is a distribution realizing the distance of P from $\mathcal{P}$, i.e. a distribution $Q \in \mathcal{P}_n$ for which $d(P, Q) = d(P, \mathcal{P}_n)$. In particular, the infimum in Definition 8 is a minimum.*

2.2 The testing model

This model is defined in [12]. We use an equivalent definition which will be the "baseline" for our restricted adaptivity variants.

The input is a distribution P over Σ^n (our final theorems will be for $\Sigma = \{0, 1\}$, but some intermediate arguments require other finite Σ). An algorithm $\mathcal{A}$ gets random oracle access to s samples that are independently drawn from P. Then it is allowed to query individual bits of the samples. The output of the algorithm is either ACCEPT or REJECT. For convenience we identify the samples with an $s \times n$ matrix, so for example the query "(i, j)" returns the jth bit of the ith sample.

The input size n and the number of samples s are hard-coded in the algorithm. As with boolean circuits, an algorithm for an arbitrarily sized input is defined as a sequence of algorithms, one for each n.

For a given algorithm we define another measure of complexity, which is the total number of queries that the algorithm makes. Without loss of generality, we always assume that every sample is queried at least once (implying that $q \geq s$).

For a property $\mathcal{P}$ and $\varepsilon > 0$, we say that an algorithm $\mathcal{A}$ is an ε-test if:

- For every $P \in \mathcal{P}$, $\mathcal{A}$ accepts the input P with probability higher than $\frac{2}{3}$.
- For every P that is ε-far from $\mathcal{P}$, $\mathcal{A}$ accepts the input P with probability less than $\frac{1}{3}$.

We say that $\mathcal{A}$ is an ε-test with one sided error if:

- For every $P \in \mathcal{P}$, $\mathcal{A}$ accepts the input P with probability 1.
- For every P that is ε-far from $\mathcal{P}$, $\mathcal{A}$ accepts the input P with probability less than $\frac{1}{2}$.

The choice of the probability bounds in the above definition are somewhat arbitrary. For the one sided error definition $\frac{1}{2}$ is more convenient than $\frac{1}{3}$. We also note that for non-ε-far inputs that are not in $\mathcal{P}$, any answer by $\mathcal{A}$ is considered to be correct.

2.3 Restricted models

As observed by Yao in [16], every probabilistic algorithm can be seen as a distribution over the set of allowable deterministic algorithms. This simplifies the algorithmic analysis, since we only have to consider deterministic algorithms (a distinction between public and private coins can break this picture, but this will not be the case here). We will use Yao's observation to define every probabilistic algorithmic model by defining its respective set of allowable deterministic algorithms. The following definitions formalize the description of the models introduced in Section 1.

▶ **Definition 10** (Fully adaptive algorithm). *Every deterministic algorithm can be described as a full decision tree T and a set A of accepted leaves. Without loss of generality we assume that all leaves have the exactly the same depth (we use dummy queries if "padding" is needed). Every internal node of T consists of a query $(i, j) \in \{1, \dots, s\} \times \{1, \dots, n\}$ (the jth bit of the ith sample), and every edge corresponds to an outcome element (in Σ). The number of queries q is defined as the height of the tree. Every leaf can be described by the string of length q detailing the answers given to the q queries, corresponding to its root-to-leaf path. Thus we can also identify A with a subset of Σ^q.*

Now that we have defined the most general form of a deterministic algorithm in the Huge Object model, we formally define our models for varying degrees of adaptivity.

▶ **Definition 11** (Non-adaptive algorithm). *We say that an algorithm is* non-adaptive *if it chooses its queries in advance, rather than deciding each query location based on the answers to its previous ones. Formally, every deterministic non-adaptive algorithm is described as a pair (Q, A) such that $Q \subseteq \{1, \dots, s\} \times \{1, \dots, n\}$ (for some sample complexity s) is the set of queries, and $A \subseteq \Sigma^Q$ is the set of accepted answer functions. The query complexity is defined as $q = |Q|$.*

▶ **Definition 12** (Locally-bounded adaptive algorithm). *We call an algorithm* locally-bounded *if it does not choose its queries to a sample based on answers to queries in other samples. Formally, every s-sample deterministic locally-bounded algorithm is a tuple $(T_1, \dots, T_s; A)$, where every T_i is a decision tree of height q_i (where $q = \sum_{i=1}^{s} q_i$ is the total number of queries) that is only allowed to query the ith sample, and $A \subseteq \Sigma^q$ represents a set of accepted superleaves, where a superleaf is defined as the concatenation of the $q_1, \dots, q_s$ symbol long sequences that represent the leaves of trees $T_1, \dots, T_s$ respectively.*

