Instance Complexity and Unlabeled Certificates in the Decision Tree Model

Tomer Grossman* Ilan Komargodski† Moni Naor‡

Abstract

Instance complexity is a measure of goodness of an algorithm in which the performance of one algorithm is compared to others per input. This is in sharp contrast to worst-case and average-case complexity measures, where the performance is compared either on the worst input or on an average one, respectively.

We initiate the systematic study of instance complexity and optimality in the query model (a.k.a. the decision tree model). In this model, instance optimality of an algorithm for computing a function is the requirement that the complexity of an algorithm on any input is at most a constant factor larger than the complexity of the best correct algorithm. That is we compare the decision tree to one that receives a certificate and its complexity is measured only if the certificate is correct (but correctness should hold on any input). We study both deterministic and randomized decision trees and provide various characterizations and barriers for more general results.

We introduce a new measure of complexity called *unlabeled*-certificate complexity, appropriate for graph properties and other functions with symmetries, where only information about the structure of the graph is known to the competing algorithm. More precisely, the certificate is some permutation of the input (rather than the input itself) and the correctness should be maintained even if the certificate is wrong. First we show that such an unlabeled certificate is sometimes very helpful in the worst-case. We then study instance optimality with respect to this measure of complexity, where an algorithm is said to be instance optimal if for every input it performs roughly as well as the best algorithm that is given an unlabeled certificate (but is correct on every input). We show that instance optimality depends on the group of permutations in consideration. Our proofs rely on techniques from hypothesis testing and analysis of random graphs.

*Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100, Israel. Email: tomer.grossman@weizmann.ac.il.

†NTT Research. Email: ilan.komargodski@ntt-research.com. Parts of this work were done at Cornell Tech, supported in part by an AFOSR grant FA9550-15-1-0262, and at the Weizmann Institute of Science, supported in part by grants from the Israel Science Foundation and by a Levzion Fellowship.

‡Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100, Israel. Email: moni.naor@weizmann.ac.il. Supported in part by grant from the Israel Science Foundation (no. 950/16). Incumbent of the Judith Kleeman Professorial Chair.

Contents

1 Introduction

Worst-case analysis is the hallmark of theoretical computer science. An algorithm is evaluated based on its performance on the worst input and we want to find the algorithm whose performance is the best according to this criterion. Nevertheless, in some cases one might not be interested only in worst-case complexity of a function f. For instance, when the worst case is inherently "very bad" for all algorithms (and then measuring the algorithms by its worst case input gives meaningless comparisons), when the worst case is very rare, or when there is some prior information about the distribution of the inputs.

In this work we consider a new measure of complexity for problems we call *the instance complexity*. Here, we compare a given algorithm to one that has additional information about the input. That is, we are interested in the best algorithm for a given problem that performs as well as possible on every input compared to the best algorithm that *knows* (something about) the input. The instance complexity (denoted IC) of a problem is the overhead of solving the problem on every input vs. solving the problem on a given instance, provided that algorithms the specific instance is a correct algorithm (on all inputs). Here is one way to define this notion. Suppose that we have a collection of algorithms $\mathcal{A}$ that are correct on all on inputs.

Definition 1.1 (Instance complexity)**.** *Let f be a function, and A an algorithm that evaluates f. Let* $\mathrm{cost}(A, x)$ *be the cost of algorithm A on input x, where cost is some non-negative value that we want to optimize (e.g. runtime, space, etc.). We say that f is* instance optimizable *and that A is* instance optimal *if for every input x, and for every algorithm $A' \in \mathcal{A}$ (that evaluates f correctly on every input) the following holds:*

$$\mathrm{cost}(A, x) \in O\left(\mathrm{cost}(A', x)\right).$$

In the definition above, we compare the cost of A on every input with the cost of an algorithm that has the input hardwired. One can relax the hint given to the competing algorithm A' so that it only knows *something* about the input (e.g., an unknown permutation of the input, a subset of the coordinates, etc). Finally, note that both algorithms A and A' should be correct on all inputs, but the competing algorithm is evaluated in terms of cost *only when the hint it is given is correct w.r.t. the input.*

The term "Instance optimality" was coined by Fagin, Lotem and Naor [FLN03] in the context of finding items with the top k aggregate scores in a database of sorted lists. It has appeared in the theoretical computer science literature in several other contexts and forms. For previous works on instance complexity in other settings see Appendix B.

We initiate the systematic study of instance complexity in the *query* model (a.k.a. the *decision tree* model). In this model the input is given through an oracle, and the goal is to minimize the number of queries made to the input oracle. The query complexity model helps us understand the power of various resources (such as randomness and non-determinism) and inherent difficulties in computing a function. Not only is it a simple and clean model, but sometimes results in this model carry over to different models (i.e., "lifting" [RM99, GPW17]). See Section 2 for more details about the decision tree model, formal definitions, and known results.

We study two different notions of instance complexity. In the first notion the hint is simply the full input. The instance complexity, IC, of an algorithm is the maximal ratio between its complexity on a given input and the complexity of any other algorithm that *knows* the specific input (but is correct on all inputs). In the second notion the hint is some *unlabeled version of the input*; namely,

the input is given in an (adversarially) permuted order. This is relevant when the function we wish to compute treats the input as symmetric, e.g., as in graph properties.[1] We call this notion *unlabeled instance complexity*, uIC,[2] and for a given algorithm define it to be the maximal ratio between its complexity on a given input and the complexity of any other algorithm that *knows* an *unlabeled version of the input* (but is correct on all inputs).

Models for instance complexity. Instance optimality captures the fact that an algorithm is not only optimal for a worst-case input or a random one, but that it is optimal for *every* input (up to constant factors). In the context of decision tree complexity, we formalize this in the following way: An algorithm is instance optimal if for every input x its complexity is at most a constant factor larger than the complexity of any decision tree whose goal is to be as efficient as possible on x (yet correct on all inputs). This naturally leads to comparing the efficiency of the decision tree on an input to the complexity of a decision tree that gets that input as a certificate. Thus, a function is instance optimizable if there is an algorithm such that for **every** input (not just the worst case input) its complexity on that input is at most a constant factor larger than the certificate complexity of the function on the same input. To show that an algorithm is *not* instance optimal it is possible to argue about inputs where the running time is not the worst case.

For example, the parity function is instance optimal, since any algorithm, on any input must read the entire input (see Example 3.1 for detail). The OR function on the other hand is not instance optimal. This is because for any algorithm, A, there exists an input, x, and another algorithm A^*, such that A does n queries on input x while A^* does 1 query (see Example 3.3 for detail).

We wish to emphasize that instance optimality is a fundamentally different notion than worst case complexity. Notice that for the OR function, in the worst case the certificate complexity is $\Theta(n)$ and any algorithm without a certificate also requires $\Theta(n)$ queries yet the OR function is not instance optimal.

We define a similar notion for randomized algorithms. Namely, an algorithm is said to be randomized instance optimal if for every input x its expected complexity is at most a constant factor larger than the randomized certificate complexity of the function on x. We will consider randomized algorithms with two-sided error.

We introduce the notion of **unlabeled certificate complexity** in which the certificate is not the input, but a permutation thereof. This is relevant when the function is invariant under a *family of permutations* (e.g., a graph property or a symmetric function, such as threshold). We show that this is sometimes very helpful, *even in the worst-case*, in the sense that there are functions where the unlabeled certificate complexity is much smaller than the randomized complexity. Having this notion in hand, it is natural to define (randomized-) unlabeled instance optimality, where the standard (randomized) decision tree algorithm is required to have complexity (input-by-input) at most a constant factor larger than the best algorithm that gets a permutation (from the family) of the input as a certificate and computes the value of the function correctly. The algorithm that gets the unlabeled certificate is required to be correct on all inputs, but its complexity is measured only in case the certificate is correct.

[1] A graph property is a set of graphs closed under graph isomorphism; see Section 2.3.

[2] The closest notion considered in previous literature is in geometric algorithms by Afshani et al. [ABC17], see Appendix B.

1.1 Our Results

We lay the foundation for the study of instance optimality and unlabeled certificate complexity in the decision tree model by providing a host of results including various examples, relations, separations, and more. Our results fall short of a (complete) characterization of which functions are instance optimal and which are not and when unlabeled certificates help and when they do not. However, our results demonstrate that the theory of instance optimality and unlabeled complexity is rich, and contains non-trivial and sometimes surprising phenomena. Thus we believe that this theory deserves further research, and hope that this work will inspire such research.

Below, our results are stated in the order corresponding to the above models: deterministic and randomized instance optimality (Section 1.1.1), and then unlabeled certificate complexity (Section 1.1.2).

1.1.1 Deterministic and Randomized Instance Optimality

We start off by observing that even in the simplest setting of deterministic decision trees, instance optimality is a non-trivial property: there are functions that are instance optimizable and functions that are not. Also, monotonicity does not directly affect instance optimizability. Furthermore, we construct a function that is *strictly* instance optimal (*strict* means that the decision tree has no additional overhead over the one that has a certificate), but once composed with itself, results in a function that is not instance optimal.

Theorem 1.2 (Deterministic instance optimality)**.** *The following holds:*

1. *There is a monotone function that is* not *deterministic instance optimal while the Majority function is deterministic instance optimizable (and monotone).*
2. *The Addressing and Parity functions are examples of non-monotone functions that are strictly instance optimizable.*
3. *Among the monotone functions, the only ones that can be* strictly *instance optimizable are functions that depend on 0 or 1 variables.*
4. *The composition of two (strictly) instance optimal functions is not necessarily instance optimal.*

In the randomized case, the situation is somewhat different. While the parity and addressing functions behave similarly, the majority function becomes *not* instance optimizable! But, there *are* other instance optimizable monotone functions. Also, there are (non-monotone) graph properties that are instance optimizable. We further provide two results that capture a wide range of functions: (1) random functions are instance optimal and (2) a relation between instance optimizability and properties related to *sensitivity* (see Sections 3.2.1 and 3.2.2). As in the deterministic setting, the composition of instance optimal functions is not necessarily instance optimal.

One notable question left open is whether there is a *monotone* graph property that is randomized instance optimizable. We conjecture that such functions do not exist.

Theorem 1.3 (Randomized instance optimality)**.** *The following holds:*

1. *The Addressing and Parity functions are randomized instance optimizable functions.*

2. *The Majority function is* not *randomized instance optimizable. Thus, deterministic instance optimality does not imply randomized instance optimality. The other direction is true as well: There exists a function that is randomized instance optimizable but not deterministic instance optimizable (see Section 3.2.3).*

3. *There exists a* monotone *function that is randomized instance optimizable.*

4. *There exists a graph property that is randomized instance optimizable.*

5. *A random function is randomized instance optimizable with high probability.*

6. *The composition of two randomized instance optimal functions is not necessarily randomized instance optimal.*

We consider the computational complexity of figuring out whether a function is instance optimizable and whether a given algorithm is instance optimal. A good time complexity in this setting is polynomial in the size of the representation of the function, i.e. exponential in n for a Boolean function with input $\{0,1\}^n$. We do not have an algorithm for the former problem (deciding whether a function is instance optimizable), but for the latter we show that an algorithm can be *verified* to be instance optimal in time that is polynomial in the domain size of the problem. Naively, one has to work in time *exponential* in the domain size and go over all possible decision trees and test against each one (see Section 3.5).

Theorem 1.4 (Complexity of checking if an algorithm is instance optimal)**.** *Given a function f on n input variables (given by a truth-table of size 2^n), and a randomized algorithm T, there exists an algorithm U that decides in time $2^{O(n)}$ whether T is an instance optimal algorithm.*

We also consider *proximity property testing*[3] where the goal is to decide whether an object has some property or whether it is *far* from having it [GGR98, Gol17]. Here it is possible to characterize the properties that are randomized instance optimizable: it is exactly those that can be tested in $O(1)$ queries (see Section 3.6).

1.1.2 Unlabeled Certificate Complexity

We introduce the study of the **unlabeled certificate complexity**. We show that an unlabeled certificate can help significantly in the worst case for graph properties. That is, we construct a function for which an algorithm that has an isomorphic copy of the graph as a certificate outperforms an algorithm that isn't given any certificate.

Theorem 1.5 (Unlabeled certificate complexity)**.** *In both the randomized and deterministic setting, there exists a graph property for which the unlabeled certificate complexity is $O(n \log n)$ while any algorithm with no certificate requires $\Omega(n^2)$ queries.*

Having the notion of unlabeled certificates we define **unlabeled randomized instance complexity and optimality**. Here the situation changes again. Whereas the majority function is

[3]We use the term "proximity property testing" so as not to confuse the reader with the case where we are testing whether the given graph satisfies a property, rather than being close to a graph that satisfies the property.

not randomized instance optimizable, it *is* (almost) unlabeled randomized instance optimizable.[4] However, when viewed as a *graph property*, the majority function becomes *not* unlabeled randomized instance optimizable. The group of permutations we consider in the latter is the group of all isomorphic copies of the graph. In addition, we show that being a graph property is not directly related to unlabeled instance optimality. We say a function is **labelling instance optimal** if the unlabeled certificate performs as well as the labeled certificate for every input up to a constant. All instance optimal functions are also labelling instance optimal. We show that (up to log factors) the converse is not true: there exists a function which is within $O(\log n)$ of being labelling instance optimal, but not instance optimal in the normal sense.

Theorem 1.6 (Unlabeled instance optimality)**.** *The following holds:*

1. *The majority function is unlabeled randomized instance optimizable within* $O(\log \log n)$*.*
2. *The graph property of having more edges than non-edges is far from being unlabeled randomized-instance optimizable – it is at best within* $\tilde{\Omega}(n)$ *from being unlabeled randomized-instance optimizable, where* n *is the number of vertices in the graph.*
3. *The graph property of having at least one edge is unlabeled randomized instance optimizable and not labelling instance optimal.*
4. *There exists a function that is* $O(\log n)$ *away from being labelling instance optimal, but is not instance optimal in the normal sense.*

An impatient reader can skip ahead to Section 4, which is about unlabeled complexity, and contains the proofs of the items in Theorem 1.5 and Theorem 1.6.

The saga of the majority: As we shall see, the *majority* function behaves differently under different notions of instance optimality. While it is deterministic instance optimizable, it is *not* randomized instance optimizable (Lemma 3.5). Having only a permutation of the input as a certificate does not provide additional power to the one receiving it, i.e., it is (almost) unlabeled randomized instance optimizable (Lemma 4.4). On the other hand, viewing majority as a graph property, getting a permutation of the graph may help and majority becomes a function that is *not* unlabeled randomized instance optimizable (Lemma 4.5).

2 The Query Model and Instance Optimality

The query model is one of the simplest computational models and has been studied widely. See, for example, Buhrman and de Wolf [BdW02], Jukna [Juk12, Chapter 14] and O'Donnell [O'D14, Chapter 8.6] for a survey. We give some standard definitions of *decision trees* and several related complexity measures such as *deterministic*, *randomized* and *certificate* complexity, and then briefly recall the known connections between them.

A decision tree is an algorithm for computing the value of a function on an a-priori unknown input. Let $f\colon \{0,1\}^n \to \{0,1\}$ be a Boolean function over a set of n input variables $x_1, \ldots, x_n$. A

[4] In this result, the optimality ratio is a super-constant function ($\log \log$) of the input size. In such cases, where the optimality ratio is a super constant function $g(n)$ of the input size, we will say that the function is *within* $g(n)$ *of being (unlabeled /randomized-) instance optimizable.*

deterministic decision tree T over a set of input variables $x_1, \ldots, x_n$ is a rooted binary tree. Every internal vertex is labeled with an input variable. The leaves of the tree are labeled by 0 or 1, and every internal vertex has one outgoing edge labeled by 0 and the other by 1. The computation of the tree is done in the natural way from the root to the leaves according to the assignment of the input variables. We denote by $T(x)$ the output of the decision tree T on input x. We say that the decision tree T computes f if for any input x it holds that $f(x) = T(x)$.

