Electronic Colloquium on Computational Complexity, Report No. 207 (2023)

Does Prior Knowledge Help Detect Collisions?

Omri Ben-Eliezer* Tomer Grossman† Moni Naor‡

Abstract

Suppose you are given a function $f\colon [n] \to [n]$ via (black-box) query access to the function. You are looking to find something local, like a collision (a pair $x \neq y$ s.t. $f(x) = f(y)$). The question is whether knowing the 'shape' of the function helps you or not (by shape we mean that some permutation of the function is known). Our goal in this work is to characterize all local properties for which knowing the shape may help, compared to an algorithm that does not know the shape.

Formally, we investigate the instance optimality of fundamental substructure detection problems in graphs and functions. Here, a problem is considered instance optimal (IO) if there exists an algorithm $\mathcal{A}$ for solving the problem which satisfies that for any possible input, the (randomized) query complexity of $\mathcal{A}$ is at most a multiplicative constant larger than the query complexity of any algorithm $\mathcal{A}'$ for solving the same problem which also holds an *unlabeled copy* of the input graph or function.

We provide a complete characterization of those constant-size substructure detection problems that are IO. Interestingly, our results imply that collision detection is not IO, showing that in some cases an algorithm holding an unlabeled certificate requires a factor of $\Theta(\log n)$ fewer queries than any algorithm without a certificate. We conjecture that this separation result is tight, which would make collision detection an "almost instance optimal" problem. In contrast, for all other non-trivial substructures, such as finding a fixed point, we show that the separation is polynomial in n.

1 Introduction

Efficient detection of small substructures in complex data is a fundamental challenge in multiple branches of computer science. In this work, we explore to what extent *prior knowledge* on the input may help in this fundamental task. Consider, for instance, the problem of detecting a collision in an unknown function $f\colon [n] \to [n]$ given query access to f. (Here, a collision in f is a pair of disjoint elements $x \neq y \in [n]$ so that $f(x) = f(y)$.) We ask the following question.

How does an algorithm that knows nothing about f in advance (aside from the domain size n) compare to an algorithm that has some prior knowledge an the structure of f?

*Department of Mathematics, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, USA. Email: omrib@mit.edu.

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

‡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.

ISSN 1433-8092

The prior knowledge we consider in this work takes the form of an *unlabeled copy* of f that the algorithm receives in advance [GKN20]. That is, the algorithm receives a permutation of f – the composed function $f \circ \pi$ for some unknown permutation π – as an "untrusted hint". We typically call this permutation of f an *unlabeled certificate*; we require the algorithm to be correct with good probability regardless of whether the hint is correct (i.e., even if f is not a permutation of the unlabeled certificate). However, the number of queries made by the algorithm is only measured if the true input is indeed a permutation of the unlabeled certificate.

In the worst case, clearly $\Omega(n)$ queries are necessary, whether we know anything about the structure of f or not. But are there beyond-worst-case instances where holding additional structural information on f may accelerate collision detection?

Definition 1.1 (instance optimality; informal)**.** *A randomized Las Vegas*[1] *algorithm A deciding if an unknown function $f\colon [n] \to [n]$ satisfies a property $\mathcal{P}$ is* instance optimal *if there exists an absolute constant α satisfying the following. For every function f, and any randomized algorithm A' for the same task, the following holds:*

$$Queries_A(f) \leq \alpha \cdot \max_{\pi} Queries_{A'}(f \circ \pi)$$

where the $Queries_A(\cdot)$ operator refers to the expected number of queries that an algorithm A makes on a certain input.

Finally, we say that $\mathcal{P}$ is instance optimal if there exists an instance optimal algorithm for it.

Note the order of the quantifiers in the definition: for every f, the algorithm A has to compete with an algorithm A' that "specializes" to functions of the form $f \circ \pi$. In other words, an algorithm A is instance optimal if it performs as well as every algorithm A', that knows the structure of f, but not the actual labels. Note that the correctness of algorithm A' is unconditional – that is A' must be correct even if the structure of f doesn't match the certificate A' receives.

