Sublinear time algorithms

Ravi Kumar [*]   Ronitt Rubinfeld[†]

With the recent tremendous increase in computational power and cheap storage, we are blessed with a multitude of available, and possibly useful, information. It is always nice to have something for (almost) nothing. However, this blessing is also something of a curse, for we may also be asked to do something meaningful with all of this data. The scale of these data sets, coupled with the typical situation in which there is very little time to perform our computations, raises the question of which computations could one hope to accomplish extremely quickly? In particular, what can one solve in *sublinear time*?

Sublinear time is a daunting goal since it allows one to read only a miniscule fraction of the input. Still, there are problems for which deterministic exact sublinear time algorithms are known. However, since any sublinear time algorithm can only view a small portion of the input, for most natural problems the algorithm must use randomization and must give an answer which is in some sense approximate. There is a growing body of work aimed at finding sublinear time algorithms for various problems. Recent results have shown that there are optimization problems whose values can be approximated in sublinear time. In addition, property testing, an alternative notion of approximation for decision problems, has been applied to give sublinear algorithms for a wide variety of problems. One can also test various properties of distributions, where access to the distribution is given through samples generated according to the distribution, in time sublinear in the size of the support of the distribution. Several useful techniques, including the use of the Szemerédi Regularity lemma and low rank approximations of matrices, have emerged for designing sublinear algorithms. Still, the study of sublinear algorithms is very new, and much remains to be understood about their scope.

We attempt to give some flavor of the types of results that one can achieve in sublinear time. We will not survey the known results or give much historic perspective of their development. For such treatments, we refer the reader to some very good surveys on property testing [15, 27, 11]. Instead, we give a few simple examples meant to illustrate the range and scope of techniques that have been used in this area.

There is also much interest in the related area of streaming algorithms, where the input is to be read just once and the workspace (though not the time) is restricted to be sublinear in the size of the input. We will not discuss streaming algorithms any further here, though we recommend the surveys of [4, 25].

1   Traditional approximation algorithms

In recent years, sublinear time algorithms have been developed to estimate the quality of the optimal solution for several optimization problems concerning graphs, strings, etc. For many problems, sublinear time approximations do not exist. However, there are often plausible assumptions that

---

[*]IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120. Email: `ravi@almaden.ibm.com`

[†]NEC Laboratories America Inc., 4 Independence Way, Princeton, NJ 08540. Email: `ronitt@nec-labs.com`

can be made on the input which allow one to make a sublinear time approximation. For example, one can approximate the size of the maximum cut in constant time when the input is a dense graph [17, 14], and in fact, one can approximate all dense problems in Max-SNP in constant time [3, 14, 2]. It is interesting to note that often, but not always, sublinear time algorithms can output an implicit representation of a near optimal solution.

In the following we describe two sublinear time approximation algorithms, for clustering and a linear programming problem respectively, both of which sample the data and use the solution on the subproblem to estimate the optimum value of the whole problem. In the former example, a representation of a near optimal solution is given.

1.1   Clustering

The question of sublinear time algorithms for k-median clustering on bounded metric spaces is considered by Mishra, Oblinger, and Pitt [23]. They analyze an algorithm that samples points in the space and obtains an approximate k-median clustering of the sample points. They show that using a number of samples that is sublinear in the size of the input, one can achieve an output that is quite close to the optimum. (For other results on various clustering problems see [1, 7, 24]).

Consider a finite metric space X with distance function $d : X \times X \to [0, M]$. In the k-median problem, we are given a set $R \subseteq X$ of points and are required to find k cluster centers such that the sum of distances of each point in R to its closest cluster center is minimized. The *discrete* version of this problem requires each cluster center to be in R while the *continuous* version permits the cluster centers to be in $X \backslash R$. Given k cluster centers $C = \{c_1, \ldots, c_k\}$, it is natural to think of a clustering function $f_C(x) = \min_{c \in C} d(x, c)$; the cost of this clustering is $\sum_{x \in X} f_C(x) = |X| \cdot E_X(f_C)$. In other words, the k-median problem can be thought of as choosing a clustering function $f_C(\cdot)$ such that $E_X(f_C)$ is minimized. We use C^* (resp. C_R^*) to denote the optimum in the continuous (resp. discrete) case. It is an easy fact using triangle inequality (e.g., [21]) that the continuous and discrete optimum differ only by a factor of two: $E_X(f_{C^*}) \leq E_X(f_{C_R^*}) \leq 2 \cdot E_X(f_{C^*})$. An algorithm is an α-factor approximation algorithm for k-median if it outputs $\tilde{C}$ such that $E_X(f_{\tilde{C}}) \leq \alpha \cdot E_X(f_{C^*})$.

