Something for (Almost) Nothing: New Advances in Sublinear-Time Algorithms

Eric Blais Ronitt Rubinfeld

March 10, 2015

1 Introduction

What computational problems can we solve when we only have time to look at a tiny fraction of the data? This general question has been studied from many different angles in statistics. More recently, with the recent proliferation of massive datasets, it has also become a central question in computer science as well. Essentially all of the research on this question starts with a simple observation: except for a very small number of special cases, the only problems that can be solved in this very restrictive setting are those that admit *approximate* solutions.

In this chapter, we focus on a formalization of approximate solutions that has been widely studied in an area of theoretical computer science known as *property testing*. Let X denote the underlying dataset, and consider the setting where this dataset represents a combinatorial object. Let P be any property of this type of combinatorial object. We say that X is ϵ-*close* to having property P if we can modify at most an ϵ fraction of X to obtain the description X' of an object that does have property P; otherwise we say that X is ϵ-*far* from having the property. A randomized algorithm A is an ϵ-*tester* for P if it can distinguish with large constant probability[1] between datasets that represent objects with the property P from those that are ϵ-far from having the same property. (The algorithm A is free to output anything on inputs that don't have the property P but are also not ϵ-far from having this property; it is this leeway that will enable property testers to be so efficient.)

There is a close connection between property testing and the general parameter estimation. Let X be a dataset and $\theta = \theta(X)$ be any parameter of this dataset. For every threshold t, we can define the property of having $\theta \leq t$. If we have an efficient algorithm for testing this property, we can also use it to efficiently obtain an estimate $\widehat{\theta}$ that is close to θ in the sense that $\widehat{\theta} \leq \theta$ and the underlying object X is ϵ-close to another dataset X' with $\theta(X') = \widehat{\theta}$. Note that this notion of closeness is very different from the notions usually considered in parameter estimation; instead of determining it as a function $L(\theta, \widehat{\theta})$ of the true and estimated values of the parameter itself, here the quality of the estimate is a function of the underlying dataset.

Clustering: an illustrative example. Let X be a set of n points in $\mathbb{R}^d$. A fundamental question is how well the points in X can be clustered. This question can be formalized as follows.

[1] That is: with probability $1 - \delta$ for some fixed constant $\delta < \frac{1}{2}$. With standard techniques, the success probability of a tester can easily be boosted to any arbitrary amount, so throughout this chapter we will simply fix δ to be a small enough constant (say, $\delta = \frac{1}{3}$) and write "with large constant probability" to mean with probability $1 - \delta$.

Let $r > 0$ be a fixed radius, and let θ_r be the minimum number of clusters of radius r that are required to capture all the points in X. The problem of determining θ_r exactly is NP-hard, as is the problem of estimating its value up to any constant factor [23, 40, 20]. But Alon, Dar, Parnas and Ron [1] show that the closely related problem of distinguishing datasets that can be clustered into at most k clusters of radius r from those that are far from having this property can be solved in time that is *independent* of the number of points in X.[2]

Theorem 1 (Alon et al. [1])**.** *There is an algorithm that examines at most $\widetilde{O}(\frac{k \cdot d}{\epsilon})$ points in X and with large probability (i) accepts if the points in X can be grouped into at most k clusters of radius r and (ii) rejects if no subset of $(1-\epsilon)n$ points in X can be grouped into k clusters.*

The testing algorithm for clustering can be used to obtain an efficient estimator of the clustering parameter θ_r in the following sense.

Corollary 1. *There is an algorithm that examines at most $\widetilde{O}(\frac{\theta_r \cdot d}{\epsilon})$ points in X and returns a value $\widehat{\theta}_r$ such that with large probability $\widehat{\theta}_r \leq \theta_r$ and all but at most ϵn points in X can be partitioned into $\widehat{\theta}_r$ clusters of radius r.*

We reiterate that, unlike in the standard parameter estimation setting, we have no guarantee on the closeness of θ_r and the estimate $\widehat{\theta}_r$ returned by the algorithm. Instead, we are guaranteed that *most* of the points can be clustered in $\widehat{\theta}_r$ clusters. This notion of approximation has a number of advantages.

First, it is robust to noise. Adding a small number of outlier points can have a significant impact on the clustering parameter θ_r of a dataset. However, since the algorithm only examines a constant number of points, with high probability it does not observe any of the outliers and so these noisy points do not affect the estimator. Furthermore, many of the property testing algorithms can be made robust to higher noise rates so that even if outliers are observed, they do not affect the result by much. Similarly, the algorithm is appropriate when the underlying dataset is constantly changing; the same robustness means that the estimate changes smoothly as points are added and removed from the dataset.