▶ **Definition 13** (Forward-only adaptive algorithm). *We call an algorithm* forward-only *if it cannot query a sample after querying a later one. Formally, a forward-only algorithm for s samples of n-length strings is defined as a pair (T, A), where T is a decision tree over $\{1, \dots, s\} \times \{1, \dots, n\}$ and $A \subseteq \Sigma^q$ (as with general adaptive algorithms), additionally satisfying that for every internal node of T that is not the root, if its query is (i, j) and its parent query is (i', j'), then $i' \le i$.*

▶ **Definition 14** (Weak memory-bounded adaptive algorithm). *We say that an algorithm is* weak m-memory bounded *if it can only query a sliding window of the m most recent samples at a time. Formally, a weak m-memory-bounded adaptive algorithm using s samples of n-length strings is defined as a pair (T, A), where T is a decision tree over $\{1, \dots, s\} \times \{1, \dots, n\}$ and $A \subseteq \Sigma^q$ (as with general adaptive algorithms), additionally satisfying that for every internal node of T that is not the root, if its query is (i, j), then for every ancestor whose query is (i', j'), it holds that $i' - m < i$.*

▶ **Definition 15** (Strong memory-bounded adaptive algorithm). *A* strong memory-bounded adaptive algorithm *for s samples of n-length strings is defined as a triplet (T, A, M) where T is a decision tree, $A \subseteq \Sigma^q$ is the set of accepted answer vectors, and $M : \mathrm{nodes}(T) \to \mathcal{P}(\{1, \dots, s\})$ is the "memory state" at every node. The explicit rules of M are:*

- *For every internal node $u \in T$, $|M(u)| \le k$ (there are at most k samples in memory).*
- *For every internal node $u \in T$, if $i \in M(u)$, and if v is a child of u for which $i \notin M(v)$, then for every descendant w of v, $i \notin M(w)$ (a "forgotten" sample cannot be "recalled").*
- *For every internal node $u \in T$ whose query is (i, j), $i \in M(u)$ (the ith sample must be in memory in order to query it).*

Without loss of generality, because the samples are independent, we can assume that:

- $M(\text{root}) = \{1, \ldots, k\}$ (the algorithm has initial access to the first k samples).
- For every internal node $u \in T$ and the set V of all its ancestors, it holds that $\max(M(u)) \leq 1 + \max_{v \in V}(\max M(v))$ (new samples are accessed "in order").

3 Overview of results and methods

The following is a semi-formal overview of our work, which is described in extensive details in the full version of the paper. Most of our results are exponential separations between models (that is, $O(\log n)$ vs $n^{\Omega(1)}$ bounds).

All separations are with an exponential gap, and are achieved by properties that have an efficient 1-sided error test in one model, but do not even have an efficient 2-sided test in the other model.

Figure 1 provides a visualization of our results. More details about the difference between the weak k-memory and the strong k-memory model are provided below.

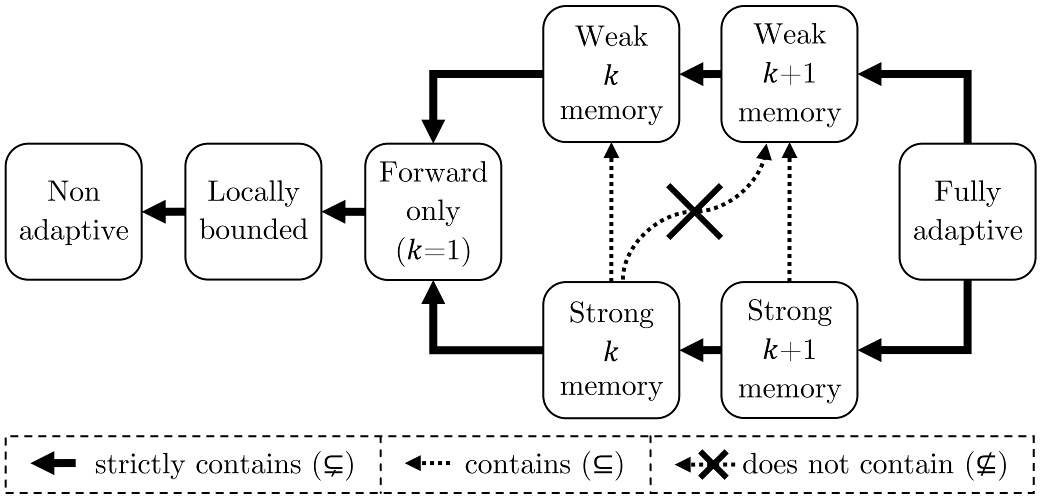

Figure 1 Graphical summary of our results.

3.1 Non-adaptive algorithms

To showcase what can be done with non-adaptive algorithms, we analyze the property of a distribution having support size at most m, and the even more basic property of a distribution being *deteministic*, that is, having support size 1.

We show that the determinism property (the property that the distribution draws a specific element with probability 1) can be tested non-adaptively using $O(\varepsilon^{-1})$ queries, consisting of $O(\varepsilon^{-1})$ samples (as in the classic model) and $O(1)$ queries per sample.