An algorithm being instance optimal means it always performs as well (up to a constant) as the algorithm that knows the structure of the input. If there is no instance optimal algorithm, that means there exists some function where knowing the structure of the function is helpful. Thus, instance optimality is a strong requirement: If a property $\mathcal{P}$ is instance optimal that means that knowing the structure of the input function f *never helps*. When a property is not instance optimal, it will sometimes be useful to discuss its "distance" from instance optimality.

Definition 1.2 (distance from instance optimality; informal)**.** *Consider the setting of Definition 1.1. For a function $\omega(n)$ that grows to infinity as $n \to \infty$, we say that $\mathcal{P}$ is ω-far from instance optimality if for every algorithm $n \in \mathbb{N}$ and A there exist a function f and an algorithm A' satisfying*

$$Queries_A(f) \geq \omega(n) \cdot \max_{\pi} Queries_{A'}(f \circ \pi).$$

Similarly, $\mathcal{P}$ is ω-close to instance optimality if the above inequality holds with $\leq$ instead of $\geq$.

We may now rephrase our initial question about collisions in the language of instance optimality. Is collision detection an instance optimal problem? I.e., is the property of containing a collision instance optimal? Is it far from instance optimality? Suppose that we have query access to a

[1] For simplicity we consider in this paper Las Vegas randomized algorithms, but all of the results apply also to Monte Carlo type algorithms (that allow some error in the returned value).

function $f\colon [n] \to [n]$ and are interested in finding a collision. There are two fundamental types of queries to f that one can make: the first option is to query an element x that we have already seen in the past, by which we mean that we have already queried some element y satisfying that $f(y) = x$. This option amounts to extending a "walk" on the (oriented) graph of f. The second option is to query a previously unseen element x, which amounts to starting a new walk. The question, then, is the following: is there a universal algorithm A (which initially knows nothing about f) for choosing when to start new walks, and which walks to extend at any given time, that is competitive with algorithms A' that know the unlabeled structure of f?

Substructure detection problems There are many other types of natural problems in computer science that involve small (i.e., constant-sized) substructure detection. A natural generalization of a collision is a k-collision (or multi-collision), where we are interested in finding k different elements $x_1, \ldots, x_k$ satisfying $f(x_1) = \ldots = f(x_k)$. Fixed points, i.e., values x for which $f(x) = x$, are important in local search and optimization problems, in particular for the study of local maxima or minima in an optimization setting.

Subgraph detection in graphs is also a fundamental problem in the algorithmic literature. Motifs (small subgraphs) in networks play a central role in biology and the social sciences. In particular, detecting and counting motifs efficiently is a fundamental challenge in large complex networks, and a substantial part of the data mining literature is devoted to obtaining efficient algorithms for these tasks. It is thus natural to ask: is it essential to rely on specific properties of these networks in order to achieve efficiency? In other words, are subgraph detection and counting instance optimal problems?

Similarly, the problem of finding collisions is a fundamental one in cryptography. Many cryptographic primitives are built around the assumption that finding a collision for some function, f is hard (e.g. efficiently signing large documents, commitments with little communication and of course distributed ledgers such as blockchain). If one wants to break such a cryptographic system, should one spend resources studying the structure of f? If finding collisions is instance optimal, that would mean that any attempt to find collisions by studying the structure of a function is destined to be futile.

In this work we focus on the instance optimality of constant-size substructure detection problems in graphs and functions. Before stating our results, let us briefly discuss these data models.

Models We consider two different types of data access in our work. The first type is that of functions. In this case the input is some function f, and the goal is to determine whether f satisfies a certain property (e.g., whether it contains a collision or a fixed point). In this case the goal of an instance optimal algorithm is to perform as well as an algorithm that receives, as an untrusted hint, the unlabeled structure of the algorithm without the actual assignment of labels. Here the complexity is measured as the number of queries an algorithm makes, where each query takes an input x and returns $f(x)$.

The second type of data is of graphs. Here the goal is to find a constant-sized subgraph. An instance optimal algorithm should perform as well as an algorithm that is given an isomorphism of the graph as an "untrusted hint". For simplicity, we focus on the standard adjacency list model (e.g., [GRS11]). Here for each vertex the algorithm knows the vertex set V in advance, and can query the identity of the i-th neighbor of a vertex v (for v and i of its choice, and according to some arbitrary ordering of the neighbors), or the degree of v. We note that all of the results also hold in

other popular graph access models, including the adjacency matrix model and the neighborhood query model.