A second advantage is that it leads to *extremely* efficient algorithms. The algorithms are so efficient that they can be used as a preprocessing step, even in situations where more accurate estimates of the clustering parameter are required. This is the case, for example, in settings where we are interested in learning how well the data can be clustered; here, the testing algorithm can be used in a preprocessing step to quickly identify point sets that are not even close to being clusterable into a reasonable number of clusters. In the case that the preprocessing algorithm identifies such a point set, alternative learning algorithms can be applied to those datasets instead.

The study of property testing was initiated in [6, 56, 26] and has been extremely active ever since. In the rest of this survey, we highlight some of the properties of point sets, graphs, and functions that can be tested efficiently. Our goal is simply to provide a glimpse of the results that can be obtained with the tools that have been developed in this research area. More detailed discussions on these topics can be found in many other more comprehensive surveys [12, 34, 24, 51, 52, 53, 55, 54].

[2]Here and throughout the rest of the chapter, running times are in the model where samples or queries to X are completed in a constant amount of time. We will also use the tilde notation $\tilde{O}(m)$ to hide polylog factors in m for clarity of presentation.

2 Testing Properties of Point Sets

A common machine learning problem involves extracting structural information from a set X of points. As we have already seen in the introduction, sublinear-time algorithms can be used to efficiently perform preprocessing tasks for these problems. In fact, sublinear-time algorithms have been shown to be useful in testing properties of both finite sets of points and (implicit representations of) infinite sets of points as well. To illustrate the former, we revisit the clustering problem in a bit more detail; for the latter, we show how sublinear-time algorithms can be used to estimate the surface area of continuous sets.

2.1 Clustering

Let us return to the problem of clustering points. To illustrate the variety of formalizations of this problem that are efficiently testable, let us now consider the setting where X is a set of points in a general metric space and we bound the *diameter* of the clusters (i.e., the maximum distance between any two points in the cluster) by some parameter d.

We can test if X can be clustered into at most k clusters of diameter d very efficiently in the following way.

Theorem 2 (Alon et al. [1])**.** *Let X be a set of points in a metric space. There is an algorithm that queries $O(\frac{k \log k}{\epsilon})$ points in X and with large constant probability (i) accepts if X can be clustered into k clusters of diameter d and (ii) rejects if we cannot cluster the points into k clusters of diameter $2d$ even if we remove an ϵ fraction of the points in X.*

The algorithm that achieves the bound in Theorem 1 maintains a set of cluster centers by repeatedly picking a sample point that is not covered by previously chosen cluster centers. If more than k cluster centers are chosen by this process, then the algorithm rejects, otherwise it accepts. The work of [1] shows that this simple algorithm satisfies the theorem's requirements. Note that if the point set cannot be clustered into k clusters of diameter d, but can be clustered into k clusters of diameter $2d$ if less than ϵ fraction of the points are thrown out, then it is ok for the algorithm to either accept or reject.

In many situations, we might know that the points are not clusterable into k clusters, but would still like to know if a very large fraction of the points can be clustered into k clusters. Note that our standard definition of property testers is of no use in this *tolerant* version of the clustering problem. The tolerant problem is in general more difficult, but several tolerant clustering problems can also be solved with sublinear-time algorithms [49, 12]. For example, there is a tolerant analogue of Theorem 2.

Theorem 3 (Parnas, Ron, and Rubinfeld [49])**.** *Given $0 < \epsilon_1 < \epsilon_2 < 1$. Let X be a set of points in a metric space. There is an algorithm that queries at most $\tilde{O}(\frac{k}{(\epsilon_2 - \epsilon_1)^2})$ points in X, and with large constant probability (i) accepts if at least $(1 - \epsilon_1)$ fraction of X can be clustered into k clusters of diameter d, and (ii) rejects if we cannot cluster more than $(1 - \epsilon_2)$ fraction of the points into k clusters of diameter $2d$.*

For more variants of the clustering problem considered in the property testing framework, see for example [1, 41, 11, 49, 12].

2.2 Surface Area

The property testing framework can also be applied to settings where the dataset represents a subset of a continuous space. Let $S \subseteq [0,1]^n$ be a subset of the unit n-dimensional hypercube, and consider any representation X of this subset for which algorithms can query an input $x \in [0,1]^n$ and observe whether $x \in S$ or not.

A fundamental parameter of the set S is its *surface area*, defined by $\text{surf}(S) = \lim_{\delta \to 0} \frac{\left|\partial S^{(\delta/2)}\right|}{\delta}$ where $\partial S^{(\delta/2)} \subseteq [0,1]^n$ is the set of points at distance at most $\delta/2$ from the boundary of S. In the one-dimensional case, $S \subseteq [0,1]$ is a union of intervals and $\text{surf}(S)$ corresponds to the number of disjoint intervals in S. And when $n = 2$, $\text{surf}(S)$ corresponds to the perimeter of the set S.