▶ **Observation 16.** *The property of drawing a fixed string has a one-sided error non-adaptive ε-test that uses $O(\varepsilon^{-1})$ queries.*

We also show a non-adaptive m-support test.

▶ **Theorem 17.** *The property of being supported on a set of at most m elements has a one-sided error non-adaptive ε-test that uses $O(\varepsilon^{-2} m \log m)$ queries.*

Algorithm 1 demonstrates this upper bound.

Algorithm 1 One sided ε-test for m-bounded support, non adaptive, $O(\varepsilon^{-2} m \log m)$ queries.

```
take $s = 1 + \lceil 8\varepsilon^{-1} m \rceil$ samples.
let $t = \lceil 4\varepsilon^{-1}(\ln m + 2) \rceil$
choose $j_1, \dots, j_t \in [n]$ uniformly and independently at random.
let $J = \{j_1, \dots, j_t\}$
for $i$ from 1 to $s$ do
    query sample $i$ at $j$ for every $j \in J$, giving substring $y^i$ of length $|J|$.
if $|\{y^1, \dots, y^s\}| > m$ then
    return REJECT
return ACCEPT
```

As described in detail in the full version of the paper, in which we prove the correctness of Algorithm 1, our ε-test for the m-support property needs more than a fixed number of queries per sample. Though not necessarily optimal, this algorithm demonstrates the core difference between the Huge Object model and the classic one: the limited ability to distinguish different samples. This limitation holds for adaptive algorithms as well, even though the adaptivity can reduce the number of queries per sample for some properties. A concurrent work [1] shows a lower bound of $\Omega(\varepsilon^{-1} \log \varepsilon^{-1})$ queries for non-adaptive support testing even for $m = 2$, showing that this limitation is unavoidable.

Locally bounded adaptive algorithms

The locally-bounded adaptive model (Definition 12) allows the algorithm to pick its queries based on answers to previous queries for every *fixed* sample, but lacks the ability to pass information between samples. The ability of being adaptive allows the algorithm more ways to query its samples, but it still lacks the ability to test *relations* between the samples.

Analysis method

To analyze the locally-bounded model, we define an intermediate model of string testing which we call the *split-adaptive model.*

▶ **Definition 18** (Split adaptive algorithm). *For a fixed k, a k-*split adaptive *deterministic algorithm for n-long strings (where n is divisible by k) over some alphabet Σ is a sequence of k decision trees $T_1, \dots, T_k$, where the tree T_i can only query at indexes between $(i-1)k+1$ and ik, and a set of accepted answer sequences. The query complexity of the algorithm is defined as the sum of heights of its trees.*

In this model, we test properties of k-tuples of strings, where the queries are made separately for every entry of the tuple (that is, every entry is processed using an adaptive algorithm that is oblivious of the other entries). To obtain a reduction, we consider every s-sample locally-bounded algorithm over an input distribution P as a split-adaptive algorithm whose input is drawn from P^s (that is, an s-tuple whose entries are independently drawn from P).

Exponential separation from the non-adaptive model

Naturally, there is an exponential separation between the locally-bounded model and the non-adaptive model of the Huge Object model. The property **CPal** (defined below) demonstrates this separation.

▶ **Definition 19** (string property **cpal**, see [8], [3]). *For any fixed n, the property* **cpal** *is defined over $\{0,1,2,3\}^n$ as the set of n-long strings that are concatenations of a palindrome over $\{0,1\}$ and a palindrome over $\{2,3\}$ (in this order).*

The following lemma is well-known (the adaptive bound, using binary search, is described in [8]).

▶ **Lemma 20.** *Property* **cpal** *does not have a non-adaptive $\frac{1}{5}$-test using $o(\sqrt{n})$ queries, while having an adaptive ε-test using $O(\log(n) + 1/\varepsilon)$ many queries.*

In [8] this was made into a distribution property by using "distributions" that are deterministic.

▶ **Definition 21** (Distribution property **CPal**, see [8]). *For a fixed, even n, the property* **CPal** *is defined as the set of distributions over $\{0,1\}^n$ that are deterministic (have support size 1), whose support is an element that belongs to* **cpal***, with respect to the encoding $(0,1,2,3) \mapsto (00,01,10,11)$.*

▶ **Lemma 22.** **CPal** *has a locally-bounded ε-test that uses $O(\mathrm{poly}(\varepsilon^{-1})\log n)$ queries for every $\varepsilon > 0$, but there exists some $\varepsilon_0 > 0$ for which any non-adaptive ε_0-test requires $\Omega(\mathrm{poly}(n))$ queries.*

This is an almost-direct corollary of a result from [12] regarding converting string testing problems to the Huge Object model. Essentially, the Huge Object model "contains" the string testing one, and the conversion produces locally adaptive algorithms out of their respective adaptive string algorithms.