Theorem 1 ([23]). *Let $\mathcal{A}$ be an α-factor k-median algorithm that runs in $T(n)$ time. Then, there is an algorithm that runs in $T\left(\left(\frac{8\alpha M}{\epsilon} \right)^2 \left(k \ln n + \ln \frac{4}{\delta} \right) \right)$ time and outputs $\tilde{C}_S$ such that with probability at least $1 - \delta$, $E_X(f_{\tilde{C}_S}) \leq 2\alpha \cdot E_X(f_{C^*}) + \epsilon$.*

Proof. The algorithm is the following. Choose a set S of i.i.d. samples from X such that $|S| \geq \left(\frac{8\alpha M}{\epsilon} \right)^2 \left(k \ln n + \ln \frac{4}{\delta} \right)$. Run the algorithm $\mathcal{A}$ on S and declare the output of $\mathcal{A}$ as the final answer. The running time of this algorithm is $T(|S|)$.

The main idea in the proof of correctness is to relate the clustering cost in S to the clustering cost in X. Such a relationship is made possible by the following *uniform convergence lemma*, which says that the mean of a function f on a domain X, $E_X(f)$, can be approximated by its mean on a random subset S of the domain, $E_S(f)$, if the subset S is sufficiently large.

Lemma 1 (Uniform convergence [22, 26]). *Let $\mathcal{F}$ be a finite family of functions on X with $0 \leq f(x) \leq M$ for all $f \in \mathcal{F}, x \in X$. If S is a set of i.i.d. samples from X such that $|S| \geq \frac{M^2}{2\epsilon^2} \left(\ln |\mathcal{F}| + \ln \frac{2}{\delta} \right)$, then $\Pr[\exists f \in \mathcal{F}, |E_X(f) - E_S(f)| \geq \epsilon] \leq \delta$.*

The function family we will be working with is $\mathcal{F} = \{f_C \mid C \subseteq [n], |C| = k\}$, so $\ln |\mathcal{F}| \leq k \ln n$. Applying the uniform convergence lemma, the choice of $|S|$ yields $E_X(f_{\tilde{C}_S}) \leq E_S(f_{\tilde{C}_S}) + \epsilon/(4\alpha)$, with probability at least $1 - \delta/2$. Using the fact that $\mathcal{A}$ is an α-factor approximation algorithm and the relationship between continuous and discrete optima, $E_S(f_{\tilde{C}_S}) \leq \alpha \cdot E_S(f_{C_S^*}) \leq 2\alpha \cdot E_S(f_{C^*})$.

Applying the uniform convergence lemma one more time, $E_S(f_{C^*}) \leq E_X(f_{C^*}) + \epsilon/(4\alpha)$, with probability at least $1 - \delta/2$. Putting these together, the proof is complete. $\square$

1.2 Linear programming

In this section we describe a result by Alon, Fernandez de la Vega, Kannan, and Karpinski [2] which says the following: Given a Linear Program (LP) where the variables are constrained to the range $[0, 1]$ and the coefficients in the constraints are bounded, one can get a good indication of the optimal value of the LP by choosing uniformly at random a small number of the variables and solving the induced subprogram. We use the notation $(x)^- = \min(x, 0)$ and if for a vector x, $\forall i, 0 \leq x_i \leq 1$, then we say that $0 \leq x \leq 1$. For a vector $x = x_1, \ldots, x_n$, let $\|x\|_\infty$ denote $\max_{i=1}^n |x_i|$ and let $\|x\|_1$ denote $\sum_{i=1}^n |x_i|$.

