Electronic Colloquium on Computational Complexity, Report No. 10 (2004)

ECCC

Tolerant Property Testing and Distance Approximation

Michal Parnas
The Academic College
of Tel-Aviv-Yaffo
Tel-Aviv, ISRAEL
michalp@mta.ac.il

Dana Ron
Department of EE – Systems
Tel-Aviv University
Ramat Aviv, ISRAEL
danar@eng.tau.ac.il

Ronitt Rubinfeld
NEC Research Institute
Princeton, NJ
ronitt@research.nj.nec.com

January 26, 2004

Abstract

A standard property testing algorithm is required to determine with high probability whether a given object has property $\mathcal{P}$ or whether it is ϵ-*far* from having $\mathcal{P}$, for any given distance parameter ϵ. An object is said to be ϵ-*far* from having property $\mathcal{P}$ if at least an ϵ-fraction of the object should be modified so that it obtains $\mathcal{P}$. In this paper we study a generalization of standard property testing where the algorithms are required to be more *tolerant*. Specifically, a *tolerant property testing algorithm* is required to accept objects that are ϵ_1-close to having a given property $\mathcal{P}$ and reject objects that are ϵ_2-far from having $\mathcal{P}$, for some parameters $0 \leq \epsilon_1 < \epsilon_2 \leq 1$. Another related natural extension of standard property testing that we study, is *distance approximation*. Here the algorithm should output an estimate $\hat{\epsilon}$ of the distance of the object to $\mathcal{P}$, where there are certain provable upper and lower bounds on $\hat{\epsilon}$ in terms of the correct distance.

We first formalize the notions of tolerant property testing and distance approximation and discuss the relationship between the two tasks, as well as their relationship to standard testing. We then study two problems: The first is distance approximation for monotonicity, and the second is tolerant testing of clustering. We present and analyze algorithms whose query complexity is either polylogarithmic or independent of the size of the input. Our distance approximation algorithm for monotonicity works by defining a certain tree structure by which upper and lower bounds on the distance of the function to monotonicity can be obtained, and then constructing only a small number of random paths in the tree. Our tolerant testing algorithms for clustering exploit a general framework (based on [CS02]) which may be applicable for tolerant testing of other cost measures for clustering, as well as for tolerant testing of other properties.

ISSN 1433-8092

1 Introduction

The past decade or so has seen a surge of research in the area of *property testing* [RS96, GGR98]. Property testing can be viewed as a relaxation of *decision problems*. In a typical decision problem it is required to determine whether an object has or does not have a given property $\mathcal{P}$. In property testing it is required to determine with high probability whether the object has property $\mathcal{P}$ or whether it is *far* from having $\mathcal{P}$. An object is said to be ϵ-*far* from having property $\mathcal{P}$ if at least an ϵ-fraction of the object should be modified so that it obtains $\mathcal{P}$. This relaxation allows for very efficient algorithms, whose complexity is sublinear in the input size and in many cases independent of the input size. In the past few years, many results have been given in this framework. Examples of objects for which testing algorithms were developed are functions, graphs, strings, and geometrical objects (see [Gol98, Fis01, Ron01] for surveys).

In all that follows we refer to the original notion of property testing as *standard* property testing. As defined above, a standard property tester should accept with high probability objects that have the property, but it is allowed to reject objects that are very close to having the property. For example, if the object is a function $f : \{1, \ldots, n\} \to \Re$ and the property is monotonicity, then the algorithm may reject the function even if it can be made monotone by modifying the function on just a single point.

In this paper we study a generalization of standard property testing where the algorithms are required to be more *tolerant*. Specifically, a *tolerant property testing algorithm* is required to accept objects that are ϵ_1-close to having a given property $\mathcal{P}$ and reject objects that are ϵ_2-far from having $\mathcal{P}$, for some parameters $0 \leq \epsilon_1 < \epsilon_2 \leq 1$. It is of course desirable for the tolerant algorithm to run for any given ϵ_1 and ϵ_2, and for its query complexity and running time to be sublinear in the input size and polynomial in $1/(\epsilon_2 - \epsilon_1)$. However, we allow for other variants as well, such as when ϵ_1 and/or ϵ_2 are fixed. In particular, if we set $\epsilon_1 = 0$ and allow ϵ_2 to be a parameter, then we get the standard definition of testing.

As touched upon above, this is not only a natural generalization of standard property testing but in many cases it is the desired notion of testing. For example, when evaluating the clusterability of a set of points, a standard testing algorithm is required to accept with high probability if it is possible to cluster all n input points into at most k clusters with a bounded diameter (radius), and is required to reject with high probability when more than ϵn points should be removed (or moved) so that the remaining points can be clustered as desired (or possibly with a slightly larger diameter or radius). However, in many applications we would like an algorithm that is ensured to accept with high probability even when there is no good clustering of all points but rather there is such a clustering of all but a small fraction of the points. A tolerant testing algorithm will allow exactly this.

Another natural extension of standard property testing that we study is *distance approximation*. Namely, we seek efficient algorithms that approximate the distance of an object to a given property $\mathcal{P}$. Here the algorithm should output an estimate $\hat{\epsilon}$ of the distance of the object to $\mathcal{P}$, where there are certain provable upper and lower bounds on $\hat{\epsilon}$ in terms of the correct distance. Clearly, the closer the upper and lower bounds that one can prove, the better the quality of the distance approximation.

It is not hard to verify, and we discuss this in more detail in Section 3, that every distance approximation algorithm can be used for tolerant testing (where the settings of ϵ_1 and ϵ_2 for which the tolerant testing algorithm works depend on the quality of the estimate $\hat{\epsilon}$ that the distance approximation algorithm obtains). In some cases tolerant testing algorithms can be used to obtain

a distance approximation. However, this is not always the case, and hence there is a need to allow for both notions.

As we further discuss subsequently, there are quite a few tolerant testing results and distance approximation results implied by previous work. These results are mostly implicit and sometimes weak (in terms of the tolerance or approximation quality), but there are some explicit and stronger results. In this paper we are interested in putting the spotlight on these generalizations of standard property testing as well as presenting new results that cannot be derived from known results for standard testing.

1.1 Our Results

As stated above, we define the notions of tolerant property testing and distance approximation and discuss the relationship between the two tasks. We then present the following results.

Distance Approximation for Monotonicity. Our first result deals with the property of monotonicity, which has been studied quite extensively in the context of standard property testing [EKK+00, GGL+00, DGL+99, BRW99, FLN+02, HK03b]. Our algorithm is given query access to a function $f : [n] \to \Re$, and outputs an estimate $\hat{\epsilon}$ such that with probability at least 2/3 it holds:

$$(1/2)\epsilon_{\mathrm{mon}}(f) - \delta \ \leq \ \hat{\epsilon} \leq \epsilon_{\mathrm{mon}}(f) + \delta$$

where $\epsilon_{\mathrm{mon}}(f)$ is the distance of f to the closest monotone function, and δ is any given parameter. The query complexity and running time are polynomial in $\log n$ and $1/\delta$.

In higher dimensions, that is when $f : [n]^d \to \Re$, we can use a dimension-reduction lemma of Halevy and Kushilevitz [HK03a] to obtain an algorithm that outputs an estimate $\hat{\epsilon}$ such that with probability at least 2/3 it holds: $(1/(d4^{d-1}))\epsilon_{\mathrm{mon}}(f) - \delta \ \leq \ \hat{\epsilon} \leq \epsilon_{\mathrm{mon}}(f) + \delta$. The running time of the algorithm is polynomial in $\log n$ and $1/\delta$. Thus this extension is appropriate for small values of d.

Tolerant Testing of Clustering. We present efficient tolerant testing algorithms for clustering for both general metrics and the Euclidean metric under the diameter cost. A set X of points is said to be (k, b)-*clusterable* under the diameter cost if X can be partitioned into k clusters so that the diameter of each cluster (i.e., the maximum distance between a pair of points in the cluster) is at most b. The algorithms we present accept with probability at least 2/3 if X is ϵ_1-close to (k, b)-clusterable, and reject with probability at least 2/3 if X is ϵ_2-far from $(k, (1+\beta)b)$-clusterable, where β is either a fixed constant or a parameter to the algorithm. Specifically,

- In the case of general metrics the query complexity of the algorithm is $\Theta\left(\frac{k}{(\epsilon_2-\epsilon_1)^2} \cdot \log \frac{k}{\epsilon_2-\epsilon_1}\right)$ and $\beta = 1$. We also note that an analogous result holds for clustering with respect to the radius cost.
- When X is in d-dimensional Euclidean space then for any given $0 < \beta \leq 1$, the query complexity is $\tilde{\Theta}\left(\frac{k}{(\epsilon_1-\epsilon_1)^2} \cdot (1 + (2/\beta))^d\right)$. As shown in [ADPR03], the exponential dependence on d, as well as some dependence on $1/\beta$, are unavoidable.

Furthermore, as is the case for many standard testing algorithms, our tolerant testing algorithm can actually be used to obtain an **approximately good clustering** of the input points. More precisely, if the set of points is ϵ_1-close to being (k, b)-clusterable, then with high probability the

algorithm outputs an implicit representation of a $(k, (1+\beta)b)$-clustering of all but an ϵ_2-fraction of the points. In an "implicit representation" we mean a partition of a small subset of points that can be used to induce the desired (approximately good) clustering.

A general framework for tolerant testing. Our tolerant testing algorithms for clustering exploit a general framework which may be applicable for tolerant testing of other cost measures for clustering, as well as for tolerant testing of other properties. The framework presented in this paper builds on the abstract combinatorial programs of Czumaj and Sohler [CS02] and expands the ideas presented there to fit the requirements of tolerant property testing. For the ease of the presentation, and since our application of the framework is to clustering, we focus on presenting the framework in the context of clustering. We later show how the framework can be easily adapted to tolerant testing of hereditary graph partition properties.

1.2 Techniques

Monotonicity and an imposed tree structure. Most standard property testing algorithms work by trying to find evidence that the tested object does not have the property in question: If evidence is found then the algorithm rejects, otherwise it accepts. Clearly, a single piece of evidence does not suffice for a tolerant testing algorithm to make a decision or for a distance approximation algorithm to come up with a good estimate for the distance to the property. A natural extension of this idea would be to make a decision, or find an estimate, based on the "amount of evidence" found in the answers to the queries. In the case of distance approximation to monotonicity, a naive suggestion would be to simply take a uniform random sample of points and compute the relative distance to monotonicity on the sample. It is not hard to verify that this algorithm fails miserably (even for standard testing). Alternatively, one may consider taking a biased random sample (as done in standard testing algorithms for monotonicity). The difficulty with this approach is that the function can actually be close to monotone, but the biased sample may tend to select those points which give rise to violations of monotonicity.

Here we suggest a different approach: We define a certain tree structure that can be determined for any given function $f : [n] \to \Re$. This tree contains information that can be used to obtain upper and lower bounds on the distance of the function to monotonicity. While reading all information in the tree would take linear time, we show that it actually suffices to obtain information from a small number of tree nodes that are selected randomly (and furthermore, one does not need the exact information from each node). The difficulty is that we do not actually have query access to the tree but rather to the function. However, we can show that by querying the function, the algorithm can effectively query the tree. Namely, the algorithm uses queries to the function in order to adaptively construct a small random partial tree and to estimate the information at its nodes so as to obtain the desired estimate.

The framework for tolerant testing. As stated in the previous subsection, our tolerant testing algorithms for clustering exploit a general framework which is suitable for proving the correctness of certain tolerant testing algorithms and which builds on the work of Czumaj and Sohler [CS02]. Below we briefly discuss the ideas behind this framework.

Assume for the sake of discussion that the object tested is a function and the property tested is $\mathcal{P}$. Many of the previous standard testing algorithms applied the "natural" algorithm, which selects a small sample and checks if the sub-function defined by this sample has property $\mathcal{P}$. Although

the algorithm is simple, it is usually not an easy task to prove that it works correctly (if at all). The common method of proof applied was to view the sample as two sub-samples, where the first sub-sample is used as a *skeleton* that induces certain constraints on many points in the domain of the function. These constraints are *always* satisfied in case the function has the property. The heart of the proof is then focused in showing that in case the function is far from having $\mathcal{P}$, then necessarily there are many points that violate the constraints induced by the skeleton. The second sub-sample is then used to provide *witnesses* to these violations. In order to circumvent the use of two sub-samples, Czumaj and Sohler defined a few conditions that should hold regarding the skeletons and the witnesses (once these are defined properly), from which the correctness of the natural algorithm follows.[1] This allows to separate between the combinatorial structure of the proof and the error probability analysis.

In the case of tolerant property testing, the framework of [CS02] described above is not suitable as it is, since there may be witnesses to violations also when the function is ϵ_1-close to $\mathcal{P}$. Thus the natural algorithm has to be modified so that it accepts also samples that are close to having $\mathcal{P}$. This change requires to prove that if the function is ϵ_1-close to $\mathcal{P}$ then the sub-function induced by a small sample is also close (as a function of ϵ_1) to $\mathcal{P}$, and on the other hand if the function is ϵ_2-far from $\mathcal{P}$ then all skeletons will provide many (as a function of ϵ_2) witnesses to the violations. We would like to emphasize that while this framework points the way to the claims that need to be proved, it is still necessary to define in an appropriate manner (which is problem specific) the skeletons and witnesses, and to prove that all necessary requirements are met.

1.3 Related Results Implied by Previous Work

Many standard testing algorithms ask queries that are uniformly distributed (though not necessarily independent). It is not hard to show, as we do in Section 3.2, that any such standard testing algorithm A implies a tolerant testing algorithm with roughly the same query complexity. The tolerant testing algorithm works for every ϵ_1 and ϵ_2 such that $\epsilon_1 = O(1/Q_A(\epsilon, n))$ and $\epsilon_2 = \epsilon$, where $Q_A(\epsilon, n)$ is the query complexity of the standard testing algorithm (with parameters ϵ and n). Since in many cases the query complexity of the standard testing algorithm grows like $1/\epsilon^d$ where $d \geq 2$, the tolerance we get is quite weak. In cases where the query complexity is only linear in $1/\epsilon$ (e.g., for linearity [BLR93]) we get a stronger result: A constant ratio between ϵ_1 and ϵ_2.

In what follows we briefly mention a few specific examples of results that we are aware of, which are either implicit or explicit in previous work. We stress that this is not a comprehensive list of all tolerant testing and distance approximation results that can be deduced from previous work.

Tolerant Testing for the Edit Distance. Consider the property of pairs of strings that have small edit distance. Note first that it is trivial to construct a standard property tester which passes pairs of strings that are identical and fails strings that have high enough edit distance. However, there are many practical situations, such as that of trying to determine which DNA sequences are similar, in which some degree of tolerance is required, since the similar DNA sequences are almost never identical. In [BEK+03], a tolerant property tester is constructed which given a pair of strings, each of length n, and a fixed parameter $\alpha < 1$, accepts when their edit distance is $O(n^\alpha)$, and rejects when their edit distance is $\Omega(n)$. The running time of the algorithm is $\tilde{O}(n^{\max\{\alpha/2, 2\alpha-1\}})$.

[1] We note that Czumaj and Sohler use a slightly different terminology (of "bases" and "violation functions").

Distance Approximation for k-Colorability and Other Partition Problems. The standard testing algorithm for k-colorability (and in particular for bipartiteness) in dense graphs [GGR98, AK02] works by taking a uniformly selected sample of vertices of size poly$(1/\epsilon)$, and querying on all pairs of vertices selected in order to obtain the induced subgraph. Thus its queries are uniformly distributed and by the discussion at the start of this subsection it is actually a tolerant testing algorithm. However, since it asks poly$(1/\epsilon)$ queries, its tolerance is quite low. Nonetheless, stronger tolerance, and moreover, a good distance approximation algorithm, can be obtained. Specifically, one should consider algorithms that approximate the size of the maximum k-cut to within an additive error of δn^2 (for any given $0 < \delta < 1$, using poly$(1/\delta, k)$ queries). Such algorithms are given in [GGR98, FK99]. Since every partition that maximized the number of cut edges, minimized the number of ("violating") edges within each part of the partition, such algorithms can be used to obtain estimates to the distance to k-colorability. Similar distance approximation results can be obtained for other partition properties [GGR98] such as being a clique of a certain size.

Approximate Clustering with Outliers. The recent work [COP03] deals, among other things, with streaming algorithms for clustering with outliers. One of their results is an algorithm for finding approximately good k-clustering with respect to the radius cost under a general metric. Specifically, similarly to what we obtain (for general metrics and the radius cost), if the input set X is ϵ_1-close to being (k, b)-clusterable, then their algorithm finds (with high probability) k centers of a $(k, 3b)$ clustering of all but an ϵ_2-fraction of the points. The sample size used by the algorithm depends polynomially on $1/(\epsilon_2 - \epsilon_1)$, but as opposed to our algorithm, also has a logarithmic dependence on $n = |X|$ (even when required to work only with high constant probability) while we have no such dependence.

Testing Properties of Distributions. In a somewhat different model that is related to property testing, one is given samples of a distribution and would like to determine whether the distribution has, or is close to having, a certain property. For example, given a pair of distributions whose support is of size n, one would want to distinguish distributions that are close from those that are far in some distance such as the L_1 or L_2 distance. Using the techniques of [GR00], one can approximate the L_2 distance between pairs of distributions in time independent of n. In [BFR$^+$00] a tester is given which passes distributions that are within $\epsilon/n^{1/3}$ from each other in L_1 norm, and fails distributions that are further than ϵ in L_1 norm.