Forward only adaptive algorithms

In the forward-only model (Definition 13), the algorithm virtually gets a stream of samples, and is allowed to query only the current sample without any restriction (but further queries to past samples are not allowed), based on answers to all past queries. In contrast to the locally bounded model, forward algorithms can test a richer collection of binary relations between samples, due to the ability to query one sample and then use the gathered data to choose the queries for the next one.

Exponential separation from the locally-bounded model

We use the ability of forward-only algorithms to consider a richer collection of relations between samples, as compared to locally-bounded algorithms, to show an exponential separation between these models. The property **Inv*** (defined below) demonstrates this separation.

In [9] it was shown that ε-testing two functions over $\{1,\ldots,n\}$ for being inverses of each other is possible with $O(\varepsilon^{-1})$ many queries, while testing a single function for having an inverse is harder and requires a polynomial number of queries. Formally, we cite the function property **inv**:

▶ **Definition 23** (Function property **inv**). *For a fixed n, the property* **inv** *is defined over $[n]^{[2n]}$ as the set of ordered pairs of functions $f, g : [n] \to [n]$ such that either $f(i) = g(i)$ for every $1 \leq i \leq n$ or $g(f(i)) = i$ for every $1 \leq i \leq n$.*

Note that we modified the definition of the property slightly from the original, by allowing also the case $f = g$. This technical change makes it possible to construct a test using forward-only adaptivity that is also with one-sided error.

Here we separate the two functions by setting them in a probability space with support size 2. If we allow forward-only adaptivity, then the original inverse test can be implemented, as it works by verifying that $g(f(i)) = i$ for sufficiently many is. We can call the first sample "f", and after writing down our $f(i_1), \ldots, f(i_q)$, we "wait" for a sample of g and then verify that $g(f(i_j)) = i_j$ for $i_1, \ldots, i_q$. Formally, we define the property **Inv**:

▶ **Definition 24** (Distribution property **Inv**). *For a fixed n, the property* **Inv** *is defined as the set of distributions over $[n]^{[n]}$ that are supported by a set of the form $\{f, g\}$ such that $(f, g) \in$* **inv**. *Note that in particular all deterministic distributions satisfy* **Inv**, *since we allow $f = g$ to occur.*

To make the above work for binary strings (rather than an alphabet of size n) we use an appropriate large distance encoding of the values.

▶ **Definition 25** (Distribution property **Inv***). *For a fixed n, let $C_n : [n] \to \{0,1\}^{2\lceil \log_2 \rceil n}$ be an error-correction code whose distance is at least $\frac{1}{3}$. We define* **Inv*** *as the property of distributions over $\{0,1\}^{2\lceil \log_2 n \rceil n}$ that can be constructed by the following procedure: beginning with some $P \in$* **Inv**, *we let P^* denote the distribution that draws $x \in [n]^n$ according to P, and then outputs the concatenation $C_n(x_1) \cdots C_n(x_n)$.*

The lower bound against locally-bounded adaptivity requires an intricate analysis of the model. Essentially we use the split-adaptive string-testing model to show that when querying each of f and g "in solitude", being adaptive over a function that is drawn at random does not provide an advantage over a non-adaptive algorithm. In particular, the values of a uniformly drawn permutation are "too random" to allow the implementation of a meaningful query strategy without getting some information from the inverse function, even if we allow to "coordinate in advance" the query strategy.

▶ **Theorem 26.** *Property* **Inv*** *has a forward-only ε-test that uses $O(\varepsilon^{-2} \log n)$ queries for every $\varepsilon > 0$, but any locally-bounded adaptive $\frac{1}{5}$-test requires $\Omega(\sqrt{n})$ queries.*

The upper bound for forward-only testing of **Inv** is demonstrated in Algorithm 2. Applying Algorithm 2 to **Inv*** is pretty straightforward.

Algorithm 2 One sided ε-test for **Inv**, forward only, $O(\varepsilon^{-2})$ queries.

```
Treat samples as $n$-long strings over $[n]$.
let $s = 1 + \lceil 3\varepsilon^{-2} \rceil$.
choose $j_2, \ldots, j_s \in [n]$, uniformly at random and independently.
choose $k_2, \ldots, k_s \in [n]$, uniformly at random and independently.
query sample 1 at $j_2, \ldots, j_s$, giving $f(j_2), \ldots, f(j_s)$.
query sample 1 at $k_2, \ldots, k_s$, giving $f(k_2), \ldots, f(k_s)$.
for $i$ from 2 to $s$ do
    query sample $i$ at $j_i$, $f(k_i)$, giving $g(j_i)$, $g(f(k_i))$.
    if $f(j_i) \neq g(j_i)$ and $g(f(k_i)) \neq k_i$ then
        return REJECT
return ACCEPT
```

The query foresight method

Some adaptive algorithms do not obey the forward only restriction but can be modified to do so, using a method we call *query foresight*. Intuitively, an adaptive algorithm that has some knowledge about the structure of the queries it may make in the future can make them speculatively at present (that is, we make all potential queries to satisfy the forward-only constraint, even though we believe that some of them will later be considered as irrelevant). The more knowledge the algorithm has about the potential future queries, the less queries are wasted on the current sample.

