Estimating the Distance to a Monotone Function[⋆]

Nir Ailon, Bernard Chazelle, Seshadhri Comandur, and Ding Liu

Department of Computer Science, Princeton University, Princeton NJ 08544, USA
{nailon,chazelle,csesha,dingliu}@cs.princeton.edu

Abstract. In standard property testing, the task is to distinguish between objects that have a property $\mathcal{P}$ and those that are ε-far from $\mathcal{P}$, for some $\varepsilon > 0$. In this setting, it is perfectly acceptable for the tester to provide a negative answer for every input object that does not satisfy $\mathcal{P}$. This implies that property testing in and of itself cannot be expected to yield any information whatsoever about the distance from the object to the property. We address this problem in this paper, restricting our attention to monotonicity testing. A function $f : \{1, \ldots, n\} \mapsto \mathbf{R}$ is at distance ε_f from being monotone if it can (and must) be modified at $\varepsilon_f n$ places to become monotone. For any fixed $\delta > 0$, we compute, with probability at least 2/3, an interval $[(1/2 - \delta)\varepsilon, \varepsilon]$ that encloses ε_f. The running time of our algorithm is $O(\varepsilon_f^{-1} \log\log \varepsilon_f^{-1} \log n)$, which is optimal within a factor of $\log\log \varepsilon_f^{-1}$ and represents a substantial improvement over previous work. We give a second algorithm with an expected running time of $O(\varepsilon_f^{-1} \log n \log\log\log n)$.

1 Introduction

Since the emergence of property testing in the nineties [12, 8], great progress has been made on a long list of combinatorial, algebraic, and geometric testing problems; see [11, 6, 4] for surveys. Property testing is a relaxation of the standard decision problem: Given a property $\mathcal{P}$, instead of determining exactly whether a given input object satisfies $\mathcal{P}$ or not, we require an exact answer only if the object satisfies the property or if it is far from doing so. This subsumes a notion of distance: Typically the object is said to be ε-far from $\mathcal{P}$ if at least a fraction ε of its description must be modified in order to enforce the property. The largest such ε is called the distance of the object to $\mathcal{P}$. In this setting, the tester can say "no" for every input object that does not satisfy $\mathcal{P}$, which precludes the leaking of any information regarding the distance of the object to the property.

This weakness has led Parnas, Ron, and Rubinfeld [10] to introduce the concept of *tolerant* property testing. Given $0 \le \varepsilon_1 < \varepsilon_2 \le 1$, a tolerant tester must accept all inputs that are not ε_1-far from $\mathcal{P}$ and reject all of those that are ε_2-far (and output anything it pleases otherwise). A related problem studied in [10]

[⋆] This work was supported in part by NSF grants CCR-998817, CCR-0306283, ARO Grant DAAH04-96-1-0181.

is that of estimating the actual distance of the object to the property within prescribed error bounds. In the model considered, all algorithms are randomized and err with probability at most 1/3. (or equivalently any arbitrarily small constant).

Testing the monotonicity of functions has been extensively studied [1–3, 5, 7, 9]. In the one-dimensional case, given a function $f : \{1, \ldots, n\} \mapsto \mathbf{R}$, after querying $O(\log n)/\varepsilon$ function values, we can, with probability at least 2/3, accept f if it is monotone and reject it if it is ε-far from being monotone [3]. These methods do not provide for tolerant property testing, however. Very recently, Parnas, Ron and Rubinfeld [10] designed sublinear algorithms for tolerant property testing and distance approximation for two problems: function monotonicity and clustering. If ε_f denotes the distance of f to monotonicity, their algorithm computes an estimate $\hat{\varepsilon}$ for ε_f that satisfies $(1/2)\varepsilon_f - \delta \leq \hat{\varepsilon} \leq \varepsilon_f + \delta$ with high probability. The query complexity and running time of their algorithm are both $\tilde{O}((\log n)^7/\delta^4)$ (the $\tilde{O}$ notation hides a factor of $(\log\log n)^{O(1)}$). The algorithm maintains and queries a data structure called an "index-value tree." Since the running time is sublinear, the tree is stored implicitly and only relevant portions are constructed whenever necessary, using random sampling to make approximate queries on the tree. Their construction is sophisticated and highly ingenious, but all in all quite involved.