Denote by $\mathcal{T} = \mathcal{T}_n$ the set of all decision trees on functions of n inputs. The complexity (i.e., cost) of a decision tree $T \in \mathcal{T}$ on input x, denoted by $\mathrm{D}(T, x)$, is the length of the path from the root to the leaf corresponding to x, or in other words, the number of queries made by the algorithm to the input x. For a function f, denote by $\mathcal{T}_f$ the set of all decision trees in $\mathcal{T}$ that compute f:

$$T \in \mathcal{T}_f \iff \forall x \colon T(x) = f(x).$$

The following two definitions deal with the worst-case deterministic and certificate decision tree complexity measures. Roughly speaking, the deterministic complexity measures the smallest possible cost over all decision trees that compute f.

Definition 2.1 (Worst-case deterministic complexity)**.** *The* ***worst-case deterministic*** *decision tree complexity of* $f \colon \{0,1\}^n \to \{0,1\}$*, denoted by* $\mathrm{D}(f)$*, is*

$$\mathrm{D}(f) = \min_{T \in \mathcal{T}_f} \max_{x \in \{0,1\}^n} \mathrm{D}(T, x).$$

The certificate complexity measures the minimal number of bits a decision tree has to query in order to *verify* the output of the function on a specific input (rather than to compute it). That is, we still want the decision tree to compute f correctly on all inputs, but we measure its complexity only with respect to a fixed input.

Definition 2.2 (Worst-case certificate complexity)**.** *The* ***certificate*** *complexity of* f *on input* x *is*

$$\mathrm{C}(f, x) = \min_{T \in \mathcal{T}_f} \mathrm{D}(T, x),$$

The ***worst-case certificate*** *complexity of* f *is* $\mathrm{C}(f) = \max_x \mathrm{C}(f, x)$*.*

Throughout this paper we use the term "decision tree" and "algorithm" synonymously. Since in the decision tree model we are only interested in the number of queries made by a given algorithm, the two terms are equivalent. We also use the terms "certificate complexity", "queries made by an algorithm with a certificate" and "queries made by any algorithm" synonymously. "Certificate complexity" and "an algorithm with a certificate" are synonyms since when arguing that the certificate complexity of a given function is some value, q one has to prove that some algorithm with access to a certificate only has to make q queries. This is the same as "queries made by any algorithm" since the certificate complexity is the decision tree with the lowest complexity, that is the algorithm that makes fewest queries.

We proceed with the randomized variants of the above two definitions. In randomized complexity it is common to talk about zero-error (Las Vegas), one-sided and two-sided (BPP style) algorithms. We will discuss only the latter for simplicity. There are two (equivalent) ways to view a *randomized* decision tree: one as a tree that has vertices that are coin flips. The other, which we will mostly use, as a probability distribution Δ over the set of all deterministic decision trees

$\mathcal{T}$. Given an input x, the algorithm first samples a decision tree from the distribution and then executes it. The randomized decision tree complexity of a distribution (i.e., a randomized decision tree) Δ on input x, denoted by $\mathrm{R}(\Delta, x)$, is the expected number of queries made by the chosen tree on input x.

A distribution Δ is *correct* for a function f if for any input x, a randomly chosen decision tree $T \leftarrow \Delta$ outputs the value $f(x)$ with probability $2/3$ over the choice of T.[5] We denote by $\mathbf{\Delta}_f$ the set of all distributions Δ that are correct for f:

$$\Delta \in \mathbf{\Delta}_f \iff \forall x\colon \Pr_{T \leftarrow \Delta}[T(x) = f(x)] \geq 2/3.$$

Analogously to the deterministic setting, the worst-case randomized complexity measures the smallest possible (expected) cost over all randomized decision trees that compute f.

Definition 2.3 (Worst-case randomized complexity)**.** *The* ***worst-case randomized*** *decision tree complexity of f, denoted by $\mathrm{R}(f)$, is*

$$\mathrm{R}(f) = \min_{\Delta \in \mathbf{\Delta}_f} \max_x \underbrace{\mathop{\mathbb{E}}_{T \leftarrow \Delta}[\mathrm{D}(T, x)]}_{\mathrm{R}(\Delta, x)}.$$

For randomized certificate complexity, we still want the decision tree to be correct on every input with high probability, but the complexity is measured only on a fixed input.

Definition 2.4 (Worst-case randomized certificate complexity)**.** *The* ***randomized certificate*** *complexity of a function f on input x is*

$$\mathrm{RC}(f, x) = \min_{\Delta \in \mathbf{\Delta}_f} \mathop{\mathbb{E}}_{T \leftarrow \Delta}[\mathrm{D}(T, x)],$$

The worst-case randomized certificate complexity of f is $\mathrm{RC}(f) = \max_x \mathrm{RC}(f, x)$.

2.1 Instance Complexity and Optimality

Adapting the general notion of instance optimality (see Definition 1.1) to the setting of query complexity is done by fixing a class of query algorithms (say, deterministic or randomized ones) and then asking whether there exists an algorithm in this class that is "the best" on every input. Such an algorithm must have complexity at most a constant factor larger than the complexity of every other algorithm on *every* input. In particular, it has to be at most a constant factor worse than an algorithm that is designed for any specific x^*, and if it finds any inconsistencies between the given input x and x^*, reads the whole input. This leads us to the observation that a sufficient and necessary condition for instance optimality of a function in the query model is the existence of a deterministic (resp. randomized) decision tree whose complexity is at most a constant factor larger than the deterministic (resp. randomized) *certificate* decision complexity of the function per input.

The strongest notion one could hope for is what we call *strict* instance optimality, where the certificate does not help, even in low order constant factors (i.e., the optimality ratio is 1 and there is no additive term). More precisely, there exists a deterministic decision tree such that on every input x the decision tree complexity on x is upper bounded by the certificate decision tree complexity of f on input x.

[5] The value of $2/3$ is arbitrary, since we can get any constant by amplification (i.e., by sampling several decision trees from Δ, executing them and taking the majority result).

Definition 2.5 (Strict optimality)**.** *A function f is* strict *instance optimizable if there exists a deterministic decision tree T such that for every input x:*

$$\mathrm{D}(T, x) = \mathrm{C}(f, x).$$

Being optimal on every input without any overhead is a very strong requirement (and indeed not many non-trivial functions satisfy it; see below), so we relax it by allowing multiplicative and additive slack in the overhead for the decision tree without the certificate. For this we need to talk about a sequence of functions.

Definition 2.6 (D-Instance optimality)**.** *A sequence of functions $f = \{f_n\}_{n\in\mathbb{N}}$ is* D-instance optimizable *if there exists a sequence of deterministic decision trees $T = \{T_n\}_{n\in\mathbb{N}}$, where each $T_n \in \mathcal{T}_{f_n}$, and (universal) constants $c_1, c_2 \geq 0$ such that on every input $x \in \{0,1\}^n$, it holds that*

$$\mathrm{D}(T_n, x) \leq c_1 \cdot \mathrm{C}(f_n, x) + c_2.$$

Definition 2.7 (R-Instance optimality)**.** *A sequence of functions $f = \{f_n\}_{n\in\mathbb{N}}$ is* R-instance optimizable *if there exists a sequence of distributions over decision trees $\Delta = \{\Delta_n\}_{n\in\mathbb{N}}$, where each $\Delta_n \in \boldsymbol{\Delta}_{f_n}$, and (universal) constants $c_1, c_2 \geq 0$ such that on every input $x \in \{0,1\}^n$, it holds that*

$$\mathrm{R}(\Delta_n, x) \leq c_1 \cdot \mathrm{RC}(f_n, x) + c_2.$$

We refer to the *constant factor* c_1 above as the *optimality ratio*. The above deterministic (resp. randomized) algorithm is called *instance optimal*.

While we mostly talk about being within a constant of the best on any instance we will also consider being within some function $g(n)$. If we have an algorithm T such that for $g(n)$ for every input $x \in \{0,1\}^n$ it hold that

$$\mathrm{D}(T, x) \leq g(n) \cdot \mathrm{C}(f_n, x),$$

then we say that f is within g of being instance optimizable.

2.2 Unlabeled Complexity and Optimality

We introduce a new complexity measure for decision trees that we call *unlabeled certificate complexity*. Roughly speaking, the unlabeled complexity is the amount of queries an algorithm that is given some *permutation* of the certificate needs to perform.

This notion only makes sense for functions that are invariant under some permutation group Γ, such as symmetric properties or graph properties. In the case of symmetric functions, we consider the group of all permutations over the inputs, and in the case of graph properties, we focus on the group of permutations over the vertices (i.e., all isomorphic copies of the graph). We denote by Γ the group of permutations.

The idea behind the notion is to model the situation where the algorithm has a lot of knowledge about the input, such as the graph structure (e.g. the degree sequence), but not the actual labels of the graph. We refer to a decision tree that gets such a certificate as an *unlabeled certificate* decision tree.

As before, we expect an unlabeled decision tree T to correctly compute f on every input, but we will measure its complexity over all inputs that are consistent with the certificate under the given group of permutations.

Definition 2.8 (Worst-case unlabeled certificate complexity)**.** *The* ***unlabeled certificate*** *complexity of a function f that is invariant under Γ on input x is*

$$\mathrm{AC}(f,\Gamma,x) = \min_{T\in\mathcal{T}_f}\max_{\pi\in\Gamma}\mathrm{D}(T,\pi(x)).$$

The ***worst-case certificate complexity*** *of f under Γ is $\mathrm{AC}(f,\Gamma) = \max_x \mathrm{AC}(f,\Gamma,x)$.*

For randomized unlabeled certificate complexity, the definition is analogous but we allow the decision tree to be randomized and to err.

Definition 2.9 (Worst-case randomized unlabeled certificate complexity)**.** *The* ***randomized unlabeled certificate*** *complexity of a function f invariant under Γ on input x is*

$$\mathrm{RAC}(f,\Gamma,x) = \min_{\Delta\in\boldsymbol{\Delta}_f}\max_{\pi\in\Gamma}\mathop{\mathbb{E}}_{T\leftarrow\Delta}[\mathrm{D}(T,\pi(x))],$$

The ***worst-case randomized unlabeled certificate complexity*** *of f under Γ is $\mathrm{RAC}(f,\Gamma) = \max_x \mathrm{RAC}(f,\Gamma,x)$.*

It follows immediately that the unlabeled certificate complexity of a function is lower bounded by its certificate complexity and upper bounded by its deterministic complexity. That is, for any Γ such that f is invariant under Γ we have

$$\mathrm{C}(f) \le \mathrm{AC}(f,\Gamma) \le \mathrm{D}(f).$$

The same holds in the randomized case:

$$\mathrm{RC}(f) \le \mathrm{RAC}(f,\Gamma) \le \mathrm{R}(f).$$

Throughout the paper we let the default Γ be the set of all graph isomorphisms. That is, if our function is a graph property, and Γ is not specified, then Γ is the set of all graph isomorphisms.

Given this new notion of unlabeled certificates one can define instance optimality with respect to it. Here, we are trying to compete with the best algorithm that is given some permutation of the certificate, rather than the exact certificate as in the definitions of Section 2.1. If an algorithm is instance optimal in this sense, this means that even the knowledge about the graph structure does not help improve the performance.

Namely, we can ask whether having access to a permutation of a certificate reduces the complexity by more than a constant factor compared to a decision tree that gets no certificate at all.

Definition 2.10 (Unlabeled instance optimality)**.** *A sequence of functions $f = \{f_n\}_{n\in\mathbb{N}}$ invariant under a sequence of family of permutations $\{\Gamma_n\}_{n\in\mathbb{N}}$, is* unlabeled D-instance optimizable *(resp.* unlabeled R-instance optimizable*) if there exists a sequence of deterministic (resp. randomized) decision trees $T = \{T_n\}_{n\in\mathbb{N}}$, where each $T_n \in \mathcal{T}_{f_n}$, and constants $c_1, c_2 \ge 0$ such that on every input x, it holds that*

$$\mathrm{D}(T,x) \le c_1\cdot\mathrm{AC}(f_n,\Gamma_n,x) + c_2 \qquad \textit{(resp. } \mathrm{R}(T,x) \le c_1\cdot\mathrm{RAC}(f_n,\Gamma_n,x) + c_2\textit{)}$$

We can also compare the labeled certificate to the unlabeled certificate. A function is labelling instance optimal if the number of queries required to evaluate the function given an unlabeled certificate is at most a constant times the number queries required to evaluate the function with a labeled certificate.

Definition 2.11 (labelling instance optimality)**.** *A sequence of functions $f = \{f_n\}_{n\in\mathbb{N}}$ invariant under a sequence of family of permutations $\{\Gamma_n\}_{n\in\mathbb{N}}$, is* labelling D-instance optimizable *(resp.* labelling R-instance optimizable*) if there exists a sequence of deterministic (resp. randomized) decision trees $T = \{T_n\}_{n\in\mathbb{N}}$, where each $T_n \in \mathcal{T}_{f_n}$, and constants $c_1, c_2 \geq 0$ such that on every input x, it holds that*

$$\mathrm{AC}(T, x) \leq c_1 \cdot \mathrm{C}(f_n, \Gamma_n, x) + c_2 \qquad \textit{(resp. } \mathrm{RAC}(T, x) \leq c_1 \cdot \mathrm{RC}(f_n, \Gamma_n, x) + c_2\textit{)}$$

2.3 Additional Definitions & Known Relations

Graph properties. In some of our result we will be interested in graph properties. A graph property is a set of graphs closed under graph isomorphism. That is Ψ is a graph property if for every graph $G = (V, E)$ and every permutation π over V, it holds that $G \in \Psi$ if and only if $\pi(G) \in \Psi$, where $\pi(G) = (V, E')$ and $E' = \{(\pi(u), \pi(v)) \mid (u, v) \in E\}$).

Sensitivity. Two central notions in the analysis of Boolean functions are the *sensitivity* and *block sensitivity*, introduced by Cook, Dwork and Reischuk [CDR86] and Nisan [Nis91], respectively. They both measure how sensitive a function f is to changes in its input. Let $f\colon \{0,1\}^n \to \{0,1\}$ be a function and $x \in \{0,1\}^n$ be an input. The *sensitivity* of f on x, denoted $\mathsf{s}(f, x)$, is the number of bit positions i such that $f(x) \neq f(x \oplus e_i)$, where $e_i \in \{0,1\}^n$ is a vector that is 1 at coordinate i and 0 everywhere else. The sensitivity of f, denoted by $\mathsf{s}(f)$, is $\max_x \mathsf{s}(f, x)$.

Block sensitivity is a useful generalization of sensitivity. The *block sensitivity* of f on x, denoted $\mathsf{bs}(f, x)$, is the maximum number t such that there is collection $B = \{B_1, \ldots, B_t\}$ of *disjoint* subsets of $[n]$ with $f(x) \neq f(x \oplus B_i)$ for all $i \in [t]$. The overall block sensitivity $\mathsf{bs}(f)$ of f is the maximum of $\mathsf{bs}(f, x)$ over all x. Note that sensitivity is just block sensitivity where the blocks are restricted to be of size 1.

It is not hard to see that $\mathsf{s}(f) \leq \mathsf{bs}(f) \leq \mathrm{C}(f)$. The biggest known gap between $\mathsf{s}(f)$ and $\mathsf{bs}(f)$ is quadratic (see Rubinstein [Rub95] and Ambainis and Sun [AS11]). Showing that this gap is at most polynomial has been a major open problem for many years, until it had been resolved recently by [Hua19], who showed that for all f, $\mathsf{bs}(f)$ is bounded by $\mathsf{s}(f)^4$.

More relations. In general, all worst-case complexity measures discussed above are known to have a polynomial relationship and figuring out the precise relationships is a major research program.