As an example to the query foresight method, we analyze and convert a fully adaptive algorithm for the m-support property (Algorithm 3)

Algorithm 3 One sided ε-test for m-bounded support, strong $m+1$-memory, $O(\varepsilon^{-1}m^2)$ queries.

```
Memory storage for samples: $z^1, \dots, z^m; x$, all initialized to NULL.
Extra cell: We have another syntactic "write-only" memory storage $z^{m+1}$ which we never query.
take $s = 1 + \lceil 2\varepsilon^{-1}m \rceil$ samples.
set $c, t \leftarrow 0$.
set $j_1, \dots, j_m \leftarrow$ NULL
for $k$ from 1 to $s$ do
    Invariant 1 $c = m$ or $z^{c+1} =$ NULL.
    Invariant 2 for $1 \le i \le c$, $z^i_J$ are distinct where $J = \{j_1, \dots, j_t\}$.
    store $x \leftarrow$ sample $k$.
    query $x$ at $j_1, \dots, j_t$, giving substring $y^k$.
    for $i$ from 1 to $c$ do
        query sample $z^i$ at $j_1, \dots, j_t$ giving substring $y^i$.      ▷ the $y^i$s are distinct
    choose $j \in [n]$ uniformly at random.
    query $x$ at $j$, giving $x_j$.
    if $\exists i : y^i = y^k$ then                                          ▷ if exists it is unique
        query sample $z^i$ at $j$ giving $z^i_j$.
        if $x_j \ne z^i_j$ then
            store $z^{c+1} \leftarrow x$.
            set $j^{t+1} \leftarrow j$.                                     ▷ keep Invariant 2
            set $t \leftarrow t + 1$ and $c \leftarrow c + 1$.              ▷ keep Invariant 1
    else
        store $z^{c+1} \leftarrow x$.                                        ▷ Invariant 2 still holds
        set $c \leftarrow c + 1$.                                            ▷ keep Invariant 1
    if $c > m$ then
        return REJECT
return ACCEPT
```

▶ **Theorem 27.** *Algorithm 3 is a one-sided ε-test for being supported by at most m elements.*

We observe that the general structure of Algorithm 3's queries is highly predictable, and provide a modified version thereof (Algorithm 4) which is also forward-only, without increasing its worst-case query complexity.

The idea is straightforward: we simulate the run of an adaptive algorithm. Every time that the simulation is about to query a new sample, we make additional speculative queries in the current sample, before dropping it as per the requirement of a forward-only algorithm.

If the simulated algorithm makes a query to an old sample, we feed it with the answer of the corresponding speculative query. If such a speculative query does not exists, we either accept (for one-sided algorithms) or behave arbitrarily (for two-sided algorithms). If the prediction is conservative, that is, the speculated queries are ensured to cover all queries to past samples, then the construction guarantees the exact acceptance probability for every individual input. This is not guaranteed when the prediction is not conservative, and in this case we need to analyze the effect of bad speculations.

Algorithm 4 One sided ε-test for m-bounded support, forward only, $O(\varepsilon^{-1}m^2)$ queries.

```
take $s = 1 + \lceil 2\varepsilon^{-1}m \rceil$ samples.
choose $j_1, \dots, j_s \in [n]$ uniformly and independently at random.
let $M$ be an uninitialized $m \times n$ sparse matrix $\{0,1\}$.    ▷ storage for speculative queries
let $A$ be an empty list over $[n]$.
$c \leftarrow 0$.
for $k$ from 1 to $s$ do
    Invariant $M_{i,j}$ is initialized for all $1 \le i \le c$ and $j \in \{j_1, \dots, j_s\}$.
    for all $j$ in $A$ do                                            ▷ simulation of $y^k$
        query sample $k$ at $j$, giving $x^k_j$.
    set found $\leftarrow 0$.
    for $i$ from 1 to $c$ do
        if $\bigwedge_{j \in A} (M_{i,j} = x^k_j)$ then                ▷ simulation of the $y^i$s
            set found $\leftarrow 1$.
            $j \leftarrow j_k$.
            query sample $k$ at $j$, giving $x^k_j$.
            if $M_{i,j} \ne x^k_j$ then
                $c \leftarrow c + 1$.
                add $j$ to $A$.
                query sample $k$ at $j_1, \dots, j_s$, giving $M_{c,j_1}, \dots, M_{c,j_s}$.  ▷ speculative queries
                                                                     ▷ keep the invariant
    if found $= 0$ then
        $c \leftarrow c + 1$.
        query sample $k$ at $j_1, \dots, j_s$, giving $M_{c,j_1}, \dots, M_{c,j_s}$.   ▷ speculative queries
                                                                     ▷ keep the invariant
    if $c > m$ then
        return REJECT
return ACCEPT
```