Since the surface area of a set S can always be increased by an arbitrary amount while modifying S itself on a set of measure 0, it is impossible to estimate the surface area of S within any additive or multiplicative factor with any finite (or even countable) number of queries. We can, however, test whether S has small surface area with a small number of queries.

Theorem 4 (Neeman [42])**.** *Fix any $0 < \alpha < 1$. There is an algorithm that queries at most $O(1/\epsilon)$ points and with large probability (i) accepts S when it has surface area at most α, and (ii) rejects S when every set S' that differs from S on at most an ϵ measure has surface area $\text{surf}(S') > \alpha$.*

Neeman's theorem [42] was obtained by improving the original analysis of an algorithm due to Kothari, Nayyeri, O'Donnell, and Wu [32], who were in turn building on previous work of Kearns and Ron [31] and Balcan et al. [3]. The algorithm itself works in a way that is conceptually similar to Buffon's classic needle problem: it drops a (virtual) needle onto the space $[0,1]^n$ and queries the two endpoints of the needles. If exactly one of those endpoints is in S, then we know that the needle crossed the boundary of S. Computing the probability of this event gives us a measure of the noise sensitivity of S, and connections between the surface area and noise sensitivity then lead to the proof of correctness of the algorithm.

3 Testing Properties of Graphs

The setting in which the underlying dataset X represents a graph G is particularly well suited for the design of sublinear-time algorithms. In this section, we explore two problems—a connectivity problem and an optimization problem—to illustrate what sublinear-time algorithms can do.

3.1 Diameter

The famous *small world* theory states that in any social group, every two members of the group are connected to each other via a short chain of acquaintances [13]. In particular, the *six degrees of separation* theory posits that everyone on Earth is connected by a chain of acquaintances of length at most 6. (See [16] for an introduction to these and many other interesting topics on graphs.) How efficiently can we test the small world theory on a given social network?

The study of the small world theory can be formalized in terms of the diameter of graphs. Let G be a graph where each person in the social group corresponds to a vertex, and an edge is placed between vertices that correspond to people that know each other. The *diameter* of G is the maximum distance between any two vertices in the graph. We can test whether the small world theory holds for the social group represented by G by testing whether the diameter of this graph is

small. Parnas and Ron have shown that we can perform this task in the property testing framework with a number of queries that is *independent* of the size of the graph.

Theorem 5 (Parnas, Ron [47])**.** *Let G be a graph with maximum degree d. There is an algorithm that queries $\tilde{O}(\epsilon^{-3})$ edges in G and with large probability (i) accepts if G has diameter at most D, and (ii) rejects if every subgraph G' obtained by removing at most an ϵ fraction of the vertices in G has diameter greater than $2D$.*

This result should look at least a little surprising at first glance: with a number of queries that is sublinear in n, we can't even determine the distance between any two fixed vertices v, w in G. So any algorithm for testing the diameter of a graph must instead test G for some other global property that distinguishes graphs with diameter D from those that are far from having diameter $2D$. That's what the algorithm in [47] does. Specifically, the algorithm estimates the *expansion* of G by sampling several start vertices, performing $O(1/\epsilon)$ steps of a breadth-first search from each of these vertices, and estimating the size of the neighborhoods around the vertices that have been sampled.

The algorithm for testing small-diameter graphs can also be used to estimate the diameter θ_{diam} of a graph G.

Corollary 2. *Let G be a graph with n vertices and maximum degree d. For every $\gamma > 1$, there is an algorithm that queries at most $\tilde{O}(\epsilon^{-3} \log_\gamma \theta_{\text{diam}})$ edges of G and outputs an estimate $\widehat{\theta}_{\text{diam}}$ such that with large probability $\widehat{\theta}_{\text{diam}} \leq \theta_{\text{diam}}$ and every subgraph G' with at least $(1-\epsilon)n$ nodes has diameter at least $\frac{\widehat{\theta}_{\text{diam}}}{2\gamma}$.*

Sublinear time algorithms for several variants of the diameter task, including tolerant versions, have been considered in [47, 7].

3.2 Vertex Cover

A *vertex cover* of the graph $G = (V, E)$ is a subset $V' \subseteq V$ such that every edge in E is adjacent to some vertex in V'. Let $\theta_{\text{VC}} = \theta_{\text{VC}}(G)$ denote the size of the minimum vertex cover of G. While the vertex cover problem is NP-complete [30], there is a linear-time algorithm which always outputs an estimate $\widehat{\theta}_{\text{VC}}$ that is at most twice as large as the true value of the minimum vertex cover of G. (This algorithm was independently discovered by Gavril and Yannakakis. See, e.g., [46] for the details.)