1.4 Organization

- **Section 2:** Definitions of tolerant testing and distance approximation.
- **Section 3:** Discussion of the relation between tolerant testing and distance approximation, and between standard testing and tolerant testing.
- **Section 4:** A distance-approximation algorithm for monotonicity.
- **Section 5:** A tolerant property testing algorithm for clustering.
- **Section 6:** Here we show how to modify the framework developed for tolerant clustering in order to test hereditary graph partition properties.

2 Definition of Problems

We consider two types of related problems. We describe them with respect to functions, but they are easily adaptable to other objects whose natural representation is not necessarily functional. We first define the distance between functions and the distance of a function to a property. In what follows n is the size of the domain of the functions. With a slight abuse of notation we let $g \in \mathcal{P}$ mean that the function g has the property $\mathcal{P}$.

Definition 1 (Distance to a Property) *For two functions f and g over the same domain X, we let* $\mathrm{dist}(f,g) \stackrel{\text{def}}{=} \Pr[f(x) \neq g(x)]$ *denote the* distance *between the two functions, where x is chosen according to the uniform distribution over the domain X.*

For a function f and a property $\mathcal{P}$, we let $\mathrm{dist}(f,\mathcal{P}) \stackrel{\text{def}}{=} \min_{g\in\mathcal{P}} \mathrm{dist}(f,g)$ *denote the distance of f to having property $\mathcal{P}$. We shall sometimes use the shorthand $\epsilon_{\mathcal{P}}(f)$ for* $\mathrm{dist}(f,\mathcal{P})$. *For a distance parameter $0 \leq \epsilon < 1$ we say that f is ϵ-*far *from $\mathcal{P}$ if $\epsilon_{\mathcal{P}}(f) > \epsilon$, otherwise f is ϵ-*close *to $\mathcal{P}$.*

We are now ready to define tolerant testing and distance approximation.

Definition 2 (Tolerant Property Testing) *Let $\mathcal{P}$ be a property, and let $0 \leq \epsilon_1 < \epsilon_2 \leq 1$. An (ϵ_1,ϵ_2)-*Tolerant Testing *algorithm for property $\mathcal{P}$ is given query access to an unknown function f. The algorithm should accept with probability at least $2/3$ if f is ϵ_1-close to property $\mathcal{P}$, and should reject with probability at least $2/3$ if f is ϵ_2-far from the property.*

Ideally we would like the algorithm to work for *every* given pair ϵ_1, ϵ_2 such that $\epsilon_1 < \epsilon_2$, where we allow the complexity of the algorithm to grow with $1/(\epsilon_2 - \epsilon_1)$. However, one may also want to consider tolerant testers that work for some specific values of ϵ_1 and ϵ_2, or it may be the case that the algorithm works for every ϵ_2 but where $\epsilon_1 = h(\epsilon_2, n)$ for some function $h : [0,1] \times \mathcal{N} \to [0,1]$ (e.g. $\epsilon_1 = \epsilon_2^2/\log n$). Note that in the latter case, if we let h be the all-0 function then we get the standard (non-tolerant) definition of property testing.

Definition 3 (Distance Approximation) *Let $h_1, h_2 : [0,1] \times \mathcal{N} \to [0,1]$ be any two functions that are monotonically increasing in their first parameter and monotonically non-decreasing in their second parameter. An (h_1,h_2)-*Distance Approximation *algorithm for property $\mathcal{P}$ is given query access to an unknown function f. The algorithm should output an estimate $\hat{\epsilon}$ to the distance of f to the property $\mathcal{P}$, such that with probability at least $2/3$ it holds that $h_1(\epsilon_{\mathcal{P}}(f), n) \leq \hat{\epsilon} \leq h_2(\epsilon_{\mathcal{P}}(f), n)$.*

It is desired of course for the functions h_1 and h_2 to be independent of n. In particular, the strongest definition of distance approximation is obtained when $h_1(\epsilon_{\mathcal{P}}(f), n) = (1-\gamma)\cdot\epsilon_{\mathcal{P}}(f)$ and $h_2(\epsilon_{\mathcal{P}}(f), n) = (1+\gamma)\cdot\epsilon_{\mathcal{P}}(f)$, so that we actually consider a family of pairs of functions where each pair in the family is determined by the parameter γ. In such a case the complexity of the algorithm would be allowed to grow with $1/\epsilon_{\mathcal{P}}(f)$ as well as with $1/\gamma$. We will be interested in a slightly weaker notion in which h_1 and h_2 are of the form $h_1(\epsilon_{\mathcal{P}}(f), n) = c_1 \cdot \epsilon_{\mathcal{P}}(f) + \delta$ and $h_2(\epsilon_{\mathcal{P}}(f), n) = c_2 \cdot \epsilon_{\mathcal{P}}(f) - \delta$ where c_1 and c_2 are constants, and $0 < \delta < 1$ is a parameter. In this case we allow the complexity of the algorithm to depend on $1/\delta$. Ideally for functions of this form, we would like that $c_1 = c_2 = 1$, but we may only be able to achieve weaker multiplicative approximations.

Parameterized Properties

For some properties we may want to consider the case that $\mathcal{P} = \{\mathcal{P}_s\}$ is a parameterized family of properties. For example, we consider testing whether a set of points can be clustered into at most k clusters, each having diameter b. Then the size parameters of the property are k and b. In such a case we may want to relax the definition of tolerant testing so as to allow a small modification in the size parameter.

Definition 4 (Tolerant Testing of Parameterized Properties) *Let $\mathcal{P} = \{\mathcal{P}_s\}$ be a parameterized family of properties where $s \in \Re^\ell$ is a vector of size parameters, and let $0 \leq \epsilon_1 < \epsilon_2 \leq 1$ and $\beta : \Re^\ell \to \Re^\ell$. An $(\epsilon_1, \epsilon_2, \beta)$-*Tolerant Testing *algorithm for the family $\mathcal{P}$ is given $s \in \Re^\ell$ and query access to an unknown function f. The algorithm should accept with probability at least $2/3$ if f is ϵ_1-close to property $\mathcal{P}_s$, and should reject with probability at least $2/3$ if f is ϵ_2-far from $\mathcal{P}_{\beta(s)}$.*

For example, in the case of clustering, we ask that the algorithm accept if at most an ϵ_1-fraction of the points should be removed so that the remaining points can be clustered into k clusters with diameter at most b, and we ask that it reject if more than an ϵ_2-fraction of the points should be removed so that the remaining points can be clustered into k clusters with diameter at most b', for b' not much larger than b.

3 General Observations

3.1 On the Relation between Tolerant Testing and Distance Approximation

Given any distance approximation algorithm, we can directly obtain a corresponding tolerant testing algorithm. The actual values of ϵ_1 and ϵ_2 for which the tolerant testing algorithm can be applied, depend of course on the functions h_1 and h_2 for which the distance approximation algorithm works.

Claim 1 *Let A be an (h_1, h_2)-distance approximation algorithm. Then for every ϵ_1 and ϵ_2 and n that satisfy $h_2(\epsilon_1, n) < h_1(\epsilon_2, n)$, we can obtain an (ϵ_1, ϵ_2)-tolerant testing algorithm B, where the query complexity of B is the same as that of A.*

Proof: Let ϵ_1 and ϵ_2 be such that $h_2(\epsilon_1, n) < h_1(\epsilon_2, n)$. Algorithm B runs the distance approximation algorithm A to get an approximation $\hat{\epsilon}$. If $\hat{\epsilon}$ satisfies $\hat{\epsilon} \leq h_2(\epsilon_1, n)$, then B accepts, otherwise B rejects.

By definition of A, if $\epsilon_{\mathcal{P}}(f) \leq \epsilon_1$ then with probability at least $2/3$,

$$\hat{\epsilon} \leq h_2(\epsilon_{\mathcal{P}}(f), n) \leq h_2(\epsilon_1, n)$$

and B accepts as required, where the second inequality follows from the fact that h_2 is monotonically increasing in its first parameter.

On the other hand, if $\epsilon_{\mathcal{P}}(f) > \epsilon_2$ then with probability at least $2/3$,

$$\hat{\epsilon} \geq h_1(\epsilon_{\mathcal{P}}(f), n) > h_1(\epsilon_2, n)$$

and B rejects, as required. ■

Consider for example the special case where A works for $h_1(\epsilon_{\mathcal{P}}(f), n) = c_1 \cdot \epsilon_{\mathcal{P}}(f) - \delta$ and $h_2(\epsilon_{\mathcal{P}}(f), n) = c_2 \cdot \epsilon_{\mathcal{P}}(f) + \delta$ (so that the functions h_1 and h_2 are parameterized by δ). Let the

number of queries that algorithm A performs be $Q_A(\delta, n)$, and let its running time be $T_A(\delta, n)$. Then we obtain a tolerant testing algorithm B that works for every pair ϵ_1, ϵ_2 that satisfy $c_2 \cdot \epsilon_1 < c_1 \cdot \epsilon_2$, where the query complexity of B is $Q_A\left(\frac{c_1 \cdot \epsilon_2 - c_2 \cdot \epsilon_1}{2}, n\right)$ and its running time is $T_A\left(\frac{c_1 \cdot \epsilon_2 - c_2 \cdot \epsilon_1}{2}, n\right)$.

What about transforming tolerant testing algorithms to distance approximation algorithms? If we have a tolerant tester for a parameterized property, as in Definition 4, where the function β is not the identity function, then clearly there is no transformation. This is also the case when we have a tolerant testing algorithm that works only for specific fixed values of ϵ_1 and ϵ_2. Thus, tolerant testing may be strictly weaker than distance approximation. However, sufficiently strong tolerant testing algorithms can be transformed into corresponding distance approximation algorithms, where again, the actual functions h_1 and h_2 for which the latter algorithms work, depend on the settings of ϵ_1 and ϵ_2 for which the former algorithms work. For simplicity, below we give one such transformation.

Claim 2 *Let A be a tolerant testing algorithm that works for* every *setting of $\epsilon_1 < \epsilon_2$, whose query and time complexities are $Q_A(\epsilon_2 - \epsilon_1, n)$ and $T_A(\epsilon_2 - \epsilon_1, n)$, respectively. Then we can obtain an (h_1, h_2)-distance approximation algorithm B where $h_1(\epsilon_{\mathcal{P}}(f), n) = \epsilon_{\mathcal{P}}(f) - \delta$ and $h_2(\epsilon_{\mathcal{P}}(f), n) = \epsilon_{\mathcal{P}}(f) + \delta$, whose query complexity is $O(\log(1/\delta)(\log\log(1/\delta)) \cdot Q_A(2\delta, n)$ and whose running time is $O(\log(1/\delta)\log\log(1/\delta)) \cdot T_A(2\delta, n)$.*

Proof: We use algorithm A and a binary-search like procedure in order to find a value $\hat{\epsilon}$ such that $\epsilon_{\mathcal{P}}(f) - \delta \leq \hat{\epsilon} \leq \epsilon_{\mathcal{P}}(f) + \delta$. In each search step we decrease the search interval by a factor of $2/3$. Specifically, we start by running algorithm A $O(\log\log(1/\delta))$ times with $\epsilon_1 = 1/3$ and $\epsilon_2 = 2/3$. If A accepts in a majority of the executions then we continue the search in the interval $[0, 2/3]$, and if it rejects in a majority of the executions then we continue the search in the interval $[1/3, 1]$. Namely, in the former case we continue by running A $O(\log\log(1/\delta))$ times with $\epsilon_1 = 2/9$ and $\epsilon_2 = 4/9$, and in the latter case we do so with $\epsilon_1 = 1/2 + 2/9$ and $\epsilon_2 = 1/2 + 4/9$. We terminate the search when the search interval has size at most 2δ, and let $\hat{\epsilon}$ be the mid-point of the final interval.

Hence, the number of search steps is $O(\log(1/\delta))$. In each step the number of queries performed is at most $O(\log\log(1/\delta)) \cdot Q_A(2\delta, n)$ and the running time of each step is at most $O(\log\log(1/\delta)) \cdot T_A(2\delta, n)$. It is easy to verify that the resulting distance approximation algorithm errs only if in one of the search steps algorithm A gives an incorrect answer in a majority of its executions. Namely, $\epsilon_{\mathcal{P}}(f) \leq \epsilon_1$ but in a majority of the executions A rejects, or $\epsilon_{\mathcal{P}}(f) \geq \epsilon_2$ but in a majority of the executions A accepts. But, since the probability of error in any particular execution is at most $1/3$, by a multiplicative Chernoff bound, for each particular step the probability the majority err is at most $\exp(-\log\log(1/\delta)) = O(1/\log(1/\delta))$. The claim follows by applying a union bound. ■

We note that it is possible to get a slightly better, but more cumbersome, expression for the complexity of algorithm B, by taking into account that the differences between ϵ_1 and ϵ_2 in the different search steps, are different.

3.2 Standard Testing and Tolerant Testing

In some cases a standard tolerant testing algorithm can be easily transformed into a tolerant testing algorithm, as the following claim shows.

Claim 3 *Let A be a standard property testing algorithm for property $\mathcal{P}$, whose queries are uniformly distributed and with query complexity $Q(\epsilon, n)$. If A is a one-sided error algorithm then for any given ϵ, A is an (ϵ_1, ϵ_2)-tolerant testing algorithm for $\epsilon_2 = \epsilon$ and $\epsilon_1 = 1/(3Q(\epsilon, n))$. If A is a two-sided error algorithm then using A we can obtain an algorithm B such that for any given ϵ, B is an*

(ϵ_1, ϵ_2)-tolerant testing algorithm for $\epsilon_2 = \epsilon$ and $\epsilon_1 = 1/(c_1 Q(\epsilon, n))$, where B performs $c_2 Q_A(\epsilon, n)$ queries and c_1 and c_2 are constants.

Proof: Consider first the case that A has a one-sided error. The fact that A rejects with probability at least 2/3 every object that is ϵ_2 $(= \epsilon)$-far from having property $\mathcal{P}$ follows from the fact that A is a standard property testing algorithm. The reason that A accepts with probability at least 2/3 every object that is $\epsilon_1 = 1/3Q(\epsilon, n)$-close to having property $\mathcal{P}$ is simple as well: Consider any object O that is ϵ_1-close to having property $\mathcal{P}$, and let O' be the closest object to O which has property P. Then the probability that A performs a query that falls in the symmetric difference of the two objects is, by the union bound, at most $\epsilon_1 \cdot Q(\epsilon, n) = 1/3$. But if A does not perform any such query then it must accept, since it has a one-sided error.

We now turn to the case in which A has a two-sided error. The stated algorithm B is only a slight variant of algorithm A: it runs A for c_2 times and accepts if A accepts in a majority of the executions. Otherwise, B rejects. If the object O is ϵ_2 $(= \epsilon)$-far from $\mathcal{P}$, then the probability that B accepts (i.e., A accepts in at least $\lceil c_2/2 \rceil$ of its executions) is $3^{-\lceil c_2/2 \rceil} \cdot \binom{c_2}{\lceil c_2/2 \rceil}$, which is at most 1/3 for an appropriate choice of c_2.

Assume now that O is ϵ_1-close to $\mathcal{P}$ and let O' be the closest object to O which has property P. Then the probability that B rejects O is upper bounded by the sum of the probabilities of the following two events. The first event is that all queries of B do not fall in the symmetric difference of O and O'. Since B rejects O it must also reject O' in this case. The probability that this happens is at most $3^{-\lceil c_2/2 \rceil} \cdot \binom{c_2}{\lceil c_2/2 \rceil}$, since A should reject O' with probability at most a 1/3. The second event is that B performs a query that falls in the symmetric difference of the two objects. This occurs with probability of at most $\epsilon_1 \cdot c_2 \cdot Q(\epsilon, n) = \frac{c_2}{c_1}$. For an appropriate choice of c_1 and c_2, the total probability of the two events is a 1/3, as required. ■

Tolerant Testing for Linearity. As an example of an application of the transformation described above, which gives a relatively good ratio between ϵ_1 and ϵ_2, consider linearity testing. In [BLR93], the standard property tester for linear functions makes $O(1/\epsilon)$ uniformly distributed queries. Thus, by the above, we get a tolerant tester for linearity where the ratio between ϵ_1 and ϵ_2 is $O(1)$.

3.3 Agnostic Learning and Distance Approximation

An *Agnostic learning algorithm* [KSS94] for a class of functions H (working under the uniform distribution and which is allowed membership queries) is defined as follows. The algorithm is given an approximation parameter ϵ and a confidence parameter δ. For an unknown function f, the algorithm is given query access to f. The requirement of the algorithm is that with probability at least $1 - \delta$ (over its choice of queries), it outputs a hypothesis $h \in H$ for which the following holds: $\mathrm{dist}(h, f) \leq \min_{g \in H} \mathrm{dist}(f, g) + \epsilon$, where $\mathrm{dist}(h, f)$ is the distance between h and f measured with respect to the uniform distribution. It is easy to see that agnostic learning of H with small query complexity implies distance approximation of the property of belonging to H with small query complexity as well. Specifically, by first running the agnostic learning algorithm and then estimating the distance between f and the hypothesis h that the learning algorithm outputs (using an independent sample), we can estimate the distance of f to the closest function in H. This observation is similar to what was observed in [GGR98] concerning the relation between standard property testing and proper learning.

4 Distance Approximation to Monotonicity

In this section we present a distance approximation algorithm to monotonicity for a function f in one-dimension. That is, f is of the form $f : [n] \to \Re$, where $[n] = \{1, \ldots, n\}$. We later show how to extend the algorithm to higher dimensions (albeit with weaker performance bounds).