k-bounded memory algorithms

As per Definitions 14 and 15 we have two models of bounded memory, which we call *weak* and *strong* respectively. Intuitively, in both models, the algorithm gets a stream of samples, and it has an unrestricted access to k of these samples. When the algorithm needs an access to a new sample, it must give up the ability to access one of the past samples. In the weak model, the algorithm does not have a choice and it must drop the earliest sample. In other words, the weak model has an unrestricted access to a sliding window of the k most recent samples. In the strong model, the algorithm is allowed to choose the sample to drop.

For $k = 1$, the weak and strong models are both equal to each other and to the forward-only model. Intuitively, as k increases, the algorithm is able to consider more complicated relations between samples, especially k-ary relations, which are more challenging for $k-1$-memory algorithms.

Exponential separation from the forward-only model

We use the ability to fully consider binary relations using 2-memory algorithms, compared to the limited ability to do so using forward-only algorithms, to establish an exponential separation between them.

We define a property **Sym** that catches the idea of symmetric functions. For some symmetric function $f : [m] \times [m] \to \{0, 1\}$, a distribution in the property draws a random key $a \in [m]$ and returns a vector that contains both a (using a high distance code of length m) and all values of f at points (a, b) for $b \in [m]$. For technical reasons, we use fixed-distance systematic codes to encode a as a part of the row.

▶ **Lemma 28** (Systematic code)**.** *There exists a set $\mathcal{C}$ of error correction codes, such that for every $n \geq m \geq 10$, it has a code $C_{m,n} : [m] \to \{0, 1\}^n$ with the following properties: (1) Its minimal codeword distance is at least $\frac{1}{3}$ and (2) The projection of $C_{m,n}$ on its first $\lceil \log_2 m \rceil$ is one-to-one, that is, $C_{m,n}$ can be decoded by reading the first $\lceil \log_2 m \rceil$ bits.*

From now on, every use of systematic codes refers to the set $\mathcal{C}$ that is guaranteed by Lemma 28, usually denoted just by C (rather than the explicit notion $C_{m,n}$).

▶ **Definition 29** (Matrix property **sym**)**.** *For a fixed n, the property* **sym** *of functions with two variables $f : [n]^2 \to \{0, 1\}$ is defined as the property of being symmetric, i.e. satisfying $f(i, j) = f(j, i)$ for all $i, j \in [n]$.*

The corresponding distribution property is inspired by considering distributions over the rows of a symmetric matrix, along with properly encoded identifiers.

▶ **Definition 30** (Distribution property **Sym**)**.** *For any m and the systematic code $C : [m] \to \{0, 1\}^m$ from Lemma 28, the property* **Sym** *is defined as the set of distributions for which*

$$\Pr_{x \sim P} [\exists a \in [m] : x_{1,\dots,m} = C(a)] = 1$$

(all vectors start with an encoding of a "row identifier"), and for every $a, b \in [m]$,

$$\Pr_{x,y \sim P} [x_{1,\dots,m} = C(a) \wedge y_{1,\dots,m} = C(b) \wedge x_{m+b} \neq y_{m+a}] = 0$$

(if two "identifiers" a and b appear with positive probability, then the respective "$f(a, b)$" and "$f(b, a)$" are identical).

▶ **Theorem 31.** *There exists a one-sided weak 2-memory ε-test for* **Sym** *that makes $O(\varepsilon^{-2} \log n)$ queries, but every forward-only $\frac{1}{14}$-test for* **Sym** *must use at least $\Omega(\sqrt{m})$ queries (for sufficiently large m).*

The lower bound follows from a forward-only algorithm being given access to every sample without any knowledge about the keys of "future" samples. If the algorithm has only one accessible sample at a time, it can only "guess" the other key, but the probability to actually draw a later sample with that key is too low, unless the algorithm collects queries according to about $\sqrt{m}$ guessed keys.

For the upper-bound, Algorithm 5 performs a sequence of independent iterations using two samples at a time. In every iteration, it gathers their "keys" a_1 and a_2, verifies the correctness of their codewords, and then checks whether $f(a_1, a_2) = f(a_2, a_1)$. There are some cases that should be carefully analyzed, for example the case where the distribution does not correspond to a single f, or the case where some values for "a" appear very rarely or not at all, but these do not defeat the above algorithm (they somewhat affect its number of needed iterations).