When the input graph G has maximum degree at most d, Parnas and Ron [48] have shown that there is an algorithm which can approximate θ_{VC} even more efficiently. Specifically, for every $\epsilon > 0$, there is an algorithm which returns an estimate $\widehat{\theta}_{\text{VC}}$ that satisfies $\theta_{\text{VC}} \leq \widehat{\theta}_{\text{VC}} \leq 2\theta_{\text{VC}} + \epsilon|V|$ and that runs in time that depends only on d and ϵ, but *not* on the size of the graph G. This result has since been sharpened, yielding the following testing algorithm.

Theorem 6 (Onak et al. [44])**.** *There is an $\tilde{O}(d) \cdot \text{poly}(1/\epsilon)$ time algorithm which on input a graph $G = (V, E)$ of average degree d and a parameter ϵ, outputs an estimate $\widehat{\theta}_{\text{VC}}$ which with large constant probability satisfies $\theta_{\text{VC}} \leq \widehat{\theta}_{\text{VC}} \leq 2\theta_{\text{VC}} + \epsilon|V|$.*

The original vertex cover estimator of Parnas and Ron [48] was obtained by establishing and exploiting a fundamental connection between distributed computation and sublinear-time algorithms: if there is a k-round distributed computation for a graph problem where the degree of the input

graph is bounded by d, then—since the output of a particular node can depend only on nodes that are within a radius of at most k—the distributed computation can be simulated in sequential time $d^{O(k)}$. This connection has since become an important and widely used source of techniques in the design of sublinear-time algorithms. In recent years, distributed computation has made great strides in finding *local distributed algorithms*, which are algorithms that run in $O(1)$ rounds [33]. Such algorithms have been found for approximate vertex cover, approximate matching, approximated packing and covering problems [48, 19, 18, 36]. All of these algorithms can be turned into sublinear-time algorithms which estimate the corresponding parameter of the input graph.

The later improvements that led to Theorem 6 were obtained by a different technique introduced by Nguyen and Onak [43]. They showed that greedy algorithms can be simulated in a local fashion. This connection has also proven to be very influential in the design of sublinear-time algorithms, leading to the design of algorithms for the approximate vertex cover, approximate matching, approximate set cover, approximated packing and covering problems [43, 57, 45, 28, 44, 37, 38, 36, 50].

4 Testing Properties of Functions

The last setting we consider is the one where the dataset represents a function $f : \mathcal{X}_1 \times \cdots \times \mathcal{X}_n \to \mathcal{Y}$ mapping n features to some value in a (finite or infinite) set $\mathcal{Y}$. (We will be particularly interested in the situation where $\mathcal{X}_1 = \cdots = \mathcal{X}_n = \mathcal{Y} = \{0, 1\}$, in which case $f : \{0, 1\}^n \to \{0, 1\}$ is a Boolean function.) A common task in this scenario is the (supervised) machine learning problem, where f is a target function, and the learning algorithm attempts to identify a hypothesis function h that is close to f while observing the value of $f(x_1, \ldots, x_n)$ (i.e., the label of the example $(x_1, \ldots, x_n)$) on as few inputs as possible. In the following subsections, we examine some sublinear-time algorithms that prove to be particularly useful for this general problem.

4.1 Feature Selection

One of the preliminary tasks when learning functions over rich feature sets is to identify the set of features that are relevant. A corresponding parameter estimation task is to determine the number of relevant features for a given target function. This task has been formalized in the property testing framework as the *junta testing* problem.

The function $f : \{0, 1\}^n \to \{0, 1\}$ is a k-junta if there is a set $J \subseteq [n]$ of at most k coordinates such that the value of $f(x)$ is completely determined by the values $\{x_i\}_{i \in J}$. The coordinates in the minimal set J that satisfies this condition are called *relevant* to f; the remaining coordinates are *irrelevant*. Fischer et al. [21] showed that we can test whether f is a k-junta with algorithms that have a running time that is *independent* of the total number n of coordinates. This result was further sharpened in [4, 5], yielding the following result.

Theorem 7 (Blais [5])**.** *There is an algorithm that queries the value of $f : \{0, 1\}^n \to \{0, 1\}$ on $\widetilde{O}(\frac{k}{\epsilon})$ inputs and with large constant probability (i) accepts when f is a k-junta, and (ii) rejects when f is ϵ-far from k-juntas.*

With the same reduction that we have already seen with the parameter estimation tasks for point sets and graphs, we can use the junta testing algorithm to obtain a good estimate of the number θ_{junta} of relevant features for a given function.

Corollary 3. *There is an algorithm that queries the value of $f : \{0,1\}^n \to \{0,1\}$ on $\widetilde{O}(\frac{\theta_{\text{junta}}}{\epsilon})$ inputs and returns an estimate $\widehat{\theta}_{\text{junta}}$ such that with large probability $\widehat{\theta}_{\text{junta}} \leq \theta_{\text{junta}}$ and f is ϵ-close to being a $\widehat{\theta}_{\text{junta}}$-junta.*

The same result also applies to the much more general setting where $f : \mathcal{X}_1 \times \cdots \times \mathcal{X}_n \to \mathcal{Y}$ is a function mapping n-tuples where each attribute comes from any finite set and where $\mathcal{Y}$ is an arbitrary range. See [5] for the details.