Let $\epsilon_{\text{mon}}(f)$ denote the distance of f to the closest monotone function. Our main theorem in this section is the following:

Theorem 1 *There exists a distance approximation algorithm that given query access to a function $f : [n] \to \Re$ and an approximation parameter δ, outputs an estimate $\hat{\epsilon}$, such that with probability at least $2/3$*

$$(1/2)\epsilon_{\text{mon}}(f) - \delta \;\leq\; \hat{\epsilon} \leq \epsilon_{\text{mon}}(f) + \delta \,. \tag{1}$$

The query complexity and running time of the algorithm are $\tilde{O}((\log n)^7/\delta^4)$.

We assume from now on that all values $\{f(1), \ldots, f(n)\}$ are different. If this is not the case then we can regard the values of f as pairs $(f(i), i)$ without changing the distance to monotonicity. The following definition will be useful throughout this section.

Definition 5 (Order of an Element) *For a finite set $S \subset \Re$ and an element $i \in S$, we say that i has* order k in S *if it is the k'th smallest element in S.*

Let $err_{\text{mon}}(f) \stackrel{\text{def}}{=} n \cdot \epsilon_{\text{mon}}(f)$. In the next subsection we define an auxiliary graph in terms of which it is possible to give upper and lower bounds on $err_{\text{mon}}(f)$.

4.1 The Violation Graph

The *violation graph* of a function f was previously defined in [DGL+99]. The vertices of the graph are the indices $\{1, \ldots, n\}$ and there is an edge between every pair i, j such that $i < j$ but $f(i) > f(j)$. The *size* of a matching M in the graph is denoted by $|M|$ and is equal to the number of edges participating in the matching M. The size of a vertex-cover C in the graph is denoted by $|C|$.

Claim 4 *Let M be any matching in the violation graph of f, and let C be any vertex-cover in the graph. Then $|M| \leq err_{\text{mon}}(f) \leq |C|$.*

Proof: Clearly every monotone function differs from f on at least one of the end points of each edge in M. Since M is a matching, $err_{\text{mon}}(f) \geq |M|$.

We turn to the upper bound on $err_{\text{mon}}(f)$. We first observe that since C is a vertex-cover, there are no edges between pairs of vertices $x, y \notin C$. Namely, all points that do not belong to the vertex-cover constitute an independent set, implying that they are a monotonic increasing subsequence. Let us hence define a function g as follows. For every $x \notin C$, let $g(x) = f(x)$. For every point $y \in C$, let $g(y) = f(x)$, where x is the largest point not in C that is smaller than y (If there is no such point then we let $g(y)$ be the minimum value of f.). Clearly g is monotone and $\text{dist}(f, g) = |C|/n$. ■

4.2 The Index-Value Tree

In this subsection we describe the tree that is mentioned in the introduction and which can be used to obtain an upper bound and a lower bound on $err_{\text{mon}}(f)$. We refer to this tree as an *index-value* tree for f, and denote it by T_f. Before defining the tree in detail, we provide a high level idea of its structure and usage.

4.2.1 The idea behind the tree

Given Claim 4 we are interesting in finding good estimates of the sizes of a vertex cover C, and a matching M, in the violation graph of f, such that C is not much larger than M. Recall that an edge in the violation graph of f corresponds to a pair of indices $i < j$ such that $f(i) > f(j)$.

Consider first all edges between pairs i, j such that $i \in \{1, \ldots, n/2\}$ and $j \in \{n/2+1, \ldots, n\}$ (where we assume for simplicity that n is even). For any fixed value x, where the choice of x is discussed momentarily, consider the following two disjoint subsets of indices:

$$B_1^x = \{i \in \{1, \ldots, n/2\} : \ f(i) > x\}, \qquad B_2^x = \{j \in \{n/2+1, \ldots, n\} : \ f(j) \leq x\}.$$

Observe that for every pair i, j such that $i \in \{1, \ldots, n/2\} \setminus B_1^x$ and $j \in \{n/2+1, \ldots, n\} \setminus B_2^x$, we have that $f(i) \leq f(j)$, and so there is no edge between i and j in the violation graph. In other words, the union $B_1^x \cup B_2^x$ covers all edges between pairs i, j such that $i \in \{1, \ldots, n/2\}$ and $j \in \{n/2+1, \ldots, n\}$. Furthermore, we have a complete bipartite subgraph between B_1^x and B_2^x, and hence we have a matching of size $\min\{|B_1^x|, |B_2^x|\}$ in this subgraph. If we take x to be the *median* value among all values of f, then the ratio between the size of the resulting cover and the size of the resulting matching, is minimized. Note that if we are unable to find the exact median value, but rather can find a value x whose order in $\{f(j)\}_{j=1}^n$ is close to the median order, then we get a slightly worse ratio.

Suppose that we put all vertices in $B_1^x \cup B_2^x$ in the vertex cover C, and put a matching of size $\min\{|B_1^x|, |B_2^x|\}$ between B_1^x and B_2^x into the matching M. We can then remove $B_1^x \cup B_2^x$ from the violation graph and continue recursively on both halves of the remaining graph. Specifically, in the next step we consider the subsets $S_1^x = \{1, \ldots, n/2\} \setminus B_1^x$ and $S_2^x = \{n/2+1, \ldots, n\} \setminus B_2^x$ separately. By continuing recursively in this manner we can obtain a vertex cover of all edges of the original violation graph, and a matching that is half the size of the vertex cover.

This recursive procedure can be captured by a tree, where each node in the tree corresponds to a stage in the recursion in which we find a cover and a matching in a bipartite subgraph of the original violation graph. The set of vertices S of each such subgraph is contained in some interval $I = \{i, \ldots, j\}$, and does not include any vertex that was selected already for the vertex cover in some ancestor node in the tree. Ideally we would like to have for each node in the tree both the median index of the set of vertices S that are associated with the node, and the median value of f on these vertices. However, since we are not necessarily able to obtain these two exactly, we allow for a more general definition. Using the more general definition, we can give upper and lower bounds on $err_{\text{mon}}(f)$ in terms of the tree.

4.2.2 Definition of T_f

We associate with each node v in T_f an interval of integers $I(v) \subseteq \{1, \ldots, n\}$, and a subset of indices $S(v) \subseteq I(v)$. With each internal node we also associate two values, denoted $ind(v)$ and $val(v)$. The value of $ind(v)$ may be any *index* in $S(v)$ except the largest index, while $val(v)$ is the *value*

assigned by f to some element in $S(v)$, that is, $val(v) \in \{f(j) : j \in S(v)\}$. As is explained in more detail below, the index $ind(v)$ is used to partition the interval $I(v)$ and consequently the set $S(v)$ into indices that are smaller than $ind(v)$, which may be "passed" to the left child of v, and indices that are greater or equal to $ind(v)$, which may be passed to the right child of v. The value $val(v)$ is used to determine which indices are "good" and in fact are passed to the children, and which indices are "bad" and can serve as evidence to the non-monotonicity of the function f (these bad indices will be part of a vertex cover of the violation graph as was described in Subsection 4.2.1). All concepts that are defined below are illustrated in Figures 1 and 2.

In what follows we use the following shorthand: We say that $ind(v)$ has order k if it has order k in the set $S(v)$ (i.e., it is the k'th smallest index in the set $S(v)$), and we say that $val(v)$ has order k if it has order k in the set $\{f(j) : j \in S(v)\}$.

A Special case of Interest. A special case that will interest us is the case in which $ind(v)$ and $val(v)$ have *the same* order k. Even more specifically, it will be instructive to think of k as being the *median* of $\{1, \ldots, |S(v)|\}$, that is, $k = \lceil |S(v)|/2 \rceil$. Suppose that for every v, $ind(v)$ and $val(v)$ indeed have the same order. It is not hard to verify that if f is monotone, then in such a case, for every node v, $val(v) = f(ind(v))$. However, this is not necessarily true for every function. Consider for example the function $f : [n] \to \Re$ such that $f(i) = i$ for every $i \neq \lceil n/2 \rceil$, and $f(\lceil n/2 \rceil) = n + 1$. Let $S(v) = \{1, \ldots, n\}$, and let $k = \lceil n/2 \rceil$. Then $ind(v) = \lceil n/2 \rceil$ and $val(v) = \lceil n/2 \rceil + 1$, but $f(ind(v)) = n + 1$ which not only differs from $val(v)$ but actually has the highest order in $\{f(j) : j \in S(v)\}$.

The "Bad" Subsets. In order to define $I(v)$ and $S(v)$ in a precise manner for every node in the tree, we must also associate with each internal node v in the tree two (disjoint) subsets $B_{\leq}(v) \subseteq S(v)$ and $B_{>}(v) \subseteq S(v)$, which we refer to as the "bad" subsets of v. These are defined as follows:

$$\begin{aligned} B_{\leq}(v) &\stackrel{\text{def}}{=} \{j \in S(v) : j \leq ind(v) \text{ and } f(j) > val(v)\} \\ B_{>}(v) &\stackrel{\text{def}}{=} \{j \in S(v) : j > ind(v) \text{ and } f(j) \leq val(v)\}. \end{aligned} \tag{2}$$

For example, consider the case that $v = v_0$ is the root, and assume that $ind(v_0) = \lceil n/2 \rceil$ is the median of the set $S(v_0) = \{1, \ldots, n\}$ and $val(v_0)$ is the median value of $\{f(1), \ldots, f(n)\}$. Then $B_{\leq}(v_0)$ includes all indices $1 \leq j \leq \lceil n/2 \rceil$ such that $f(j)$ is larger than the median value, and $B_{>}(v_0)$ includes all indices $\lceil n/2 \rceil < j \leq n$ such that $f(j)$ is at most the median value. These subsets are "bad" since they violate the conditions of monotonicity, and for every such pair $i \in B_{\leq}(v_0)$ and $j \in B_{>}(v_0)$ there is an edge in the violation graph of f. It is not hard to verify that if f is monotone, then for every node v such that $ind(v)$ and $val(v)$ have the same order, both $B_{\leq}(v)$ and $B_{>}(v)$ must be empty, while if $ind(v)$ and $val(v)$ do not have the same order, then at least one of the two subsets is empty.

Defining $I(v)$ and $S(v)$. We now describe precisely how $I(v)$ and $S(v)$ are defined for every node v in the tree. We denote the root of the tree by v_0, and associate with it the interval of integers $I(v_0) = \{1, \ldots, n\}$, and the complete set of indices $S(v_0) = \{1, \ldots, n\}$. In general, the interval associated with node v will be denoted by $I(v) = \{\ell ind(v), \ldots, rind(v)\}$, where $\ell ind(v)$ and $rind(v)$ denote the *left-most* index and *right-most* index, respectively, of v's interval. It remains to explain how $I(v)$ and $S(v)$ are defined for a given v, other than the root v_0. Let p be the parent of v in the tree. Then it will hold that $I(v) \subseteq I(p)$ and $S(v) \subseteq S(p)$. Furthermore $I(v)$ and $S(v)$ are determined by $I(p)$ and $S(p)$ together with $ind(p)$ and $val(p)$ as follows:

- The interval $I(v) = \{\ell ind(v), \ldots, rind(v)\}$ is determined as follows: If v is the left child of p then $\ell ind(v) = \ell ind(p)$ and $rind(v) = ind(p)$, while if v is the right child of p then $\ell ind(v) = ind(p) + 1$, and $rind(v) = rind(p)$.

  For example, if $p = v_0$ is the root and v is the left child of v_0 then $I(v) = \{1, \ldots, ind(v_0)\}$, while if v is the right child of v_0 then $I(v) = \{ind(v_0) + 1, \ldots, n\}$.

- The set $S(v) \subseteq I(v)$ is determined as follows: If v is the left child of its parent p then $S(v) = (I(v) \cap S(p)) \setminus B_{\leq}(p)$, and if v is the right child of p then $S(v) = (I(v) \cap S(p)) \setminus B_{>}(p)$. Recalling the definitions of the bad subsets, this means that $S(v)$ includes all indices in $S(p)$ that belong to the interval $I(v)$ *except* those on which the value of f is "incorrect" (larger than $val(v)$ when it should be smaller or equal, or vice versa).

Each leaf v in the tree either corresponds to a set $S(v)$ that contains a single element, or it is the case that $S(v)$ is empty. If v is the left child of its parent p then the latter occurs when $B_{\leq}(p) = S(p) \cap I(v)$, and if v is the right child of p then this occurs when $B_{>}(p) = S(p) \cap I(v)$. For every leaf v we let $B_{\leq}(v) = B_{>}(v) = \emptyset$.

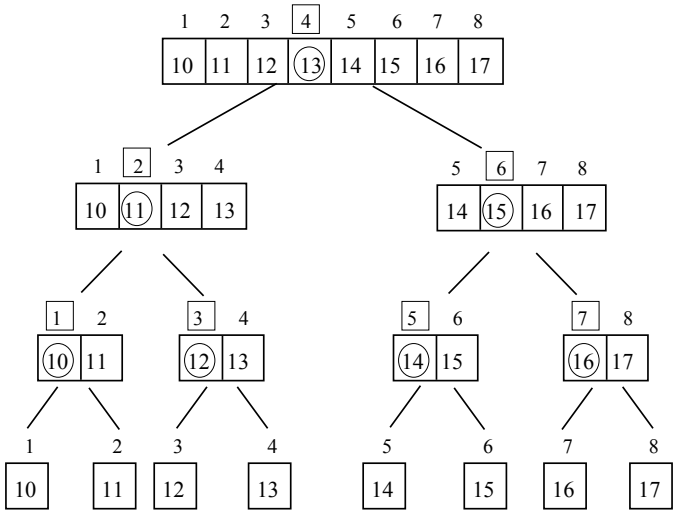

Figure 1: An illustration of an index-value tree of the monotone function f which is defined as $f(i) = i + 10$ for every $1 \leq i \leq 8$. Each node v in the tree contains an array which represents the interval $I(v)$ corresponding to v, and the values of f in this interval. The elements which correspond to indices in $S(v)$ are denoted by white cells. Since the function is monotone there are no bad indices here, and thus $S(v) = I(v)$ in this case for every v. The index $ind(v)$ is marked by a square around it, and in this example it is simply the median index of $S(v)$. The element corresponding to $val(v)$ is denoted by a circle around it, and here it is the median value of f on indices in $S(v)$.

Remarks. The following remarks will be helpful in understanding the usage of index-value trees.

1. By definition of the intervals it holds that in every level of the tree the intervals associated with the nodes in that level are disjoint. Furthermore, for every level h such that there are no leaves in levels $h' < h$ we have that the union of all intervals $I(v)$ corresponding to nodes in level h is $\{1, \ldots, n\}$.

2. If u and v are the two children of p in the tree then $S(v) \cup S(u) = S(p) \setminus (B_{\leq}(p) \cup B_{>}(p))$.

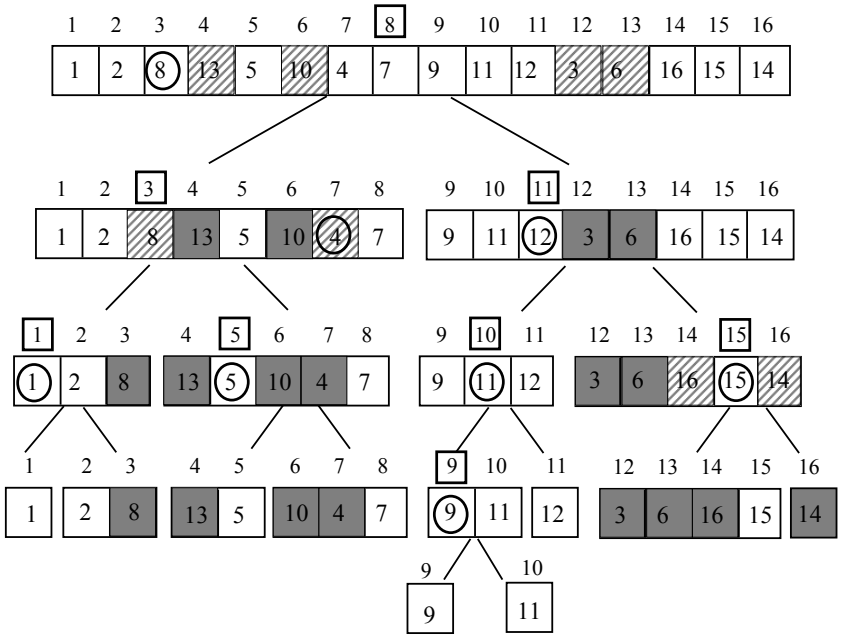

Figure 2: An illustration of an index-value tree of a non monotone function f which is described in the array in the root of the tree. Again, all relevant data of each node v is described as in Figure 1. Since this function is not monotone there are bad indices here. For each node v, the elements which correspond to indices that belong to $B_{\leq}(v)$ and $B_{>}(v)$ are denoted by striped cells, and the elements corresponding to indices in the bad subsets of the *ancestors* of v are denoted by gray cells. Also here $ind(v)$ was selected as the median index of $S(v)$ and $val(v)$ is the median value of f on indices in $S(v)$. Note for example that $B_{<}(v_0) = \{4, 6\}$ and $B_{>}(v_0) = \{12, 13\}$ where v_0 is the root of the tree.

3. All bad subsets associated with *all* nodes in the tree are disjoint.

As a further demonstration of these definitions, suppose that f is monotone. In such a case, if for every v, $ind(v)$ and $val(v)$ have the same order, then for every node v, the bad subsets $B_{\leq}(v)$ and $B_{>}(v)$ are empty. Note that if f is monotone but for some node v, $ind(v)$ and $val(v)$ do not have the same order, then at least one of the two subsets $B_{\leq}(v)$ and $B_{>}(v)$ must be empty (which subset is empty depends on whether the order of $ind(v)$ is smaller than the order of $val(v)$ or vice versa).