Interestingly, graphs and functions seem closely related in our context. Specifically, the problem of finding a claw in a graph (a star with three edges) is very similar to that of finding a collision in a function, and the results we obtain for these problems are analogous.

1.1 Our Results

Our main result in this paper characterizes which substructure detection problems in functions and graphs are instance optimal. Let us start with the setting of functions.

A structure $H = ([h], E)$ is an oriented graph where each vertex has outdegree at most one, and we say that f contains H as a substructure if there exist values $x_1, \ldots, x_h$ such that $f(x_i) = x_j$ if and only if the edge $i \to j$ exists in H. (For example, a collision corresponds to the structure $([3], \{1 \to 3, 2 \to 3\})$.) Finally, the property $\mathcal{P}_H$ includes all functions f containing the structure H. Our first theorem constitutes a characterization for instance optimality in functions.

Theorem 1.3 (Instance optimality of substructure detection in functions)**.** *Let H be a connected, constant-sized oriented graph with maximum outdegree 1, and consider the function property $\mathcal{P}_H$ of containing H as a substructure. Then $\mathcal{P}_H$ is*

1. *Instance optimal if $H = P_k$ is a simple oriented path of length k;*
2. *$n^{\Omega(1)}$-far from instance optimal for any H that contains a fixed point, two edge-disjoint collisions, or a 3-collision;*
3. *$\Omega(\log n)$-far from instance optimal for any H that contains a collision.*

Similarly, in graphs we denote by $\mathcal{P}_H$ the property of containing H as a (non-induced) subgraph. Our next theorem provides a characterization for the instance optimality of subgraph detection.

Theorem 1.4 (Instance optimality of subgraph detection in graphs)**.** *Let H be a connected, constant-sized graph with at least one edge. Then $\mathcal{P}_H$ is:*

1. *Instance optimal if H is an edge or a wedge (path of length 2);*
2. *$n^{\Omega(1)}$-far from instance optimal if H is any graph other than an edge, a wedge, or a claw;*
3. *$\Omega(\log n)$-far from instance optimal when H is a claw.*

1.2 Discussion and Open Questions

Model robustness Throughout the paper we chose to focus on specific models for convenience. However, all our results are model robust and apply in many "natural" models. In particular, in the case of functions we chose to work on the model where an algorithm can only go forward. That is, an algorithm can query $f(x)$ in a black box manner, and doesn't have the capability to make inverse/backward ($f^{-1}(x)$) queries. Similar characterization results to the graph case also apply if an algorithm can walk backwards; in fact, the model where walking backward is allowed seems to serve as a middle ground between our models for graphs and functions, in the sense that we deal with directed graph properties but are allowed to move in the graph as if it were undirected.

For convenience we wrote all our results for Las Vegas randomized algorithms. All the results in this paper also apply if we require the algorithm to be a Monte Carlo randomized algorithm, i.e., one that is allowed to err with constant probability.

In graphs, we use the popular adjacency list model (which allows sampling random vertices, querying a single neighbor, or querying the degree of a vertex) for data access. The same characterization results also apply under other types of data access, such as the adjacency matrix model or the neighborhood query model (where querying a node retrieves all of its neighbors at once).

Almost instance optimality of claws and collisions While we provide a full characterization of those substructures (or subgraphs) H for which $\mathcal{P}_H$ is instance optimal, there remains a notable open problem: is the problem of containing a collision (in functions) or a claw (in graph) "almost instance optimal", e.g., is it $O(\log n)$-close to instance optimality?

We conjecture that the answer is positive, and moreover propose a concrete algorithm that we believe may achieve this bound. The main idea behind the proposed algorithm A is to alternate at any given time between $m = O(\log n)$ parallel "walks" $W_1, \ldots, W_m$ that we maintain at different scales, each time adding a single step to one of the walks. We try to extend each W_i until it reaches length 2^i or until it has to end (either because of finding a collision or due to reaching the end of a path/cycle). In the case that W_i reaches length 2^i, we "forget" it and restart W_i at a fresh random starting point.