4.2 Model Selection

Another ubiquitous preliminary task in machine learning is determining which is the right learning algorithm to use for a given target function. This problem is known as *model selection*. The model selection task is often completed by using domain-specific knowledge (e.g., by using algorithms that have successfully learned similar functions in the past), but sublinear-time algorithms can also provide useful tools for this task when such knowledge is not available.

Consider for example the algorithms for learning decision tree representations of Boolean functions. A decision tree is a full binary tree with indices in $[n]$ associated with each internal node, the labels $0, 1$ associated with the two edges that connect an internal node with its two children, and a Boolean value associated with each leaf. Every input $x \in \{0,1\}^n$ determines a unique path from the root to one of the leaves in a decision tree by following the edge labelled with x_i from an internal node labelled with i, and the function $f_T : \{0,1\}^n \to \{0,1\}$ corresponds to a decision tree T if for every input $x \in \{0,1\}^n$, the leaf reached by x is labelled with $f_T(x)$. The *size* of a decision tree is the number of nodes it contains, and the *decision tree (size) complexity* of a function is the size of the smallest decision tree that represents it. Decision tree learning algorithms are most appropriate for functions with small decision tree complexity. As Diakonikolas et al. [14] showed, we can test whether a function has small decision tree complexity with a number of queries that is independent of the total number n of features. Chakraborty, García-Soriano, and Matsliah [9] later improved the query complexity to obtain the following tight bounds on its query complexity.

Theorem 8 (Chakraborty, García-Soriano, Matsliah [9])**.** *Fix $s \geq 1$ and $\epsilon > 0$. There is an algorithm that queries the value of $f : \{0,1\}^n \to \{0,1\}$ on $\widetilde{O}(\frac{s}{\epsilon^2})$ inputs and with large constant probability (i) accepts if f has decision tree complexity at most s, and (ii) rejects if f is ϵ-far from having decision tree complexity s.*

Diakonikolas et al. [14] obtained the first result on testing small decision tree complexity by introducing a new technique called *testing by implicit learning*. The idea of this technique is that for many classes of functions, there are extremely efficient algorithms for learning the high-level structure of the function without identifying which are the relevant features. In the case of functions with small decision tree complexity, this means that we can learn the shape of the decision tree that represents the target function—but not the labels associated with each node in the tree—with a running time that is independent of n. This technique is quite powerful, and it has been used to test many other properties as well, including the circuit complexity, minimum DNF size complexity, and Fourier degree of Boolean functions.

The same algorithms can also be used to provide good estimates $\widehat{\theta}_{\text{DT}}$ of the decision tree complexity θ_{DT} of Boolean functions.

Corollary 4. *There is an algorithm that queries the value of $f : \{0,1\}^n \to \{0,1\}$ on $\widetilde{O}(\frac{\theta_{\mathrm{DT}}}{\epsilon^2})$ inputs and returns an estimate $\widehat{\theta}_{\mathrm{DT}}$ such that with large probability $\widehat{\theta}_{\mathrm{DT}} \leq \theta_{\mathrm{DT}}$ and f is ϵ-close to being representable with a decision tree of size $\widehat{\theta}_{\mathrm{DT}}$.*

There are also other models that can be tested very efficiently using completely different techniques. For example, another well-known learning model is that of *linear threshold functions* (or halfspaces). The function $f : \{0,1\}^n \to \{0,1\}$ is a linear threshold function if there are real-valued weights $w_1, \ldots, w_n$ and a threshold $t \in \mathbb{R}$ such that for every $x \in \{0,1\}^n$, $f(x) = \mathrm{sign}(\sum_{i=1}^n w_i x_i - t)$. Matulef et al. [39] showed that we can test whether a function is a linear threshold function using a constant number of queries.

Theorem 9 (Matulef et al. [39])**.** *Fix $\epsilon > 0$. There is an algorithm that queries the value of $f : \{0,1\}^n \to \{0,1\}$ on $\mathrm{poly}(1/\epsilon)$ inputs and with large constant probability (i) accepts if f is a linear threshold function, and (ii) rejects if f is ϵ-far from being a linear threshold function.*

4.3 Data Quality

Collected data is notorious for being very noisy, and, in particular, containing entries that are erroneous for various reasons. Can we estimate the quality of a dataset? Here are some instances in which sublinear algorithms can be of assistance.

Lipschitz property. One common property of many datasets is a controlled rate of change. For example, a sensor measuring the ambient temperature will not observe extreme fluctuations within short time scales. However, in noisy datasets, the noisy entries often break this property. So one test we can do on a dataset to evaluate its quality is whether it has this property.