Theorem 2 ([2]). *There is an algorithm that, given an LP*

$$\max_x \ c^T x \quad \text{subject to} \quad Ax \leq b, \ \ 0 \leq x \leq 1, \ \ A \in \mathbb{R}^{m \times n}, \ \ c, x \in \mathbb{R}^n, \ \ b \in \mathbb{R}^m,$$

an upper bound α on the value of the LP, and an integer $0 < q \leq n$, runs in time $O(q)$ and outputs $Q \subseteq [n], |Q| = q$, such that for any $\lambda > 0$, with probability at least $1 - 4\exp(-\lambda^2/4)$,

$$\frac{q}{n}\alpha + \lambda\sqrt{q}\|c\|_\infty \geq \max_x \sum_{i \in Q} c_i x_i \quad \text{subject to} \quad \sum_{i \in Q} A_i x_i \leq \frac{q}{n}b - \lambda\sqrt{q}\|A\|_\infty \cdot 1_{m \times 1}.$$

If there is a lower bound on the value of the LP, then it is easy to obtain a lower bound on the value of the LP restricted to a random subset Q. To show the above theorem, we need the other direction—i.e., that if there is no good solution to the full LP, then there will also not be a good solution to the restricted LP. A clever use of duality [2] converts the problem back into the easy case. (For more recent results on this topic, see [9].)

Proof of Theorem. From the statement of the theorem, the following system is infeasible ($s \in \mathbb{R}, t \in \mathbb{R}^m$ are the slack variables):

$$\begin{pmatrix} c^T & -1 & 0_{1 \times m} \\ A & 0 & I_{m \times m} \end{pmatrix} \cdot \begin{pmatrix} x \\ s \\ t \end{pmatrix} = \begin{pmatrix} \alpha \\ b \end{pmatrix}, \quad \begin{pmatrix} x \\ s \\ t \end{pmatrix} \geq 0.$$

We use *Farkas' lemma*, which says that exactly one of the following holds: $\exists z, Mz = w, z \geq 0$ or $\exists z', M^T z' \geq 0, z'^T w < 0$. Applying Farkas' lemma, $\exists y' \in \mathbb{R}^{m+1}$, written as $y' = (-\beta, y)$ where β is a scalar and $y \in \mathbb{R}^m$, satisfying: $\forall i, -\beta c_i + (y^T A)_i \geq 0, y_i \geq 0, \beta \geq 0$, and $-\beta\alpha + y^T b < 0$. Clearly, for any vector $x \in \mathbb{R}^n, 0 \leq x \leq 1, \sum_{i=1}^n (-\beta c_i + (y^T A)_i)x_i \geq 0 > -\beta\alpha + y^T b$. In particular, let us choose x such that $x_i = 1$ whenever $(y^T A_i - \beta c_i) = (y^T A - \beta c)_i < 0$. This implies that $\sum_{i=1}^n (y^T A - \beta c)_i^- > y^T b - \beta\alpha$.

Also, we can bound the maximum of the entries of $(y^T A - \beta c)$. For every i,

$$\left\| (y^T A - \beta c)_i \right\|_1 = \left\| \left(\sum_j y_j A_{ji} \right) - \beta c_i \right\|_1 \leq \|A\|_\infty \|y\|_1 + \|c\|_\infty \beta.$$

This will be used to bound $\left\| y^T A - \beta c \right\|_\infty$.

3

Consider the following event $\mathcal{E}$:

$$\sum_{i \in Q} (y^T A - \beta c)_i^- \geq \frac{q}{n}(y^T b - \beta\alpha) - \lambda\sqrt{q}\left(\|A\|_\infty \|y\|_1 + \|c\|_\infty \beta\right).$$