Proposition 2.12. *For every Boolean function f we have:*

1. $\mathrm{RC}(f) \leq \mathrm{C}(f) \leq \mathrm{D}(f) \leq n$ *and* $\mathrm{R}(f) \leq \mathrm{D}(f)$.
2. $\mathrm{C}(f) \leq \mathsf{s}(f) \cdot \mathsf{bs}(f)$ *(see [Nis91] and [BdW02, Theorem 2]).*
3. $\mathrm{D}(f) \leq \mathsf{s}(f) \cdot \mathsf{bs}(f)^2 \leq \mathsf{bs}(f)^3$ *(see [BdW02, Corollary 1]).*
4. $\mathsf{bs}(f) \leq \mathsf{s}(f)^4$ *(see [Hua19]).*
5. $\mathrm{D}(f) \leq \mathrm{C}(f)^2$ *(see [BI87, Tar89] and, for example, [Juk12, Theorem 14.3]).*
6. $\mathsf{bs}(f) \leq 3\mathrm{RC}(f) \leq 3\mathrm{R}(f)$ *(see [Nis91] and [Aar08]) and* $\mathsf{bs}(f) \leq \mathrm{D}(f)$.

Proposition 2.13. *For every* monotone *Boolean function f it holds that* $\mathrm{C}(f) = \mathsf{s}(f) = \mathsf{bs}(f)$ *(see [Nis91] [BdW02, Proposition 3]) and hence* $\mathrm{D}(f) \leq \mathsf{s}(f)^2$.

Yao's Minimax Principle. When proving lower bounds on the randomized query complexity of a function f it is often easier to rephrase the problem by applying Yao's [Yao77] Minimax Principle (see [MR97] Chapter 2). In words, it says that in order to prove a lower bound on the randomized decision tree complexity, it is enough to come up with one particularly bad distribution over *inputs* which fools all deterministic algorithms. Often Yao's Minimax principle is used by saying that if t expected queries are required to distinguish between two input distributions with high probability (where the function outputs 1 where the input is chosen from one distribution but when the input is chosen from the second distribution the function outputs 0), then any deterministic algorithm must perform at least t queries.

.

3 Properties Of Instance Optimal Functions

This section is devoted to showing several results regarding properties of instance optimal function in both the deterministic and randomized setting. We prove theorem 1.2 and theorem 1.3, as well as showing, given a truth table representing a function, and an algorithm, how to test if the algorithm is instance optimal in time polynomial in the size of the truth table. Lastly we give a characterization of which functions are R-instance optimal in the proximity property testing model.

3.1 Deterministic Instance Optimality

We show that there are functions which are D-instance optimizable and there are functions which are not. We also prove that there are no non-trivial monotone functions whose optimality ratio is 1 (i.e., *strict* D-instance optimizable). The arguments in this section are simple and provide good examples for the notions discussed.

Let $\mathrm{Parity}\colon \{0,1\}^n \to \{0,1\}$ be the parity function on n variables, defined as $\mathrm{Parity}(x) = \bigoplus_{i=1}^{n} x_i$. Assume that n is a power of 2 and let $\mathrm{Addr}\colon [n] \times \{0,1\}^n \to \{0,1\}$ be the addressing function defined as $\mathrm{Addr}(i, x_1, \ldots, x_n) = x_i$. Both of these functions are D-instance optimizable. In both cases the argument is that on all instances the certificate complexity has the same value (n and $1 + \log n$, respectively) and it is possible to achieve it with a straightforward deterministic algorithm.

Example 3.1 (Item 2 of Theorem 1.2)**.** *The* Parity *function and the* Addr *function are strict D-instance optimizable.*

Proof. The proof for both Parity and Addr being strict D-instance optimizable rely on a common statement that follows from the definitions of deterministic and certificate complexity: given a function $f\colon \{0,1\}^n \to \{0,1\}$, for any decision tree $T \in \mathcal{T}_f$ and any input $x \in \{0,1\}^n$, it holds that $\mathrm{D}(T,x) \geq \mathrm{C}(f,x)$. For the Parity and Addr functions the straightforward trees $T_{\mathrm{Parity}} \in \mathcal{T}_{\mathrm{Parity}}$ (that reads the whole input) and $T_{\mathrm{Addr}} \in \mathcal{T}_{\mathrm{Addr}}$ (that reads the address part plus the resulting address) satisfy $D(T_{\mathrm{Parity}}, x) = n$ and $D(T_{\mathrm{Addr}}, x) = \log n + 1$ for every $x \in \{0,1\}^n$. Thus, it remains to prove that $\mathrm{C}(\mathrm{Parity}, x) \geq n$ and $\mathrm{C}(\mathrm{Addr}, x) \geq \log n + 1$. We do this by showing that any decision tree (that is always correct) must make at least n queries for Parity and at least $\log n + 1$ queries for addressing. We show that for any algorithm that makes less queries, we can change bits that the algorithm does not query, and this will change the value of the function, thus making the algorithm err.

For Parity, consider an algorithm that on some input x queries less than n input locations. Since there is a location that the algorithm does not query and the function's output depends on all locations, by flipping the value of an unqueried bit, the output of the function must change, but the algorithm cannot notice this and hence it is wrong either on x or on x with the flipped bit.

For Addr, consider an algorithm that on some input x reads $k \leq \log n$ values. These k bits can either fully fix the index i, but then there are two possible values for the bit in the i-th position (which determine the value of the function) that the algorithm cannot distinguish. The other case is that the algorithm did not query the value of the index i in full. Denote by $\ell > 0$ the number of bits of i that the algorithm did not query. There are 2^ℓ possibilities for the value of i. The algorithm can query only $k - (\log n - \ell)$ locations out of the possible n. Since $k - (\log n - \ell) < 2^\ell$, there must be a way to assign values to the unread location so as to fool the algorithm to output the wrong answer. □

We show that there is a *monotone* function that is D-instance optimizable (unlike the parity and addressing functions that are not monotone). However, the function is not *strict* D-instance optimizable. The function is the majority function $\mathrm{Maj}\colon \{0,1\}^n \to \{0,1\}$ defined as

$$\mathrm{Maj}(x_1, \ldots, x_n) = 1 \iff \sum_{i=1}^{n} x_i \geq \frac{n}{2}.$$

Example 3.2 (Item 1(i) of Theorem 1.2)**.** *The majority function is D-instance optimizable.*

Proof. In order to determine the output of Maj on an input $x \in \{0,1\}^n$, every decision tree has to find (at least) $n/2+1$ locations (in x) that have the same value, 0 or 1. Since the algorithm is deterministic, this can take up to n queries in the worst case. On the other hand, an algorithm with a certificate for x can query exactly $n/2+1$ locations that have the same value. Any algorithm that has a certificate for x that queries at most $n/2$ queries can be fooled to output the wrong answer. We get that for every input x it holds that $C(\mathrm{Maj}, x) = n/2+1$ and $D(\mathrm{Maj}, x) \leq n$, as needed. □

To complete the picture, we give a (monotone) function that is not D-instance optimizable.

Example 3.3 (Item 1(ii) of Theorem 1.2)**.** *The OR function (that outputs '1' iff the Hamming weight of the input is at least 1) is not D-instance optimizable.*

Proof. An algorithm with a certificate can make just one query on every input with Hamming weight 1 (to a location where there is supposedly a '1'). However, every decision tree (not getting a certificate) that is correct on every input must query all the coordinates on one of these inputs, as otherwise we can fool it to output the wrong answer. This is done by planting '0's at the points it queries and a '1' at a point that it does not query. □

Strict instance optimality. Both the addressing and parity functions were shown to be *strict* D-instance optimizable (the algorithm with the certificate could not outperform the one without the certificate even by a multiplicative small constant term). On the other hand, in our analysis of the instance optimality of the majority function, we showed that an algorithm with a certificate may perform twice as fast than the one that does not have the certificate. Namely, the optimality ratio of the majority function is 2.

These examples are no coincidence: No non-trivial (that is not constant or dictatorship) *monotone* function can be strictly D-instance optimizable.

Lemma 3.4 (Item 3 of Theorem 1.2)**.** *There are no* strict *D-instance optimizable* monotone *functions except those that depend on* 0 *or* 1 *variables.*

Proof. Consider the DNF of the function i.e. the OR of all the minimal terms that make the function '1'. Suppose that the function has only one term. It cannot be of length 1, since the function does not depend on one variable. Therefore, if all the variables in the term but one are set to '1', the certificate size is 1, but for any decision tree all the variables in the term must be read and if the one set to '0' is the last one, then the complexity on that input is greater than 1.

So suppose that there are at least two terms in the conjunction. Due to the monotonicity of the function they are not on exactly the same sets of variables. Consider the two instances, one satisfying the first term and the other satisfying the second one (and nothing else). A certificate verifier just needs to access the variables of one term, but a deterministic (or even probabilistic) will have to ask on a variable that appears in one term but not the other and hence will not have the same complexity and the instance satisfying the other term. □

3.2 Randomized Instance Optimality

The question we consider is whether randomness changes anything w.r.t. instance optimality. It is not hard to see that the parity function and the addressing function are not only D-instance optimizable but they are also R-instance optimal (the proof of Example 3.1 actually shows it). However, the situation changes for the majority function: we show that, in contrast to Example 3.2, the majority function is *not* R-instance optimizable. We complement the picture in this section with an example of a *monotone* function that is also R-instance optimal.

Lemma 3.5 (Item 2 of Theorem 1.3, first part)**.** *The majority function is* not *R-instance optimizable.*

Proof. Given that the deterministic certificate size of majority is always of size $\Theta(n)$, we must use the fact that the algorithm may err. We will give a distribution $\mathcal{D}$ on the inputs such that any algorithm that does not get a certificate (but may be tailored to inputs that come from the distribution $\mathcal{D}$) must query $\Omega(n)$ bits in order to be correct with probability at least $2/3$. On the other hand, for any input x from this distribution $\mathcal{D}$, an algorithm with a certificate for x only needs to query $O(\sqrt{n})$ of the bits to be correct with high probability.

For $x \in \{0,1\}^n$ let $wt(x)$ stands for the Hamming weight of x. The hard distribution on inputs $x \in \{0,1\}^n$ is to choose uniformly at random from the set $wt(x) \geq n/2 + \sqrt{n}$ and the set $wt(x) \leq n/2 - \sqrt{n}$ (in the former Maj is 1 and in the latter it is 0). Consider a randomized decision tree for Maj that is correct on this distribution with probability at least $2/3$, i.e. distinguishes between the case that $wt(x) \geq n/2 + \sqrt{n}$ and the case that $wt(x) \leq n/2 - \sqrt{n}$ with probability at least $2/3$.

We show that this task requires $\Omega(n)$ queries on a uniform input of Hamming weight $n/2 + \sqrt{n}$. First, by symmetry, notice that the best strategy for the algorithm is to sample random locations in x and query them. Second, the lower bound follows from the fact that $\Omega\left(1/\varepsilon^2\right)$ samples are needed to distinguish (with constant probability) a biased random coin that outputs heads with probability $1/2 + \varepsilon$ from one that outputs heads with probability $1/2 - \varepsilon$.[6]

[6] See, for example, Claim 5.6 here: http://www.tau.ac.il/~mansour/advanced-agt+ml/scribe5-lower-bound-MAB.pdf (Accessed: June 2018).

On the other hand, given a certificate for an x of Hamming weight at least $n/2+\sqrt{n}$, there is a randomized algorithm that makes $q = 10\sqrt{n}$ queries and outputs the right answer with probability at least $2/3$. The algorithm, given a certificate $x^* \in \{0,1\}^n$ and an input $x \in \{0,1\}^n$, does:

1. Query x at q random locations i where x^*_i is 1.
2. If all queries return 1, output 1.
3. Otherwise, read the whole input and output the majority value.

The algorithm makes a mistake only if at Step 2 all queries returned '1', but the Hamming weight of x was less than $n/2$. This can happen only if there is a set $I \subseteq [n]$ of coordinates of size at least $|I| \geq \sqrt{n}$ such that $x^*_i = 1$ but $x_i = 0$ for every $i \in I$, *and* the algorithm did not query at any of them. The probability that this happens is at most

$$\left(1 - \frac{\sqrt{n}}{n}\right)^q = \left(1 - \frac{1}{\sqrt{n}}\right)^{10\sqrt{n}} \leq 1/3.$$

Thus, this algorithm outputs the correct result with probability at least $2/3$, as needed. □

We give an explicit monotone function that is R-instance optimizable. We call this function the *lexicographic threshold* function.

Lemma 3.6 (Item 3 of Theorem 1.3)**.** *There exists an explicit monotone function that is R-instance optimizable.*

Proof. Consider the lexicographic order of strings in $\{0,1\}^n$ under $\geq_{\text{lex}}$. Let $b = 1010\ldots1010 \in \{0,1\}^{2n}$ and define $\text{LTF}_n \colon \{0,1\}^{2n} \to \{0,1\}$ as $\text{LTF}_n(x) = 1$ if and only if $x \geq_{\text{lex}} b$. Denote by $b_{[i]}$ the substring $b_1 \ldots b_i$ where b_i is the i-th bit of b from left to right.

We show that the randomized query complexity of LTF is roughly equal to the randomized query complexity of LTF given access to a certificate for the input. That is, we show that any randomized query algorithm for LTF that has access to a certificate for the input must work (roughly) as hard as an algorithm that does not have access to a certificate.

Our algorithm, that we denote by P_n, is to read bit by bit from left to right. Denote by $x_{[i]}$ the substring read by the algorithm (ordered left to right) until iteration i. Initially, $x_{[0]} = \perp$ and $i = 0$. At iteration $1 \leq i \leq 2n-1$, the algorithm returns 1 if $x_{[i]} >_{\text{lex}} b_{[i]}$, returns 0 if $x_{[i]} <_{\text{lex}} b_{[i]}$ and continues to the next iteration otherwise (i.e., if $x_{[i]} =_{\text{lex}} b_{[i]}$). If the algorithm reaches $i = 2n$, it returns 1 (since the last bit of b is 0). Note that the algorithm is deterministic. The number of queries that this algorithm does depends on the input and is $2n-1$ in the worst case.

We prove that for every input x it holds that $\mathbb{E}[P_n(x)] \leq 2\text{RC}(\text{LTF}, x)$.

For any x let $\mathsf{pref}(x)$ be the length of the prefix of x that algorithm P_n reads before it stops. In other words, it is the length of the prefix of x that agrees with the string b. Consider a pair of locations $2i-1$ and $2i$ that are smaller than $\mathsf{pref}(x)$. If the probability that A reads at least one of them is smaller than $2/3$, then algorithm A may err with probability at least $1/3$. If instead of input x it is given input x' that is flipped in locations $2i-1$ or $2i$, then the value of $\text{LTF}_n(x') \neq \text{LTF}_n(x)$, but A distinguishes the two inputs with probability less than $2/3$. □

Remark 3.7 (Quantum instance optimality)**.** *In the quantum case, the Parity function is* not *instance optimal. Based on Grover's algorithm [Gro96], it is known that* quantum *certificate complexity is exactly the square root of randomized certificate complexity (up to a constant factor). An explicit statement appears in Aaronson [Aar08], where this claim is shown to be true on an input-by-input basis. On the other hand, computing Parity requires $n/2$ queries (quantumly) as shown by Farhi et al. [FGGS98] and Beals et al. [BBC+01].*[7]

3.2.1 Sensitivity and Instance Optimality

We show that a function for which the decision tree complexity for every input is (roughly) equal to its sensitivity, then the function is instance optimizable. This is not true in the opposite direction.

Lemma 3.8. *Let f be an n-input Boolean function. If there is a (randomized) decision tree for f s.t. for any x the complexity on input x is $\Theta(\mathsf{s}(f,x))$, then this algorithm is instance optimal.*

Proof. Let f be an n-input Boolean function. Assume that there is a decision tree algorithm (randomized or deterministic) whose complexity for every input x is $O(\mathsf{s}(f,x))$. For the deterministic case we show that any algorithm A that does not err must query $\mathsf{s}(f,x))$ location on x: suppose that there is an index i such that $f(x) \neq f(x \oplus e_i)$ that A does not query x_i. Then, on input $x \oplus e_i$ the algorithm A makes the same decision as on input x and thus must err.