Algorithm 5 One-sided ε-test for **Sym**, weak 2-memory, $O(\varepsilon^{-2} \log n)$ queries.

```
let $m \leftarrow n/2$.
for $\lceil 8\varepsilon^{-2} \rceil$ times do
    take two samples $x$, $y$.
    query $x_1, \ldots, x_{\lceil \log_2 m \rceil}$, giving $\kappa(x)$ as $a$.
    query $y_1, \ldots, y_{\lceil \log_2 m \rceil}$, giving $\kappa(y)$ as $b$.
    choose $i \in [m]$, uniformly at random.
    query $x$, $y$ at $i$, giving $x_i$, $y_i$.
    query $\phi_x(b)$, $\phi_y(a)$.
    if $x_i \neq (C(a))_i$ or $y_i \neq (C(b))_i$ then
        return REJECT                                  ▷ rejection by key invalidity
    if $\phi_x(b) \neq \phi_y(a)$ then
        return REJECT                                  ▷ rejection by asymmetry
return ACCEPT
```

Larger memory generalization

We generalize the above theorem to state an exponential separation between the k-weak model (Definition 14) and the $k-1$-strong model (Definition 15). We define a property $\mathbf{Par}_k$ based on parity for every $k \geq 2$ for which:

▶ **Theorem 32.** *For every $k \geq 2$, there exists a one-sided weak k-memory ε-test for $\mathbf{Par}_k$ that makes $O(k\varepsilon^{-k} \log n)$ queries, but every forward-only $\frac{1}{6k}$-test for $\mathbf{Par}_k$ must use at least $\Omega(\sqrt{m})$ (for sufficiently large n), which is $\Omega(n^{1/2k})$ queries since $n \approx \binom{m}{k-1}$.*

To motivate the definition of $\mathbf{Par}_k$, suppose that $f : \binom{[m]}{k} \to \{0,1\}^k$ is a function such that $f(A)$ has zero parity for every subset $A \subseteq [m]$ of size k. We "encode" such a function as a distribution, making sure to "separate" the k bits of $f(A)$ to k different samples. A typical sample in the distribution would have an encoding (using a high distance code) of a random key $a \in [m]$, followed by some information on $f(A)$ for every A that contains a. Specifically, for each such A we supply the ith bit of $f(A)$, where i is the "rank" of a in A (going by the natural order over $[m]$).

▶ **Definition 33** (Preliminaries for distribution property $\mathbf{Par}_k$). *Let $k \geq 2$ be the degree of freedom in the represented function. Let m be the (sufficiently large) size of the input set and $n = \binom{m-1}{k-1}$. Also, consider a systematic code $C : [m] \to \{0,1\}^n$.*

For a string $x \in \{0,1\}^{2n}$, let $\kappa(x) = C^{-1}(x_{1,\ldots,n})$ be the key *of x (if the inverse function is not defined we can use an arbitrary key instead). Since we have an implicit mapping between $k-1$-subsets of $\{1, \ldots, m\} \setminus \{\kappa(a)\}$ and the indexes $\{1, \ldots, n\}$, for every $A \subseteq \{1, \ldots, m\} \setminus \{\kappa(a)\}$ of size $k-1$ we can define $\Phi_A(x)$ as the corresponding bit in $x_{n+1,\ldots,2n}$.*

▶ **Definition 34** (Distribution property $\mathbf{Par}_k$). *For k, m, n, C defined above, the property $\mathbf{Par}_k$ is defined as the set of distributions over $\{0,1\}^{2n}$ for which*

$$\Pr_{x \sim P}[\exists a \in [m] : x_{1,\ldots,n} = C(a)] = 1$$

(all vectors start with an encoding of a "row identifier"), and for every $a_1 < \ldots < a_k \in [m]$*,*

$$\Pr_{x^1,\ldots,x^k \sim P}\left[\bigwedge_{i=1}^{k} (x^i)_{1,\ldots,n} = C(a_i) \wedge \bigoplus_{i=1}^{k} \Phi_{\{a_1,\ldots,a_k\}\setminus\{a_i\}}(x^i) = 1\right] = 0$$

(if all "identifiers" $a_1, \ldots, a_k$ *appear with positive probability, then the respective concatenation of values which forms "*$f(\{a_1, \ldots, a_k\})$*" has zero parity).*

For the lower bound, if the algorithm has less than k accessible samples at a time, as with the analysis of the **Sym** property under forward-only testing, the algorithm here can only "guess" the missing key, and the probability to make the right guess is too low.

We go further, and show that even if the $k-1$-memory algorithm is allowed to choose which of the samples are retained in every stage (strong $k-1$-memory) rather than keeping a sliding window of recent history, the exponential separation still holds. The separation is achieved for an ε_k-test of the property where $\varepsilon_k = \Theta(1/k)$.

For the upper bound, Algorithm 6 makes a sequence of independent iterations of k samples at a time. In every iteration it gathers the keys $a_1, \ldots, a_k$ and verifies their codewords. If they are all different, the algorithm constructs the value of $f(\{a_1, \ldots, a_k\})$ and checks its parity.