The first claim is that $\mathcal{E}$ implies the conclusion of the theorem. To see this, assume the contrary. Then, we have that there is a solution x to the sub-program that satisfies $\sum_{i \in Q} c_i x_i > (q/n)\alpha + \lambda\sqrt{q}\|c\|_\infty$ with the constraint $\sum_{i \in Q} A_i x_i \leq (q/n)b - \lambda\|A\|_\infty \sqrt{q} \cdot 1_{m \times 1}$. Multiplying the first inequality by $-\beta$ and the second inequality by y^T (keeping in mind that both β and y^T are non-negative) and summing them yields

$$\sum_{i \in Q} (y^T A - \beta c)_i x_i \leq (q/n)(y^T b - \beta\alpha) - \lambda\sqrt{q}(\|A\|_\infty \|y\|_1 + \|c\|_\infty \beta).$$

But, $\sum_{i \in Q}(y^T A - \beta c)_i^- \leq \sum_{i \in Q}(y^T A - \beta c)_i x_i$. This contradicts $\mathcal{E}$.

The second claim is that $\Pr[\mathcal{E}] \geq 1 - 4\exp(-\lambda^2/4)$. This is by Chernoff–Hoeffding bounds of the following form: if $Z_1, \ldots, Z_q$ are i.i.d. random variables $\|Z_i\|_1 \leq 1$ and $Z = Z_1 + \cdots + Z_q$, then $\Pr[\|Z - E[Z]\|_1 \geq \delta] \leq 4\exp(-\delta^2/(4q))$. Setting $Z_i = (y^T A - \beta c)_i^- / \|y^T A - \beta c\|_\infty$, $\delta = \lambda\sqrt{q}$, and noting that

$$E[Z] = \frac{q}{n}\frac{\sum_{i=1}^n (y^T A - \beta c)_i^-}{\|y^T A - \beta c\|_\infty} \geq \frac{q}{n}\frac{y^T b - \beta\alpha}{\|y^T A - \beta c\|_\infty} \geq \frac{q}{n}\frac{y^T b - \beta\alpha}{\|A\|_\infty \|y\|_1 + \|c\|_\infty \beta},$$

the claim follows. $\qquad\square$

2   Property testing

Given an input, a *property tester* tries to distinguish whether the input is "in the ballpark" or "out of the ballpark" with respect to having a certain property. More formally, given a set of strings P with a fixed property and an input x, the goal is to decide whether x has the property (i.e., $x \in P$) or x is ϵ-*far* from having the property (i.e., x has large distance from every member of P, according to a specified distance metric, such as Hamming distance). Property testing is defined in [28, 17].

Because of the nature of the approximation, property tests with running times and query complexity which are sublinear in the size of the input can often be achieved. In fact, there are properties with testers that use a number of queries to the input which is independent of the size of the input, and depends only on the distance parameter.

Property testing can be viewed as an alternate type of an approximation problem. There are situations where the information yielded by property tests is the natural question to ask. Furthermore, such an approximation might be just as good as an exact answer when the data is constantly changing, or might be used as a fast sanity check to rule out very "bad" inputs before running a slow but more exact algorithm. Property testers and the techniques behind them have been applied to constructing probabilistically checkable proof systems and program checkers. Some property testers have been converted to give sublinear time approximation algorithms, for example for the Max-Cut problem [17]. It is interesting to note that there are examples of problems that are NP-hard to approximate but for which the property testing version of the problem can be decided in constant time.

Over the past decade, researchers in the community have found property testers for a variety of algebraic, graph-theoretic, and combinatorial properties. As is the case with approximation algorithms, one of the commonly used techniques for constructing property testers for graph properties

is to sample the graph and see if the property holds on the subgraph induced by the sample [17]. In fact, it has been shown that if a graph property has a constant query tester when the graph is presented in the adjacency matrix representation, then the property has a tester of the above form [20]. Here we give two examples of property testers, one for monotonicity of lists and the other for connectivity of degree-bounded graphs.

2.1 Monotonicity

A list of numbers $\vec{x} = x_1, \ldots, x_n$, is *monotone* (increasing) if $x_i \leq x_j$ for $i < j$. A property tester for monotonicity needs to distinguish lists that are monotone from those that are ϵ-far from monotone, i.e., more than $\epsilon \cdot n$ elements of the list need to be deleted in order to make the list monotone.