For the randomized case, we show that any algorithm (that does not err with probability larger than $1/3$ on any input) must query at least as many points in expectation as $2\mathsf{s}(f,x)/3$ on input x before deciding the output of the function.

Consider two input distributions: one with the point x and the other one is on $\mathsf{s}(f,x)$ points of the form $x \oplus e_i$ for a random $i \in [n]$ such that $f(x) \neq f(x \oplus e_i)$. If an algorithm makes at most j queries, then the probability of distinguishing the two distributions is at most $j/\mathsf{s}(f,x)$. That is, the output distribution of such an algorithm when given one or the other distribution can differ by at most $j/\mathsf{s}(f,x)$. Therefore, if the expected number of queries the randomized algorithms makes on x is less than $2\mathsf{s}(f,x)/3$, then the expected difference is less than $2/3$ which means that the algorithm will err with probability at least $1/3$. □

From this lemma we can obtain many results concerning the instance optimality of many algorithms. For instance, we can recover our results for the parity, addressing and lexicographic threshold functions.

The lemma actually holds for block-sensitivity as well (with essentially the same proof). But this still does not give a tight characterization, as there are functions where the block-sensitivity is lower than the randomized certificate complexity (by some polynomial factor) [Aar08, GSS16].

3.2.2 Random Functions are Instance Optimizable

Consider the uniform distribution of Boolean functions, i.e. selecting at random one from the set of size 2^{2^n} functions $\{0,1\}^n \to \{0,1\}$. A random Boolean function $f\colon \{0,1\}^n \to \{0,1\}$ is instance optimizable with probability $1 - 2^{-cn}$ for some constant c and the naive algorithm that reads the *entire input* is an instance optimal algorithm. A proof follows.

What we need to show is that for such a random function on *all* inputs any algorithm must read a large fraction of the bits.

[7] We thank Scott Aaronson for bringing these results to our attention.

For input $x \in \{0,1\}^n$ consider all $\binom{n}{2}$ elements at (Hamming) distance 2. We have that except with probability $\ll 2^{-n}$ we will have at least $\binom{n}{2}/4$ values x' of distance 2 from x such that $f(x) \neq f(x')$.

Consider first a deterministic algorithm for x that queries k locations. If the algorithm is not to err, then it must cover all those x''s in the sense that it queries at least one of the two bits where x and x' differ. Now if it queries only $k \in o(n)$ locations it can cover at most $\binom{k}{2} + k(n-k)$ such x', but this is much smaller than $\binom{n}{2}/4$.

To get a lower bound for a randomized algorithm, consider the set of locations that have probability at least $1/8$ to be queried. If this set does not cover an x' at distance 2 from x where $f(x') \neq f(x)$, then we have that the algorithms errs with probability at least $3/4$ when the input is x'. So again we get that this set should be large and the expected number of location queried is $\Theta(n)$. So, altogether, we get that on all inputs x an algorithm tailored for x must ask $\Omega(n)$ queries and therefore the naive algorithm is instance optimal. This proves Item 5 of Theorem 1.3.

Remark 3.9. *It would have been nice if we could show the above claim (that random functions are instance optimizable) by showing that such functions have high block sensitivity* on any input*. But note that to use the union bound we need to get probability less than 2^{-n} and that just won't work. Instead, we gave a direct proof.*

3.2.3 A Separation From Deterministic Instance Optimality

As we have seen, deterministic instance optimality does not imply randomized instance optimality (e.g., the Majority function, Lemma 3.2 and 3.5). There are several examples for the other direction, utilizing cases where there is a difference between the worst case deterministic and randomized complexities. In particular the following example:

Lemma 3.10 (Item 2 of Theorem 1.3, second part)**.** *There exists a function that is R-instance optimal but not D-instance optimal.*

Proof. Let $g(x_1)$ (where $|x_1| = n$) be a function for which $\mathrm{D}(g) \in \Omega(n)$, $\mathrm{R}(g) \in o(n)$ and $\mathrm{C}(g) \in o(n)$ in the worst case. Such functions are known, for example Saks and Wigderson [SW86] or Ambainis et al. [ABB+17]. Let Parity_ℓ be the parity function of all the bits of the input string of length ℓ. Let x be the concatenation of x_1 and x_2, and let $f(x) = g(x_1) \oplus \mathrm{Parity}_\ell(x_2)$ where we set $\ell = |x_2| = \mathrm{R}(g)$. For this length we have $\mathrm{C}(\mathrm{Parity}, x_2) = \mathrm{R}(g)$ on all inputs x_2.

Regarding the deterministic complexity of f, we have that $\mathrm{D}(f) = \mathrm{D}(g) + \mathrm{D}(\mathrm{Parity}_\ell) \geq \mathrm{D}(g) \in \Omega(n)$. Similarly $\mathrm{C}(f,x) = \mathrm{C}(g,x_1) + \mathrm{C}(\mathrm{Parity}_\ell, x_2) \in o(n)$ and thus on the worst case input, the certificate outperforms the deterministic algorithm, and thus f is not D instance optimal.

On the other hand, in the randomized case: let Δ_g be the best randomized algorithm to evaluate g in the worst case. Let $\Delta_{\mathrm{Parity}_\ell}$ be the algorithm that evaluates Parity_ℓ by simply querying the entire input. Let Δ_f be the algorithm that evaluates f by doing both Δ_g and$\Delta_{\mathrm{Parity}_\ell}$. We have, for any x, $\mathrm{R}(\Delta_f, x) = \mathrm{R}(\Delta_g, x_1) + \mathrm{R}(\Delta_{\mathrm{Parity}_\ell}, x_2) \in O(\mathrm{R}(g))$. Similarly $\mathrm{RC}(f,x) = \mathrm{RC}(g,x_1) + \mathrm{RC}(\mathrm{Parity}_\ell, x_2) > \mathrm{RC}(\mathrm{Parity}_\ell, x_2) \in \Omega(R(g))$ and thus f is R-instance optimal.

□

3.3 Instance Optimality and Graph Properties

In this section we study the instance optimizability of a specific set of functions, ones that test graph properties.[8] Our motivation comes from our example in Lemma 3.6 of a monotone function which is R-instance optimizable: the function is very "far" from being a graph property as the location of each bit greatly influences its effect on the outcome of the function. This raises the question of whether non-symmetry is necessary for R-instance optimizability of monotone functions.

As a first step, we ask the following question: if we restrict our attention to graph properties, then is it possible that having a certificate *always* "helps"? In Lemma 3.11 we show that it is not the case. Specifically, we give an explicit graph property that is instance optimal. The graph property that we use is the *scorpion* property. An n vertex graph $G = (V, E)$ is a *scorpion* graph if it contains 3 special vertices: a vertex b (body) of degree $n-2$, a vertex t (tail) of degree 2 and a vertex s (sting) of degree 1. The tail is adjacent to both b and s. Edges in between the remaining $n-3$ vertices in $V \setminus \{b, s, t\}$ may be present or not.

Lemma 3.11 (Item 4 of Theorem 1.3)**.** *The scorpion graph property is R-instance optimal.*

Proof. We first show that a deterministic algorithm can test whether a given graph is a scorpion graph with $O(n)$ queries. This is a result of Best et al. [BvEBL74]. We provide a proof of this in Appendix A for completeness.

Claim 3.12 ([BvEBL74])**.** *Testing whether a given n vertex graph is a scorpion graph takes at most $O(n)$ queries.*

To complete the proof we show that on any instance G the complexity is $\Omega(n)$:

Claim 3.13. *For any graph G the best randomized algorithm on G that is correct on all inputs with probability at least $2/3$ must make $\Omega(n)$ queries in expectation.*

Proof. Consider first a graphs that is a scorpion. The sensitivity of such a graph is at least $n-1$: there is a unique vertex that is the body (of degree $n-2$). If any of its $n-2$ adjacent edges is missing, then the graph ceases to be a scorpion. So as in the proof of Lemma 3.8 any correct algorithm must make $\Omega(n)$ queries in expectation. Consider now a graph that is not a scorpion. We partition its vertices into $n/3$ groups of triples and know that if for any one of the triples we completely change the neighborhood the triple can become the b, t and s of the scorpion graph (and the corresponding graph will satisfy the property). So consider the following two distributions on graphs: one is simply G the other one starts with G, picks a triple at random and changes the neighborhoods so they become the body, tail and sting. Differentiating between these two distributions implies that the number of triples touched (i.e. at least one edge in their neighborhood is queried) is close to $n/3$, but since each edge belongs to at most two triples we get that $\Omega(n)$ edges must be queried in expectation. □

□

[8] We consider finite and undirected graphs with neither self loops nor parallel edges. A graph property is an invariant that depends only on the abstract structure of the graph (and not on specific graph representations). See Section 2.3.

3.3.1 Conjecture: monotone graph properties are not instance optimizable

One of the most famous conjectures in the literature of query complexity is the *evasiveness conjecture* (a.k.a. the Aanderaa-Rosenberg conjecture). Roughly speaking, the conjecture says that for any non-trivial graph property any deterministic decision tree algorithm must query at least a constant fraction of all the edges (in the worst case). This conjecture was resolved by Rivest and Vuillemin [RV75]. In another version, called the *Aanderaa-Karp-Rosenberg conjecture* (AKR conjecture), the specific constant is also conjectured to be 1. This was resolved for graphs of prime power order (number of vertices) by Kahn, Saks and Sturtevant [KSS84].

Yao [Yao77, §4, Question (2)] asked whether an analogue of the Rivest-Vuillemin result holds for monotone graph properties and *no-error* randomized algorithms. Namely, whether a constant fraction of edges must be queried in expectation for any non-trivial monotone graph property by any randomized decision tree algorithm. The first result related to this question was of Yao [Yao91] who showed a lower bound of $\Omega(n \cdot \log^{1/12} n)$. This was improved by King [Kin91] and then by Hajnal [Haj91] to $\Omega(n^{5/4})$ and $\Omega(n^{4/3})$, respectively. The currently best lower bound is $\Omega(n^{4/3} \cdot \log^{1/3} n)$ of Chakrabarti and Khot [CK07].

If the algorithm is allowed to make two-sided errors the best lower bound we are aware of is that of Jain and Zhang [JZ11]. One could also pose a conjecture analogous to Yao's for such algorithms:

Conjecture 3.14. *Any randomized decision tree algorithm has to query a constant fraction of edges for any non-trivial monotone graph property P, even if it is allowed to make an error with probability $1/3$. That is, $R(P)$ is $\Theta(n^2)$ for any non-trivial monotone graph property P on n-vertex graphs.*

We make a conjecture regarding instance optimality of graph properties, namely that every *monotone* graph property is not R-instance optimizable. That is, a randomized algorithm that gets a certificate will always be better than any randomized algorithm (that has no certificate) on *some* input. The examples we gave in previous sections do not rule out this conjecture: the scorpion property is not monotone, the lexicographic threshold function is not a graph property, and the majority function (which can be phrased as a graph property) is not R-instance optimizable. Recall that the majority function is D-instance optimal, so the conjecture is false for deterministic computation.

Conjecture 3.15. *Every non-trivial monotone graph property is not R-instance optimizable.*

We do not know what is the relationship between Conjectures 3.14 and 3.15.

3.4 Composition of Instance Optimal Functions

Given boolean function $f\colon \{0,1\}^n \to \{0,1\}$ and $g\colon \{0,1\}^m \to \{0,1\}$. The composition of f and g, denoted $f \circ g\colon \{0,1\}^{nm} \to \{0,1\}$, is defined as the value of $f(g(\vec{x_1}), ..., g(\vec{x_n}))$ where each $\vec{x_i}$ is an independent vector of m bits. Most classical notions in query complexity (such as deterministic and certificate complexities, degree, sensitivity and more) are known to behave well under composition (see for example Tal [Tal13, Lemma 3.1]). In this section we show that instance optimality, on the other hand, does not compose. That is, the composition of two instance optimal functions is not necessarily instance optimal.

Lemma 3.16 (Item 4 of Theorem 1.2 and Item 6 of Theorem 1.3)**.** *There exist two functions that are both strictly D-instance optimal and R-instance optimal, but their composition results in a function that is neither R nor D instance optimal.*

Proof. Let $f\colon \{0,1\}^n \to \{0,1\}$ have input of the form: $(a_1, b_1), ..., (a_{n/2}, b_{n/2})$. The function f outputs b_i where i is the first index for which $a_i = 1$. We also define $a_{n/2} = 1$, regardless of the true input. The lemma follows immediately from the claims below.

Claim 3.17. *The function f is strictly D-instance optimal.*

Proof. Consider the naive algorithm that queries each a_i in order until it finds an i for which $a_i = 1$ and then queries b_i. WLOG suppose the naive algorithm outputs 0. Suppose towards a contradiction that there exists an algorithm A that makes at most k queries for some input X, while the naive algorithm makes $k+1$ queries. We will show that by changing the values of bits that A did not query we can make A output the wrong value. If A makes at most k queries one of two things must be true:

1. For some $i < k$, A does not query any one of a_i and b_i.
2. A doesn't query a_k or A doesn't query b_k. If Item 1 does not occur, then this must occur (since if for every $i < k$, A queries either a_i or b_i, then querying both a_k and b_k would require $k+1$ total queries).

For the first case, if $a_i = b_i = 1$ then A makes an error. For the second case, we have two options: either A queries a_k or A queries b_k. Suppose A queries a_k. Note that $a_k = 1$ since the naive algorithm makes $k+1$ queries. If $b_k = 1$ then A has made an error. For the second option, suppose A queries b_k. If $a_k = 0$ but $a_{k+1} = b_{k+1} = 1$ then A has made an error. Note that the maximum number of queries the naive algorithm makes is $k + 1 = \frac{n}{2}$ since we know automatically that $a_{n/2} = 1$. □

Claim 3.18. *The function f is R-instance optimal.*

Proof sketch. This is very similar to the proof of Lemma 3.6, that the lexicographic function is R-instance optimal. Essentially, any algorithm that is correct on 2/3 of the input must query the a_i's in order, since otherwise the input can be set to fool the algorithm. □

Claim 3.19. *For any algorithm $\Delta \in \mathbf{\Delta}_{f\circ f}$ (i.e., that computes $f \circ f$ on all inputs correctly with probability 2/3) there exists an input, x, for which $\mathrm{C}(f \circ f, x) \in O(n)$ while $\mathrm{R}(\Delta, x) \in \Omega(n^2)$.*

The above claim implies that instance optimality does not compose in both the randomized and the deterministic case. This is true since our upper bound uses a deterministic decision tree while the lower bound is for any randomized decision tree.

Proof. Consider $f' \circ f''$. where both f' and f'' are f as defined above. Denote the input of f' by $[a_1^{\mathsf{out}}(\vec{x}_{1,1}), b_1^{\mathsf{out}}(\vec{x}_{1,2})], ..., [a_{n/2}^{\mathsf{out}}(\vec{x}_{n/2,2}), b_{n/2}^{\mathsf{out}}(\vec{x}_{n/2,2})]$, and similarly the input of each f'' by $\vec{x}_{i,j} = (a_1^{\mathsf{in}}, b_1^{\mathsf{in}}), ..., (a_{n/2}^{\mathsf{in}}, b_{n/2}^{\mathsf{in}})$. Note that a_i^{in} and b_i^{in} consists of a single bit. On the other hand $a_i^{\mathsf{out}}(\vec{x}_{i,1})$ and $b_i^{\mathsf{out}}(\vec{x}_{i,2})$ are functions on n bits.