Conjecture 1.5. *There exists an algorithm A for collision detection (in functions $f\colon [n] \to [n]$) that is $O(\log n)$-close to instance optimality.*

As observed in Section 3, the problems of finding a collision in a function and detecting a claw in a graph are very similar. Thus we also conjecture the following:

Conjecture 1.6. *Determining if a graph contains a claw is $O(\log n)$-close to instance optimality.*

1.3 Technical Overview: Collisions and Fixed Points

In this section we give an overview of our main ideas and techniques. Since many of the ideas are shared between functions and graphs, we shall mostly focus on the case of functions here.

Showing the polynomial separation for most graph and function properties amounts, roughly speaking, to providing constructions where a certain substructure is hidden, but where certain hints are planted along the way to help the algorithm holding a certificate to navigate within the graph. Given the constructions, which are themselves interesting and non-trivial, it is not hard to prove the separation. As an example of a polynomial separation construction and result, we discuss the case of a fixed point in functions. For more general statements and proofs regarding these separations, please refer to Sections 4 (for functions) and 5 (for graphs).

The $\Omega(\log n)$-separation for claws and collisions is the most technically involved contribution of this paper. Unlike the polynomial separation results, where the core idea revolves around describing the "right" way to hide information, here the construction is more complicated (roughly speaking): the trick of planting hints that allow the algorithm to navigate does not work well, and our arguments rely on the observation that it is sometimes essential for an algorithm without a certificate to keep track of multiple different scales in which relevant phenomena may emerge, compared to an algorithm with a certificate that knows in advance which of the scales is relevant. The proof is also more challenging, requiring us to closely track counts of intermediate substructures of interest. For

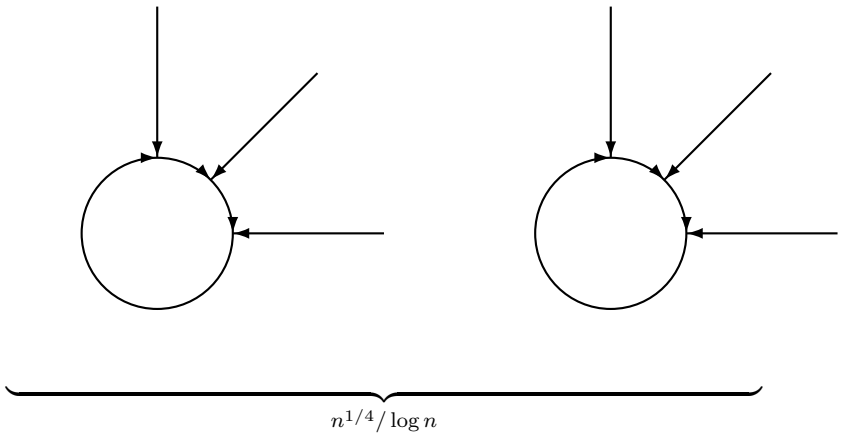

Figure 1: Each circle is of length $n^{3/4}$. Each path entering a circle is of size $n^{1/4}$. The distance between every two paths on the i-th circle is p_i.

the sake of the current discussion, we focus on collision detection, but the proof (and construction) for claws is very similar; see Section 3.

Before diving into the ideas behind fixed point and collision detection, let us briefly mention the simplest component in the characterization: an instance optimal algorithm for finding a path of length k. The algorithm chooses a random value and evaluates the function k times on successive values to see if a path of length k emerges (and not a smaller cycle, or a smaller path ending in a fixed point). This is repeated until a path is found or all values have been exhausted. It is instance optimal, since knowing the structure of the function does not help; stopping after less than k steps is meaningless, since it only saves us a constant fraction of the queries.

1.3.1 Fixed Point Detection is Polynomially Far From Instance Optimal

We give an overview of the proof that finding a fixed point is polynomially far from instance optimality. Small variations of the constructions can be used to show that the same is true for any structure containing a fixed point, a 3-collision, or two edge-disjoint collision.