We propose a simpler, faster, algorithm that is nearly optimal. Given any fixed $\delta > 0$, it outputs an interval $[(1/2 - \delta)\varepsilon, \varepsilon]$ that encloses ε_f with probability at least 2/3. The running time is $O(\varepsilon_f^{-1} \log\log \varepsilon_f^{-1} \log n)$, which is optimal within a factor of $\log\log \varepsilon_f^{-1}$. (The optimality proof is quite simple and omitted from this version.) One thing to note is the different use of δ: in our algorithm it is part of the multiplicative factor, whereas in [10] it is an additive term. To achieve the same multiplicative factor as in our algorithm, the additive term needs to be $\Theta(\delta\varepsilon_f)$. This makes the running time of Parnas et al.'s algorithm $\tilde{O}((\log n)^7/\varepsilon_f^4)$, for any fixed δ.

The starting point of our algorithm is the property tester of Ergun et al. [3], which relies on a key fact: There exist at least $\varepsilon_f n$ "critical" integers $i \in \{1, \ldots, n\}$; for i to be critical means that it is the (left or right) endpoint of an interval at least half of whose elements are in violation with i. Here i is said to violate j if either $i < j$ and $f(i) > f(j)$ or $i > j$ and $f(i) < f(j)$. By proving an upper bound on the number of critical integers, we are able to define a "signature" distribution for f which reflects its distance ε_f fairly accurately. Specifically, two functions with distances to monotonicity off by a factor of 2 (roughly) will have signatures that are distinguishable in time $O(\varepsilon_f^{-1} \log n)$. This provides us with a tolerant property tester for monotonicity. We can turn it into a distance approximator by using a one-way searching strategy, which we discuss below. Just as in [10], our algorithm extends to higher dimension.

We also present an improvement of our one-dimensional algorithm for small enough values of ε. We show how to estimate ε_f in time $O(\varepsilon_f^{-1} \log n \log\log\log n)$. Unlike in our previous algorithm, the number of steps in this one is itself a random variable; therefore, the running time is to be understood in the expected sense over the random bits used by the algorithm.

2 Estimating Distance to Monotonicity

Given two functions $f, g : \{1, \ldots, n\} \mapsto \mathbf{R}$, let $d(f, g) = \text{Prob}[f(x) \neq g(x)]$ denote the distance between f and g, where $x \in \{1, \ldots, n\}$ is chosen uniformly at random. We define $\varepsilon_f = \min_{g \in \mathcal{M}} d(f, g)$, where $\mathcal{M}$ is the set of monotone functions from $\{1, \ldots, n\}$ to $\mathbf{R}$.

Theorem 1. *For any fixed $\delta > 0$, we can compute an interval $[(1/2 - \delta)\varepsilon, \varepsilon]$ that encloses ε_f with probability at least $2/3$. The running time is $O(\varepsilon_f^{-1} \log\log \varepsilon_f^{-1} \log n)$.*

It is not entirely clear from the theorem that amplifying the probability of success can be achieved by simply repeating the algorithm enough times and taking a majority vote. What if we get different candidate intervals every time? We do not. As will soon become obvious, majority voting does, indeed, boost the probability of success arbitrarily close to 1.

It is easy to reduce the search for such an interval to a "distance separation" decision problem. Suppose that, given any $\varepsilon > 0$, one can tell in $O(\varepsilon^{-1} \log n)$ time and with probability at least $2/3$ whether $\varepsilon_f > \varepsilon$ or $\varepsilon_f < (1/2 - \delta)\varepsilon$. If $(1/2 - \delta)\varepsilon \leq \varepsilon_f \leq \varepsilon$, the algorithm can report anything. For each $k = 1, 2, \ldots$, we run the algorithm $c \log(k + 1)$ times with ε set to $\varepsilon_k = (1/2 - \delta)^k$, where c is a large enough constant, and we take a majority vote. We continue until we hear the report that $\varepsilon_f > \varepsilon_\ell$. By Chernoff's bound, the probability that $\varepsilon_{\ell+1} \leq \varepsilon_f \leq \varepsilon_{\ell-1}$ is at least $1 - \sum_{k \geq 0} O(1/ck^2) > 2/3$. The running time of $\sum_{1 \leq k \leq \ell} O(\log(k+1))\varepsilon_k^{-1} \log n$, which is $O(\varepsilon_f^{-1} \log\log \varepsilon_f^{-1} \log n)$ time, as claimed.