It is easy to construct examples showing that a constant number of tests of the form "is $x_i \leq x_j$?" or "is $x_i \leq x_{i+1}$?" for randomly chosen i, j will not suffice to test the property of monotonicity. In fact, the problem is known to require $\Omega(\log n)$ queries [10, 12]. The following logarithmic time algorithm from the work of Ergün, Kannan, Kumar, Rubinfeld, and Viswanathan [10] tests if $\vec{x}$ has a long monotone increasing subsequence, which in turn gives a property tester for monotonicity. Very efficient monotonicity testers for functions defined over posets, and more specifically for functions defined over hypercubes of high dimension, are given in [16, 8, 13].

For simplicity, let us assume that the elements in $\vec{x}$ are distinct. The last assumption is without loss of generality, since one can append the index of an item to the least significant bits of its value in order to break ties.

Theorem 3 ([10]). *There is an algorithm that, given a sequence $\vec{x} = x_1, \ldots, x_n$ and an $\epsilon > 0$, runs in $O((1/\epsilon) \log n)$ time and outputs (1) PASS, if $\vec{x}$ is monotone and (2) FAIL, with constant probability, if $\vec{x}$ does not have an increasing subsequence of length at least $(1 - \epsilon)n$ (in particular, if $\vec{x}$ is ϵ-far from monotone).*

Proof. The algorithm is the following. Let $i_1, \ldots, i_\ell$ be indices in $[n]$ chosen uniformly at random, where $\ell = O(1/\epsilon)$. For each such chosen index i_j, perform binary search in $\vec{x}$ as if to determine whether x_{i_j} is present in $\vec{x}$ or not. Output FAIL if the binary search fails to find x_{i_j}. Output PASS if all the ℓ binary searches succeed.

The running time of the algorithm is $O((1/\epsilon) \log n)$. Moreover, if $\vec{x}$ is monotone, then the algorithm would output PASS as each of the binary searches would succeed. Now, we assume that the input is such that the algorithm outputs PASS with probability at least $2/3$ and show that $\vec{x}$ has a long increasing subsequence. Let $G \subseteq [n]$ denote the set of indices for which the binary search would succeed, i.e., $i \in G$ if and only if x_i can be found by a binary search on $\vec{x}$. Since the algorithm outputs PASS with probability at least $2/3$, we know that $|G| \geq (1 - \epsilon)n$. We now argue that the restriction of $\vec{x}$ to the indices in G is an increasing subsequence, which would complete the proof. Let $i, j \in G$ and $i < j$. Let k be the least common ancestor index where the binary searches for x_i and x_j diverge. Then $x_i < x_k$ and $x_k < x_j$, which implies $x_i < x_j$. $\qquad\square$

2.2 Graph connectivity

In the work of Goldreich and Ron [19], testers for the properties of graph connectivity, k-edge connectivity and k-vertex connectivity are given for the adjacency list model. This model describes undirected graphs on n nodes and degree bounded by d. For such a graph G, $f_G(u, i)$ is defined to be the i-th neighbor of vertex u, if it exists, and 0 otherwise. The distance between two such graphs G, H is defined to be the fraction of places where f_G and f_H disagree, i.e., $|(u, i) : u \in$

$[n], i \in [d], f_G(u,i) \neq f_G(v,i)|/(dn)$. Note that every edge in the symmetric difference of the graphs is counted twice.

Here we describe their result for graph connectivity. A graph G is said to be (ϵ, d)-*far from being connected* if its distance to every connected graph (on the same number of nodes) of bounded degree d is at least ϵ.

Theorem 4 ([19]). *There is an algorithm that, given a graph G of bounded degree d, and an $\epsilon > 0$, runs in $O(1/(\epsilon^2 d))$ time and outputs (1) PASS, if G is connected and (2) FAIL, with constant probability, if G is (ϵ, d)-far from being connected.*

Proof. The main idea is to pick a random vertex s and see how large is the component C_s containing s. If G is connected, then $C_s = G$, whereas if G is far from being connected, then for most s, $|C_s|$ would be quite small. This is formalized in the following lemma:

Lemma 2. *Let $d \geq 2$. If G is (ϵ, d)-far from being connected, then it has at least $\epsilon dn/8$ connected components each with at most $8/(\epsilon d)$ vertices.*