In order to obtain such a result we provide a distribution of functions that have several fixed points with a secret parameter so that an algorithm with a certificate (knowing the parameter in this case) can find a fixed point in $n^{\frac{3}{4}}$ queries while any algorithm that does not know the secret parameter (i.e. without a certificate) requires $\tilde{\Omega}(n)$ queries to find a fixed point.

The idea is to construct a function f with $\tilde{\theta}(n^{1/4})$ cycles of size roughly $n^{3/4}$, where one random value x in one of the cycles is turned into a fixed point (which effectively turns the said cycle into a path ending at x). It is quite clear that for such a distribution finding the fixed point take time $\tilde{\Omega}(n)$. But we want to add some information or hint that will allow a certificate holder to find out which is the "correct" cycle.

To give such a hint we add to each cycle many paths of length $n^{1/4}$ entering it. The distance between two paths entering the ith cycle is some (unique) prime p_i where p_i is of size roughly $n^{1/4}$ (so roughly $n^{1/2}$ paths enter the cycle). See Figure 1 for a drawing of this construction.

The hint is the value p_i associated with the unique cycle that ends up with a fixed point. The algorithm (with the hint) we propose will check many (about $\sqrt{n}$) 'short' (length $n^{1/4}$) paths and see when they collide with another set of paths that is supposed to be on the cycles (these are $n^{1/4}$ 'long' paths of length $\sqrt{n}$). Once our algorithm finds three paths entering the same cycle which are of distances that are all a multiple of p_i, the algorithm will conclude that this is the unique path that at its end the fixed point resides and will continue on the path. On the other hand, for any algorithm that does not know which of the p_j's is the chosen one and hence the which path ends in a fixed point, each x residing in a cycle is equally likely to be a fixed point, and thus the algorithm requires $\tilde{\Omega}(n)$ queries in expectation.

1.3.2 Finding Collisions is $\Omega(\log n)$ Far From Instance Optimal

The distribution constructed above will not work for collision detection, since functions generated according to this distribution will inherently have many collisions. Below we describe a substantially different construction demonstrating that collision detection is (at least) logarithmically far from instance optimality. We note that the same proof outline, and same construction idea can also be used to show that finding a claw in a graph is not instance optimal.

In order to obtain such a result we provide a distribution of functions that have several collisions, again, with a secret parameter, so that an algorithm with a certificate (knowing the parameter in this case) can find a collision in n^c queries for some constant $c < 1/2$, while any algorithm that does not know the secret parameter (i.e. without a certificate) requires $\Omega(n^c \log n)$ queries to find a collision.

The hard distribution is as follows: there are $\log n$ length scales. For scale i we have $n/2.2^i$ cycles, each of length 2^i (note that the total number of nodes in all cycles is $O(n)$). For a uniformly randomly chosen scale t we turn $n^{1-c}/1.1^t$ of the cycles to be a path ending in a loop of size 2 at the end (this is a collision).

The secret parameter is the value of t. The algorithm with a certificate simply picks a value at random and follows the path it defines for 2^t steps. The algorithm stops if (i) a collision is discovered or (ii) the path has reached length 2^t without a collision or (iii) the path created a cycle of length $2^i < 2^t$. The probability of picking a node on a good path (one ending in a collision of length 2^t) is

$$\frac{n^{1-c} \cdot 2^t}{n \cdot 1.1^t}$$

(since there are $n^{1-c}/1.1^t$ such cycles, each of size 2^t). The cost (in terms of queries) of picking a value on the wrong size cycle, say of size 2^i, is $\min(2^i, 2^t)$. It is possible to show that the total expected work when picking the wrong value is $O(2^t/1.1^t)$.[2] Therefore the expected amount of work until a good path is found is

$$\frac{2^t}{1.1^t} \cdot \frac{1.1^t \cdot n}{n^{1-c} \cdot 2^t} = n^c.$$

The result is $O(n^c)$ queries in expectation.

We next show that any algorithm that does not know t requires $\Omega(n^c \log n)$ queries, which results in a logarithmic separation from the algorithm with a certificate. In essence, this means that the algorithm needs to spend a substantial effort at all possible scales (instead of just one scale, t) in order to find the collision.