This does not quite do the job. Indeed, we are now left with the knowledge that ε_f falls in the interval $[\varepsilon_{\ell+1}, \varepsilon_{\ell-1}]$, which unfortunately is too big for our purposes. It is enclosed in the interval $[\varepsilon_0/5, \varepsilon_0]$, for some $0 < \varepsilon_0 < 1$, which we must now shrink to the right size. To do this we simply use the previous "distance separation" algorithm for the values $(1 - k\delta)\varepsilon_0$, for $0 \leq k \leq 1/\delta$. This allows us to pinpoint ε_f within an interval of the form $[(1/2 - O(\delta))\varepsilon, \varepsilon]$. Rescaling δ gives us the desired result. It thus suffices to prove the following lemma:

Lemma 1. *For any fixed $\varepsilon, \delta > 0$, we can decide, in time $O(\varepsilon^{-1} \log n)$ and with probability at least $2/3$, whether $\varepsilon_f > \varepsilon$ or $\varepsilon_f < (1/2 - \delta)\varepsilon$. If $(1/2 - \delta)\varepsilon \leq \varepsilon_f \leq \varepsilon$, the algorithm can report anything.*

2.1 A Separation Oracle

As mentioned in the introduction, the key to estimating the distance to monotonicity is to approximate the number of "critical" integers (to be defined in the next section). To identify a critical integer i, we need to find an interval starting or ending at i such that there are many violations with i in the interval. This is done through random sampling, to ensure a sublinear running time. The motivation for the following definitions on joint distributions of random variables will be made clear later in this section.

Let $\mathcal{D}$ be the joint distribution of m independent 0/1 random variables $x_1, \ldots, x_m$, which can be sampled independently. If $\mathbf{E}\, x_i \leq a$ for all i, then $\mathcal{D}$ is called *a-light*; else it is *a-heavy*. We describe an algorithm light-test which, given any $a < b$, determines whether a distribution is a-light or b-heavy.

Lemma 2. *If $\mathcal{D}$ is either a-light or b-heavy, for some fixed $a < b$, then with probability $2/3$ we can tell which is the case in $O(bm/(b-a)^2)$ time.*

Proof. Call light-test$(\{x_1, \ldots, x_m\}, c_0)$, where c_0 is chosen so that $c_1 \overset{\text{def}}{=} c_0(b-a)^2/b$ is a large enough constant. The algorithm runs in time proportional to $\sum_{k \geq 0} c_0 k(m/2^k) = O(c_0 m)$. To see why it works, we begin with a simple observation. Suppose that $\mathbf{E}\, x_i > b$, then at the k-th recursive call we sample x_i (if at all) exactly $c_0 k$ times; therefore, by Chernoff's bounds,

$$\text{Prob}[\hat{x}_i \leq (a+b)/2] = 2^{-\Omega(c_1 k)}$$

The same upper bound holds for the probability that $\hat{x}_i > (a+b)/2$, assuming that $\mathbf{E}\, x_i \leq a$. Suppose now that :

- $\mathcal{D}$ IS b-HEAVY: Let x_i be such that $\mathbf{E}\, x_i > b$. At the k-th recursion call, the probability that S' is empty is $2^{-\Omega(c_1 k)}$. Summing up over all k bounds the likelihood of erring by 1/3.
- $\mathcal{D}$ IS a-LIGHT: The probability that any given $\hat{x}_i$ exceeds $(a+b)/2$ is at most 1/3 (conservatively) and so erring any time before the size of S is recursively reduced to below c_1 is $\sum_{c_1 \leq k < |S|} 2^{-\Omega(k)} = 2^{-\Omega(c_1)} < 1/6$. After that stage, the probability of reaching a b-heavy verdict is at most $O(c_1(\log c_1) 2^{-\Omega(c_1)}) < 1/6$.

□

light-test (S, k)

```
For each $x \in S$, sample it $k$ times and compute the average $\hat{x}$;
Form $S' = \{ x \in S \mid \hat{x} > (a+b)/2 \}$.
If $|S'| = 0$, then output "$a$-light".
If $|S'| \geq |S|/2$, then output "$b$-heavy".
light-test$(S', k + c_0)$
```

2.2 Distance Separation: The Facts

Given $0 < \delta < 1/2$, the integer i is called *δ-big* if there exists $j > i$ such that

$$\left| \left\{ i \leq k \leq j \mid f(k) < f(i) \right\} \right| \geq (1/2 - \delta)(j - i + 1)$$

or, similarly, $j < i$ such that

$$\left| \left\{ j \leq k \leq i \mid f(k) > f(i) \right\} \right| \geq (1/2 - \delta)(i - j + 1).$$

Intuitively, integer i is big if $f(i)$ violates monotonicity with an abundance of witnesses. In the following we show that when δ is small, the number of δ-big integers approximates $\varepsilon_f n$ to within a factor of roughly 2.