On the other hand, if f is not monotone, then it is possible that the subsets $B_{\leq}(v)$ and $B_{>}(v)$ are both non-empty (even if $ind(v)$ and $val(v)$ have the same order). However, as we shall see in the next subsection, if we consider all indices that do not belong to any bad subset (i.e., are located in one of the leaves), then the function f is monotonically increasing on them.

4.2.3 The Index-Value Tree and Monotonicity

In this subsection we show that given an index-value tree T_f, we can use the sizes of all bad subsets in the tree to get upper and lower bounds on the distance of f from monotone (using Claim 4). For any node v in T_f, let $b_{\leq}(v) \stackrel{\text{def}}{=} |B_{\leq}(v)|$, and let $b_{>}(v) \stackrel{\text{def}}{=} |B_{>}(v)|$ denote the sizes of the bad

subsets that are associated with v. Let L_h denote the set of nodes in level h of the tree T_f, and let

$$B \stackrel{\text{def}}{=} \bigcup_{v \in T_f} (B_{\leq}(v) \cup B_{>}(v)) \ .$$

Lemma 5 *Let T_f be any index-value tree for f. Then the values of f on all indices that do not belong to B form a monotonically increasing sequence, and*

$$\sum_{v \in T_f} \min\{b_{\leq}(v), b_{>}(v)\} \ \leq \ err_{\text{mon}}(f) \ \leq \sum_{v \in T_f} (b_{\leq}(v) + b_{>}(v)) \ .$$

Proof: We first show that B is a vertex-cover of the violation graph. Namely, we show that for every pair $i, j \notin B$, such that $i < j$, we also have $f(i) < f(j)$ (that is, there is no edge between i and j in the violation graph). Consider any such fixed pair $i < j$, $i, j \notin B$. By definition of an index-value tree, since both i and j do not belong to B, there exist leaf nodes u and w such that $S(u) = \{i\}$ and $S(w) = \{j\}$. Let v be their least common ancestor. Then, once again by definition of the tree, we have $i, j \in S(v)$ and $i \leq ind(v) < j$. By definition of the set B, if $i \leq ind(v) < j$ and $i, j \notin B$ then $f(i) \leq val(v) < f(j)$, and so $f(i) < f(j)$, as required. The first part of Lemma 5 concerning the values of f on indices $i \notin B$ directly follows, and the upper bound on $err_{\text{mon}}(f)$ follows from Claim 4.

To establish the lower bound on $err_{\text{mon}}(f)$, consider any node v, and the bad subsets $B_{\leq}(v)$ and $B_{>}(v)$ associated with it. By definition of these sets, for every $i \in B_{\leq}(v)$ and $j \in B_{>}(v)$, $i < j$, while $f(j) \leq val(v) < f(i)$. Thus there is an edge between i and j in the violation graph. It directly follows that there exist a matching of size $\min\{b_{\leq}(v), b_{>}(v)\}$ between the two subsets. Furthermore the bad subsets (of all nodes in the tree), are disjoint. Therefore by taking the union over all nodes v of these matchings between pairs of bad subsets, we obtain a matching in the violation graph, which has size $\sum_{v \in T_f} \min\{b_{\leq}(v), b_{>}(v)\}$. The lower bound follows by applying Claim 4. ■

We next consider index-value trees that are approximately balanced (as defined below), and use Lemma 5 to obtain an upper bound on $err_{\text{mon}}(f)$ in terms of $\sum_{v \in T_f} \min\{b_{\leq}(v), b_{>}(v)\}$.

Definition 6 (γ-Balanced Trees) *We say that an index-value tree T_f is* γ-balanced, *if for every node v in T_f we have that the orders of $ind(v)$ and $val(v)$ are each at least $\lceil |S(v)|/2 \rceil - \min\{\gamma |I(v)|, \frac{1}{8}|S(v)|\}$ and at most $\lceil |S(v)|/2 \rceil + \min\{\gamma |I(v)|, \frac{1}{8}|S(v)|\}$.*

Lemma 6 *Let T_f be an index-value tree for f that is γ-balanced for some $\gamma = \gamma(\delta, n)$. Then there are at most $2 \log n$ levels in T_f and*

$$err_{\text{mon}}(f) \ \leq 2 \sum_{v \in T_f} \min\{b_{\leq}(v), b_{>}(v)\} + 2\gamma \cdot \log n \cdot n \ .$$

In order to prove Lemma 6, we shall need the following claim.

Claim 7 *Let v be a node in an index-value tree T_f. If the orders of $ind(v)$ and $val(v)$ differ by at most r, then $|b_{\leq}(v) - b_{>}(v)| \leq r$.*

Proof: Let $S(v) = \{i_1, \ldots, i_k\}$, let the the order of $val(v)$ be s, and let the order of $ind(v)$ be t. Thus, $ind(v) = i_t$ and $val(v)$ is the s'th smallest value in $\{f(i_1), \ldots, f(i_k)\}$.

We look at two partitions of the set $S(v)$. One is the partition $\{S_1, S_2\}$ such that S_1 contains all indices $j \in S(v)$ for which $f(j) \leq val(v)$, and the set S_2 contains all indices $j \in S(v)$ for which $f(j) > val(v)$. Thus $|S_1| = s$. The second partition $\{T_1, T_2\}$ of $S(v)$ is such that T_1 contains all indices $j \leq ind(v)$ in $S(v)$ and T_2 contains all indices $j > ind(v)$ in $S(v)$. Thus $|T_1| = t$.

Since

$$|S_1 \cap T_1| + |S_1 \cap T_2| = |S_1| = s$$

while

$$|T_1 \cap S_1| + |T_1 \cap S_2| = |T_1| = t \ ,$$

we have that

$$| \ |S_1 \cap T_2| - |S_2 \cap T_1| \ | = |s - t| \ .$$

But $S_1 \cap T_2 = B_>(v)$ whereas $S_2 \cap T_1 = B_\leq(v)$ and $|s - t| \leq r$. The claim follows. ■

Proof of Lemma 6: The claim concerning the number of levels in T_f follows directly from the definition of an approximately balanced tree (since for every node v, $|S(v)| \leq (5/8)|S(p)|$ where p is the parent of v). We hence turn to the upper bound on $err_{\mathrm{mon}}(f)$. We first observe that by the premise of Lemma 6 and by Claim 7, for every node v we have that

$$(b_\leq(v) + b_>(v)) \leq 2 \min\{b_\leq(v), b_>(v)\} + \gamma \cdot |I(v)| \ .$$

Therefore,

$$\sum_{v \in T_f} (b_\leq(v) + b_>(v)) \leq 2 \sum_{v \in T_f} \min\{b_\leq(v), b_>(v)\} + \sum_{v \in T_f} \gamma \cdot |I(v)| \ .$$

Using the upper bound in Lemma 5, it remains to show that

$$\sum_{v \in T_f} \gamma \cdot |I(v)| \leq 2\gamma n \log n$$

But this follows directly from the fact that for every level h, $\sum_{v \in L_h} |I(v)| \leq n$, and the upper bound of $2 \log n$ on the number of levels in T_f. ■

4.3 The Distance Approximation Algorithm

Notice that if we can construct a γ-balanced index-value tree for $\gamma = O(\frac{\delta}{\log n})$, then using Lemma 6 and Lemma 5, we get the following lower and upper bounds on $err_{\mathrm{mon}}(f)$:

$$\sum_{v \in T_f} \min\{b_\leq(v), b_>(v)\} \ \leq err_{\mathrm{mon}}(f) \ \leq 2 \sum_{v \in T_f} \min\{b_\leq(v), b_>(v)\} + 2\delta \cdot n \ .$$

In other words, setting $\hat{\epsilon} = \frac{1}{n} \sum_{v \in T_f} \min\{b_\leq(v), b_>(v)\}$ we get an approximation $\hat{\epsilon}$ to monotonicity that satisfies: $\frac{1}{2}\epsilon_{\mathrm{mon}}(f) - \delta \leq \hat{\epsilon} \leq \epsilon_{\mathrm{mon}}(f)$.

This immediately suggests a distance approximation algorithm for monotonicity. However this algorithm is not sublinear in n since it requires us to fully construct the tree, and for each node in the tree to find the exact values of $b_\leq(v)$ and $b_>(v)$. In the next two subsections we show how to overcome these problems. We do so in two stages.

Stage 1: A Mental Experiment. We first assume, as a mental experiment, that our algorithm has *approximate* query access to a γ-balanced index-value tree for f (where $\gamma = O(\frac{\delta}{\log n})$). Namely, the algorithm can choose to query any node v in the tree, and by querying v it obtains estimates $\hat{b}_{\leq}(v)$ and $\hat{b}_{>}(v)$ of $b_{\leq}(v)$ and $b_{>}(v)$, respectively. We show in Subsection 4.3.1 that under certain conditions on these estimates at the nodes, it is possible to obtain a good estimate of $err_{\mathrm{mon}}(f)$ by performing only $\tilde{O}(\log^3 n/\delta^2)$ queries into the tree.

Stage 2: The Actual Algorithm. The main insight gleaned from the above mental experiment is that while constructing an index-value tree (or even an approximate one) takes linear time, it suffices to access only $\mathrm{poly}(\log n/\delta)$ nodes in the tree. Hence, our algorithm, which is described in detail in Subsection 4.3.1, will construct only parts of such a tree according to its "needs" (i.e., the approximate queries it would like to perform). Furthermore, for each node or path that it constructs in the partial tree, it will perform only $\mathrm{poly}(\log n/\delta)$ queries to the function f. Thus our algorithm can be viewed as emulating the mental experiment algorithm with small overhead in terms of the number of queries to the function that are performed per query to the tree. As a result we get a distance approximation algorithm whose query complexity is $\mathrm{poly}(\log n/\delta)$.

4.3.1 A Mental Experiment: Performing Approximate Queries

In this and the next subsection we complete the description of the algorithm for distance approximation to monotonicity. As discussed briefly earlier, we start by describing, as a mental experiment, an algorithm that has query access to an index-value tree with certain properties. Details follows.

Let T_f be a γ-balanced index-value tree for $\gamma = O(\frac{\delta}{\log n})$, where δ is the given approximation parameter, as in Equation (1). Suppose that it was possible to perform "approximate queries" of the following form. For any given node v in the tree, we could obtain non-negative estimates $\hat{b}_{\leq}(v)$ and $\hat{b}_{>}(v)$ in the range $\{0, \ldots, |I(v)|\}$ which are good estimates of $b_{\leq}(v)$ and $b_{>}(v)$, respectively. Since eventually we will obtain these estimates by sampling, a slight complication arises for nodes v for which the size of $S(v)$ is too small (with respect to $I(v)$). For such nodes, obtaining a good estimate to the size of their bad subsets may not be possible using a sample that is not sufficiently large. For reasons that will become more clear in the next section, we let "smallness" be a hereditary property. Namely:

Definition 7 (γ-Small Nodes) *A node v in an index-value tree is called* γ-small *(or simply* small*) if for some ancestor w of v, $|S(w)| \leq \gamma |I(w)|$ (where v is considered an ancestor of itself).*

We can now define good estimates as follows:

Definition 8 (γ-Estimate) *We say that $\hat{b}_{\leq}(v)$ and $\hat{b}_{>}(v)$ are a* γ-estimate *for $b_{\leq}(v)$ and $b_{>}(v)$ if either (i) $|\hat{b}_{\leq}(v) - b_{\leq}(v)| \leq \gamma \cdot |I(v)|$ and $|\hat{b}_{>}(v) - b_{>}(v)| \leq \gamma \cdot |I(v)|$; or (ii) v is γ-small and $\hat{b}_{\leq}(v) = \hat{b}_{>}(v) = 0$.*

We next show that by performing $\tilde{O}(\log^3 n/\delta^2)$ queries to a γ-balanced index-value tree that contains in each node such γ-estimates, we can obtain, with high probability, an estimate $\hat{\epsilon}$ to $\epsilon_{\mathrm{mon}}(f)$ that satisfies Equation (1).

Recall that L_h denotes the set of nodes in level h of the tree T_f. Let $m = \frac{c}{\gamma^2} \cdot \log\log n$ for some sufficiently large constant c that will be specified below. The algorithm will sample (with repetitions) m nodes from each level, and will use the $\hat{b}$ estimates for the selected nodes.

Algorithm 1 (Approximation of distance to monotone: Mental Experiment)

1. For levels $h = 0$ to $2\log n$ do:
   - Initialize $\hat{b}_h = 0$.
   - Repeat the following m times: Select node $v \in L_h$ with probability $|I(v)|/n$. If $\sum_{v \in L_h} |I(v)| < n$ then with probability $1 - (\sum_{v \in L_h} |I(v)|)/n$ no node is selected. If a node v is selected then update $\hat{b}_h = \hat{b}_h + \frac{\min\{\hat{b}_<(v), \hat{b}_>(v)\}}{|I(v)|}$.
   - Let $\hat{\epsilon}_h = \frac{1}{m} b_h$.
2. Set $\hat{\epsilon} = \sum_h \hat{\epsilon}_h$.

Observe that the number of queries made into the tree is $O(m \log n) = \tilde{O}(\log^3 n/\delta^2)$. We note that in levels $h \leq \log m$ we could actually consider all (at most m) nodes in L_h, without need for sampling. However, for simplicity we perform the same sampling process in all levels. We can now prove the following lemma about the quality of the estimate $\hat{\epsilon}$ that is output by Algorithm 1.

Lemma 8 *Let T_f be an index-value tree that is γ-balanced and assume that for every v in T_f the estimates $\hat{b}_\leq(v)$ and $\hat{b}_>(v)$ are γ-estimates of $b_\leq(v)$ and $b_>(v)$, respectively, where $\gamma \leq \delta/(5 \log n)$. Then with probability at least $9/10$, the output $\hat{\epsilon}$ of Algorithm 1 satisfies:*

$$(1/2)\epsilon_{\mathrm{mon}}(f) - \delta \;\leq\; \hat{\epsilon} \leq \epsilon_{\mathrm{mon}}(f) + \delta.$$

Proof: We first observe that by Lemma 6 the number of levels in T_f is indeed at most $2\log n$ so that the algorithm considers all levels in the tree. For each level h, let

$$\epsilon_h \stackrel{\mathrm{def}}{=} \frac{1}{n} \sum_{v \in L_h} \min\{b_\leq(v), b_>(v)\}\ . \tag{3}$$

By Lemmas 5 and 6,

$$(1/2)\epsilon_{\mathrm{mon}}(f) - \gamma \cdot \log n \leq \sum_h \epsilon_h \leq \epsilon_{\mathrm{mon}}(f) \tag{4}$$

We would like to show that with high probability over the choice of the samples selected by the algorithm, for all levels h, $\hat{\epsilon}_h$ is a good estimate to ϵ_h.

For a fixed level h, let $\chi_i, \ldots, \chi_m$ be independent random variables, where each χ_i attains with probability $|I(v)|/n$ the value $\frac{\min\{\hat{b}_<(v), \hat{b}_>(v)\}}{|I(v)|}$, for $v \in L_h$, and with probability $1 - \frac{1}{n}\sum_{v \in L_h} |I(v)|$, $\chi_i = 0$. The expected value of each χ_i is hence:

$$\mathrm{Exp}[\chi_i] = \sum_{v \in L_h} \frac{|I(v)|}{n} \cdot \frac{\min\{\hat{b}_\leq(v), \hat{b}_>(v)\}}{|I(v)|} \;= \frac{1}{n} \cdot \sum_{v \in L_h} \min\{\hat{b}_\leq(v), \hat{b}_>(v)\}. \tag{5}$$

Let us denote this expected value by μ_h. We first show that by our assumption on $\hat{b}_\leq(v)$ and $\hat{b}_>(v)$, we have that

$$\epsilon_h - \gamma \leq \mu_h \leq \epsilon_h + \gamma. \tag{6}$$

To establish Equation (6) we partition the nodes in level h into two disjoint subsets. The first subset, V_h^1, consists of those nodes v in level h for which $|\hat{b}_\leq(v) - b_\leq(v)| \leq \gamma \cdot |I(v)|$ and $|\hat{b}_>(v) - b_>(v)| \leq \gamma \cdot |I(v)|$. The second subset, V_h^2, consists of those nodes v that do not belong

to V_h^1, are γ-small, and satisfy $\hat{b}_{\leq}(v) = \hat{b}_{>}(v) = 0$. Since for every v in T_f the estimates $\hat{b}_{\leq}(v)$ and $\hat{b}_{>}(v)$ are assumed to be γ-estimates of $b_{\leq}(v)$ and $b_{>}(v)$ respectively (recall Definition 8), we have that $V_h^1 \cup V_h^2 = L_h$. By definition of V_h^1,

$$\left| \frac{1}{n} \sum_{v \in V_h^1} \min\{\hat{b}_{\leq}(v), \hat{b}_{>}(v)\} - \frac{1}{n} \sum_{v \in V_h^1} \min\{b_{\leq}(v), b_{>}(v)\} \right| \leq \frac{1}{n} \cdot \sum_{v \in V_h^1} \gamma |I(v)| \; . \tag{7}$$

Turning to the nodes in V_h^2, we have that for every $v \in V_h^2$, $\hat{b}_{\leq}(v) = \hat{b}_{>}(v) = 0$ so that

$$\sum_{v \in V_h^2} \min\{\hat{b}_{\leq}(v), \hat{b}_{>}(v)\} = 0 \; .$$

On the other hand, for every v, $B_{\leq}(v) \cup B_{>}(v) \subseteq S(v)$, and thus:

$$\sum_{v \in V_h^2} \min\{b_{\leq}(v), b_{>}(v)\} \leq \sum_{v \in V_h^2} |S(v)| \; .$$

Since all vertices in V_h^2 are γ-small, we can deduce that

$$\sum_{v \in V_h^2} |S(v)| \leq \gamma \cdot \sum_{v \in V_h^2} |I(v)|$$

and so

$$\left| \frac{1}{n} \sum_{v \in V_h^2} \min\{\hat{b}_{\leq}(v), \hat{b}_{>}(v)\} - \frac{1}{n} \sum_{v \in V_h^2} \min\{b_{\leq}(v), b_{>}(v)\} \right| \leq \frac{1}{n} \cdot \gamma \sum_{v \in V_h^1} |I(v)| \; . \tag{8}$$

Equation (6) follows by combining Equations (7) and (8) with the fact that $\sum_{v \in L_h} |I(v)| \leq n$.