Consider an input of the following form: for every i, $\vec{x}_{i,2}$ has $a_1^{\mathsf{in}} = 1$ (and thus for all i, $b_i^{\mathsf{out}}(\vec{x}_{i,2})$ can be computed in $O(1)$). Suppose that for all $i \neq \frac{n}{2}$, $\vec{x}_{i,2}$ has $b_1^{\mathsf{in}} = 0$, (and thus for all $i \neq \frac{n}{2}$, $b_i^{\mathsf{out}}(\vec{x}_{i,2}) = 0$), and $\vec{x}_{n/2,2}$ has $b_1^{\mathsf{in}} = 1$ (and thus $b_{n/2}^{\mathsf{out}}(\vec{x}_{n/2,2}) = 1$). Suppose that for every i, $\vec{x}_{i,1}$ has $a_k^{\mathsf{in}} = 0$ for every $k \neq \frac{n}{2}$, and $a_{n/2}^{\mathsf{in}} = 1$ (and thus for all i, computing $a_i^{\mathsf{out}}(\vec{x}_{i,1})$ requires $\Omega(n)$ queries). Suppose there exists a single i^*, which is chosen uniformly at random from $[1, \frac{n}{2} - 1]$, such that $\vec{x}_{i^*,1}$ has $b_{n/2}^{\mathsf{in}} = 1$ (and thus $a_{i^*}^{\mathsf{out}}(\vec{x}_{i^*,1}) = 1$). Similarly, suppose $\vec{x}_{n/2,1}$ has $b_{n/2}^{\mathsf{in}} = 1$ (and

thus $a^{\mathsf{out}}_{n/2}(\vec{x}_{n/2,1}) = 1$). For all $i \neq i^*, \frac{n}{2}$ suppose $\vec{x}_{i,1}$ has $b^{\mathsf{in}}_{n/2} = 0$ (and thus for $i \neq i^*, \frac{n}{2}$, we have $a^{\mathsf{out}}_i(\vec{x}_{i,1}) = 0$).

Consider the following certificate algorithm: First the algorithm determines all of $b^{\mathsf{out}}_i(\vec{x}_{i,2})$. To do this, for every $\vec{x}_{i,2}$ the algorithm queries a^{in}_1, b^{in}_1. If for any $i \neq \frac{n}{2}$, either $a^{\mathsf{in}}_1 \neq 1$ or $b^{\mathsf{in}}_1 \neq 0$ then the input doesn't match the certificate (and in this case the algorithm can query everything, and we don't care about the number of queries). This requires $O(n)$ queries, since for every $b^{\mathsf{out}}_i(\vec{x}_{i,2})$ the algorithm makes two queries and there are $O(n)$ such b^{out}_i. Since for all $i < \frac{n}{2}$, $b^{\mathsf{out}}_i(\vec{x}_{i,2}) = 0$, if the certificate algorithm finds a single $i < \frac{n}{2}$ for which $a^{\mathsf{out}}_i(\vec{x}_{i,1}) = 1$ then the algorithm with a certificate knows that the output is 0. Thus the algorithm with a certificate can query all of the bits of $\vec{x}_{i^*,1}$ in $O(n)$ queries. If the certificate is truthful all of the $a^{\mathsf{in}}_i = 0$, except for $i = \frac{n}{2}$ for which $a^{\mathsf{in}}_{n/2} = 1$ and $b^{\mathsf{in}}_{n/2} = 1$, and thus $a^{\mathsf{out}}_{i^*}(\vec{x}_{i^*,1}) = 1$ (once again, if the queries do not match the certificate, then the certificate can query everything, and the number of queries is irrelevant). Thus the certificate algorithm outputs 0 in $O(n)$ queries.

On the other hand any randomized algorithm without a certificate will take $\Omega(n^2)$ queries. Suppose that for every i the randomized algorithm is given the value of each $b^{\mathsf{out}}_i(\vec{x}_{i,2})$, without any queries. The output of the function is '1' iff there exists an $i < \frac{n}{2}$ for which $a^{\mathsf{out}}_i(\vec{x}_{i,1}) = 1$. That is, the algorithm needs to output the OR function of $\frac{n}{2} - 1$ many $a^{\mathsf{out}}_i(\vec{x}_{i,1})$. In order to compute a single $a^{\mathsf{out}}_i(\vec{x}_{i,1})$ correctly with probability $\frac{2}{3}$ any randomized algorithm must make at least $k \in \Omega(n)$ queries due to Claim 3.18. That is $k = cn$ for some constant c. Consider any randomized algorithm that makes at most $\frac{cn^2}{10}$ queries. This algorithm, can determine at most $\frac{cn^2/10}{cn} = \frac{n}{10}$ many $a^{\mathsf{out}}_i(\vec{x}_{i,1})$ correctly with probability $\frac{2}{3}$ (since the input $\vec{x}_{i,1}$ and $\vec{x}_{i',1}$ are independent for each i and i'). Suppose the algorithm has determined $\frac{n}{10}$ many $a^{\mathsf{out}}_i(\vec{x}_{i,1})$ correctly with probability 1 (which is more than what any randomized algorithm can do). Since the single i^* that satisfies $a^{\mathsf{out}}_{i^*}(\vec{x}_{i^*,1}) = 1$ is distributed uniformly at random among all the possible $\frac{n}{2} - 1$ many i's, and since any randomized algorithm knows the value of at most $\frac{n}{10}$ many $a^{\mathsf{out}}_i(\vec{x}_{i,1})$, and since $\frac{n}{10} < \frac{2}{3}(\frac{n}{2} - 1)$, any randomized algorithm that makes at most $\frac{cn^2}{10}$ will succeed with probability less than $\frac{2}{3}$.

□

□

3.5 Testing Instance Optimality

In this section, we deal with the computational complexity of the problem of constructing and verifying instance optimal algorithms. The input is a function, i.e., the truthtable and the output is a decision tree.

We give an algorithm[9] U (for *universal*) that for a function f and an instance x, outputs an *optimal* randomized certificate tree for f on x. Thus, if one wants to test whether a given decision tree T is instance optimal for a problem f, then it is enough to compare its complexity on input x to the complexity of the output of U on f and input x for every $x \in \{0,1\}^n$. A different interpretation of U is that if one wants to prove that f is instance optimizable, then one has to give a decision tree that performs just as well as the decision tree generated by U for every input.

[9]Throughout the paper the term "algorithm" was used synonymously with the term "decision tree". In this section the term "algorithm" has a different meaning. Specifically, in this section the model the algorithm works in is RAM rather than the query model.

The running time of U for a problem f and input x is polynomial in the domain size of f and thus the total running time of testing whether a decision tree is instance optimal is a polynomial in the domain size of f

Theorem 3.20 (Theorem 1.4 rephrased)**.** *There exists an algorithm U that gets as an input a function $f\colon \{0,1\}^n \to \{0,1\}$, and a certificate y for the input to the function, and generates in time $2^{O(n)}$ a correct randomized decision tree T_y for f such that $\mathrm{R}(\mathrm{T}_y, y) \in O(\mathrm{RC}(f,y))$.*

Proof. Let $f\colon \{0,1\}^n \to \{0,1\}$ be a Boolean function and $y \in \{0,1\}^n$ be a certificate for the input. For a subset $S \subseteq [n]$ let $y_S \in \{0,1\}^n$ be the incidence vector of S, i.e., a string whose i-th bit is:

$$(y_S)_i = \begin{cases} y_i & \text{if } i \notin S \\ \bar{y_i} & \text{if } i \in S \end{cases}$$

The algorithm U is given as Algorithm 1. On a high level, it uses the truth table of f to construct a linear program whose variables are all the possible inputs to f and the output is a set of n numbers $p_1, \ldots, p_n \in [0,1]$ that correspond to the probability that the certificate verification tree should query the ith input bit.

Each such bit should be queried independently and the result is a non-adaptive verifier. The algorithm takes exponential time (its input is a function f whose description may be exponential and it solves an exponentially large linear program) but the representation of the decision tree is small—just a set of probabilities $\{p_i\}_i$.

1: **procedure** $U(f,y)$
2: Solve the linear program

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} p_i \\ \text{subject to} \quad & 0 \le p_i \le 1 \ , \forall 1 \le i \le n \\ & \sum_{i \in S} p_i \ge 1 \ , \forall S \subseteq [n] \text{ such that } f(y_S) \neq f(y) \end{aligned}$$

3: **end procedure**

The certificate verification algorithm is:

1: Repeat c times:
2: **for** $1 \le i \le n$ **do**
3: query x at location i with probability p_i.
4: **if** the result is equal to y_i **then** continue.
5: **else** query x at all locations and output $f(x)$.
6: **end if**
7: **end for**
8: output $f(y)$.

Algorithm 1: Randomized Decision Tree Generation Algorithm

First note that there is always a solution for the linear program by setting all the p_i's to 1.

We begin by proving that Algorithm 1 satisfies correctness. Namely, that the resulting decision tree computes f correctly on every input (with sufficiently high probability). If $y = x$, then clearly the decision tree generated by Algorithm 1 outputs the correct value with probability 1 (since $f(x) = f(y)$). Otherwise, assume that $y \neq x$. Either the decision tree finds the differing point or not. In the former, it outputs $f(x)$ as needed. In the latter, there are two cases again: either $f(x) \neq f(y)$ or $f(x) = f(y)$. In the latter, we are done again by construction.

We are left with calculating the probability that the differing point is not recognized in case $f(x) \neq f(y)$. Let $S = \{i_1, \ldots, i_k\} \subseteq [n]$ be a subset of size $k \leq n$ of indices such that $x_i \neq y_i$ for $i \in S$. Algorithm 1 is wrong only if it does not query on any x_i such that $i \in S$. But since $f(y) \neq f(y_S)$ (since $y_S = x$) Algorithm 1 ensures that $\sum_{i \in S} p_i \geq 1$. Let $q_i = 1 - p_i$ (i.e., q_i is the probability that the decision tree does not query on x_i) and the probability of the algorithm being wrong is $\prod_{i \in S} q_i$. By the inequality of arithmetic and geometric means, we get that

$$\prod_{i \in S} q_i \leq \left(\frac{\sum_{i \in S} q_i}{k} \right)^k \leq \left(\frac{k-1}{k} \right)^k \leq e^{-1} < 0.4.$$

Hence, Algorithm 1 succeeds with probability at least 0.6. By standard amplification (running the algorithm repeatedly c times and outputting $f(y)$ only if there was never a disagreement), we get an algorithm whose success probability is at least $2/3$.

To prove that Algorithm 1 is optimal in terms of the number of queries it makes, let T' be a randomized algorithm that computes f. Assume towards contradiction that $\mathrm{R}(\mathrm{T}', x) \in o(\mathrm{R}(\mathrm{T}, x))$, where T is the randomized decision tree generated by Algorithm 1. We will reach a contradiction by giving an input x' to T' for which it must error with probability at least $1/3$. By definition, $\mathrm{R}(\mathrm{T}, x) = \sum_{i=1}^{n} p_i$ and denote by p'_i the probability that T' queries x_i (on input x). Since $\mathrm{R}(\mathrm{T}', x) \in o(\mathrm{R}(\mathrm{T}, x))$, there exists some $S \subseteq [n]$ for which $\sum_{i \in S} p'_i \in o(1)$ and $f(x) \neq f(x')$ where $x' = x_S$. This is true since otherwise the p'_i's (times some constant) would be a solution to the linear program. Clearly, T' computes f correctly on x' with probability at most $o(1)$, contradicting the assumption. □

The conclusion is therefore that when given a function and a decision tree, in order to check whether the decision tree is instance optimal (with a given constant and some slackness) one may go over all inputs, for each one compute the certificate complexity as in Algorithm 1 and compare it to the complexity of the decision tree on that input. So only time proportional to polynomial in the truth table size is required.

Remark 3.21. *Aaronson [Aar08, Lemma 5] showed that there is a non-adaptive verifier whose query complexity is similar to the best adaptive verifier. While Aaronson starts with the adaptive verifier and defines from it a non-adaptive one, we construct the non-adaptive verifier directly from the function.*

We leave open the question of the complexity of testing whether a given function is instance optimizable.

Question 3.22. *What is the complexity of testing whether a given function f is instance optimizable (within some c).*

3.6 Instance Optimality of Proximity Property Testing

Consider instance optimality in *proximity-to-property testing*: the task in proximity property testing, as defined by Goldreich, Goldwasser and Ron [GGR98] (see [Gol17] for a current survey), is to decide whether an object has some property or whether it is *far* from having it (according to some measure of distance). This is actually a function with "don't cares": if the object is close to satisfying the property but does not satisfy it, then any answer is acceptable; in this sense it is different than the other functions considered in the paper which are full domain.

There are several examples and characterizations of problems that can be tested within query complexity that only depends on the proximity parameter (see, for example, [GGR98, AFNS09, GR10]). In this case, when the proximity parameter is constant, the number of queries is also constant which means that every such property is "trivially" instance optimizable: both the tester that has a certificate and the tester that does not have it can decide the property within a constant number of queries.

We observe that these properties, namely the ones that are testable within a constant number of queries, are *the only properties that are R-instance optimizable*: the tester that gets a certificate can always efficiently check (using $O(1/\delta)$ queries sampled at random) that its certificate is $\delta/2$-close to the object. If the certificate satisfies the property or if it $\delta/2$ close to satisfying the property, then it outputs "accept". Otherwise it outputs "reject". Here, we are using the fact that we are allowing two-sided errors.

4 Unlabeled Certificates

In this section we switch gears and study the notion of *unlabeled certificates* (see Section 2.2 for formal definitions). In the previous section the competing algorithm received as a certificate the entire input (this can be though of as an "untrusted hint" where the hint is the entire input), whereas in this section the certificate is a *permutation of the input* (the "untrusted hint" is a permutation of the input, where the permutation is chosen adversarially). The unlabeled certificate makes sense especially for functions that are symmetric under some class of permutations. For instance, if the input is a graph property, the unlabeled certificate will be an isomorphic copy of the graph. That means that the algorithm should always be correct (even if the unlabeled certificate is not an isomorphic copy of graph), but the runtime of the algorithm is only measured when the unlabeled certificate is indeed isomorphic to the graph.

We study both the worst case complexity and the instance complexity of unlabeled certificates. For the worst case, roughly speaking, suppose we have a graph property, and suppose we are given an isomorphic copy of the graph as a certificate. It turns out that there exists a function for which such a certificate helps almost as much as an unlabeled certificate can possibly help. See Section 4.1 for more details.

In the case of instance complexity of unlabeled certificates, we say a function is unlabeled instance optimal if there exists an algorithm that performs as well as the unlabeled complexity (up to a multiplicative constant) for every input. We show that the group of permutations considered is important. Specifically, the majority property is randomized unlabeled instance optimal (where the unlabeled certificate is simply the hamming weight, since the function is symmetric under any permutation of the input), whereas the graph property of having more edges than non edges (now the group of allowed permutations are all isomorphisms of the graph) is far from being unlabeled instance optimal. See Section 4.2 for more details.

4.1 Unlabeled Certificates In the Worst Case

4.1.1 The Power of Unlabeled Certificates

We construct an example of a function (graph property) for which an algorithm that receives an isomorphic copy of the graph as a certificate outperforms any algorithm that has no certificate, thus showing a separation between algorithms receiving unlabeled certificates and algorithms with no certificate in the worst case. The separation is almost the largest possible.

Theorem 4.1 (Theorem 1.5 rephrased)**.** *There exists a graph property f for which* $\mathrm{AC}(f) \in O(n \log n)$ *while* $\mathrm{R}(f) \in \Omega(n^2)$.

That is, we show a function f for which $\mathrm{R}(f) \in \widetilde{\Omega}(\mathrm{AC}(f)^2)$. Since $\mathrm{D}(f) \geq \mathrm{R}(f)$, and $\mathrm{AC}(f) \geq \mathrm{RAC}(f)$, we also know that there exists a function where $D(f) \in \widetilde{\Omega}(\mathrm{AC}(f)^2)$ and $\mathrm{R}(f) \in \widetilde{\Omega}(\mathrm{RAC}(f)^2)$. Up to log factors, this example is tight in the deterministic setting since $\mathrm{D}(f) < \mathrm{C}(f)^2$ (see for example [Juk12, Theorem 14.3]). In the randomized setting, the best known separation between $\mathrm{R}(f)$ and $\mathrm{RC}(f)$ is $\mathrm{R}(f) \in O(\mathrm{RC}(f)^2)$, and improving this separation is a major open research problem. In fact, if $\mathrm{D}(f) \in O(\mathsf{bs}(f)^2)$ then for all total functions f, $\mathrm{R}(f) \in O(\mathrm{RC}(f)^2)$, in which case our example is also tight in the randomized setting, up to log factors.