Proof. The idea is that if G is (ϵ, d)-far from being connected, then it would require a lot of edges to hook up the connected components of G. Let $C_1, \ldots, C_\ell$ be the connected components of G. We bound the distance of G from being connected in terms of ℓ. The idea is to connect C_i and C_{i+1} in G by an edge to obtain G' which is connected. Care must be taken to respect the degree bound. Consider C_i. If there are two vertices with degree strictly less than d in C_i, then these vertices can be used to link C_i to $C_{i\pm 1}$. If there are no two such vertices, we can delete a single edge in C_i without disconnecting it—take any spanning tree T_i of C_i and consider an edge in C_i not present in T_i—and thereby create the two required vertices. Thus the symmetric difference between G and G' is at most 2ℓ. Since G is (ϵ, d)-far from being connected, we obtain $\ell \geq \epsilon dn/4$. Noting that the average number of vertices per component is $n/\ell \leq 4/(\epsilon d)$, the lemma follows by a counting argument. □

Now consider the following algorithm. Pick a vertex s uniformly at random and perform a BFS starting from s. If less than $8/(\epsilon d)$ vertices are encountered in the BFS, then output FAIL. Repeat the above test $16/(\epsilon d)$ times. If G is connected, the above will always PASS. If G is ϵ-far from connected, then by the above lemma, the algorithm is likely to output FAIL. The running time of this algorithm is $8/(\epsilon d) \cdot 16/(\epsilon d) \cdot d = O(1/(\epsilon^2 d))$. □

3 Properties of distributions

Sublinear time algorithms are also of use in testing *properties of distributions*. Suppose you are studying the occurrence of a disease and need to uncover any salient statistical properties that might hold. For example, it would be important to know if the probability of contracting the disease decreases with distance of your house from a given nuclear plant, whether the distribution on zipcodes of patients is close to the distribution for another disease, or whether a person's likelihood of contracting it is correlated with their profession. Of course, you wish to notice such trends given as few samples as possible. Such questions have arisen in many disciplines, including statistics, learning theory and data mining.

This yields a somewhat different model than the property testing model in terms of the assumption on how the data is presented—the tester is given access only to samples from the distributions (called the *generation oracle*). Properties such as closeness between two distributions, closeness to an explicitly given distribution, independence, etc., have been studied in this model [18, 6, 5]. For

many properties, well-known statistical techniques, such as the χ^2-test or the straightforward use of Chernoff bounds, have sample complexities that are at least linear in the size of the support of the underlying probability distribution. In contrast, there are algorithms whose sample complexity is sublinear in the size of the support for various properties of distributions; moreover, these algorithms are the best possible to within polylogarithmic factors [18, 6, 5].

3.1 Testing closeness to the uniform distribution

Given samples of a distribution X on $[n]$, for example all the previous winners of the New Jersey Pick 4 lottery (`http://www.state.nj.us/lottery/games/1-4-3_p4_history.shtml`), how can one tell whether X is close to uniform? For the purposes of this section, we will measure closeness in terms of the L_2 norm, i.e., $\|X\|_2 = \sum_{i \in [n]} (X(i) - 1/n)^2$. Goldreich and Ron [18] note that since $\sum_{i \in [n]} (X(i) - 1/n)^2 = \sum_{i \in [n]} X(i)^2 - 1/n$, it is enough to estimate the collision probability. They then show that this can be done by considering only $O(\sqrt{n})$ samples and counting the number of pairs that are the same. By bounding the variance of their estimator, they obtain the following:

Theorem 5 ([18]). *There is an algorithm that, given a distribution X on $[n]$ via a generation oracle, approximates $\|X\|_2$ to within a factor of $(1 \pm \epsilon)$ using $O(\sqrt{n}/\epsilon^2)$ samples, with constant probability.*

3.2 Testing identity

Let X, Y be distributions on $[n]$. Let X be available via a generation oracle and let Y be known to the algorithm (hardwired as $Y(1), \ldots, Y(n)$, and such that preprocessing operations on Y are not considered in the running time of the testing algorithm). The problem of testing if X is close to Y was considered by Batu, Fischer, Fortnow, Kumar, Rubinfeld, and White [5]. For the purposes of this section, we will measures closeness in terms of the L_1 norm, i.e., $\sum_{i \in [n]} |X(i) - Y(i)|$. We use the following notation. For $S \subseteq [n]$, let $X(S) = \sum_{i \in S} X(i)$. Let $X_{|S}$ denote the conditional distribution X w.r.t. R, i.e., $X_{|S}(i) = X(i)/X(S)$. Let U_S denote the uniform distribution on S.