We next apply an additive Chernoff bound in order to bound the deviation of $\hat{\epsilon}_h = \frac{1}{m} \cdot \sum_i \chi_i$ from the expected value μ_h. Since the χ_i's are independent random variables that attain values between 0 and 1 and whose expected value is μ_h,

$$\Pr\left[\left| \frac{1}{m} \cdot \sum_i \chi_i - \mu_h \right| > \gamma \right] < 2\exp(-2m\gamma^2) \leq \frac{1}{20 \log n} \tag{9}$$

where the last inequality holds for $m \geq \frac{2}{\gamma^2} \log\log n$. By applying a union bound over all levels h in which we sample, and using Equation (6), we get that with probability at least 9/10, for every such level h,

$$\epsilon_h - 2\gamma \leq \hat{\epsilon}_h \leq \epsilon_h + 2\gamma \; . \tag{10}$$

The lemma follows by combining Equations (4), (6) and (10) with the bound on the number of levels of the tree and the upper bound on γ. ■

A Remark concerning monotone functions. Note that if f is a monotone function then for every node v either $b_{\leq}(v) = 0$ or $b_{>}(v) = 0$, so that $\min\{b_{\leq}(v), b_{>}(v)\} = 0$. Suppose that we slightly strengthen the definition of a γ-estimate and require that $\hat{b}_{\leq}(v) = 0$ whenever $b_{\leq}(v) = 0$, and that $\hat{b}_{>}(v) = 0$ whenever $b_{>}(v) = 0$. It directly follows that if f is monotone then the output, $\hat{\epsilon}$, of the mental experiment algorithm is 0.

4.3.2 The Sublinear Algorithm

Observe that while the size of a fully constructed index-value tree for f is linear in n, the above mental-experiment algorithm required to access only $\text{poly}(\log n/\delta)$ nodes in the tree. Hence, instead of fully constructing the index-value tree, and then accessing a small fraction of it, as done above, our algorithm will construct only parts of such a tree according to its "needs" (i.e., the approximate queries it would like to perform). Furthermore, for each node or path that it constructs in the partial tree, it will perform only $\text{poly}(\log n/\delta)$ queries to the function f. Thus we get a distance approximation algorithm whose query complexity is $\text{poly}(\log n/\delta)$.

The Partial Tree. We associate, as before, with every node v in the partial tree (with the exception of nodes that are *small*, in a sense closely related to that of Definition 7) an interval $I(v)$, a subset $S(v) \subseteq I(v)$, subsets $B_{\leq}(v), B_{>}(v) \subseteq S(v)$, an index $ind(v) \in S(v)$, and a value $val(v) \in \{f(j) : j \in S(v)\}$. We shall ensure that with high probability $ind(v)$ and $val(v)$ do not deviate by much from the median index in $S(v)$ and the median value of f in $S(v)$, respectively. We also assign to each node v two additional values: $\ell val(v)$ and $rval(v)$ which constitute a lower bound and an upper bound, respectively, on the value of f on indices in $S(v)$. These values can be determined from existing values associated with ancestor nodes of v, but it will be convenient to explicitly associate them with v. (In fact, this claim about redundancy is true of all information associated with the nodes with the exception of $ind(v)$ and $val(v)$, which determine all other information.)

Specifically, for the root v_0 we set $\ell val(v_0) = -\infty$ and $rval(v_0) = \infty$. For any other node v in the tree, let p denote the parent of v. If v is the left child of p then $\ell val(v) = \ell val(p)$ and $rval(v) = val(p)$, and if v is the right child of p then $\ell val(v) = val(p)$ and $rval(v) = rval(p)$. It is not hard to verify that for every v,

$$S(v) = \{j \in I(v) : \ell val(v) < f(j) \leq rval(v)\} .$$

Sampling into the Tree. One issue that should be addressed when the partial tree is constructed is how to sample nodes in each level according to $|I(v)|/n$. This issue is easily handled as follows.

In order to select a node v in level h of the tree with probability $|I(v)|/n$, we first uniformly select a point $x \in \{1, \ldots, n\}$. We then go down the tree, starting from the root, and find the node v in level h for which $x \in I(v)$ (if such a node v exists), constructing nodes that were not yet constructed as we go down the tree. Clearly each node v in level h is selected with the appropriate probability. The case in which there is no node v in level h such that $x \in I(v)$ because a leaf was reached before getting to level h, corresponds to the case in the mental experiment algorithm where a node is not selected in some trial.

In "constructing a node" u we mean computing $ind(u)$ and $val(u)$, which with high probability will have orders close to $\lceil |S(u)|/2 \rceil$. These index and value are used for determining the rest of the path to the node v in level h such that $x \in I(v)$. If along this path we reach a node u for which $|I(u)|$ is smaller than some threshold $(4/\gamma)$, then we stop sampling and actually construct all the subtree rooted at u by viewing all values $f(j)$ of points $j \in I(v)$ (and using exact median indices and values).

Finally, once we reach v we use $ind(v)$ and $val(v)$ (as well as $\ell ind(v)$ and $rind(v)$) to compute estimates $\hat{b}_{\leq}(v)$ and $\hat{b}_{>}(v)$ of $b_{\leq}(v)$ and $b_{>}(v)$, respectively.

Empirically Small Nodes. As mentioned briefly in the previous subsection, a second technical issue that should be addressed is dealing with nodes v for which $S(v)$ is very small compared to $I(v)$. For such nodes it is not possible to obtain with a sample that is not sufficiently large an index $ind(v)$ and a value $val(v)$, whose order is close to $\lceil |S(v)|/2 \rceil$. To clarify this point, observe that given $\ell ind(v)$ and $rind(v)$ we can easily obtain uniform samples from $I(v)$ since $I(v) = \{\ell ind(v), \ldots, rind(v)\}$. However, in order to obtain uniform samples from $S(v)$, so as to set $ind(v)$ and $val(v)$, we need to first sample from $I(v)$ and then, for each point j selected, check whether it belongs to $S(v)$ by checking whether $\ell val(v) < f(j) \leq rval(v)$. Since we require that the orders of $ind(v)$ and $val(v)$ deviate by at most $\min\{\frac{1}{8}|S(v)|, \gamma|I(v)|\}$ from $\lceil |S(v)|/2 \rceil$, the size of the sample required grows with $|I(v)|/|S(v)|$.

To deal with this issue we first compute an estimate $\hat{s}(u)$ of $|S(u)|$ for each new node u constructed. If this estimate is below a certain threshold, where the threshold is set at $(3\gamma/4)|I(u)|$, then we consider the node u and all its descendents as *empirically small* nodes. For each empirically small node u we do not compute $ind(u)$ and $val(u)$ (and these values remain undefined), and we simply set $\hat{b}_{\leq}(u) = \hat{b}_{>}(u) = 0$. Since we do not compute $ind(u)$ and $val(u)$, we cannot continue and construct the subtree rooted at u. However, since every possible descendent v of u is empirically small and we always set $\hat{b}_{\leq}(v) = \hat{b}_{>}(v) = 0$, we do not actually need to construct this subtree because we can use these 0 estimates. The notion of an empirically small node is clearly closely related to the notion of a small node in Definition 7. Specifically, with high probability a node that is $(\gamma/2)$-small as in Definition 7, will be empirically small, and a node that is not γ-small will not be empirically small.

Procedures. The approximation algorithm will use two procedures: Procedure 1 is used to approximate the size of the subsets $S(u)$, $B_{\leq}(u)$, and $B_{>}(u)$ for the nodes u we construct, and Procedure 2 is used to obtain $ind(u)$ and $val(u)$. These simple sampling procedures are described following the algorithm. Since Procedure 1 can be used, given a node u, to estimate both the size of $S(u)$ and the sizes of the bad subsets $B_{\leq}(u)$ and $B_{>}(u)$, it receives also as parameters two indices ind_1 and ind_2 and two values val_1 and val_2. The indices determine the interval of indices in which to sample, and the values determine the range of values allowed in this interval according to the type of subset whose size we are trying to estimate.

We assume that if for some node u in the tree, the size of $S(u)$ has already been estimated, then this size estimate, denoted $\hat{s}(u)$, is maintained at the node for future use. The algorithm and the accompanying procedures are given below.

Let $\gamma = \frac{\delta}{5 \log n}$ and let $m = \frac{c}{\gamma^2} \cdot \log\log n$ be as in the previous subsection. Also define $s = \frac{c'}{\gamma^2} \cdot \log(n)$ for an appropriate constant c'.

Algorithm 2 (Approximation of distance to monotone)

- For level $h = 0$:
  1. Construct the root node v_0 of the tree: Let $I(v_0) = \{1, \ldots, n\}$, $ind(v_0) = \lceil n/2 \rceil$, $\ell ind(v_0) = 1$, $rind(v_0) = n$, $\ell val(v_0) = -\infty$, $rval(v_0) = \infty$, and let $\hat{s}(v_0) = n$. Set $val(v_0)$ by calling Procedure 2 for v_0 (and ignoring the index it returns for $ind(v_0)$).
  2. Estimate $\hat{b}_{\leq}(v_0)$ by calling Procedure 1 with parameters: $ind_1 = 1$, $ind_2 = \lceil n/2 \rceil$, $val_1 = val(v_0)$ and $val_2 = \infty$. Estimate $\hat{b}_{>}(v_0)$ by calling Procedure 1 with parameters: $ind_1 = \lceil n/2 \rceil + 1$, $ind_2 = n$, $val_1 = -\infty$ and $val_2 = val(v_0)$.
  3. Set $\hat{\epsilon}_h = \frac{1}{n} \cdot \min\{\hat{b}_{\leq}(v_0), \hat{b}_{>}(v_0)\}$.
- For levels $h = 1$ to $2 \log n$ do:
  1. Initialize $\hat{b}_h = 0$.
  2. Uniformly and independently select m points $x_h^1, \ldots, x_h^m$ from $\{1, \ldots, n\}$. Denote the multi-set of points selected by X_h. (These points are used to select nodes in level h.)
  3. For each point $x_h^j \in X_h$ do:
     (a) Find the leaf w in the current partial tree such that $x_h^j \in I(w)$. Set $z \leftarrow w$.
     (b) While $h(z) < z$, z is not an empirically small node and $|I(z)| > s$:
        i. If $x_h^j \leq ind(z)$ then add a left child to z, and if $x_h^j > ind(z)$ then add a right child. Let u denote this new child. Set $\ell ind(u)$, $rind(u)$, $\ell val(u)$ and $rval(u)$ based on $\ell ind(z)$, $rind(z)$, $\ell val(z)$ $rval(z)$ and $val(z)$, as described earlier in this section.
        ii. Compute $\hat{s}(u)$ by calling Procedure 1 with parameters: $ind_1 = \ell ind(u), ind_2 = rind(u), val_1 = \ell val(u)$ and $val_2 = rval(u)$. If $\hat{s}(u) \geq (3\gamma/4) \cdot |I(u)|$, then set $ind(u)$ and $val(u)$ by calling Procedure 2 for u. Else u is an empirically small node.
        iii. Set $z \leftarrow u$.
     (c) If $|I(z)| \leq s$ then construct all nodes in the subtree rooted at z by looking at all values $\{f(j) : j \in I(z)\}$ and for each node v in this subtree selecting $ind(v)$ to be the median index in $S(v)$ and $val(v)$ to be the median value. If there exists a node v in this subtree belonging to level h such that $x_h^j \in I(v)$ then update $\hat{b}_h = \hat{b}_h + \frac{\min\{b_<(v), b_>(v)\}}{|I(v)|}$. (Note that $b_{\leq}(v)$ and $b_{>}(v)$ can be calculated exactly in this case.)
     (d) Otherwise, if $h(z) = h$ and z is not empirically small then do the following: Estimate $\hat{b}_{\leq}(z)$ by calling Procedure 1 with parameters: $ind_1 = \ell ind(z), ind_2 = ind(z), val_1 = val(z)$ and $val_2 = \infty$; Estimate $\hat{b}_{>}(z)$ by calling Procedure 1 with parameters: $ind_1 = ind(z)+1, ind_2 = rind(z), val_1 = -\infty$ and $val_2 = val(z)$; Update $\hat{b}_h = \hat{b}_h + \frac{\min\{\hat{b}_<(z), \hat{b}_>(z)\}}{|I(z)|}$.
  4. Set $\hat{\epsilon}_h = \frac{1}{m} \cdot \hat{b}_h$.
- Return $\hat{\epsilon} = \sum_h \hat{\epsilon}_h$.

Procedure 1 (Given indices $ind_1 < ind_2$ and values $val_1 < val_2$, approximate the size of the set $T = \{j : ind_1 \leq j \leq ind_2$ and $val_1 < f(j) \leq val_2\}$)

1. Uniformly and independently select s indices from $I = \{ind_1, \ldots, ind_2\}$. Denote the subset of indices sampled by Z.
2. Let α be the fraction of indices j in Z for which $val_1 < f(j) \leq val_2$.
3. Let $\hat{t} = \alpha \cdot |I|$, and return $\hat{t}$.

Procedure 2 (Approximate median index and median value for node u)

1. Uniformly and independently select s indices from $I(u) = \{\ell ind(u), \ldots, rind(u)\}$, and denote the set of indices selected by Z. Let $Y \subseteq Z$ be the subset of indices j for which $\ell val(u) < f(j) \leq rval(u)$.
2. If Y is non-empty then let $ind(u)$ be the median index in Y, and let $val(u)$ be the median value of f on indices $j \in Y$. If Y is empty then exit the procedure and the algorithm with an error message.
3. Return $ind(u)$ and $val(u)$.

We first establish two claims concerning Procedure 1 and Procedure 2, respectively.

Lemma 9 *Let $ind_1 < ind_2$ and $val_1 < val_2$ be inputs to Procedure 1, and let*

$$T = \{j : ind_1 \leq j \leq ind_2 \text{ and } val_1 < f(j) \leq val_2\} .$$

If the sample size used by Procedure 1 is $s = \Theta(\log(n)/\gamma^2)$, then with probability at least $1 - \frac{1}{\text{poly}(n)}$,

$$|T| - (\gamma/2)|I| \;\leq\; \hat{t} \;\leq\; |T| + (\gamma/2)|I|$$

where $I = \{ind_1, \ldots, ind_2\}$ and $\hat{t}$ is the output of the procedure.

Proof: The Lemma follows immediately by applying an additive Chernoff bound. For each selected sample point $j \in I$, the probability that $j \in T$ is $|T|/|I|$, so that the expected value of α is $|T|/|I|$. Since the sample is of size $s = \Theta(\log n/\gamma^2)$, the probability that α deviates by more than $\gamma/8$ from its expected value is $\exp(-\Theta(\log n)) = \frac{1}{\text{poly}(n)}$. Since $\hat{t} = \alpha \cdot |I|$, the claim follows. ■

Lemma 10 *Let u be a node in the partial tree constructed by Algorithm 2 such that $|S(u)| \geq (\gamma/2)|I(u)|$. If the sample size used by Procedure 2 is $s = \Theta(\log n/\gamma^2)$, then with probability at least $1 - \frac{1}{\text{poly}(n)}$, the orders of $ind(u)$ and $val(u)$ each deviate by at most $(\gamma/16) \cdot |I(u)|$ from $\lceil |S(u)|/2 \rceil$.*

Proof: Let $\ell = |S(u)|$ and recall that by the premise of the lemma, $\ell \geq (\gamma/2)|I(u)|$. Also recall that Y is the subset of indices sampled by Procedure 2 that belong to $S(u)$, and denote $\bar{s} = |Y|$. Similarly to what was shown in Lemma 9, we can ensure that with probability at least $1 - \frac{1}{\text{poly}(n)}$, $\bar{s}/s \geq \ell/|I| - (\gamma/4)$, implying that Y is non-empty. Assume from now on that this is in fact true.

We next prove that with high probability, the order of $ind(u)$ deviates by at most $(\gamma/16) \cdot |I(u)|$ from $\lceil \ell/2 \rceil$. The claim concerning the order of $val(u)$ is proved analogously. Let the indices in $S(u)$ be $j_1 < j_1 < \ldots < j_\ell$. Define

$$\begin{aligned} S_1(u) &= \{j_1, \ldots, j_{\lceil \ell/2 \rceil - (\gamma/16)|I(u)|}\} \\ S_2(u) &= \{j_{\lceil \ell/2 \rceil - (\gamma/16)|I(u)|+1}, \cdots, j_{\lceil \ell/2 \rceil + (\gamma/16)|I(u)|}\} \\ S_3(u) &= \{j_{\lceil \ell/2 \rceil + (\gamma/16)|I(u)|+1}, \cdots, j_\ell\} \end{aligned}$$

We want to show that with high probability, $ind(u) \in S_2(u)$. By applying an additive Chernoff bound once more, we have that with probability at least $1 - \frac{1}{\text{poly}(n)}$ the number of sample points that belong to $S_1(u)$ does not deviate by more than $(\gamma/16) \cdot s$ from its expected value $\left(\frac{\lceil \ell/2 \rceil}{|I(u)|} - (\gamma/16)\right) \cdot s$, and a similar claim holds for $S_2(u)$ and $S_3(u)$. But in such a case, since $ind(u)$ is chosen to be

the median index in $Y \subset S(u)$, we have that $ind(u) \in S_2(u)$, as desired. The lemma follows by summing all the probabilities of failure. ■

We are now ready to prove that Algorithm 2 satisfies Theorem 1, which was stated at the beginning of Section 4.