[2] the constant 1.1 is a bit arbitrary, and other constants larger than 1 will also work.

Consider an algorithm without a certificate, and suppose that we choose the secret parameter t in an online manner as follows. Our initial construction starts with $n/2.2^i$ cycles of length i. For each such i, we pick $n^{1-c}/1.1^i$ of the cycles of length 2^i, and color one of the nodes in each such cycle by red (call these points the "i-red points". Note that at this time we have no information whatsoever on t. Now, each time that a red point on some cycle of length 2^i is encountered, we flip a coin with an appropriate probability (which initially is of order $1/\log n$) to decide whether the current value of i is the secret parameter t or not. If it is, then we turn all i-red points (for this specific value of i) into collisions as described above, and remove the color from all other red points (in paths of all possible lengths). Otherwise, we remove the color from all i-red points (for this specific i) and continue.

It turns out that this construction produces the same distribution as we described before (where t was chosen in advance). However, it can also be shown that to find a collision with constant probability, $\Omega(\log n)$ red points need to be encountered along the way. The rest of the analysis provides an amortized argument showing that the expected time to find each red vertex by any algorithm is $\Omega(n^c)$. The main idea of the amortized analysis (which we will not describe in depth here) is to treat cycles in which we made many queries – at least a constant fraction of the cycle – differently from cycles where we made few queries. For cycles of the first type, the probability to find a red point (if one exists) is of order $2^i/n^c$, but the amount of queries that we need to spend is proportional to 2^i. For cycles of the second type, each additional query only has probability $O(1/n^c)$ to succeed finding a red point, but the query cost is only 1. In both cases, the rate between the success probability and the query cost is of order $1/n^c$.

1.4 Related Work

The term instance optimality was coined by Fagin, Lotem and Naor [FLN03]. If an algorithm always outperforms every other algorithm (up to a constant), particularly the algorithm that is aware of the input distribution, it is defined as being instance optimal. This definition is very strict, and thus there are numerous situations in which it cannot be meaningfully attained. As a result, several works (including ours) address this by necessitating that an instance optimal algorithm be competitive against a certain class of algorithms (for example, those that know an a permutation of the input, rather than the input itself). This notion of instance optimality is sometimes referred to as "unlabeled instance optimality".

Unlabeled instance optimality Afshani, Barbay, and Chan [ABC17] considered and presented unlabeled instance-optimal algorithms for locating the convex hull and set maxima of a set of points. Valiant and Valiant [VV16] developed an unlabeled instance optimal algorithm for approximating a distribution using independent samples from it (with the cost function being the number of samples). Later [VV17], they provided an algorithm for the identity testing problem. Here the problem is determining, given an explicit description of a distribution, whether a collection of samples was selected from that distribution or from one that is promised to be far away. More recent works on instance optimality in distribution testing include, for example, the work of Hao et al. [HOSW18, HO20].

Grossman, Komargodski, and Naor examined unlabeled instance optimality in the query model [GKN20]. Their work relaxes the definition of instance optimality by no longer requiring an optimal algorithm to compete against an algorithm that knows the entire input, but rather against an algorithm that knows *something* about the input. Arnon and Grossman [AG21] define the notion

of min-entropic optimality, where instead of relaxing the "for-all" quantifier over algorithms, they relax "for-all" quantifier over inputs. That is, for an algorithm to be optimal it is still required to perform as well any other algorithm; however it is no longer required to be optimal on every input distribution, but rather only on a certain class of inputs.

Instance optimality in graphs Subgraph detection and counting has not been thoroughly investigated from the perspective of instance optimality; establishing a unified theory of instance optimality in this context remains an intriguing open problem. However, instance optimality has been investigated for other graph problems of interest. For example, Haeupler, Wajc and Zuzic [HWZ21] investigate instance optimality and a related notion called universal optimality in a family of classical and more "global" distributed graph problems such as computing minimum spanning trees and approximate shortest paths.