Our starting point in the construction is the property that every vertex has degree at least $\log n$. If one has a labeled certificate of the graph, then for each vertex it is easy to check $\log n$ neighbors and see that they indeed exist. But the problem with an unlabeled certificate is that there is no clear way to figure out who the vertices in the graph correspond to in the certificate and hence finding the $\log n$ neighbors might be lengthy and require $\Omega(n)$ queries per vertex. To overcome this we add some gadgets in order to uniquely define the vertices in a manner that allows to find their "identities" with relatively few queries.

Construction 4.2. *The function outputs '1' iff all of the following conditions holds:*

1. *There are* exactly 2 *vertices of degree* 1, T_1 *and* T_2*, both of which have an edge to a special vertex, denote this vertex by* P.

2. *There is a vertex of degree* $n-3$ *that is adjacent to all vertices except* T_1 *and* T_2*. Call this vertex* B.

3. *Vertex* P *is the root of a "binary tree" of size* $10 \log n$*. Every "right" child consists of an intermediate vertex. See Figure 1 where such a tree is shown. All vertices in the tree are also adjacent to* B*, this is not shown in the figure. Condition 4 is also shown in the figure.*

4. *There are* 2 *extra vertices, both adjacent to* P*'s right child, and one of which is also adjacent to* P*'s left child. This makes both of* P*'s immediate children also have degree* 5*. See Figure 1.*

5. *All the* $10 \log n$ *leaves have degree different than* 5*. This makes* P *and its immediate children the only vertices of degree* 5.

6. *We call the set of remaining vertices* ($n - O(\log n)$ *in number) the "crowd" (denoted by* C*). The neighbor of every vertex in the crowd either also belongs to the crowd, or is a leaf of the tree described in Condition 3 or is the vertex* B *from Condition 2. Furthermore, all vertices*

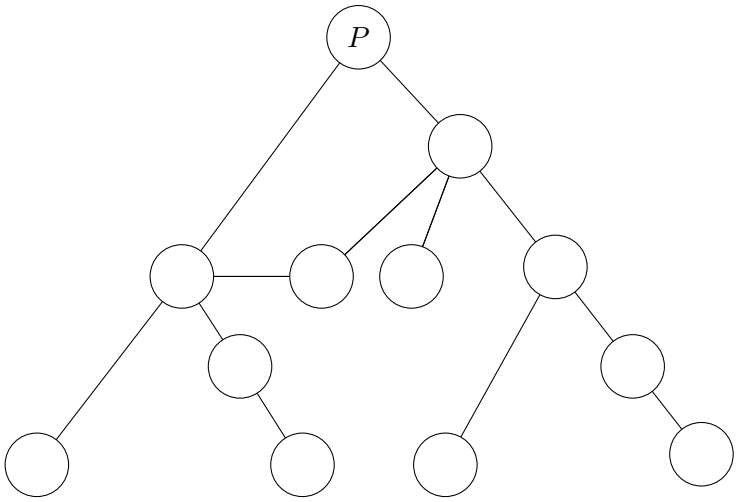

Figure 1: Example of binary tree illustrating Conditions 3 and 4.

in the crowd must have degree at least $\log n$ *on the subgraph induced by* C *(that is, every vertex in* C *must have at least* $\log n$ *neighbors also in* C*).*[10]

7. *No two vertices in the crowd have the same neighborhood when restricted to the leaves of the tree. Note that this is possible since we have more than* $\log n$ *leaves.*

See Figure 2 for an illustration.

Proof. Note that if the function outputs 1, then T_1 and T_2 are the only vertices of degree 1. Thus, once we find T_1 or T_2 we can also find P in $O(n)$ queries by finding T_1's (or T_2's) only neighbor. If we know which vertex is B, then in $O(n)$ queries we can find T_1 and T_2, as they are the only vertices not adjacent to B. Once we find P we can check if Condition 1 is met by querying all of P's neighbors, and for each one making $O(n)$ queries to see if their only neighbor is P. If only 2 of P's 5 neighbors have this property then Condition 1 is met. Similarly once we find P, T_1 and T_2 we can check in $O(n)$ queries that Condition 2 is met by checking that one of P's neighbor has degree $n-3$ and is not adjacent to T_1 and T_2 as a neighbor. We can similarly check that Condition 4 is met in $O(n)$ queries given P. If we find a vertex, v, of degree 5 we can find P in $O(n)$ queries by checking if v has exactly two neighbors of degree 5, if it does then $v = P$, otherwise $v \neq P$ since P has 2 neighbors of degree 5, and these two neighbors don't have an edge between them.

The first phase of the algorithm is to find out either T_1, T_2 or B, and afterwards in $O(n)$ queries we also find P. We do this without using the unlabeled certificate similar to the way we check if a graph is a scorpion [BvEBL74], see Appendix A. Start by picking an arbitrary vertex, v, and querying all of its neighbors. If $\deg(v) \in \{1, 5, n-3\}$ we are done, as v must be T, P (or one of P's immediate children) or B respectively. If $\deg(v) > n-3$ then output 0 since then we can't have two vertices of degree 1. So assume $\deg(v) \notin \{1, 5, n-3, n-2, n-1\}$. Let N_0 denote the set of all of v's neighbors, and let $N_1 = \overline{N_0} \setminus \{v\}$. Notice that if the function outputs 1 then $B \in N_0$ and $T_1, T_2 \in N_1$ since T_1 and T_2 are only adjacent to P, and $v \neq P$. Furthermore $P \in N_1$ because all

[10] This condition can be replaced with any condition that can be solved in $O(n \log n)$ queries with a labeled certificate, but requires $\Omega(n^2)$ queries without a certificate.

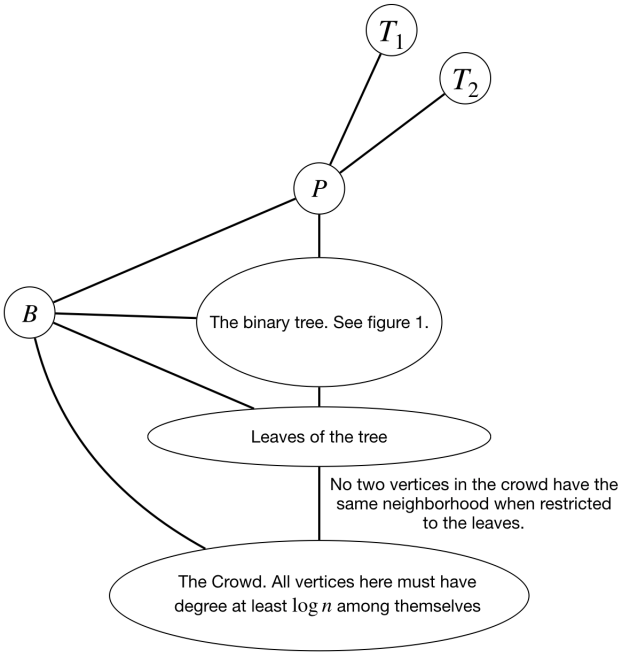

Figure 2: An illustration of Construction 4.2

of P's neighbors have degree $1, 5$ or $n-3$. Iteratively pick $x \in N_0$ and $y \in N_1$, and make a query to see if x is adjacent to y. If x and y are adjacent then y can't be T_1 or T_2 (since T_1 and T_2 only have an edge to P which is also in N_1), so remove y from N_0. If x and y are not adjacent then x can't be B unless y is T_1 or T_2, so remove x from N_0. Eventually N_0 will be empty, in which case y must be either T_1 or T_2. Thus in $O(n)$ queries we find P, and using $O(n)$ further queries we can check if Conditions 1, 2 and 4 are met.

Once we find P we proceed to find the leaves of the tree in $O(n \log n)$ queries. If at any point the structure does not match the tree (each vertex having one child of degree $2+1$ and one of degree $3+1$, where the $+1$ if for the edge with B) output 0 (Condition 3). Once we find the leaves of the tree, for each leaf we make $n-1$ queries to check if each leaf has degree different than 5 (Condition 5) and to find the neighborhood of all the leaves. Having the neighborhood of each leaf allows us to check if Condition 7 is met, that no two vertices in the crowd have the same neighborhood when restricted to the leaves. If it is met then we can determine the isomorphism (for the vertices in C [11]), between the unlabeled certificate and the true (labeled) certificate. Note that the leaves of the tree are distinguishable since we know exactly which is a right child and which is a left child due to the intermediate right children, and this allows us to determine the isomorphism. With the (labeled) certificate we can check if Condition 6 is met in $O(n \log n)$.

Lower bound for randomized algorithms: It remains to show that the randomized complexity of this function is $\Omega(n^2)$. Suppose the algorithm is given (without any queries) all of the vertices

[11] We do not have the full isomorphism, but we do know the mapping of the isomorphism for each of the vertices in the crowd – the vertices for which we want to check if they all have degree at least $\log n$ on the subgraph induced on C.

of degree less than $\log n$, their neighbors, the vertex of degree $n-3$, as well as being told which vertices are the leaves of the tree. Suppose that all conditions except 6 are met. It remains to check if Condition 6 is met – that all vertices in C have at least $\log n$ neighbors also in C. Consider the following two input distributions:

1. The vertices in C are split into 3 sets, C_1, C_2 and C_3, where $|C_1| = \log(n) - 1$ and $|C_2| = |C_3| \in \Omega(n)$. The subgraphs induced by C_1 and C_3 are cliques, and every vertex in C_1 is connected to every vertex in C_2. For every vertex in C_2 we pick a vertex at random from C_3 and add an edge. Pick an edge uniformly at random from all edges that have one vertex in C_2 and one in C_3. Call the pair of endpoints of the edge the "marked pair".

2. Same as distribution 1, except the edge which is the marked pair is removed.

For the first distribution Condition 6 is met, since every vertex in C_2 has degree $\log n$ (on the subgraph induced by C) because it has $\log n - 1$ neighbors from C_1 and a single neighbor from C_3. In the second distribution Condition 6 is not met since there exists a single vertex in C_2 with degree $\log n - 1$.

Differentiating between the first distribution and the second distribution requires $\Omega(n^2)$ queries: Suppose that the algorithm A that attempts to distinguish between the two cases is given the partition of the vertices in C into C_1, C_2 and C_3 "for free", as well as being given for free that C_1 and C_3 are cliques. Furthermore, suppose that each query not only tells us if (u, v) is an edge, but also tells us if (u, v) is the marked pair. Note that the two distributions are identical, except that the second distribution does not contain the marked pair as an edge, and thus it is possible to tell apart the two distributions with any advantage (even if algorithm is allowed to err) if and only if the marked pair is queried.

Consider an arbitrary algorithm A. Let $N = |C_2| = |C_3| \in \Omega(n)$. At any point during the execution of A, call a vertex $u \in C_2$ 'unmatched' if its adjacent edge has not been discovered yet, call it 'thin' if thus far A has made at most $\frac{N}{2}$ queries that involve u, and call it 'thick' if A has made at least $\frac{N}{2}$ queries that involve u. We can also think of the choice of the edges as well as the marked one as happening when the queries are made (principle of deferred decision): given an unmatched vertex u with q_u queries made so far, when a new slot is queried, it is matched with probability $1/(n - q_u - 1)$ and if it matched it is chosen as the marked one with probability proportional to the number of unmatched vertices.

We do not charge the algorithm for queries made to thick vertices: once a vertex u becomes thick the next time A decides to query u it queries all the slots adjacent to it and gets the edge (or the marked pair); this modification does not result in a big change in the complexity, since it at most halves the total number of queries we charge A for making in any given execution. Thus, when considering an execution of A, we assume it can only choose among thin vertices.

We will first show that if A has made at most $q = \frac{1}{200}N^2$ queries so far, then the number of unmatched vertices is at least $N/2$ whp. Consider q independent trials (corresponding to the queries), each successful with probability at most $2/N$ (a vertex becoming matched). By the Chernoff Bound, the probability that q independent (and identical) trials, each successful with probability $\frac{2}{N}$ have more than $\frac{N}{2}$ successes is[12]

$$\Pr[\text{More than } N/2 \text{ vertices become matched}] \leq 2^{-N/2}.$$

[12]We use the following form of Chernoff: $\forall t \geq 2e\,\mathbb{E}[X]$, $\Pr[X \geq t] \leq 2^{-t}$. Indeed in this case $\mathbb{E}[X] = q\frac{2}{N} = \frac{N}{100}$. And $\frac{N}{2} > 2e\frac{N}{100}$.

Thus any algorithm that thus far has made less than $\frac{1}{200}N^2$ queries, has at least $\frac{N}{2}$ unmatched vertices with high probability.

We conclude that under the assumption that the number of unmatched vertices is at least $N/2$ the probability that A queries the marked pair in the next round is bounded by $4/N^2$: The probability that a given thin vertex u is part of the matched pair is $\frac{2}{N}$ (since there are at least $N/2$ unmatched vertices), and given that we query a vertex that is part of the marked pair, the probability that we query the marked pair is at most $2/N$ (since u is thin). So we get that the probability that A finds the marked pair in q rounds is bounded by $4q/N^2$ (union bound) plus the probability of not having $N/2$ unmatched vertices (which is negligible). So altogether, an algorithm operating with $q = \frac{1}{200}N^2$ queries has a limited probability of success and we get that the worst-case randomized complexity of f (Definition 2.3) is $\Omega(N^2) = \Omega(n^2)$.

□

Remark 4.3. *The technique or gadget used in the above construction in order to make each vertex unique has additional applications. For instance, it can be used to turn an arbitrary function into a graph property while the complexity increases by an additive $O(n \cdot \log n)$ factor, resulting in the first known graph property with separation between the deterministic and randomized complexity. We explore this in future work.*

4.2 Unlabeled Instance Optimality

4.2.1 Unlabeled Certificates of Majority

Our first example regarding unlabeled instance complexity is that while the majority function is not R-instance optimizable (Lemma 3.5), it is (almost) *unlabeled* R-instance optimizable. Namely, knowing the Hamming weight of the input does not help in significantly reducing the number of queries made to the input.[13]

Here, the optimality ratio is super-constant, that is, the unlabeled decision tree has an asymptotic advantage over the standard decision tree but it is only doubly-logarithmic in the input size (and this is tight).

Lemma 4.4 (Item 1 of Theorem 1.6)**.** *The majority function is within $O(\log \log n)$ of being unlabeled R-instance optimizable.*

Proof. Consider an algorithm with an unlabeled certificate, which in the case of majority can be thought of as the number of 1's in the input. Suppose that the Hamming weight is $n/2 + \varepsilon n$. Then the algorithm must make at least $\Omega(1/\varepsilon^2)$ queries to the input. This follows by fixing the distribution that with probability $1/2$ outputs a random input of Hamming weight $n/2 + \varepsilon n$ and with probability $1/2$ outputs a random input of Hamming weight $n/2 - \varepsilon n$. By the same fact we used in Lemma 3.5, $\Omega(1/\varepsilon^2)$ queries are required to distinguish the two cases.

We show that without a certificate, we can work with almost the same number of samples: $\Theta(1/\varepsilon^2 \cdot \log \log(1/\varepsilon))$. The decision is obviously based on where the majority of the coordinates we read vote, but the issue is when to stop. This is a case of *sequential hypothesis testing* and the danger is that we are trying too many hypotheses and may fail in one of the trials, even though there is decent a probability of not failing in any particular one.

[13] The majority is a symmetric function, and so in this case Γ contains all permutations of the n inputs, thus the full information is given by the hamming weight of the input.