Lemma 3. *(i) At least $\varepsilon_f n$ integers are 0-big; (ii) no more than $(2 + 4\delta/(1 - 2\delta))\varepsilon_f n$ integers are δ-big.*

Proof. Note that, for any $i < j$ such that $f(i) > f(j)$, either i or j (or both) is 0-big. Therefore, if we were to remove all the 0-big integers from the domain $\{1, \ldots, n\}$, the function f would become monotone; hence (i).

To prove (ii), let C be a set of $\varepsilon_f n$ integers in the domain of f over which the function can be modified to become monotone. An element i of C is called *high-critical* (resp. *low-critical*) if there is $j \notin C$ such that $j > i$ and $f(j) < f(i)$ (resp. $j < i$ and $f(j) > f(i)$). Note that the two definitions are exclusive. For each δ-big i, we choose a unique witness j_i to its bigness (which one does not matter). If $j_i > i$, then i is called *right-big*; else it is *left-big*. (Obviously, the classification depends on the choice of witnesses.)

To bound the number of right-bigs, we charge low-criticals with a credit scheme. (Then we apply a similar procedure to charge left-bigs.) Initially, each element of C is assigned 1 credit. For each right-big $i \notin C$ among $n, \ldots, 1$ in this order, *spread* one credit among all the low-criticals k such that $i \leq k \leq j_i$ and $f(k) < f(i)$. We use the word "spread" because we do not simply drop one credit into one account. Rather, viewing the accounts as buckets and credits as water, we pour one unit of water one infinitesimal drop at a time, always pouring the next drop into the least filled bucket. (There are other ways to describe this charging scheme, none of them quite as poetic.)

We now show that no low-critical ever receives an excess of $2 + 4\delta/(1 - 2\delta)$ credits. Suppose by contradiction that this were the case. Let i be the right-big that causes the low-critical k's account to reach over $2 + 4\delta/(1 - 2\delta)$. By construction i is not low-critical; therefore, the excess occurs while right-big i is charging the l low-criticals k such that $i < k \leq j_i$ and $f(k) < f(i)$. Note that, because $i \notin C$, any k satisfying these two conditions is a low-critical and thus gets charged. With the uniform charging scheme (remember the water?), this ensures that all of these l low-criticals have the same amount of credits by the time they reach the excess value, which gives a total greater than $l(2 + 4\delta/(1 - 2\delta))$. By definition of right-bigness, $l \geq (1/2 - \delta)(j_i - i + 1)$. But none of these accounts could be charged before step j_i; therefore,

$$(1/2 - \delta)(j_i - i + 1)(2 + 4\delta/(1 - 2\delta)) < j_i - i + 1,$$

which is a contradiction.

We handle left-bigs in a similar way by running now from left to right, ie, $i = 1, \ldots, n$. Since no integer can be both left-critical and right-critical, part (ii) of the lemma follows. □

2.3 Distance Separation: The Algorithm

We need one piece of terminology before describing the distance separation algorithm. Given an interval in $[u, v]$, we define two 0/1 random variables $\alpha[u, v]$

and $\beta[u, v]$: given random $i \in [u, v] \cap \{1, \dots, n\}$, $\alpha[u, v] = 1$ (resp. $\beta[u, v] = 1$) iff $f(u) > f(i)$ (resp. $f(i) > f(v)$). With probability at least 2/3, distance-separation (f, ε, δ). reports that $\varepsilon_f > \varepsilon$ (resp. $\varepsilon_f < (1/2 - \delta)\varepsilon$) if it is, indeed the case, and anything it wants if $(1/2 - \delta)\varepsilon \le \varepsilon_f \le \varepsilon$.

distance-separation (f, ε, δ)

```
Pick $s = \lceil (1 + \delta/2)\varepsilon^{-1} \ln 2 \rceil$ random $i \in \{1, \dots, n\}$.
For each $1 \le k \le (5/\delta) \ln n$, define $x^{(i)}_{2k-1} = \alpha[i, i + (1 + \delta/4)^k]$
and $x^{(i)}_{2k} = \beta[i - (1 + \delta/4)^k, i]$.
Let $\mathcal{D}$ be the distribution of $(x^{(1)}_1, x^{(1)}_2, \dots, x^{(2)}_1, x^{(2)}_2, \dots, x^{(s)}_1, x^{(s)}_2, \dots)$.
If $\mathcal{D}$ is $(1/2 - \delta/4)$-heavy, then output "$\varepsilon_f > \varepsilon$".
If $\mathcal{D}$ is $(1/2 - \delta/3)$-light, then output "$\varepsilon_f < (1/2 - \delta)\varepsilon$".
```