Proof of Theorem 1: We start by bounding the sample size. The algorithm considers $O(\log n)$ levels, and for each level it selects at most m nodes. For each node selected the algorithm needs to construct at most all $O(\log n)$ ancestors of this node, where the construction requires a sample of size $O(s)$. Hence the total query complexity is $O(\log^2 n \cdot m \cdot s) = \tilde{O}((\log n)^7/\delta^4)$. The running time of the algorithm is linear in the query complexity.

We next turn to prove the correctness of the algorithm. Instead of directly analyzing Algorithm 2, we analyze a process that is different (and in particular does not have a sublinear sample and time complexity), but whose output is distributed exactly the same as that of Algorithm 2. By analyzing this alternative process we are able to separate between the errors Algorithm 2 incurs in the construction of the nodes (e.g., deviations of the indices $ind(v)$ from the median or deviations in the estimates $\hat{b}_{\leq}(v)$ and $\hat{b}_{>}(v)$), and the errors it incurs due to the random choice of nodes in the tree. While it is possible that a direct analysis of Algorithm 2 can be performed, care is needed in the slightly subtle probabilistic aspects of this analysis.

Consider the following two-stage process: First we fully construct the tree T_f, together with estimates $\hat{b}_{\leq}(v)$ and $\hat{b}_{>}(v)$ for every node v (in a manner that is explained momentarily), and then we apply Algorithm 1, which samples nodes from each level h of the constructed tree. Note that this sampling of nodes in level h can be implemented in a similar way to what is done in Algorithm 2. That is, the algorithm uniformly selects points $x \in \{1, \ldots, n\}$ and for each x selected it finds the node v in level h such that $x \in I(v)$.

CONSTRUCTION OF THE TREE. The tree is constructed in the same manner as the partial tree is constructed in Algorithm 2 with the following two exceptions: (1) For every level h, we consider all nodes u in level $h-1$ that are not empirically small, and for each such node u we add both its children to the tree. For each such child v we compute $\hat{s}(v)$, and if $\hat{s}(v) \geq (3\gamma/4)|I(v)|$ (so that v is not empirically small), then we compute $ind(v)$, $val(v)$, $\hat{b}_{\leq}(v)$, and $\hat{b}_{>}(v)$. (2) Whenever the algorithm adds a node u such that the estimated size $\hat{s}(u)$ of $S(u)$, is less than $(3\gamma/4)|I(u)|$ (i.e., u is empirically small, but its parent is not), then the subtree rooted at u is constructed as follows. For every descendent v of u (starting from u itself and continuing down the subtree), we let $ind(v)$ be the median of $S(v)$ and we let $val(v)$ be the median value in $\{f(j) : j \in S(v)\}$. Finally, we set $\hat{b}_{\leq}(v) = \hat{b}_{>}(v) = 0$. Recall that in the construction of the partial tree, if we have a node u for which $\hat{s}(u) < (3\gamma/4)|I(u)|$ then every descendent v of u in the partial tree is considered empirically small, and we set the estimates $\hat{b}_{\leq}(v) = \hat{b}_{>}(v) = 0$ as in the tree built by the process whose construction is described above.

By definition of Algorithm 2 and Algorithm 1, the distribution over the output $\hat{\epsilon}$ according to Algorithm 2 is exactly the same as in this two-stage process. The important things to note here are the following: (1) A node v in level h is indeed selected with probability $|I(v)|/n$ by Algorithm 2 (unless it is a descendent of an empirically small node, in which case its actual identity is immaterial since for such a node $\hat{b}_{\leq}(v) = \hat{b}_{>}(v) = 0$); (2) The random choice of m nodes selected in level h, is determined by a uniform choice of m points in $\{1, \ldots, n\}$, and is hence independent of the randomized construction of the partial/fully constructed tree; (3) The setting of m is the same in both algorithms. In other words, if we first fully construct the tree and only then select the nodes (as done in the two step process), or we first select the nodes (implicitly, by selecting the x_h^j's) and then construct the partial tree, then we get exactly the same distribution on the estimates $\hat{b}_{\leq}(v)$

and $\hat{b}_>(v)$.

Therefore, in order to establish Theorem 1 it suffices to prove that if we construct a tree T_f as described above, then with probability at least 9/10 it satisfies all the properties required by Lemma 8. The remainder of the proof is dedicated to establishing this claim.

PROPERTIES OF THE TREE. Recall that the premise of Lemma 8 is that for $\gamma \leq \delta/(5 \log n)$ the tree T_f is γ-balanced and that for every node v in the tree, $\hat{b}_\leq(v)$ and $\hat{b}_>(v)$ are γ-estimates of $b_\leq(v)$ and $b_>(v)$, respectively (where γ-estimates are as in Definition 8). Let us refer to these requirements as requirements R1 and R2, where we slightly strengthen the first requirement. That is, for every node v we have:

R1. If $|S(v)| < (\gamma/2)|I(v)|$ then the orders of $ind(v)$ and $val(v)$ are each $\lceil |S(v)|/2 \rceil$, and otherwise the orders of $ind(v)$ and $val(v)$ are a least $\lceil |S(v)|/2 \rceil - (\gamma/16)|I(v)|$ and at most $\lceil |S(v)|/2 \rceil + (\gamma/16)|I(v)|$.

R2. $\hat{b}_\leq(v)$ and $\hat{b}_>(v)$ are γ-estimates of $b_\leq(v)$ and $b_>(v)$, respectively.

It is easy to verify that if R1 holds for all nodes v, then the tree is γ-balanced. We shall add one more requirement that will aid us in our analysis.

R3. If the size $\hat{s}(v)$ of $S(v)$ is estimated by the algorithm, then $|\,\hat{s}(v) - |S(v)|\,| \leq \frac{\gamma}{2} \cdot |I(v)|$.

The tree is constructed top-down, starting from the root. We shall prove that for each node u constructed, conditioned on requirements R1–R3 holding for all its ancestors, the probability that one of the requirements is violated for node u is $1/\text{poly}(n)$. By a union bound, with probability at least $1 - (1/\text{poly}(n))$ all requirements hold for all nodes in the tree, as desired.

We start with the root node, v_0, whose construction is slightly different from that of all other nodes. By construction, $ind(v_0) = \lceil n/2 \rceil$ and $I(v_0) = S(v_0) = \{1, \ldots, n\}$. Hence the first part of requirement R1, concerning the order of $ind(v_0)$, always holds. The second part of requirement R1 concerning the order of $val(v_0)$ holds with probability $1 - 1/\text{poly}(n)$ by Lemma 10 (using the fact that $|S(v_0)| = |I(v_0)|$). Requirements R2 and R3 hold with probability $1 - 1/\text{poly}(n)$ by Lemma 9.

Next consider the case in which u is an empirically small node: Requirement R1 holds by construction (since for an empirically small node, the orders of $ind(u)$ and $val(u)$ are $\lceil |S(u)|/2 \rceil$). It remains to establish requirements R2 and R3 in this case:

1. If the parent p of u is not empirically small, we know that $\hat{s}(u)$ was computed so that we need to establish R3. But by Lemma 9, requirement R3 holds with probability $1 - 1/\text{poly}(n)$. Conditioned on R3 holding, since u is set to be empirically small because $\hat{s}(u) < (3\gamma/4)|I(u)|$, we know that $|S(u)| < \gamma|I(u)|$, and so the estimates $\hat{b}_\leq(u) = \hat{b}_>(u) = 0$ satisfy R2.

2. If on the other hand the parent p of u is empirically small, then we do not need to establish R3. To establish R2 in this case, consider the empirically small ancestor w of u which is furthest from u. That is, $\hat{s}(w) < (3\gamma/4)|I(w)|$, and so $|S(w)| < \gamma|I(w)|$ (since R3 holds for this ancestor). Hence R2 holds for u in this case as well.

Finally, if u is not empirically small: by applying Lemma 9 we know that R2 and R3 hold with probability $1 - 1/\text{poly}(n)$. Conditioned on R3 holding, and since by our assumption that u is not empirically small we know that $\hat{s}(u) \geq (3\gamma/4)|I(u)|$, we have that $|S(u)| \geq (\gamma/2)|I(u)|$. Hence we can apply Lemma 10 to establish R1 with probability $1 - 1/\text{poly}(n)$. ■

4.4 Monotonicity in Higher Dimensions

In this section we consider testing whether a function $f : [n]^d \to \Re$, where $d > 1$, is monotone.

The following notation will be useful.

Definition 9 *Let f be a function from $[n]^d$ to $\Re$, where we denote the distance of f to the closest monotone function over $[n]^d$ by $\epsilon^d_{\mathrm{mon}}(f)$.*

For $j \in \{1, \ldots, d\}$ and $\eta \in [n]^{d-1}$ we define the one-dimensional projection *of f determined by j and η as follows: For each $x \in [n]$, $f_{j,\eta}(x) = f(\eta_1, \ldots, \eta_{j-1}, x, \eta_{j+1}, \ldots, \eta_{d-1})$.*

Finally, for any fixed index $j \in \{1, \ldots, d\}$ we let $\mathrm{Exp}^j_{\mathrm{mon}}(f) = \mathrm{Exp}_\eta[\epsilon_{\mathrm{mon}}(f_{j,\eta})]$, and we let $\mathrm{Exp}_{\mathrm{mon}}(f) = \mathrm{Exp}_{j,\eta}[\epsilon_{\mathrm{mon}}(f_{j,\eta})]$.

For example, when $d = 2$ and we view f as a two-dimensional $n \times n$ matrix, then each function $f_{j,\eta}(\cdot)$ corresponds to either a row (in case $j = 1$) or a column (in case $j = 2$), $\mathrm{Exp}^1_{\mathrm{mon}}(f)$ is the average distance to monotonicity taken over all rows of the matrix, and $\mathrm{Exp}_{\mathrm{mon}}(f)$ is the average taken over all rows and columns.

It is easy to verify that for every index $j \in \{1, \ldots, d\}$, $\epsilon^d_{\mathrm{mon}}(f) \geq \mathrm{Exp}^j_{\mathrm{mon}}(f)$, and so, in particular,

$$\epsilon^d_{\mathrm{mon}}(f) \geq \mathrm{Exp}_{\mathrm{mon}}(f) \; . \tag{11}$$

Halevy and Kushilevitz [HK03a] prove the following *dimension reduction* lemma.

Lemma 11 ([HK03a]) *For every function $f : [n]^d \to \Re$,*

$$\mathrm{Exp}_{\mathrm{mon}}(f) \geq \frac{1}{d \cdot 4^{d-1}} \epsilon^d_{\mathrm{mon}}(f) \; .$$

Thus for example, when $d = 2$, the expected distance to monotonicity, taken over all one-dimensional projections of a function $f : [n]^2 \to \Re$, is at least an $1/8$ of its distance to monotonicity.

This lemma immediately suggests a distance approximation algorithm for higher dimensions: Simply obtain an estimate for $\mathrm{Exp}_{\mathrm{mon}}(f)$ and use it as an estimate for $\epsilon^d_{\mathrm{mon}}(f)$.

Specifically, let $(j_1, \eta_1), \ldots, (j_t, \eta_t)$ be $t = \Theta(1/\delta^2)$ uniformly and independently selected pairs where $j_\ell \in \{1, \ldots, d\}$ and $\eta_\ell \in [n]^{d-1}$ for each $1 \leq \ell \leq t$. We can define t random variables, $\chi_1, \ldots, \chi_t$ as follows: For each $1 \leq \ell \leq t$, $\chi_\ell = \epsilon_{\mathrm{mon}}(f_{j_\ell,\eta_\ell})$. By definition, for each ℓ, $\mathrm{Exp}[\chi_\ell] = \mathrm{Exp}_{\mathrm{mon}}(f)$. Let $\bar{\epsilon} = \frac{1}{t}\sum_\ell \chi_\ell$ be the average of these random variables. By an additive Chernoff bound and our choice of t, with probability at least $9/10$,

$$\mathrm{Exp}_{\mathrm{mon}}(f) - \delta/2 \leq \bar{\epsilon} \leq \mathrm{Exp}_{\mathrm{mon}}(f) + \delta/2. \tag{12}$$

Now, for each pair (j_ℓ, η_ℓ), by running our algorithm for one-dimensional functions (Algorithm 2), we can obtain a value $\hat{\epsilon}_\ell$ such that with high probability

$$(1/2)\chi_\ell - \delta/2 \leq \hat{\epsilon}_\ell \leq \chi_\ell + \delta/2 \; . \tag{13}$$

In "with high probability" we mean with probability at least $1 - 1/(6t)$: It is not hard to verify that the success probability of Algorithm 2 can be increased from $2/3$ to $1 - 1/(6t)$ at a multiplicative cost of $O(\log t) = O(\log(1/\delta))$ in the query complexity.

Let $\hat{\epsilon} = \frac{1}{t}\sum_\ell \hat{\epsilon}_\ell$. It follows that with probability at least $2/3$ (taken over the selection of the η_ℓ's and over the t executions of Algorithm 2), we get that

$$(1/2)\mathrm{Exp}_{\mathrm{mon}}(f) - \delta \leq \hat{\epsilon} \leq \mathrm{Exp}_{\mathrm{mon}}(f) + \delta. \tag{14}$$

But then, by Lemma 11 and Equation (11) we have that

$$\frac{1}{2d4^{d-1}}\epsilon^d_{\text{mon}}(f) - \delta \leq \hat{\epsilon} \leq \epsilon^d_{\text{mon}}(f) + \delta. \tag{15}$$

5 Tolerant Property Testing of Clustering

Let X be a set of n points, and let $d : X \times X \to \Re$ be a distance function defined over pairs of points in X. Let $C : 2^X \to \Re$ be a *cost measure* on sets of points, defined based on the underlying distance function $d(\cdot,\cdot)$. For a given k-way partition $P_X = \{X_i\}_{i=1}^k$ of X, the cost of the partition P_X is defined as $\max_i C(X_i)$, and with a slight abuse of notation is denoted by $C(P_X)$. [2]

In particular, we shall be interested in the case when $C(\cdot)$ is the *diameter* cost measure. Namely, for a given subset $S \subseteq X$, $C(S) = \max_{x,y \in S}\{d(x,y)\}$. Another cost measure we consider is the *radius* cost measure, for which $C(S) = \min_x \max_{y \in S} d(x,y)$.

Definition 10 ((k,b)-clusterable) *Let k be an integer and let b be a real value. We say that a set X is* (k,b)-clusterable *with respect to the cost function $C(\cdot)$ (and the underlying distance function $d(\cdot,\cdot)$), if there* exists *a k-way partition $P_X = \{X_i\}_{i=1}^k$ of X such that $C(P_X) \leq b$.*

Definition 11 (Hereditary Cost Measures) *We say that the cost measure $C(\cdot)$ is an* hereditary *clustering cost if for every k and b, whenever X is (k,b)-clusterable with respect to $C(\cdot)$, then so is every subset Y of X.*

Note for example that the diameter cost measure is hereditary.

Definition 12 (ϵ-far) *A set X is said to be* ϵ-far *from $(k,(1+\beta)b)$-clusterable with respect to $C(\cdot)$ for a given $0 \leq \epsilon \leq 1$ and $\beta \geq 0$, if for* every *subset $Y \subseteq X$ of size at most $(1-\epsilon)|X|$, and for every k-way partition $P_Y = \{Y_i\}_{i=1}^k$ of Y, we have $C(P_Y) > (1+\beta)b$.*

We consider the following "natural" algorithm for tolerant testing of clustering.

Algorithm 3 (Tolerant Testing Algorithm for (k,b)-Clustering with respect to cost $C(\cdot)$)

1. Uniformly at random select m points from X (where m will be specified later). Denote the set of points selected by U.
2. Let $\delta = \epsilon_2 - \epsilon_1$. If U is $(\epsilon_1 + \delta/2)$-close to (k,b)-clusterable with respect to $C(\cdot)$ then accept, otherwise reject.

Recall that a tolerant testing algorithm should accept X with probability at least $2/3$ if X is ϵ_1-close to (k,b)-clusterable with respect to $C(\cdot)$. The following lemma, which can be easily proved using an additive Chernoff bound, ensures that this is the case whenever the cost measure $C(\cdot)$ is an hereditary clustering cost and the sample size m is sufficiently large.

Lemma 12 *Let $C(\cdot)$ be an hereditary clustering cost, and let X be a set of points that is ϵ_1-close to being (k,b)-clusterable with respect to $C(\cdot)$. Then with probability at least $2/3$ over the choice of a random subset $U \subseteq X$ of size $m = \Omega(1/\delta^2)$, the subset U is $(\epsilon_1 + \delta/2)$-close to being (k,b)-clusterable with respect to $C(\cdot)$.*

[2] An alternative is to consider the *average* cost of the subsets in the partition, where the average may be weighted according to the sizes of the subsets X_i. For simplicity, and since this will be the case of our particular applications, we consider only taking the maximum cost over all subsets.

The focus of the analysis of Algorithm 3 is hence on proving that, for an appropriate choice of the sample size m, if X is ϵ_2-far from being $(k,(1+\beta)b)$-clusterable with respect to $C(\cdot)$, then it is rejected with probability at least 2/3.