We use a recent result of Daskalakis and Kawase [DK17] who studied the following problem: Given sample access to an unknown distribution p over $\{0,1\}$ and an explicit distribution q over the same domain. How many samples are needed to reject the hypothesis "$p = q$" when $p \neq q$, while never rejecting when $p = q$. Daskalakis and Kawase showed that $\Theta(1/\mathsf{d}_{p,q}^2 \cdot \log\log(1/\mathsf{d}_{p,q}))$ queries are needed and sufficient, where $\mathsf{d}_{p,q}$ is the total variation distance between p and q. We sketch the upper bound directly for our case next.

The algorithm samples a coordinate $i \leftarrow [n]$ uniformly at random and queries its input at i. Let t be the number of queries made so far and let t_0 be the number of 0's and t_1 the number of 1's (where $t_0 + t_1 = t$). Whenever t is a power of 2, the algorithm decides whether to stop or go on:

1. if $t \geq n$, read the whole input and output the majority value.
2. if $t_b > t/2 + 2\sqrt{t \cdot \log\log n}$, output the bit b.

We need to show two properties of the algorithm. First, that after enough iterations, it stops with the right answer. Second, that the probability that it stops too early, before making enough queries, is small. For the latter, since our test is done only when t is a power of two, the test is performed at most $\log n$ times. Thus, if we wish that no error will be made by stopping too early we need to reduce the probability of error per test to be at most $1/(3\log n)$. Assume that the input has $n/2 + \varepsilon n$ 1's (the case with 0 majority is analogous). We start by showing that the probability that the algorithm halts and outputs the wrong answer is small. Denote by $T_b^{(t)}$ the random variable corresponding to the number of b's seen until query t, for $b \in \{0,1\}$. ($T_b^{(t)}$ is the sum of t independent random variables.) By a union bound, the probability that the algorithm makes a mistake it at most

$$\sum_{t=2^i} \Pr\left[T_0^{(t)} \geq \frac{t}{2} + 2\sqrt{t \cdot \log\log n}\right],$$

where the sum is over all t's that are powers of 2. We bound each of these terms separately using Chernoff's bound.[14] Since the input contains $n/2 + \varepsilon n$ 1's, it holds that $\mathbb{E}[T_0^{(t)}] = t/2 - \varepsilon t$. Hence, for every $t \in [n]$:

$$\begin{aligned}
\Pr\left[T_0^{(t)} \geq \frac{t}{2} + 2\sqrt{t \cdot \log\log n}\right] &= \Pr\left[T_0^{(t)} \geq \left(\frac{t}{2} - \varepsilon t\right) + \left(\varepsilon\sqrt{t} + 2\sqrt{\log\log n}\right) \cdot \sqrt{t}\right] \\
&\leq \exp\left(-2\left(\varepsilon\sqrt{t} + 2\sqrt{\log\log n}\right)^2\right) \\
&\leq \exp\left(-8\log\log n\right) \leq \log^{-8} n.
\end{aligned}$$

Thus, the overall probability that the algorithm makes a mistake is less than $\log n \cdot \log^{-8} n < 1/3$.

We show that with probability at least $2/3$ our algorithm halts (with the correct output) after at most $t' = \min\{2n, 25(\log\log n)/\varepsilon^2\}$ queries. This will finish the proof of the claim. Again, consider an input with $n/2 + \varepsilon n$ 1's. If ε is such that $t' \geq n$, then we are done. Otherwise, by the same

[14] We use the additive version of Chernoff's bound [AS08, Appendix A.1]. Let $X = \sum_{i=1}^n X_i$ be a sum of identically distributed independent random variables $X_1, \ldots, X_n \in \{0,1\}$. Let $\mu = \mathbb{E}[X] = \sum_{i=1}^n \mathbb{E}[X_i]$. It holds that for $a > 0$, $\Pr[X > \mu + a] \leq \exp(-2a^2/n)$.

Chernoff bound:

$$\Pr\left[T_1^{(t')} < \frac{t'}{2} + 2\sqrt{t' \cdot \log\log n}\right] = \Pr\left[T_1^{(t')} \leq \left(\frac{t}{2} + \varepsilon t\right) - \left(\varepsilon\sqrt{t} - 2\sqrt{\log\log n}\right) \cdot \sqrt{t}\right]$$
$$\leq \exp\left(-2\left(\varepsilon\sqrt{t} - 2\sqrt{\log\log n}\right)^2\right) < 1/3.$$

□

4.2.2 Unlabeled Certificates of Graph Properties

Our next result is that there exists a graph property which is *not* unlabeled R-instance optimizable. It is our 'old friend' the majority property, but this time the group of permutations Γ considered is that of the vertices (rather than the edges), which demonstrates the significance of picking the most appropriate Γ.

Lemma 4.5 (Item 2 of Theorem 1.6)**.** *There exists a graph property – majority of edges – which is not unlabeled R-instance optimizable. There is a distribution where any correct algorithm takes $\Omega(n^2)$ queries but an algorithm receiving an unlabeled certificate requires only $O(n \log n)$ queries.*

Proof. The graph property is that the graph $G = (V, E)$ with n vertices contains at least $\binom{n}{2}/2$ edges. The hard distribution for (certificateless) randomized decision trees will be a random graph that has either $\binom{n}{2}/2 + n$ edges or $\binom{n}{2}/2 - n$ edges.

On the one hand, a randomized algorithm that has no certificate cannot distinguish random graphs that have $\binom{n}{2}/2 + n$ edges from random graphs that have $\binom{n}{2}/2 - n$ edges unless it makes $\Omega(n^2)$ queries to the input and recovers a constant fraction of the graph. This is the fact we used in the proof of Lemma 3.5. The rest of the proof is devoted to showing that using an unlabeled certificate can significantly help.

We design a decision tree algorithm that has access to an unlabeled certificate and makes only $O(n \cdot \log n)$ queries to the input in expectation on the above distribution and decides the property correctly on all inputs with probability 2/3. The algorithm works in two steps after which the algorithms recovers the *exact* graph (and not a permutation thereof) or figures out that it was given the wrong certificate. Then, we are back in the case of Lemma 3.5 where the (exact) certificate helps compared to the algorithm without the certificate.

Denote by $G = (V, E)$ the real graph we are querying and denote by $G' = (V, E')$ the permuted graph we have as a certificate. That is, $E' = \{(\pi(u), \pi(v)) \mid (u, v) \in E\})$ for some unknown permutation π. The first step of our algorithm is to sample a random set of $k = c \cdot \log n$ vertices $A = \{v_1, \ldots, v_k\}$ (for some large enough constant $c > 0$) and query all edges (v_i, v_j) for all $i, j \in [k]$. We call the set A the *anchor*. This costs at most $k^2 \in O(\log^2 n)$ queries. Denote by G_A the induced subgraph that the algorithm just queried. The algorithm then tests how many isomorphic subgraphs there are to G_A in G'. If there are no copies, we query all the edges in the graph, as the certificate is not a permutation of the input graph. If there is more than one copy, then we are unlucky (see below) and again we query all edges. If there is exactly one isomorphic subgraph, we have just identified the vertices in A and we can use them as an *anchor* to identify the other vertices.

The next step of the algorithm is to query the input on all possible neighbors for each vertex in the anchor A. This costs $n \cdot k \in O(n \cdot \log n)$ additional queries. For a vertex $v \in V \setminus A$, let

$v(A) = (E(v, v_1), \ldots, E(v, v_k))$ be the incidence vector w.r.t. neighborhood in A. For each such $v \in V \setminus A$, look for a vertex in the certificate with the same "fingerprint", i.e., same list of vertices in A (again, if we find no match, we output $\perp$).

At this point, the algorithm completely recovers the permutation between the input and the certificate and thus we are almost done: as in Lemma 3.5, the algorithm checks using $O(n)$ random queries that the input graph corresponds to the certificate; if it does the algorithm answers accordingly (i.e., by the majority of the edges of the ceritificate) and if not the algorithm queries all the edges and answers according to the result. By the correctness of Lemma 3.5 we get that the algorithm is correct with high probability.

The fact that the algorithm is efficient, that is, if it is given a permutation of the true graph then the expected number of queries it makes is $O(n \log n)$, follows from the two claims below. They show that we will indeed recognize the vertices in A uniquely and then each other vertex in $V \setminus A$ (with very good probability). The overall complexity of the algorithm is $O(n \cdot \log n)$ queries with probability $1 - 1/n$ (the probability is over the distribution of the graphs and the random choices of the algorithm).

Claim 4.6. *With all but $1/n$ probability, there is only one induced subgraph of G that is isomorphic to G_A.*

Proof. In Alon-Spencer [AS08, §10], Subsection titled "THE PROBABILISTIC LENS: Counting Subgraphs" the following is shown. Let $G = (V, E)$ be a $G(n, p)$ graph (i.e., a graph on n vertices with edge probability $p \in (0, 1)$) and let $S \subseteq V$ be a subset of k vertices. Let G_S be the induced subgraph of G to the vertices in S. Then, the probability that there is an isomorphic copy to G_S in G besides the one induced by S is at most

$$\sum_{g=1}^{t} \left(n^2 \cdot p^{1/3 - k/6} \right)^g .$$

Letting $p = 1/2 \pm n/\binom{n}{2}$ and $k = 100 \log n$, we get that the above probability is bounded by $1/n^2$.

The above holds for a $G(n, p)$ graph while our graph is sampled from a different distribution. Our distribution is such that the given graph is a random one conditioned on having $\binom{n}{2}/2 \pm n$ edges. But, a random $G(n, p)$ graph has inverse-polynomial probability of ending up with $p \cdot \binom{n}{2}$ edges so we can get the result for these graphs by conditioning. More generally, it is known that random graphs with m edges have similar properties to ones with edge probability $p = m/\binom{n}{2}$; see, for example, Frieze and Karonski [FK15, Lemma 1.2]. □

Claim 4.7. *With probability less than $1/n$, there exist distinct $v, v' \in V \setminus A$ such that $v(A) = v'(A)$.*

Proof. For every two distinct $v, v' \in V \setminus A$ the probability that $v(A) = v'(A)$ is 2^{-k}. Applying a union bound over all possible such pairs, we get that there exist distinct $v, v' \in V \setminus A$ such that $v(A) = v'(A)$ with probability at most

$$|V \setminus A|^2 \cdot 2^{-k} \leq n^2 / 2^{c \cdot \log n} \leq 1/n^2,$$

where the last inequality follows for $c > 4$. □

This concludes the proof that majority is not instance optimal in this sense. □

Lastly, we show that even when considering graph properties (that is, the permutation group considered is over the vertices of the graph), there is a (monotone) graph property which is unlabeled R-instance optimizable:

Lemma 4.8 (Item 3 of Theorem 1.6)**.** *There exists a monotone graph property ("there exists at least one edge") which is unlabeled R-instance optimizable. But not labelling R-instance optimal.*

Proof. The graph property is that the graph $G = (V, E)$ contains at least one edge. Let $n = |V|$, and let m be the number of edges.The instance optimal randomized decision tree algorithm is the one that just samples random edges in the graph (without repetition) and queries on them until it either finds an edge or until it exhausts all the edges in the graph. If the input graph has $m \geq 1$ edges, the expected number of queries the algorithm makes is bounded by $\binom{n}{2}/m$.

In the unlabeled certificate case, we need to argue that for any $m \in \{0, \ldots, \binom{n}{2}\}$ and any graph G no randomized decision tree algorithm given G', an isomorphic copy of G, as an unlabeled certificate can be correct (on all graphs) with probability at least $2/3$ by querying in expectation $o\left(\binom{n}{2}/m\right)$.

Consider a distribution $\mathcal{D}_G$ on graphs such that with probability $1/2$ the output is a random isomorphic copy of the graph G and with probability $1/2$ the output is the empty graph (no edges at all). Consider the first k queries of any decision tree T. We will argue that for $k = \binom{n}{2}/(8m)$, with high probability T will not query any edge (when the graph it queries is a random isomorphic copy of G), and thus have no way of telling if the graph is indeed an isomorphic copy of G' or the empty graph. Note that as long as T doesn't query any edges, T is non-adaptive. Since T makes k queries, the probability that it queries at least one edge is the same as the probability that a k-edge graph H and a random permutation of a m-edge graph G intersect. We argue that this is roughly on the order of $k \cdot m/n^2$. More precisely, by a union bound and assuming the first $i-1$ rounds returned no edges, if T makes k queries, then with probability at most:

$$\sum_{i=1}^{k} \frac{m}{\binom{n}{2} - i + 1} \leq \frac{2mk}{n^2 - 2n - 2k} \leq 1/4$$

it will query an edge. Therefore, with probability at least $2/3$ it will not query any edge and thus have no way of knowing whether the graph has an edge or not. This means that $\Omega\left(\binom{n}{2}/m\right)$ queries must be made in expectation, same as the naive algorithm that does not get a certificate nor does it know m and therefore the property of having at least one edge is unlabeled R-instance optimizable.

The function is not labelling R-instance optimal, since if there exists one edge the algorithm with a (labeled) certificate requires only $O(1)$ queries. □

4.2.3 Labelling Instance Optimal

All instance optimal functions are also labelling instance optimal since for every x $\mathrm{C}(f, x) \leq \mathrm{AC}(f, \Gamma, x) \leq \mathrm{D}(f, x)$ and $\mathrm{RC}(f, x) \leq \mathrm{RAC}(f, \Gamma, x) \leq \mathrm{R}(f, x)$. We show that, up to log factors, the converse is not true; i.e, there exists a function that is within $O(\log(n))$ of being labelling D-instance optimal (resp. labelling R-instance optimal) but is not D-instance optimal (resp. R-instance optimal).

Lemma 4.9 (Item 4 of Theorem 1.6)**.** *The function defined in Construction 4.2 is within $O(\log(n))$ of being labelling D-instance optimal, and labelling R-instance optimal, but is neither D-instance optimal nor R-instance optimal.*

Proof. We know the function isn't D nor R instance optimal since in the worst case the algorithm that has access to an unlabeled certificate does significantly better than the algorithm that has no certificate due to theorem 4.1.

We will show that for every input $\mathrm{RC}(f,x) \in \Omega(n)$. Since for all inputs $\mathrm{AC}(f,x) \in \widetilde{O}(n)$ both D and R instance optimality follows. Notice that all the conditions except for Condition 3 make a requirement about the degree of a given vertex (or multiple vertices). $\Omega(n)$ queries are necessary to determine the degree of a given vertex with high probability. Condition 3 claims that we have a binary tree of size $O(\log(n))$ and thus this condition also requires $\Omega(n)$ queries with a certificate. □

4.2.4 Unlabeled Instance Optimality of Proximity Property Testing

It turns out that *every* proximity property, given an (unlabeled) certificate, can be done with at most $\tilde{O}(\sqrt{n})$ queries (with two-sided errors). This follows from a result of Fischer and Matsliah [FM08] who studied the query complexity of graph isomorphism in the proximity property testing model. One of their results determines the query complexity of this problem where one of the graphs is "known" to the algorithm and while allowing two-sided errors (this is in the dense graph model). They showed that, in this setting, $\tilde{O}(\sqrt{n})$ queries suffice and $\Omega(\sqrt{n})$ queries are necessary in general. Thus, in the unlabeled model, we can use this algorithm to first test proximity between the object and the given unlabeled certificate (using $\tilde{O}(\sqrt{n})$ queries). Then, if they are close, answer according to whether the certificate possess the property. If they are far, query the whole object and just compute the output precisely.

So, for a proximity property to be instance optimizable, it is necessary to prove that $O(\sqrt{n})$ queries are sufficient in the worst case.

5 Open Questions and Future Research

This work raises several open problems and research directions. Conjecture 3.15 is about the lack of monotone graph properties which are R-instance optimal.