Theorem 6 ([5]). *For any distribution Y on $[n]$, there is an algorithm that, given distribution X on $[n]$ via a generation oracle and $\epsilon > 0$, uses $\tilde{O}(\sqrt{n} \cdot \mathrm{poly}(1/\epsilon))$ samples and outputs (1) PASS, with constant probability, if $X = Y$ and (2) FAIL, with constant probability, if $\|X - Y\|_1 \geq 6\epsilon$.*

Proof. The main idea in the algorithm is to do *bucketing* of Y. Bucketing is a tool to decompose an arbitrary distribution via an evaluation oracle into a small collection of distributions that are nearly uniform. The decomposition is done by placing indices into one of logarithmically many buckets $B_0, B_1, \ldots, B_k$, where bucket $B_i, i > 0$ is designated to contain indices j such that $(1 + \epsilon)^{i-1}/(n \log n) \leq Y(j) < (1+\epsilon)^i/(n \log n)$; note that $k = 2/\log(1+\epsilon) \log n$. The bucket B_0 contains those indices j such that $Y(j) < 1/(n \log n)$; this bucket has sufficiently negligible mass so that it can be safely ignored for the rest of the proof. Since within a bucket, the maximum and minimum probability values are off only by a factor of $(1 + \epsilon)$, the distribution Y restricted to each bucket is close to the uniform distribution: $\|Y_{|B_i} - U_{B_i}\|_1 \leq \epsilon$.

Next for each of the buckets, the goal is to test if $X_{|B_i}$ is also close to uniform. Using Cauchy–Schwarz,

$$\|X_{|S} - U_S\|_1 \leq \sqrt{|S|} \cdot \|X_{|S} - U_S\|_2 = \sqrt{|S|} \cdot (\|X_{|S}\|_2^2 - \|U_S\|_2^2)^{1/2} = \sqrt{|S| \cdot (\|X_{|S}\|_2^2 - 1/|S|)}.$$

Thus, if $\|X_{B_i}\|_2^2 \leq (1 + \epsilon^2)/|B_i|$, then $\|X_{|B_i} - U_{B_i}\|_1 \leq \epsilon$. The problem boils down to estimating the two norm of $X_{|B_i}$, which is possible by Theorem 5.

7

The algorithm estimates $\|X_{B_i}\|_2$ for each i, using only a total of $\tilde{O}(\sqrt{n}/\epsilon^2)$ samples. The algorithm outputs FAIL if for any i, $\|X_{|B_i}\|_2^2 > (1 + \epsilon^2)/|B_i|$. If none of these steps fails, then we know that for all $i \in [k]$, $\|X_{|B_i} - Y_{|B_i}\|_1 \leq \epsilon$. One final step is for the algorithm to test if the distributions induced by X and Y on $[k]$ by collapsing all indices in a bucket into a 'super' index, are close in L_1-norm as well; this can be accomplished by brute-force as k is small. It is a simple argument to put these together and conclude that if the algorithm outputs PASS, then the distributions are in fact close to each other.

Conversely, suppose $X = Y$. Then the distributions $X_{|B_i} = Y_{|B_i}$ and so $\|X_{|B_i} - U_{B_i}\|_1 \leq \epsilon$. It is an easy calculation to see that in this case, $\|X_{|B_i}\|_2^2 \leq (1 + \epsilon^2)/|B_i|$ and so the algorithm will never output FAIL. $\qquad\square$

If both distributions are given via generation oracles, then the complexity of the problem changes—$\Theta(n^{2/3})$ samples are necessary and sufficient, up to polylogarithmic factors, [6].