In what follows we describe a framework under which such claims can be proved. As stated in the introduction, this framework generalizes the abstract combinatorial programs of Czumaj and Sohler [CS02]. We later apply this framework to establish the correctness of Algorithm 3 when: (1) The cost measure is either the diameter cost or the radius cost, and the distance function $d(\cdot,\cdot)$ is a general metric, that is, it obeys the triangle inequality; (2) The cost measure is the diameter cost and the distance function $d(\cdot,\cdot)$ is the Euclidean metric, where $X \subset \Re^d$.

We also discuss in Section 6 how the framework can be generalized to tolerant testing of graph properties.

5.1 A Framework for tolerant testing of clustering: Skeletons and Witnesses

We start by introducing the notions of skeletons and witnesses as described briefly in the introduction.

Definition 13 (Skeletons) *A* skeleton *defined over a set X of points is a partition $P_S = \{S_i\}_{i=1}^t$ of a subset $S \subseteq X$. We let $\mathcal{S}_{s,p}(X)$ be any set of skeletons over X, such that for every skeleton $P_S \in \mathcal{S}_{s,p}(X)$, we have that $|S| \leq s$, and for every subset $S \subseteq X$, there are at most p skeletons $P_S \in \mathcal{S}_{s,p}(X)$.*

Intuitively we may think of skeletons as being representatives of partitions (clusterings) of all points in X, or of almost all points in X, where each S_i is a subset of a different cluster X_i. For example, we may let a skeleton simply be a partition of at most k points into singletons, where each point represents a different potential cluster.

Definition 14 (Witnesses) *Let $w : \mathcal{S}_{s,p}(X) \times X \to \{0,1\}$ be a* witness *function. If $w(P_S,x) = 1$ then we say that x is a* witness *for P_S.*

Intuitively, witnesses are points that in some sense provide evidence to imperfections of skeletons. In continuation to the example described above, a witness may simply be a point that is at distance greater than b from every skeleton point.

Definition 15 (Good Subsets) *A subset $U \subseteq X$ is $\boldsymbol{\alpha}$-*good *with respect to $\mathcal{S}_{s,p}(X)$, if there exists a skeleton $P_S \in \mathcal{S}_{s,p}(X)$ such that $S \subseteq U$, and there are at most $\alpha \cdot |U|$ points $u \in U$, such that $w(P_S,u) = 1$ (that is, u is a witness for P_S). Otherwise the subset U is $\boldsymbol{\alpha}$-*bad *with respect to $\mathcal{S}_{s,p}(X)$.*

The following Lemma will allow us to prove the correctness of Algorithm 3 under certain conditions on the set of skeletons $\mathcal{S}_{s,p}(X)$.

Lemma 13 *Let $\mathcal{S}_{s,p}(X)$ be a set of skeletons defined over X, and let $0 < \alpha, \gamma < 1$. Suppose that every skeleton in $\mathcal{S}_{s,p}(X)$ has at least $(\alpha+\gamma)\cdot|X|$ points $x \in X$ that are witnesses for it. Consider selecting, uniformly and independently, a subset U of $m = \Omega\left((\log p + s\log(s/\gamma))/\gamma^2\right)$ points from X. Then with probability at least 2/3, the subset U is α-bad with respect to $\mathcal{S}_{s,p}(X)$.*

Proof: Let $m = s + m'$, where m' is set subsequently, and let $u_1, \ldots, u_m$ be the m elements selected uniformly and at random. For each subset of indices $I \subset [m]$, $I = \{i_1, \ldots, i_s\}$, of size s, let U_I be the (multi)set $\{u_{i_1}, \ldots, u_{i_s}\}$. Note that if we take the union over all $I \subset [m]$ of the skeletons $P_S \in \mathcal{S}_{s,p}(X)$ such that $S \subseteq U_I$, then we get all skeletons contained in U (where some skeletons are counted more than once).

When considering any particular I, it will be convenient to think of the selection process of U as first selecting U_I and then $U_{[m]\setminus I}$. For a fixed subset of indices I, and a fixed skeleton P_S, where $S \subseteq U_I$. Let E_{I,P_S} denote the event that among the $m' = m - s$ vertices in $U_{[m]\setminus I}$ there are less than $(\alpha + \gamma/2)m'$ witnesses to P_S. By an additive Chernoff bound,

$$\Pr[E_{I,P_S}] < \exp(-2m'(\gamma/2)^2)$$

Let E_I denote the union over all E_{I,P_S}, for a fixed I. That is, it is the event that for some skeleton P_S, $S \subseteq U_I$, there are less than $(\alpha + \gamma/2)m'$ witnesses to P_S in $U_{[m]\setminus I}$. Since the number of such skeletons is at most $2^s \cdot p$, by a union bound,

$$\Pr[E_I] < 2^s \cdot p \cdot \exp(-2m'(\gamma/2)^2)$$

Finally, let E denote the union over all I of E_I. Then

$$\Pr[E] \leq (2m)^s \cdot p \cdot \exp(-2m'(\gamma/2)^2)$$

If we select $m' = \Omega(\gamma^{-2}(\log p + s\log(s/\gamma)))$ then we get that with probability at least $2/3$, for every skeleton in U, there are at least $(\alpha + \gamma/2)m'$ witnesses for the skeleton. Since $m' \geq 2s/\gamma$, this is at least $\alpha \cdot m$, as desired. ■

It is now possible to prove the correctness of Algorithm 3, assuming that there exists a set of skeletons $\mathcal{S}_{s,p}(X)$ which satisfies the two conditions that are specified below. The theorem also requires the sample size m taken by the algorithm to be some large enough function of ϵ_1, ϵ_2 and of s and p.

Theorem 2 *Let $C(\cdot)$ be a hereditary clustering cost, and let $0 < \beta \leq 1$. Suppose that $\mathcal{S}_{s,p}(X)$ is a set of skeletons that satisfy the following:*

1. *For any given $0 \leq \alpha \leq 1$, if X is α-far from $(k, (1+\beta)b)$-clusterable with respect to $C(\cdot)$, then every skeleton in $\mathcal{S}_{s,p}(X)$ has at least $\alpha \cdot |X|$ witnesses for it in X.*
2. *For any given $0 \leq \alpha \leq 1$, if a subset $U \subseteq X$ is α-bad with respect to $\mathcal{S}_{s,p}(X)$, then U is α-far from (k, b)-clusterable with respect to $C(\cdot)$.*

Let $m = \Omega\left((\log p + s\log(s/\delta))/\delta^2\right)$ where $\delta = \epsilon_2 - \epsilon_1$. If X is ϵ_1-close to (k, b)-clusterable with respect to $C(\cdot)$, then Algorithm 3 accepts with probability at least $2/3$, and if X is ϵ_2-far from $(k, (1+\beta)b)$-clusterable with respect to $C(\cdot)$, then Algorithm 3 rejects with probability at least $2/3$.

We note that it is possible to slightly relax the two requirements on the set of skeletons that are stated in the theorem, and still obtain the same result. In relaxing we mean that in the first item we require that there be a least $\alpha'|X|$ witnesses in X, where α' is slightly smaller than α (say, $\alpha' = \alpha - \delta/4$) and in the second item to require that U be α'-far from (k, b)-clusterable.

Proof: The first part of the theorem directly follows from Lemma 12, since $C(\cdot)$ is assumed to be hereditary, and $m = \Omega(1/\delta^2)$. We turn to the second part.

Suppose that X is ϵ_2-far from $(k,(1+\beta)b)$-clusterable with respect to $C(\cdot)$. By the first item in the premise of the theorem, every skeleton in $\mathcal{S}_{s,p}(X)$ has at least $\epsilon_2 \cdot |X|$ witnesses for it in X. By Lemma 13, if we set $\alpha = \epsilon_2 - \delta/2$ and $\gamma = \delta/2$, then with probability $2/3$ over the choice of the sample U, the set U is $(\epsilon_2 - \delta/2)$-bad. But by the second item in the premise of the theorem, for each such U, the set U is $(\epsilon_2 - \delta/2)$-far from being (k,b)-clusterable. Since $\epsilon_2 - \delta/2 = \epsilon_1 + \delta/2$, each such U would cause Algorithm 3 to reject, and the second part of the theorem follows. ■

Note that in order to use this theorem to prove the correctness of Algorithm 3 for specific cost measures, such as the diameter cost measure, we have to define a skeleton set $\mathcal{S}_{s,p}(X)$ and witnesses for the specific cost measure at hand. Furthermore, we must prove that the two conditions in Theorem 2 hold, and that s and p are bounded so that the sample size m will not be too large. In the next sub-sections we show that it is possible to do so for the diameter and the radius cost measures when the distance function is a general metric, and for the diameter cost measure when the distance function is the Euclidean metric.

5.2 Clustering Under a General Metric

In this subsection we show how to apply Theorem 2 so as to obtain the following theorem for the diameter cost. At the end of this subsection we give an analogous theorem for the radius cost. Recall that $\delta = \epsilon_2 - \epsilon_1$.

Theorem 3 *Let $C(\cdot)$ be the diameter cost, and let $d(\cdot,\cdot)$ be any underlying distance function that obeys the triangle inequality. Suppose that we run Algorithm 3 with $m = \Theta((k/\delta^2)\log(k/\delta))$. If X is ϵ_1-close to (k,b)-clusterable, then with probability at least $2/3$ Algorithm 3 accepts X, while if X is ϵ_2-far from $(k,2b)$-clusterable the with probability at least $2/3$ it rejects X.*

In order to apply Theorem 2 we define skeletons and witnesses as follows:

Definition 16 (Skeletons and Witnesses for General Metrics) *A skeleton P_S is a partition of a subset $S \subseteq X$, $|S| \leq k$ into singletons. A point $x \in X$ is a witness for a skeleton P_S, if it is at distance greater than b from every point in S.*

Since the partition P_S is uniquely defined by the set S, we simply use S to denote a skeleton. Let $\mathcal{S}(X)$ denote the set of all skeletons over X, where the subscripts s and p (which in this case are k and 1, respectively) are eliminated for sake of succinctness.

The next two lemmas establish that the two items in Theorem 2 hold for $\mathcal{S}(X)$ as defined above. Theorem 3 directly follows using the fact that $s = k$ and $p = 1$.

Lemma 14 *Let $0 \leq \alpha \leq 1$. If X is α-far from $(k,2b)$-clusterable then every skeleton in $\mathcal{S}$ has at least $\alpha|X|$ witnesses in X.*

Proof: Assume, contrary to the claim, that there exists a skeleton S with less than $\alpha|X|$ witnesses in X. Consider the subset $Y \subset X$ that consists of all points in X that are not witnesses for S. By definition of skeletons and witnesses in this case, each point in Y is at distance at most b from some point in S. We can now assign each point in Y to the closest point in the skeleton S. All points that are assigned to the same skeleton point must be at distance at most $2b$ from each other (using the triangle inequality). Thus Y is $(k,2b)$-clusterable, and has size at least $(1-\alpha)|X|$. But this means that X is α-close to $(k,2b)$-clusterable, and we have reached a contradiction, as desired. ■

Lemma 15 *Let $0 \leq \alpha \leq 1$. If a subset $U \subseteq X$ is α-bad with respect to $\mathcal{S}(X)$, then U is α-far from (k, b)-clusterable.*

Proof: Assume, contrary to the claim, that U is α-close to (k, b)-clusterable. We will show that U is α-good (thus reaching a contradiction) by presenting a skeleton $S \subseteq U$ with at most $\alpha|U|$ witnesses in U with respect to S.

Since U is α-close to (k, b)-clusterable, there exists a subset $W \subseteq U$, of size at least $(1 - \alpha)|U|$ that can be clustered into k clusters of diameter at most b. We now choose a point from each cluster in W to obtain a skeleton $S \subseteq W \subseteq U$. The skeleton S has at most $\alpha|U|$ witnesses in U (that is, the points that do not belong to W), as claimed. ■

5.2.1 Clustering with respect to the radius cost

We next show how a very similar analysis can be applied to clustering with respect to the radius cost.

Theorem 4 *Let $C(\cdot)$ be the radius cost, and let $d(\cdot, \cdot)$ be any underlying distance function that obeys the triangle inequality. Suppose that we run Algorithm 3 with $m = \Theta((k/\delta^2)\log(k/\delta))$. If X is ϵ_1-close to (k, b)-clusterable, then with probability at least $2/3$ Algorithm 3 accepts X, while if X is ϵ_2-far from $(k, 2b)$-clusterable the with probability at least $2/3$ it rejects X.*

In order to apply Theorem 2 and establish Theorem 4 we need to slightly modify the definition of witnesses that was given for the diameter cost. The definition of skeletons remains the same.

Definition 17 (Skeletons and Witnesses for General Metrics and the Radius Cost) *A skeleton P_S is a partition of a subset $S \subseteq X$, $|S| \leq k$ into singletons. A point $x \in X$ is a witness for a skeleton P_S, if it is at distance greater than $2b$ from every point in S.*

It is easy to verify that Lemmas 14 and 15 hold for the radius cost using the above definition and the triangle inequality. Theorem 4 directly follows using the fact that $s = k$ and $p = 1$.

5.3 Clustering Under the Euclidean Metric

In this subsection we consider the case that the set of points X lies in Euclidean space and the underlying distance function is the Euclidean distance. The cost measure $C(\cdot)$ is the diameter cost.

Theorem 5 *Let $C(\cdot)$ be the diameter cost, let $X \subset \Re^d$ for some integer d, and let the underlying distance $d(\cdot, \cdot)$ be the Euclidean distance between points. Then for any given $0 < \beta \leq 1$, if we run Algorithm 3 with $m = \tilde{\Theta}(k \cdot \delta^{-2} \cdot (1 + (2/\beta))^d)$, then with probability at least $2/3$ it accepts X if X is ϵ_1-close to (k, b)-clusterable, and with probability at least $2/3$ it rejects X if X is ϵ_2-far from $(k, (1 + \beta)b)$-clusterable.*

Here too we shall apply Theorem 2, but we shall need slightly more sophisticated notions of skeletons and witnesses. The definitions used here are taken from [CS02], which in turn are partly based on ideas introduced in [ADPR03].

Definition 18 (Intersections of Balls) *For any subset $Y \subseteq X$ let $I(Y)$ denote the intersection of all d-dimensional balls of radius b centered at the points in Y.*

Definition 19 (Violating and Influential Points) *Let $Y \subseteq X$, such that $I(Y) \neq \emptyset$. A point $x \in X$ is* violating *for $Y \subseteq X$ if $x \notin I(Y)$. The point x is* influential *with respect to Y if $x \in I(Y)$ and for every $y \in Y$ it holds that $dist(x,y) > \beta b$.*

Definition 20 (Skeletons and Witnesses for the Euclidean Distance) *A* skeleton *is defined inductively as follows:*

1. *The k-partition $P_\emptyset = \{\emptyset, \ldots, \emptyset\}$ is a skeleton (that is, all k parts are empty).*
2. *If $P_S = \{S_1, \ldots, S_k\}$ is a skeleton and $x \in X \setminus S$ is an influential point with respect to S_i for some $1 \leq i \leq k$, then $P_{S \cup \{x\}} = \{S_1, \ldots, S_{i-1}, S_i \cup \{x\}, S_{i+1}, \ldots, S_k)$ is a skeleton. (Note that there may be more than one way to add x to the skeleton P_S.)*

A point $x \in X$ is a witness *for a skeleton $P_S = \{S_1, \ldots, S_k\}$, if for every $1 \leq i \leq k$ the point x is either violating or influential with respect to S_i.*

Here to we denote by $\mathcal{S}(X)$ the set of all skeletons defined over X. Clearly, for every set S, if $|S| \leq s$, then the number of partitions $P_S \in \mathcal{S}(X)$ is bounded by k^s (where this upper bound will suffice for our purposes). The following lemma which bounds the size s of the sets S that participate in skeletons is from [CS02].

Lemma 16 *Let $P_S = \{S_i\}_{i=1}^k$ be a skeleton in $\mathcal{S}(X)$. Then $|S| \leq k(1 + (2/\beta))^d$.*

Lemmas 17 and 18, stated and proved below, establish the two items in Theorem 2 for the set of skeletons $\mathcal{S}(X)$ as defined above. Theorem 5 readily follows (using lemma 16).

Lemma 17 *Let $0 \leq \alpha \leq 1$. If X is α-far from $(k, b(1+\beta))$-clusterable, then every skeleton in $\mathcal{S}(X)$ has at least $\alpha|X|$ witnesses in X.*

Proof: Assume, contrary to the claim, that there exists a skeleton $P_S = \{S_i\}_{i=1}^k$ with less than $\alpha|X|$ witnesses in X. Let $Z \subseteq X$ be the subset of all points that are not witnesses with respect to P_S. Hence $|Z| > (1-\alpha)|X|$. We next show that Z is $(k, (1+\beta)b)$-clusterable, and reach a contradiction to the premise of the lemma.

For each $z \in Z$ there exists an index i, $1 \leq i \leq k$ such that z is not violating and non-influential with respect to S_i. We assign such a point z to the i'th cluster, where initially the i'th cluster consists of S_i. We next show that the distance between any two points in the i'th cluster is at most $(1+\beta)b$.

First observe that all points in S_i are at distance at most b from each other. This holds by construction of skeletons, since a point x can be added to S_i only if it is influential with respect to S_i, which in particular requires that $x \in I(S_i)$. As to points $z \in Z$ that were assigned to the i'th cluster: note that any such point z was non-violating and non-influential with respect to S_i. Hence $z \in I(S_i)$ and there exists a point $y \in S_i$ such that $dist(z,y) \leq \beta b$. Using the triangle inequality, we get that for any other point x in the i'th cluster it holds that $dist(z,x) \leq dist(x,y) + dist(y,z) \leq (1+\beta)b$. Thus Z is $(k, (1+\beta)b)$-clusterable, as claimed. ■

Lemma 18 *Let $0 \leq \alpha \leq 1$. If a subset $U \subseteq X$ is α-bad with respect to $\mathcal{S}(X)$, then U is α-far from (k,b)-clusterable.*

Proof: Assume, contrary to the claim, that U is α-close to (k,b)-clusterable. That is, there exists a subset $W \subseteq U$, $|W| \geq (1-\alpha)|U|$, and a partition, $\{W_i\}_{i=1}^k$, such that the diameter of each W_i is at most b.