Algorithm 6 One-sided ε-test for $\mathbf{Par}_k$, weak k-memory, $O(\varepsilon^{-k} k \log n)$ queries.

```
let $m$ be such that $\binom{m-1}{k-1} = n$.
for $\lceil 4\varepsilon^{-k} k \rceil$ times do
    take $k$ new samples $x^1, \ldots, x^k$.
    for $t$ from 1 to $k$ do
        query $x^t_1, \ldots, x^t_{\lceil \log_2 m \rceil}$, giving $\kappa(x^t)$ as $a^t$.
        choose $i \in [m]$, uniformly at random.
        query $x^t$ at $i$, giving $x^t_i$.
        if $x^t_i \neq (C(a^t))_i$ then
            return REJECT                                    ▷ reject by key invalidity
    if $|\{a^1, \ldots, a^k\}| = k$ then
        for $t$ from 1 to $k$ do
            query $\Phi_{x^t}(\{a^1, \ldots, a^k\} \setminus \{a^t\})$, giving $s^t$.
        if $\bigoplus_{i=1}^{k} s^t = 1$ then
            return REJECT                                    ▷ reject by parity-invalidity
return ACCEPT
```

4 Remaining open problems

It is an open problem whether the weak k-memory model is indeed strictly weaker than the strong k-memory model (for the same k). And if so, is the separation exponential? Also, we do not know whether or not for every k there exists k^* such that the k^*-weak model contains the k-strong one.

We believe that there exist some $\varepsilon_0 > 0$ and $0 < \alpha < 1$ such that for every sufficiently large k, there is an exponential separation between the weak k-memory model and the strong αk-memory model, with respect to an ε_0-test, rather than the separation for $\varepsilon_k = \Theta(1/k)$ that we show for $k-1$ vs k memory.

Another interesting open problem is whether the fully adaptive model has a simultaneous exponential separation from all fixed k-memory models. That is, whether there exists a property $\mathcal{P}$ and some $\varepsilon_0 > 0$ such that ε_0-testing of $\mathcal{P}$ would require $\Omega(\text{poly}(n))$ queries in every k-memory model (the polynomial degree possibly depending on k), but $\mathcal{P}$ is ε-testable using $O(\log n)$ queries using a fully adaptive algorithm for every fixed $\varepsilon > 0$.

References

1 Tomer Adar, Eldar Fischer, and Amit Levi. Support testing in the huge object model. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2024, August 28-30, 2024, London, United Kingdom*, volume 317. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024.

2 Maryam Aliakbarpour, Andrew McGregor, Jelani Nelson, and Erik Waingarten. Estimation of entropy in constant space with improved sample complexity. In *Proceedings of the 34th Annual Conference on Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

3 Noga Alon, Michael Krivelevich, Ilan Newman, and Mario Szegedy. Regular languages are testable with a constant number of queries. *SIAM Journal on Computing*, 30(6):1842–1862, 2001.

4 Tugkan Batu, Eldar Fischer, Lance Fortnow, Ravi Kumar, Ronitt Rubinfeld, and Patrick White. Testing random variables for independence and identity. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 442–451. IEEE, 2001.

5 Tugkan Batu, Lance Fortnow, Ronitt Rubinfeld, Warren D Smith, and Patrick White. Testing that distributions are close. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 259–269. IEEE, 2000.

6 Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 73–83, 1990.

7 Clément L. Canonne. *A Survey on Distribution Testing: Your Data is Big. But is it Blue?* Number 9 in Graduate Surveys. Theory of Computing Library, 2020. doi:10.4086/toc.gs.2020.009.

8 Sourav Chakraborty, Eldar Fischer, Arijit Ghosh, Gopinath Mishra, and Sayantan Sen. Testing of index-invariant properties in the huge object model. *CoRR*, abs/2207.12514, 2022. doi:10.48550/arXiv.2207.12514.

9 Funda Ergün, Ravi Kumar, and Ronitt Rubinfeld. Fast approximate pcps. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 41–50, 1999.

10 Oded Goldreich. *Introduction to property testing.* Cambridge University Press, 2017.

11 Oded Goldreich and Dana Ron. On testing expansion in bounded-degree graphs. *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*, pages 68–75, 2011.

12 Oded Goldreich and Dana Ron. Testing distributions of huge objects. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPIcs*, pages 78:1–78:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ITCS.2022.78.

13 David Peleg. *Distributed computing: a locality-sensitive approach.* SIAM, 2000.

14 Ronitt Rubinfeld and Madhu Sudan. Self-testing polynomial functions efficiently and over rational domains. In *SODA*, pages 23–32, 1992.

15 Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, 1996.

16 Andrew Chi-Chin Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pages 222–227, 1977.