Acknowledgments

References

[1] N. Alon, S. Dar, M. Parnas, and D. Ron. Testing of clustering. *SIAM Journal on Discrete Mathematics*, 16(3):393–417, 2003.

[2] N. Alon, W. Fernandez de la Vega, R. Kannan, and M. Karpinski. Random sampling and approximation of MAX-CSP problems. In *Proc. 34th Annual ACM Symposium on Theory of Computing*, pages 232–239, 2002.

[3] S. Arora, D. R. Karger, and M. Karpinski. Polynomial time approximation schemes for dense instances of NP-hard problems. *Journal of Computer and System Sciences*, 58(1):193–210, 1999.

[4] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proc. 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 1–16, 2002.

[5] T. Batu, E. Fischer, L. Fortnow, R. Kumar, R. Rubinfeld, and P. White. Testing random variables for independence and identity. In *Proc. 42nd IEEE Conference on Foundations of Computer Science*, pages 442–451, 2001.

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

[7] M. Charikar, L. O'Callaghan, and R. Panigrahy. Better streaming algorithms for clustering problems. In *Proc. 35th Annual ACM Symposium on Theory of Computing*, pages 30–39, 2003.

[8] Y. Dodis, O. Goldreich, E. Lehman, S. Raskhodnikova, D. Ron, and A. Samorodnitsky. Improved testing algorithms for monotonicity. In *Proc. 3rd International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 97–108, 1999.

[9] P. Drineas, R. Kannan, and M. Mahoney. Manuscript, 2003.

[10] F. Ergün, S. Kannan, R. Kumar, R. Rubinfeld, and M. Viswanathan. Spot checkers. *Journal of Computer and System Sciences*, 60:717–751, 2000.

[11] E. Fischer. The art of uninformed decisions: A primer to property testing. *Bulletin of the European Association for Theoretical Computer Science*, 75:97–126, 2001.

[12] E. Fischer. On the strength of comparisons in property testing. Technical Report TR01-008, Electronic Colloquium on Computational Complexity, 2001.

[13] E. Fischer, E. Lehman, I. Newman, S. Raskhodnikova, R. Rubinfeld, and A. Samorodnitsky. Monotonicity testing over general poset domains. In *Proc. 34th Annual ACM Symposium on Theory of Computing*, pages 474–483, 2002.

[14] A. Frieze and R. Kannan. Quick approximation to matrices and applications. *Combinatorica*, 19(2):175–220, 1999.

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

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

[17] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4):653–750, 1998.

[18] O. Goldreich and D. Ron. On testing expansion in bounded-degree graphs. Technical Report TR00-020), Electronic Colloquium on Computational Complexity, 2000.

[19] O. Goldreich and D. Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002.

[20] O. Goldreich and L. Trevisan. Three theorems regarding testing graph properties. In *Proc. 42nd IEEE Conference on Foundations of Computer Science*, pages 460–469, 2001.

[21] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *Proc. 41st IEEE Conference on Foundations of Computer Science*, pages 359–366, 2000.

[22] D. Haussler. Decision-theoretic generalizations of the PAC model for neural net and other learning applications. *Information and Computation*, 100(1):78–150, 1992.

[23] N. Mishra, D. Oblinger, and L. Pitt. Sublinear time approximate clustering. In *Proc. 12th Annual ACM–SIAM Symposium on Discrete Algorithms*, pages 439–447, 2001.

[24] N. Mishra, D. Ron, and R. Swaminathan. On finding large conjunctive clusters. In *Proc. 16th Annual Conference on Learning Theory*, 2003.

[25] S. Muthukrishnan. Data streams: algorithms and applications. In *Proc. 14th Annual ACM–SIAM Symposium on Discrete Algorithms*, page 413, 2003.

[26] D. Pollard. *Convergence of Stochastic Processes*. Springer–Verlag, 1984.

[27] D. Ron. Property testing (a tutorial). In S. Rajasekaran, P. M. Pardalos, J. H. Reif, and J. D. P. Rolim, editors, *Handbook on Randomization, Volume II*, pages 597–649. Kluwer Academic Press, 2001.

[28] R. Rubinfeld and M. Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, 1996.