We next prove that there exists a skeleton $P_S = \{S_i\}_{i=1}^k$, $S_i \subseteq W_i \subseteq U$, with at most $\alpha|U|$ witnesses in U. We will build P_S in the following iterative manner:

1. We start with the skeleton $P_\emptyset = \{\emptyset, \ldots, \emptyset\}$.
2. Let $P_{S^j} = \{S_i^j\}_{i=1}^k$ be the skeleton at the beginning of the j'th iteration. If there exists an index i and a point $x \in W_i$ that is an influential point with respect to S_i^j, then we let $S^{j+1} = S^j \cup \{x\}$ and $P_{S^{j+1}} = \{S_1^j, \ldots, S_{i-1}^j, S_i^j \cup \{x\}, S_{i+1}^j, \ldots, S_k^j\}$.
3. When for every i, the subset W_i does not contain any influential points with respect S_i then we stop.

Let $P_S = \{S_i\}_{i=1}^k$ be the final resulting skeleton. Notice that for every i, the subset W_i does not contain a violating point with respect to $S_i \subseteq W_i$, because all points in W_i are at distance at most b from each other. Also when we finish building $P_S = \{S_i\}_{i=1}^k$, the subset W_i contains no influential points with respect to S_i. Thus all points in W_i are not witnesses with respect to P_S. Since there are at most $\alpha|U|$ points in U that do not belong to any cluster W_i, then there are at most $\alpha|U|$ witnesses with respect to P_S. ■

5.4 Finding Approximately Good Clusterings

As stated in the introduction, our tolerant testing algorithm for clustering can be easily modified to obtain an algorithm that outputs an approximately good clustering of most input points. More precisely, for any given parameters $\epsilon_1 < \epsilon_2$, if the set of points X is ϵ_1-close to being (k,b)-clusterable, then with high probability the modified algorithm outputs an implicit representation of a $(k,(1+\beta)b)$-clustering of all but an ϵ_2-fraction of the points of X. In an "implicit representation" we mean a partition of a small subset of points that can be used to determine the cluster to which each point in X belongs to (but at most $\epsilon_2|X|$ of the points). Specifically, we prove:

Theorem 6 *Let $C(\cdot)$ be a hereditary clustering cost, and let $0 < \beta \leq 1$. Suppose that $\mathcal{S}_{s,p}(X)$ is a set of skeletons for X that satisfy the following:*

1. *For any given skeleton $P_S \in \mathcal{S}_{s,p}(X)$, the subset of all points in X that are not witnesses for P_S is $(k,(1+\beta)b)$-clusterable.*
2. *For any given $0 \leq \alpha \leq 1$, if a subset $U \subset X$ is α-close to (k,b)-clusterable with respect to $C(\cdot)$, then U is α-good with respect to $\mathcal{S}_{s,p}(X)$.*

Then the following holds for Algorithm 4 specified below: for any given parameters $\epsilon_1 < \epsilon_2$, if X is ϵ_1-close to (k,b)-clusterable with respect to $C(\cdot)$, then with high constant probability Algorithm 4 outputs an implicit representation of a $(k,(1+\beta)b)$-clustering of all but at most an ϵ_2-fraction of the points in X. By an "implicit representation" we mean a skeleton in $\mathcal{S}_{s,p}(X)$ that can be used to determine the cluster to which each point in X belongs to (but at most $\epsilon_2|X|$ of the points).

Algorithm 4 (Approximate Clustering Algorithm given a set of skeletons $\mathcal{S}_{s,p}(X)$)

1. Uniformly at random select $m = \Theta\left((\log p + s\log(s/\delta))/\delta^2\right)$ points from X, where $\delta = \epsilon_2 - \epsilon_1$.
2. If U is $(\epsilon_1 + \delta/2)$-close to (k, b)-clusterable with respect to $C(\cdot)$ then do:
   - Find a subset $S \subseteq U$ such that $P_S \in \mathcal{S}_{s,p}(X)$ and such that there exist at most $(\epsilon_1 + \delta/2) \cdot |U|$ witnesses to P_S in U.
3. Output P_S.

Proof of Theorem 6: Let X be a set of points that is ϵ_1-close to (k, b)-clusterable with respect to $C(\cdot)$, and recall that $C(\cdot)$ is an hereditary cost measure. By Lemma 12, given the size m of the sample U that is selected by Algorithm 4, with high probability the subset U is $(\epsilon_1 + \delta/2)$-close to being (k, b)-clusterable. By Condition (2) in Theorem 6, this implies that the subset U is $(\epsilon_1 + \delta/2)$-good with respect to $\mathcal{S}_{s,p}(X)$. That is, it contains a subset S such that $P_S \in \mathcal{S}_{s,p}(X)$ and such that there are at most $(\epsilon_1 + \delta/2) \cdot |U|$ witnesses to P_S in U.

By the proof of Lemma 13 we can ensure that with high probability over the choice of U, for every skeleton $P_S \in \mathcal{S}_{s,p}(X)$ such that $S \subseteq U$ and there are more than $\epsilon_2|X|$ witnesses to P_S in X, the fraction of witnesses to P_S in U is more than $\epsilon_2 - \delta/2 = \epsilon_1 + \delta/2$. Assuming that this event holds, then for every subset $S \subseteq U$ such that $P_S \in \mathcal{S}_{s,p}(X)$ and U contains at most $(\epsilon_1 + \delta/2) \cdot |U|$ witnesses to P_S, the skeleton P_S has at most $\epsilon_2|X|$ witnesses in X. By Condition (1) in Theorem 6, every such skeleton (and in particular the one output by Algorithm 4) is an implicit representation of a $(k, (1+\beta)b)$-clustering of all but an ϵ_2-fraction of the points in X. ■

Applying Algorithm 4 to specific cost measures. It is not hard to verify that the conditions in Theorem 6 hold for each of the cost measures studied in the previous subsections (Subsections 5.2 and 5.3). Specifically, observe that the first condition in Theorem 6 is a slight strengthening of the first condition in Theorem 2, and the second condition is equivalent to the second condition in Theorem 2. We have shown that the second condition holds for the cases we have studied in the previous subsections. It is easily verified that the proofs that the first condition in Theorem 2 holds for these cases, in fact give the first condition in Theorem 6. Hence for these cases, the query complexity of Algorithm 4 is the same as that of the tolerant testing algorithm.

It is also easy to verify that for the cases studied in the previous subsections, given a skeleton $P_S \in \mathcal{S}_{s,p}(X)$ and a point $x \in X$, it is possible to efficiently determine $w(P_S, x)$ (that is, to determine whether x is a witness for P_S). Moreover, if $w(P_S, x) = 0$ then it is possible to efficiently assign x to a cluster (so that we obtain a $(k, (1+\beta)b)$-clustering of all points $\{x : w(P_S, x) = 0\}$, as required in the first condition in Theorem 6). In particular, all we need to do in order to determine $w(P_S, x)$ (and if $w(P_S, x) = 0$, to determine the cluster that x belongs to) is to compute the distances between x and the points in S.

The running time of Algorithm 4 for a specific cost measure, depends, as before, on the complexity of determining whether U is $(\epsilon_1 + \delta/2)$-close to (k, b)-clusterable. In addition it now depends on the complexity of finding the desired P_S. In the cases studied in the previous subsections, the latter task can be done efficiently provided that a clustering of all but an $(\epsilon_1 + \delta/2)$-fraction of the points in U is indeed found (see the proofs to Lemmas 15 and 18).

6 Tolerant Testing of Graph Properties

Let $G = (V, E)$ be a graph where we denote by G_U the subgraph of G induced by a subset $U \subseteq V$. If Q is a property of graphs, then the following is a "natural" candidate for a tolerant testing algorithm for graph property Q.

Algorithm 5 (Tolerant Testing Algorithm for Graph Property Q)

1. *Uniformly and independently select m vertices from V and denote the resulting subset by U.*
2. *Let $\delta = \epsilon_2 - \epsilon_1$. If G_U is $(\epsilon_1 + \delta/2)$-close to having property Q then accept, otherwise reject.*

The framework described in Subsection 5.1 can be adapted to graphs and partition properties of graphs, similarly to what was done for non-tolerant testing by Czumaj and Sohler [CS02]. Using their framework they obtain results for standard property testing of colorability (of dense graphs). In this case the set of points X is the set of graph vertices V, and skeletons are partitions of subsets of vertices.

Definition 21 (Hereditary) *A property Q of graphs is* hereditary *if for every graph G that has property Q, it is the case that every subgraph of G also has the property Q.*

Lemma 19 *Let Q be an hereditary property of graphs, and let $0 < \epsilon_1, \delta < 1$. If G is ϵ_1-close to having property Q, then with probability at least $2/3$ over the choice of a random subset U of size $m = \Omega(\log(1/\delta)/\delta^2)$, the subgraph G_U is $(\epsilon_1 + \delta/2)$-close to having Q.*

Proof: Let G' be a graph having property Q such that G is at most ϵ_1-far from G'. That is, $|\Delta\{E(G), E(G')\}| \leq \epsilon_1 n^2$, where

$$\Delta\{E(G), E(G')\} \stackrel{\text{def}}{=} \{E(G) \setminus E(G')\} \cup \{E(G') \setminus E(G)\}$$

denotes the symmetric difference between the two sets of edges. For convenience of our analysis, we view each edge in $E(G)$ (similarly, $E(G')$), as an *ordered* pair. Recall that since Q is hereditary, then every subgraph of G' has property Q. We would like to show that with probability at least $2/3$ over the choice of U, the subgraphs G_U and G'_U are $(\epsilon_1 + \delta)$-close. The lemma directly follows.

For each vertex v, let $D(v)$ denote the subset of vertices u, such that $(u, v) \in \Delta\{E(G), E(G')\}$, and let $d(v) = |D(v)|/n$. By these definitions, and our assumption on G, we have that

$$\frac{1}{n} \sum_{v \in V} d(v) \leq \epsilon_1 \tag{16}$$

Let $U = \{v_1, \ldots, v_m\}$, where each v_i is chosen uniformly and independently. Since the expected value of $d(v_i)$, for every i, is at most ϵ_1, then by an additive Chernoff bound,

$$\Pr\left[\frac{1}{m} \sum_{i=1}^{m} d(v_i) > \epsilon_1 + \delta/4\right] < \exp(-2m(\delta/4)^2) \tag{17}$$

which for $m = \Omega(1/\delta^2)$, is at most $1/10$. Thus we have that with sufficiently high probability, the average value of $d(v_i)$ taken over the sample U, is not much larger than ϵ_1.

For each $i = 1, \ldots, m$, let $U_i = U \setminus \{v_i\}$, and define the event E_i as follows:

$$E_i : \qquad \frac{|D(v_i) \cap U_i|}{m-1} > d(v_i) + \delta/4 \tag{18}$$

Once again, if we apply an additive Chernoff bound then we get that for each $1 \leq i \leq m$,

$$\Pr[E_i] < \exp(-2(m-1)(\delta/4)^2) \tag{19}$$

For $m = \Omega(\log(1/\delta)/\delta^2$, this probability is at most $1/(10m)$. By a union bound, with probability at least $9/10$ none of the events E_i holds and hence for every $1 \leq i \leq m$, we have that

$$\frac{|D(v_i) \cap U_i|}{m-1} \leq d(v_i) + \delta/4 \tag{20}$$

Combining the above we have that with probability at least $2/3$

$$\begin{aligned}
\Delta\{E(G_U), E(G'_U)\} &= \sum_{i=1}^{m} |D(v_i) \cap U_i| \\
&\leq m \cdot \sum_{i=1}^{m} (d(v_i) + \delta/4) \\
&\leq m \cdot \sum_{i=1}^{m} d(v_i) + m^2(\delta/4) \\
&\leq m^2(\epsilon_1 + \delta/4) + m^2(\delta/4) \\
&\leq (\epsilon_1 + \delta/2)m^2
\end{aligned} \tag{21}$$

as desired. ■

As was done above for clustering, it is possible to define skeletons and witnesses, however here these concepts are defined with respect to the set of vertices V.

Theorem 7 *Let Q be an hereditary partition property of graphs. Suppose that $\mathcal{S}_{s,p}(V)$ is a set of skeletons defined over V, which satisfy the following:*

1. *For any given $0 \leq \alpha \leq 1$, if G is α-far from property Q then for every skeleton in $\mathcal{S}_{s,p}(V)$ there are at least $\alpha \cdot |V|$ witnesses in V.*
2. *For any given $0 \leq \alpha \leq 1$, if a subset $U \subseteq V$ is α-bad with respect to $\mathcal{S}_{s,p}(V)$, then G_U is α-far from Q.*

Let $m = \Omega\left((\log p + s\log(s/\delta))/\delta^2\right)$ where $\delta = \epsilon_2 - \epsilon_1$. If G is ϵ_1-close to having property Q then Algorithm 5 accepts with probability at least $2/3$, and if G is ϵ_2-far from having property Q then Algorithm 5 rejects with probability at least $2/3$.

Proof: The first item in the theorem directly follows from Lemma 19.

To prove the second item, consider a graph G that is $(\epsilon_2 = \epsilon_1 + \delta)$-far from Q. By the first item in the premise of the theorem, every skeleton in $\mathcal{S}_{s,p}$ has at least $\epsilon_2 \cdot |V|$ elements $v \in V$ that are witnesses for it. By Lemma 13, if we set $\alpha = \epsilon_2 - \delta/2$ and $\gamma = \delta/2$, then with probability $2/3$ over the choice of U, the set U is $(\epsilon_2 - \delta/2)$-bad. But by the second item in the premise of the theorem, for each such U, the graph G_U is $(\epsilon_2 - \delta/2)$ far from Q. Since $\epsilon_2 - \delta/2 = \epsilon_1 + \delta/2$, each such U would cause Algorithm 5 to reject, and the second item of the theorem follows. ■

References

[ADPR03] N. Alon, S. Dar, M. Parnas, and D. Ron. Testing of clustering. *SIAM Journal on Discrete Math*, pages 393–417, 2003.

[AK02] N. Alon and M. Krivelevich. Testing k-colorability. *SIAM Journal on Discrete Math*, 15(2):211–227, 2002.

[BEK+03] T. Batu, F. Ergun, J. Kilian, A. Magen, S. Raskhodnikova, R. Rubinfeld, and Rahul Sami. A sublinear algorithm for weakly approximating edit distance. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on the Theory of Computing*, pages 316–324, 2003.

[BFR+00] T. Batu, L. Fortnow, R. Rubinfeld, W. Smith, and P. White. Testing that distributions are close. In *Proceedings of the Forty-First Annual Symposium on Foundations of Computer Science*, pages 259–269, 2000.

[BLR93] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *JACM*, 47:549–595, 1993.

[BRW99] T. Batu, R. Rubinfeld, and P. White. Fast approximate PCPs for multidimensional bin-packing problems. In *Proceedings of RANDOM*, pages 245–256, 1999.

[COP03] M. Charikar, L. O'Callaghan, and R. Panigraphy. Better streaming algorithms for clustering problems. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on the Theory of Computing*, pages 30–39, 2003.

[CS02] A. Czumaj and C. Sohler. Abstract combinatorial programs and efficient property testers. In *Proceedings of the Forty-Third Annual Symposium on Foundations of Computer Science*, pages 83–92, 2002.

[DGL+99] 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.

[EKK+00] F. Ergun, S. Kannan, S. R. Kumar, R. Rubinfeld, and M. Viswanathan. Spot-checkers. *JCSS*, 60(3):717–751, 2000.

[Fis01] 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.

[FK99] A. Frieze and R. Kanan. Quick approximation to matrices and applications. *Combinatorica*, 19(2):175–220, 1999.

[FLN+02] E. Fischer, E. Lehman, I. Newman, S. Raskhodnikova, R. Rubinfeld, and A. Samrodnitsky. Monotonicity testing over general poset domains. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on the Theory of Computing*, pages 474–483, 2002.

[GGL+00] O. Goldreich, S. Goldwasser, E. Lehman, D. Ron, and A. Samordinsky. Testing monotonicity. *Combinatorica*, 20(3):301–337, 2000.

[GGR98] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *JACM*, 45(4):653–750, 1998.

[Gol98] O. Goldreich. Combinatorial property testing - a survey. In *Randomization Methods in Algorithm Design*, pages 45–60, 1998.

[GR00] O. Goldreich and D. Ron. On testing expansion in bounded-degree graphs. *Electronic Colloqium on Computational Complexity*, 7(20), 2000.

[HK03a] S. Halevy and E. Kushilevitz. Private Communications, 2003.

[HK03b] S. Halevy and E. Kushilevitz. Distribution-free property testing. In *Proceedings of RANDOM*, pages 302–317, 2003.

[KSS94] M. J. Kearns, R. E. Schapire, and L. M. Sellie. Toward efficient agnostic learning. *Machine Learning*, 17(2-3):115–141, 1994.

[Ron01] D. Ron. Property testing. In *Handbook on Randomization, Volume II*, pages 597–649, 2001.

[RS96] R. Rubinfeld and M. Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, 1996.

ECCC ISSN 1433-8092
http://www.eccc.uni-trier.de/eccc
ftp://ftp.eccc.uni-trier.de/pub/eccc
ftpmail@ftp.eccc.uni-trier.de, subject 'help eccc'