Formally, the property of having a bounded rate of change is known as the *Lipschitz* property. The function $f : \{1, \ldots, n\} \to \mathbb{R}$ is t-Lipschitz for some $t \geq 0$ if for every $x, x' \in [n]$, we have $|f(x) - f(x')| \leq t \cdot |x - x'|$. As Jha and Raskhodnikova showed [29], we can test whether a function is Lipschitz with sublinear-time algorithms.

Theorem 10 (Jha, Raskhodnikova [29])**.** *Fix any $t > 0$. There is an algorithm that queries $f : \{1, 2, \ldots, n\} \to \mathbb{R}$ on $O(\frac{1}{\epsilon} \log n)$ inputs and with large constant probability (i) accepts f when it is t-Lipschitz, and (ii) rejects f when it is ϵ-far from t-Lipschitz.*

One natural idea for testing the Lipschitz property of a function f may be to select some neighboring indices $i, i+1 \in [n]$, verify that $|f(i) - f(i+1)| \leq t$, and repeat this test some number of times. This idea, however, will not work: a function that has a single, enormous jump in value will appear to be Lipschitz on any pair that does not cross the jump even though it is very far from Lipschitz. Jha and Raskhodnikova bypass this possible barrier by considering a completely different testing algorithm, based on the monotonicity testing algorithms we describe below. They also show how the algorithm can be extended to test the Lipschitz property for functions over many other domains as well.

Monotonicity. A second property that is common to many types of non-noisy datasets is *monotonicity*. For example, data that represents the cumulative flow that has passed through a sensor will be a non-decreasing monotone sequence. Formally, $f : [n] \to \mathbb{R}$ is *monotone* if $f(1) \leq f(2) \leq \cdots \leq f(n)$. We can also test the monotonicity of a function with a sublinear-time algorithm.

Theorem 11 (Ergün et al. [17]). *There is an algorithm that queries $f : \{1, 2, \ldots, n\} \to \mathbb{R}$ on $O(\frac{1}{\epsilon} \log n)$ inputs and with large constant probability (i) accepts f when it is monotone, and (ii) rejects f when it is ϵ-far from monotone.*

As was the case with the Lipschitz-testing problem, the idea of choosing neighboring pairs $i, i+1 \in [n]$ and verifying $f(i) \leq f(i+1)$ does not lead to a valid sublinear-time algorithm for testing monotonicity. Another natural idea is to choose m elements uniformly at random from $[n]$, call them $i_1 \leq \ldots \leq i_m \in [n]$ and verify that $f(i_1) \leq \cdots \leq f(i_m)$. This approach *does* lead to a sublinear-time algorithm, but one that is not nearly as efficient as the one which gives the bound in Theorem 11. It can be shown that this approach requires $\Omega(\sqrt{n})$ queries to correctly reject all the functions that are far from monotone. Ergün et al. [17] obtained an exponentially more efficient monotonicity tester by taking a completely different approach in which the tester simulates a binary search. It chooses $i \in [n]$ uniformly at random, queries $f(i)$, and then searches for the value $f(i)$ in the sequence $f(1), \ldots, f(n)$ under the assumption that f is monotone. When f is indeed monotone, this binary search correctly leads to i. Interestingly, for *any* function that is far from monotone, this binary search simulation will identify a witness of non-monotonicity (that is, a pair $i < j \in [n]$ of indices for which $f(i) > f(j)$) with reasonably large probability.

There are also sublinear-time algorithms for testing the monotonicity of functions over many other domains. For some of the results in this direction, see [25, 15, 22, 8, 10].

4.4 Alternative Query and Sampling Models

The results described in this section have all been in the setting where the algorithm can query the value of the target function on any inputs of its choosing. This setting corresponds to the *membership query model* in machine learning. For many applications in machine learning, however, the membership query model is unrealistic and algorithms must operate in more restrictive query models. Two alternative query models have received particular attention in the machine learning community: the (passive) *sampling* model and the *active query* model. In the sampling model, the inputs on which the algorithm observes the value of the target function are drawn at random from some fixed distribution. In the active query model, a larger number of inputs are drawn from some distribution over the function's domain, and the algorithm can query the value of the target function on any of the points that were drawn.

Many efficient learning algorithms have been devised for both the sampling and active query models. Similarly, sublinear-time algorithms have been introduced to test fundamental properties of Boolean functions (and other combinatorial objects) in the sampling and active query models as well. For example, linear-threshold functions can be tested with a sublinear number of samples or active queries.

Theorem 12 (Balcan et al. [3]). *Fix $\epsilon > 0$. There is an algorithm that observes the value of $f : \mathbb{R}^n \to \{0, 1\}$ on $\tilde{O}(\sqrt{n}) \cdot \mathrm{poly}(1/\epsilon)$ inputs drawn independently at random from the n-dimensional standard normal distribution and with large constant probability (i) accepts if f is a linear threshold function, and (ii) rejects if f is ϵ-far from being a linear threshold function.*

For more on sublinear-time algorithms and on their limitations in the sampling and active query models, see [26, 3, 2, 27].