The algorithm assumes that both δ and ε/δ are suitably small. The requirement on δ is nonrestrictive. To make ε small, however, we use an artifice: set $f(i) = +\infty$ for $i = n + 1, \dots, O(n/\delta)$. We also need to assume that the algorithm used for distinguishing between light and heavy succeeds with probability at least $1 - \delta^2$ (instead of 2/3); to do that iterate it $\log \delta^{-1}$ times and take a majority vote. To prove the correctness of the algorithm, it suffices to show that:

– If $\varepsilon_f > \varepsilon$, then $\mathcal{D}$ is $(1/2 - \delta/4)$-heavy with probability $1/2 + \Omega(\delta)$:

  By Lemma 3 (i), more than εn integers are 0-big, so the probability of hitting at least one of them in the first step (and hence, of ensuring that $\mathcal{D}$ is $(1/2)/(1 + \delta/4)$-heavy) is at least $1 - (1 - \varepsilon)^s > 1/2 + \Omega(\delta)$.

– If $\varepsilon_f < (1/2 - \delta)\varepsilon$, then $\mathcal{D}$ is $(1/2 - \delta/3)$-light with probability $1/2 + \Omega(\delta)$:

  By Lemma 3 (ii), the number of $\delta/3$-big integers is less than $(1 - \delta)\varepsilon n$; therefore, the probability of missing all of them (and hence, of ensuring that $\mathcal{D}$ is $(1/2 - \delta/3)$-light) is at least $(1 - (1 - \delta)\varepsilon)^s > 1/2 + \Omega(\delta)$.

By running the whole algorithm $O(1/\delta^2)$ times and taking a majority vote, we can boost the probability of success to 2/3. By Lemma 2, the running time is $O(\varepsilon^{-1} \log n)$, as claimed (for fixed δ). This completes the proof of Lemma 1 and hence of Theorem 1.

2.4 A Faster Algorithm for Small Distances

We show in this section how to slightly improve the query complexity of the algorithm to

$$O(\min\{\log\log \varepsilon_f^{-1}, \log\log\log n\}\, \varepsilon_f^{-1} \log n).$$

The running time is now expected (over the random bits used by the algorithm). To do this, we need the following theorem:

Theorem 2. *We can compute an interval* $[\Omega(\varepsilon/\log n), \varepsilon]$ *that encloses* ε_f *with probability at least* $2/3$*. The expected running time is* $O(\varepsilon_f^{-1} \log n)$*.*

Using this theorem, it is clear that the factor $\log\log \varepsilon_f^{-1}$ in the distance estimation algorithm can be replaced by $\min\{\log\log \varepsilon_f^{-1}, \log\log\log n\}$. Indeed, instead of taking $k = 1, 2, 3, \ldots$, and running the separation oracle for each value of ε_k a number of times (ie, $c \log(k+1)$ times), we redefine ε_k to be $(1/2-\delta)^k\varepsilon$, where ε is the estimate returned by Theorem 2. Because the maximum value of k is now $O(\log\log n)$, the running time drops to $O(\min\{\log\log \varepsilon_f^{-1}, \log\log\log n\}\varepsilon_f^{-1} \log n)$.

To prove Theorem 2, we turn to a construction introduced by Goldreich et al. [7]. Define a subset P of pairs of integers: $(i, j) \in P$ if $j > i$, and $j - i$ is at most t, where t is the largest power of 2 that divides either i or j. This set has the following two properties:

- $|P| = \Theta(n \log n)$.
- For any $i < j$, there exists k $(i < k < j)$ such that both $(i, k) \in P$ and $(k, j) \in P$. This means, in particular, that for any violation (i, j) of f, there exists a "witness" (i, k) or (k, j) of the violation in the subset P.

Now, for a function f, let M be a maximum matching in the violation graph (the undirected graph whose vertex set is $\{1, \ldots, n\}$ and where i is connected to j if $i < j$ and $f(i) > f(j)$). It is known [7] that $|M| = \Theta(\varepsilon_f n)$; to be precise, $\frac{1}{2}\varepsilon_f n \leq |M| \leq \varepsilon_f n$. Let $Q \subseteq P$ be the set of violations of f in P. Consider the bipartite graph G with M on the left and Q on the right. Connect an edge between $(i, j) \in M$ and $(a, b) \in Q$ if $\{i, j\} \cap \{a, b\} \neq \emptyset$. By the second property above, and from the definition of a maximum matching, every node on the right has degree either 1 or 2, and every node on the left has degree at least 1; therefore, the cardinality of the right side is $\Omega(|M|)$. We would like to show that it is $O(|M| \log n)$. If we could do that, then by sampling from P and checking for violations, we could then estimate the size of Q and get the desired approximation. Unfortunately, it is not quite the case that the cardinality of the right side is always $O(|M| \log n)$. To fix this problem, we need to introduce some more randomness.