Conjecture 5.1 (Conjecture 3.15, repeated)**.** *Every non-trivial monotone graph property is not R-instance optimizable.*

In section Section 3.5 we showed that given a truth-table representing a function, we can test if a given algorithm is instance optimal in time polynomial in the size of the truth-table. It remains open if we can test if a given function is instance optimizable in time polynomial in the size of the truth table. Another question is, given a decision tree (instead of a truth table), can we test whether the function it evaluates is instance optimizable.

Question 5.2 (Question 3.22, repeated)**.** *What is the complexity of testing whether a given function f is instance optimizable (within some c).*

As we observed in Section 3.6, there is a clean characterization of (randomized) instance optimality in the context of proximity testing. Also, we know that unlabeled certificates help (with properties which require many queries to test), but we do not have a full characterization.

Question 5.3. *Are there unlabeled instance optimizable properties in the proximity testing model, with worst case complexity $\omega(1)$?*

Another question is whether analogues of the Aanderaa-Rosenberg conjecture holds for unlabeled certificate decision tree complexity. The randomized variant of this question for monotone properties is also open.

Question 5.4. *Can the result of Rivest and Vuillemin [RV75] be extended to algorithms with an unlabeled certificate? Namely, do deterministic decision tree algorithms require querying a constant fraction of the edges for any non-trivial monotone graph property, even given a permutation of the given graph?*

In this work we considered two types of hints regarding the input, i.e. side information where the competing algorithm is measured only when it is correct: the full input and a permutation of the input. A natural question is what other types of hints are useful to study, e.g. partial inputs (that do not contain a certificate), such as the degree sequence.

Acknowledgements

We thank Scott Aaronson for discussions on quantum certificates, Ron Fagin for useful comments, and Ofer Grossman for helpful discussions and Yoram Moses and Benny Applebaum for suggesting instance optimality in other settings. We thank some of the referees of the paper for helpful comments.

References

[Aar08] Scott Aaronson. Quantum certificate complexity. *J. Comput. Syst. Sci.*, 74(3):313–322, 2008.

[ABB+17] Andris Ambainis, Kaspars Balodis, Aleksandrs Belovs, Troy Lee, Miklos Santha, and Juris Smotrovs. Separations in query complexity based on pointer functions. *J. ACM*, 64(5):32:1–32:24, 2017.

[ABC17] Peyman Afshani, Jérémy Barbay, and Timothy M. Chan. Instance-optimal geometric algorithms. *J. ACM*, 64(1):3:1–3:38, 2017.

[AFNS09] Noga Alon, Eldar Fischer, Ilan Newman, and Asaf Shapira. A combinatorial characterization of the testable graph properties: It's all about regularity. *SIAM J. Comput.*, 39(1):143–167, 2009.

[AS08] Noga Alon and Joel Spencer. *The Probabilistic Method*. John Wiley, third edition, 2008.

[AS11] Andris Ambainis and Xiaoming Sun. New separation between *s(f)* and *bs(f)*. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:116, 2011.

[BBC+01] Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald de Wolf. Quantum lower bounds by polynomials. *J. ACM*, 48(4):778–797, 2001.

[BD04] Ilya Baran and Erik D. Demaine. Optimal adaptive algorithms for finding the nearest and farthest point on a parametric black-box curve. In *Proceedings of the 20th ACM Symposium on Computational Geometry, SOCG*, pages 220–229. ACM, 2004.

[BdW02] Harry Buhrman and Ronald de Wolf. Complexity measures and decision tree complexity: a survey. *Theor. Comput. Sci.*, 288(1):21–43, 2002.

[BE98] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.

[BI87] Manuel Blum and Russell Impagliazzo. Generic oracles and oracle classes (extended abstract). In *28th Annual Symposium on Foundations of Computer Science, FOCS*, pages 118–126. IEEE Computer Society, 1987.

[BvEBL74] R. Best, P. van Emde Boas, and H.W. Lenstra. *A Sharpened Version of the Aanderaa-Rosenberg Conjecture*. Mathematisch Centrum Amsterdam. Afdeling Zuivere Wiskunde: ZW. Stichting Mathematisch Centrum, 1974.

[CDR86] Stephen A. Cook, Cynthia Dwork, and Rüdiger Reischuk. Upper and lower time bounds for parallel random access machines without simultaneous writes. *SIAM J. Comput.*, 15(1):87–97, 1986.

[CK07] Amit Chakrabarti and Subhash Khot. Improved lower bounds on the randomized complexity of graph properties. *Random Struct. Algorithms*, 30(3):427–440, 2007.

[DHI+09] Erik D. Demaine, Dion Harmon, John Iacono, Daniel M. Kane, and Mihai Patrascu. The geometry of binary search trees. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 496–505, 2009.

[DK17] Constantinos Daskalakis and Yasushi Kawase. Optimal stopping rules for sequential hypothesis testing. In *25th Annual European Symposium on Algorithms, ESA*, pages 32:1–32:14, 2017.

[DLM00] Erik D. Demaine, Alejandro López-Ortiz, and J. Ian Munro. Adaptive set intersections, unions, and differences. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 743–752. ACM/SIAM, 2000.

[DM90] Cynthia Dwork and Yoram Moses. Knowledge and common knowledge in a byzantine environment: Crash failures. *Inf. Comput.*, 88(2):156–186, 1990.

[FGGS98] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. Limit on the speed of quantum computation in determining parity. *Phys. Rev. Lett.*, 81:5442–5444, Dec 1998.

[FK15] Alan M. Frieze and Michal Karonski. *Introduction to Random Graphs*. Cambridge University Press, 2015.

[FLN03] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.

[FM08] Eldar Fischer and Arie Matsliah. Testing graph isomorphism. *SIAM J. Comput.*, 38(1):207–225, 2008.

[FW98] Amos Fiat and Gerhard J. Woeginger, editors. *Online Algorithms, The State of the Art (the book grow out of a Dagstuhl Seminar, June 1996)*, volume 1442 of *Lecture Notes in Computer Science*. Springer, 1998.

[GGR98] Oded Goldreich, Shafi Goldwasser, and Dana Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45(4):653–750, 1998.

[Gol17] Oded Goldreich. *Introduction to Property Testing*. Cambridge University Press, 2017.

[GPW17] Mika Göös, Toniann Pitassi, and Thomas Watson. Query-to-communication lifting for BPP. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 132–143. IEEE Computer Society, 2017.

[GR10] Oded Goldreich and Dana Ron. Algorithmic aspects of property testing in the dense graphs model. In *Property Testing - Current Research and Surveys*, pages 295–305. Springer, 2010.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 212–219. ACM, 1996.

[GSS16] Justin Gilmer, Michael Saks, and Srikanth Srinivasan. Composition limits and separating examples for some boolean function complexity measures. *Combinatorica*, 36(3):265–311, Jun 2016.

[Haj91] Péter Hajnal. An $\omega(n^{4/3})$ lower bound on the randomized complexity of graph properties. *Combinatorica*, 11(2):131–143, 1991.

[HMW90] Joseph Y. Halpern, Yoram Moses, and Orli Waarts. A characterization of eventual byzantine agreement. In *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing*, pages 333–346. ACM, 1990.

[Hua19] Hao Huang. Induced subgraphs of hypercubes and a proof of the sensitivity conjecture. *CoRR*, abs/1907.00847, 2019.

[Juk12] Stasys Jukna. *Boolean Function Complexity: Advances and Frontiers*. Springer Berlin Heidelberg, 2012.

[JZ11] Rahul Jain and Shengyu Zhang. The influence lower bound via query elimination. *Theory of Computing*, 7(1):147–153, 2011.

[Kin91] Valerie King. An $\omega(n^{5/4})$ lower bound on the randomized complexity of graph properties. *Combinatorica*, 11(1):23–32, 1991.

[KSS84] Jeff Kahn, Michael E. Saks, and Dean Sturtevant. A topological approach to evasiveness. *Combinatorica*, 4(4):297–306, 1984.

[MP06] Silvio Micali and Rafael Pass. Local zero knowledge. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, STOC*, pages 306–315. ACM, 2006. Revision http://www.cs.cornell.edu/~rafael/papers/preciseZK.pdf.

[MR97] Rajeev Motwani and Prabhakar Raghavan. Randomized algorithms. In Allen B. Tucker, editor, *The Computer Science and Engineering Handbook*, pages 141–161. CRC Press, 1997.

[MT88] Yoram Moses and Mark R. Tuttle. Programming simultaneous actions using common knowledge. *Algorithmica*, 3:121–169, 1988.

[Nis91] Noam Nisan. CREW prams and decision trees. *SIAM J. Comput.*, 20(6):999–1007, 1991.

[O'D14] Ryan O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 2014.

[OKSW94] Pekka Orponen, Ker-I Ko, Uwe Schöning, and Osamu Watanabe. Instance complexity. *J. ACM*, 41(1):96–121, 1994.

[RM99] Ran Raz and Pierre McKenzie. Separation of the monotone NC hierarchy. *Combinatorica*, 19(3):403–435, 1999.

[Rub95] David Rubinstein. Sensitivity vs. block sensitivity of boolean functions. *Combinatorica*, 15(2):297–299, 1995.

[RV75] Ronald L. Rivest and Jean Vuillemin. A generalization and proof of the aanderaa-rosenberg conjecture. In *Proceedings of the 7th Annual ACM Symposium on Theory of Computing, STOC*, pages 6–11. ACM, 1975.

[ST85] Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, 1985.

[SW86] Michael Saks and Avi Wigderson. Probabilistic boolean decision trees and the complexity of evaluating game trees. In *27th Annual Symposium on Foundations of Computer Science*, SFCS '86, pages 29–38. IEEE Computer Society, 1986.

[Tal13] Avishay Tal. Properties and applications of boolean function composition. In *Innovations in Theoretical Computer Science, ITCS*, pages 441–454. ACM, 2013.

[Tar89] Gábor Tardos. Query complexity, or why is it difficult to seperate $NP^A \cap coNP^A$ from P^A by random oracles A? *Combinatorica*, 9(4):385–392, 1989.

[VV16] Gregory Valiant and Paul Valiant. Instance optimal learning of discrete distributions. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC*, pages 142–155. ACM, 2016.

[VV17] Gregory Valiant and Paul Valiant. An automatic inequality prover and instance optimal identity testing. *SIAM J. Comput.*, 46(1):429–455, 2017.

[Yao77] Andrew Chi-Chih Yao. Probabilistic computations: Toward a unified measure of complexity (extended abstract). In *18th Annual Symposium on Foundations of Computer Science, FOCS*, pages 222–227. IEEE Computer Society, 1977.

[Yao91] Andrew Chi-Chih Yao. Lower bounds to randomized algorithms for graph properties. *J. Comput. Syst. Sci.*, 42(3):267–287, 1991.

A Efficiently Testing Scorpion Graphs

Claim A.1 (Restatement of Claim 3.12, [BvEBL74])**.** *Testing whether a given n vertex graph is a scorpion graph takes at most $O(n)$ queries.*

Proof. Let $G = (V, E)$ be an n vertex graph. We give an algorithm that does at most $O(n)$ queries to the graph G. Denote by $d(v)$ the degree of a vertex $v \in V$. Observe that once we find one of the special vertices (i.e., the body, the tail or the sting), we can locate all other special vertices with at most $3n$ queries and then check that the graph is a scorpion graph. For example, once we find a vertex b with $d(b) = n - 2$, then the vertex must be the body (if the graph is a scorpion). Going over all the neighbors of b we locate the only vertex $s \in V$ that is not a neighbor of b. This must be the sting vertex (again, if the graph is a scorpion). Then, going over all the edges adjacent to s, we locate the tail vertex t and verify that $d(t) = 2$. In total, we made $3n$ queries to the edges in G. Thus, the goal now is to locate one of the special vertices.

We begin with an arbitrary vertex $v \in V$. If $d(v) \in \{0, n-1\}$ then G is not a scorpion graph. Otherwise, if $d(v) \in \{1, 2\}$ then either v is one of the special vertices or a neighbor of v is one of the special vertices. If $d(v) = n - 2$ then v must be the body vertex. In any such case we can identify all the special vertices and checking if the graph is a scorpion graph using at most $4n$ queries to G.

Assume now that $3 \leq d(v) \leq n-3$. Denote by N_0 the set of neighbors of v. Let $N_1 = \bar{N_0} \setminus \{v\}$. The body of G (if G is a scorpion) must be in N_0 and the sting and the tail must be in N_1. At this point we iteratively choose $x \in N_0$ and $y \in N_1$ such that if $(x, y) \in E$ then $N_1 = N_1 \setminus \{y\}$ (as y can not be the sting) and choose a new $y \in N_1$, and otherwise (i.e., if $(x, y) \notin E$), $N_0 = N_0 \setminus \{x\}$ (as x can not be the body unless y is the sting) and choose a new $x \in N_0$. This process terminates after at most n queries since after every query a vertex is deleted. Moreover, if G is a scorpion then at the end of the iterations $N_0 = \emptyset$ and y is the sting. To see this, notice that the body can not be deleted from N_0 by any vertex in N_1 that is not the sting, and once the sting is encountered all the vertices in N_0 will be deleted. □

B Instance Optimality in Other Settings

The term "Instance optimality" was coined by Fagin, Lotem and Naor [FLN03] in the context of finding items with the top k aggregate scores in a database of sorted lists.[15] It has appeared in the theoretical computer science literature in several other contexts and forms. Here are a number of examples:

- **Competitive online analysis.** The competitive ratio of an algorithm A is its worst case performance relative to the best offline (see [BE98, FW98]). This is a slightly different form of instance optimality since the comparison here is to an algorithm that is not necessarily from the same class.

- **Approximation algorithms.** Comparing the size (or value) that the best algorithm can find to the one the approximating algorithm finds.

- **Self-adjusting data structures.** Is it possible to construct an optimal binary search tree, i.e., with smallest possible search time, for an access sequence given online? Sleator and

[15] We note that the term "Instance Complexity" has been used by Orponen et al. [OKSW94] in a different meaning, one related to Kolmogorov Complexity.

Tarjan [ST85] conjectured that Splay Trees are instance optimal in this sense and resolving it has been a central issue in the area. See also Demaine et al. [DHI+09].

- **Database operations.** Demaine et al. [DLM00] studied the problem of finding intersections, unions, or differences of a collection of sorted sets. While the worst-case complexity of these problems is straightforward, they consider algorithms whose complexity depends on the particular instance.

- **Computational geometry.** Baran and Demaine [BD04] gave an instance optimal algorithm for the nearest-point-on-curve problem (where one needs to find a point on a curve that is nearest to a given point). Instance optimal algorithms for convex hull and set maxima were given by Afshani et al. [ABC17]. Their notion is particularly related to the unlabeled model we consider in Section 4, where the algorithm is competing against the best one for the given set of points in some order.

- **Learning Theory.** A fundamental question in learning theory is to produce an accurate as possible approximation of an unknown distribution (over a discrete support) given independent draws from it. Valiant and Valiant [VV16] gave an algorithm that outputs an approximation whose expected distance from the real distribution is equal to the minimum possible expected error that could be obtained by any algorithm that *knows* the true yet unlabeled distribution and simply needs to assign labels. Another work of Valiant and Valiant [VV17] studied the identity testing problem: Given the explicit description of a distribution, decide whether a set of samples was drawn from it or from a distribution with some distance from it. They gave an algorithm in which the number of queries depends on the given distribution.

- **Distributed computing.** A series of works studied the possibility of instance optimal algorithms for eventual Byzantine agreement (a.k.a. Consensus) and simultaneous Byzantine agreement [DM90, MT88, HMW90]. In these problems the inputs are the votes of the players and the failure pattern. They provided several positive and negative results for various notions of instance optimality appropriate for the setting.

- **Cryptography.** The notion of *precise zero knowledge* bounds the knowledge gained by a player in terms of its actual computation rather than standard zero knowledge that bounds the knowledge of a player in terms of his *potential* computational power (see Micali and Pass [MP06]).

In this work, we concentrated on cases where the domain is full and there is no promise regarding the input (as opposed to, say the work of Fagin et al. [FLN03], where the assumption is that each column is sorted).