5 Other Topics

A related area of study is how to understand properties underlying a distribution over a very large domain, when given access to samples from that distribution. This is a scenario that is well studied in statistics, information theory, database algorithms, and machine learning algorithms. For example – it may be important to understand whether the distribution has certain properties, such as being close to uniform, Gaussian, high entropy, or independent. It may also be important to learn parameters of the distribution, or to learn a concise representation of an approximation to the distribution. Recently, surprising bounds on the sample complexity of these problems have been achieved. Although we do not mention these problems in any further detail in this survey, we point the reader to the surveys of [35, 54] and to the chapter entitled "Learning Structured Distributions" by Ilias Diakonikolas in this volume.

References

[1] N. Alon, S. Dar, M. Parnas, and D. Ron. Testing of clustering. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:240, 2000.

[2] N. Alon, R. Hod, and A. Weinstein. On active and passive testing. *CoRR*, abs/1307.7364, 2013.

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

[4] E. Blais. Improved bounds for testing juntas. In *APPROX-RANDOM '08*, pages 317–330, 2008.

[5] E. Blais. Testing juntas nearly optimally. In *Proc. 41st Annual ACM Symposium on the Theory of Computing*, pages 151–158, 2009.

[6] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47:549–595, 1993. Earlier version in STOC'90.

[7] A. Campagna, A. Guo, and R. Rubinfeld. Local reconstructors and tolerant testers for connectivity and diameter. In *APPROX-RANDOM*, pages 411–424, 2013.

[8] D. Chakrabarty and C. Seshadhri. A o(n) monotonicity tester for boolean functions over the hypercube. In *STOC*, pages 411–418, 2013.

[9] S. Chakraborty, D. García-Soriano, and A. Matsliah. Efficient sample extractors for juntas with applications. In *Automata, Languages and Programming: 38th International Colloquium*, pages 545–556, 2011.

[10] X. Chen, R. A. Servedio, and L. Tan. New algorithms and lower bounds for monotonicity testing. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 286–295, 2014.

[11] A. Czumaj and C. Sohler. Abstract combinatorial programs and efficient property testers. pages 83–92, 2002.

[12] A. Czumaj and C. Sohler. Sublinear-time algorithms. *Bulletin of the EATCS*, 89:23–47, 2006.

[13] I. de Sola Pool and M. Kochen. Contacts and influence. *Social networks*, 1(1):5–51, 1979.

[14] I. Diakonikolas, H. Lee, K. Matulef, K. Onak, R. Rubinfeld, R. Servedio, and A. Wan. Testing for concise representations. In *Proc. 48th Annual IEEE Symposium on Foundations of Computer Science*, pages 549–558, 2007.

[15] Y. Dodis, O. Goldreich, E. Lehman, S. Raskhodnikova, D. Ron, and A. Samorodnitsky. Improved testing algorithms for monotonocity. In *Proceedings of RANDOM*, pages 97–108, 1999.

[16] D. A. Easley and J. M. Kleinberg. *Networks, Crowds, and Markets - Reasoning About a Highly Connected World*. Cambridge University Press, 2010.

[17] F. Ergün, S. Kannan, S. R. Kumar, R. Rubinfeld, and M. Viswanathan. Spot-checkers. *JCSS*, 60(3):717–751, 2000.

[18] G. Even, M. Medina, and D. Ron. Best of two local models: Local centralized and local distributed algorithms. *arXiv preprint arXiv:1402.3796*, 2014.

[19] G. Even, M. Medina, and D. Ron. Distributed maximum matching in bounded degree graphs. *arXiv preprint arXiv:1407.7882*, 2014.

[20] T. Feder and D. H. Greene. Optimal algorithms for approximate clustering. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 434–444, 1988.

[21] E. Fischer, G. Kindler, D. Ron, S. Safra, and A. Samorodnitsky. Testing juntas. *Journal of Computer and System Sciences*, 68(4):753–787, 2004.

[22] E. Fischer, E. Lehman, I. Newman, S. Raskhodnikova, R. Rubinfeld, and A. Samrodnitsky. Monotonicity testing over general poset domains. In *Proceedings of the Thirty-Fourth Annual ACM Symposium on the Theory of Computing*, pages 474–483, 2002.

[23] R. J. Fowler, M. S. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are np-complete. *IPL*, pages 434–444, 1981.

[24] O. Goldreich. Combinatorial property testing - a survey. In *Randomization Methods in Algorithm Design*, pages 45–60, 1998.