We slightly change the definition of P: for an integer $r \in [1, n]$ let P_r denote the subset of pairs defined as follows: $(i, j) \in P_r$ if $j - i$ is at most t, where t is the largest power of 2 that divides either $i + r$ or $j + r$. The set P_r still has the two properties above. In addition, if r is chosen uniformly at random then, for any i, the expected number of j such that $(i, j) \in P_r$ and j' such that $(j', i) \in P_r$ is $O(\log n)$. The expected number of edges of the corresponding bipartite graph G_r, therefore, is $O(|M| \log n)$. So the expected cardinality of the right side is $\alpha|P_r|$, where $\alpha \in [\Omega(\varepsilon_f/\log n), O(\varepsilon_f)]$. We sample P_r to form an estimation $\hat{\alpha}$ for α and return $\varepsilon = C\hat{\alpha}\log n$, for some large enough constant C, to prove

Theorem 2. The estimation follows the predictable scheme: (1) pick a random $r \in \{1, \ldots, n\}$; (2) pick a pair (i, j) uniformly at random from P_r; (3) if (i, j) is a violation of f, output `success`, otherwise `failure`. The success probability is precisely α, so repeating the sampling enough times sharpens our estimation to the desired accuracy, as indicated by the following fact.

Lemma 4. *Given a 0/1 random variable with expectation $\alpha > 0$, with probability at least $2/3$, the value of $1/\alpha$ can be approximated with a relative constant error by sampling it $O(1/\alpha)$ times on average. Therefore, α can be approximated within the same error and the same expected running time.*

Proof. Run Bernoulli trials on the random variable and define Y to be the number of trials until (and including) the first 1. It is a geometric random variable with $\mathbf{E}\,Y = 1/\alpha$, and $\mathbf{var}\,(Y) = (1-\alpha)/\alpha^2 \leq (\mathbf{E}\,Y)^2$. By taking several samples of Y and averaging we get an estimate $\widehat{1/\alpha}$ of $1/\alpha$. Using Chebyshev's inequality, a constant number of samples suffices to get a constant factor approximation.

□

References

1. Batu, T., Rubinfeld, R., White, P. *Fast approximate PCPs for multidimensional bin-packing problems*, Proc. RANDOM (1999), 245–256.
2. Dodis, Y., Goldreich, O., Lehman, E., Raskhodnikova, S., Ron, D., Samorodnitsky, A. *Improved testing algorithms for monotonicity*, Proc. RANDOM (1999), 97–108.
3. Ergun, F., Kannan, S., Kumar, S. Ravi, Rubinfeld, R., Viswanathan, M. *Spot-checkers*, Proc. STOC (1998), 259–268.
4. Fischer, E. *The art of uninformed decisions: A primer to property testing*, Bulletin of EATCS, 75: 97-126, 2001.
5. Fischer, E., Lehman, E., Newman, I., Raskhodnikova, S., Rubinfeld, R., Samorodnitsky, A. *Monotonicity testing over general poset domains*, Proc. STOC (2002), 474–483.
6. Goldreich, O. *Combinatorial property testing - A survey*, in "Randomization Methods in Algorithm Design," 45-60, 1998.
7. Goldreich, O., Goldwasser, S., Lehman, E., Ron, D., Samordinsky, A. *Testing monotonicity*, Combinatorica, 20 (2000), 301–337.
8. Goldreich, O., Goldwasser, S., Ron, D. *Property testing and its connection to learning and approximation*, J. ACM 45 (1998), 653–750.
9. Halevy, S., Kushilevitz, E. *Distribution-free property testing*, Proc. RANDOM (2003), 302–317.
10. Parnas, M., Ron, D., Rubinfeld, R. *Tolerant property testing and distance approximation*, ECCC 2004.
11. Ron, D. *Property testing*, in "Handbook on Randomization," Volume II, 597-649, 2001.
12. Rubinfeld, R., Sudan, M. *Robust characterization of polynomials with applications to program testing*, SIAM J. Comput. 25 (1996), 647–668.