[25] O. Goldreich, S. Goldwasser, E. Lehman, D. Ron, and A. Samordinsky. Testing monotonicity. *Combinatorica*, 20(3):301–337, 2000.

[26] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45:653–750, 1998.

[27] O. Goldreich and D. Ron. On sample-based testers. In *Proc. 6th Innovations in Theoretical Computer Science*, pages 337–345, 2015.

[28] A. Hassidim, Y. Mansour, and S. Vardi. Local computation mechanism design. In *Proceedings of the fifteenth ACM conference on Economics and computation*, pages 601–616. ACM, 2014.

[29] M. Jha and S. Raskhodnikova. Testing and reconstruction of lipschitz functions with applications to data privacy. *SIAM J. Comput.*, 42(2):700–731, 2013.

[30] R. M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York.*, pages 85–103, 1972.

[31] M. J. Kearns and D. Ron. Testing problems with sublearning sample complexity. *Journal of Computer and System Sciences*, 61(3):428–456, 2000.

[32] P. Kothari, A. Nayyeri, R. O'Donnell, and C. Wu. Testing surface area. In *Proc. 25th ACM-SIAM Symposium on Discrete Algorithms*, pages 1204–1214, 2014.

[33] F. Kuhn, T. Moscibroda, and R. Wattenhofer. Local computation: Lower and upper bounds. *CoRR*, abs/1011.5470, 2010.

[34] R. Kumar and R. Rubinfeld. Algorithms column: Sublinear-time algorithms. *Sigact News*, 34:57–67, 2003.

[35] R. Kumar and R. Rubinfeld. Sublinear time algorithms. *SIGACT News*, 34:57–67, 2003.

[36] R. Levi, R. Rubinfeld, and A. Yodpinyanee. Local computation algorithms for graphs of non-constant degrees. *arXiv preprint arXiv:1502.04022*, 2015.

[37] Y. Mansour, A. Rubinstein, S. Vardi, and N. Xie. Converting online algorithms to local computation algorithms. In *Automata, Languages, and Programming*, pages 653–664. Springer, 2012.

[38] Y. Mansour and S. Vardi. A local computation approximation scheme to maximum matching. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 260–273. Springer, 2013.

[39] K. Matulef, R. O'Donnell, R. Rubinfeld, and R. A. Servedio. Testing halfspaces. *SIAM J. Comput.*, 39(5):2004–2047, 2010.

[40] N. Megiddo and E. Zemel. An o(n log n) randomizing algorithm for the weighted euclidean 1-center problem. *J. Algorithms*, 7(3):358–368, 1986.

[41] N. Mishra, D. Oblinger, and L. Pitt. Sublinear time approximate clustering. pages 439–447, 2001.

[42] J. Neeman. Testing surface area with arbitrary accuracy. In *Proc. 46th Annual ACM Symposium on the Theory of Computing*, pages 393–397, 2014.

[43] H. N. Nguyen and K. Onak. Constant-time approximation algorithms via local improvements. In *Proc. 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 327–336, 2008.

[44] K. Onak, D. Ron, M. Rosen, and R. Rubinfeld. A near-optimal sublinear-time algorithm for approximating the minimum vertex cover size. In *23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012)*, Kyoto, Japan, January 2012.

[45] K. Onak, D. Ron, M. Rosen, and R. Rubinfeld. A near-optimal sublinear-time algorithm for approximating the minimum vertex cover size. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1123–1131. SIAM, 2012.

[46] C. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Dover Publications, 1998.

[47] M. Parnas and D. Ron. Testing the diameter of graphs. *Random Structures and Algorithms*, 20(2):165–183, 2002.

[48] M. Parnas and D. Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theoretical Computer Science*, 381(1–3):183–196, 2007.

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

[50] O. Reingold and S. Vardi. New techniques and tighter bounds for local computation algorithms. *arXiv preprint arXiv:1404.5398*, 2014.

[51] D. Ron. Property testing. In *Handbook on Randomization, Volume II*, pages 597–649, 2001.

[52] D. Ron. Property testing: A learning theory perspective. *Foundations and Trends in Machine Learning*, 3:307–402, 2008.

[53] R. Rubinfeld. Sublinear time algorithms. In *Proc. International Congress of Mathematicians*, volume 3, pages 1095–1111, 2006.

[54] R. Rubinfeld. Taming big probability distributions. *ACM Crossroads*, 19(1):24–28, 2012.

[55] R. Rubinfeld and A. Shapira. Sublinear time algorithms. *SIAM J. Discrete Math.*, 25(4):1562–1588, 2011.

[56] R. Rubinfeld and M. Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM Journal on Computing*, 25:252–271, 1996.

[57] Y. Yoshida, Y. Yamamoto, and H. Ito. An improved constant-time approximation algorithm for maximum matchings. In *Proc. 41st Annual ACM Symposium on the Theory of Computing*, pages 225–234, 2009.