Strong instance optimality The original, robust definition of instance optimization calls for an algorithm to be superior to every other algorithm. For getting the top k aggregate score in a database with the guarantee that each column is sorted, [FLN03] provided an instance-optimal algorithm. Demaine, López-Ortiz, and Munro [DLM00] provided instance-optimal algorithms for locating intersections, unions, or differences of a group of sorted sets. Baran and Demaine [BD04] showed an instance optimal algorithm for finding the closest and farthest points on a curve. Grossman, Komargodski and Naor [GKN20] studied instance optimality in the decision tree model.

Cryptography and complexity The problems of finding a constant sized structure in f, where f is a total function guaranteed to contain the structure at hand has been studied extensively and is a fundamental problem in computational complexity and there are complexity classes in TFNP defined around it [MP91, Pap94]. We note that we can slightly change the functions in our paper to also make them total problems, and all our proofs will still hold.

As mentioned above, the problem of finding collisions is a fundamental one in cryptography. The standard definition is that of a collision resistant hash (CRH), where finding a collision is a computationally hard problem. Such functions are very significant for obtaining efficient signature schemes and commitment to large piece of data using little communication. But other related structures are also considered in the literature: for instance, functions where it is hard to find multiple collisions [KNY18].

1.5 Organization

In Section 2 we formally define the computational models we use as well as the notions of an unlabeled certificate (i.e., the "untrusted hint") and instance optimality. In Section 3 we prove that finding a collision in functions, and a claw in a graph is not instance optimal. In Section 4 we prove that functions that are not subsets of the collisions of two paths, followed by an additional path is polynomial far from being instance optimal. Such properties include many fundamental structures such as finding a fixed point, or a multi-collision. This gives us a complete characterization in the function model. Finally, in Section 5 we complete the full characterization for the model defined on graphs. We do this by proving that finding a path of length one or two is instance optimal, and that determining if a graph contains a subgraph H is polynomially far from being instance optimal, unless H is a path of length 1 or 2, or a claw.

2 Preliminaries

2.1 Functions

Let $n \in \mathbb{N}$. A *property* $\mathcal{P}$ of functions is a collection of functions $f\colon [n] \to [n]$ that is closed under relabeling. That is, if $f \in \mathcal{P}$ then $f \circ \pi \in \mathcal{P}$ for any permutation $\pi\colon [n] \to [n]$. We sometimes say that f *satisfies* $\mathcal{P}$ when $f \in \mathcal{P}$.

In this work we will be interested in the property of containing some constant size substructure H (e.g., a collision or a fixed point). Let H be an oriented graph with h vertices. Suppose further that the outdegree of each vertex in H is at most 1.[3] The property $\mathcal{P}_H$ consists of all functions $f\colon [n] \to [n]$ satisfying the following. There exist h disjoint elements $x_1, \ldots x_h \in [n]$ and a mapping between $V(H)$ and $\{x_1, \ldots, x_h\}$, so that H contains an edge between u and v if and only if $f(x_u) = x_v$, where x_u, x_v are the mappings of u and v, respectively.

A Las Vegas (randomized) algorithm for the property $\mathcal{P}$ in the query model is a randomized decision tree that determines membership in the property $\mathcal{P}$ with probability 1 (i.e., it is always correct, and the quantity of interest is the number of queries the algorithm requires). Given a Las Vegas randomized algorithm A (which knows n) with random seed r and given $f\colon [n] \to [n]$, denote by $\mathrm{Queries}_A^{\mathcal{P}}(f, r)$ the amount of queries that A makes when evaluating if f satisfies a property using the random seed r. Usually when the property $\mathcal{P}$ is clear from context we omit it from the notation.

We write $\mathrm{Queries}_A^{\mathcal{P}}(f) = \mathbb{E}_r \mathrm{Queries}_A^{\mathcal{P}}(f, r)$ (or $\mathrm{Queries}_A(f)$) to denote the expected number of queries that the algorithm A makes over input f, where the expectation is taken over all possible random seeds.

Definition 2.1 (Unlabeled Certificate Complexity)**.** *The* Unlabeled Certificate *complexity of a property $\mathcal{P}$, and function f is:*

$$\mathrm{RAC}(\mathcal{P}, f) = \min_{A \in \mathcal{A}_{\mathcal{P}}} \max_{\pi} \mathit{Queries}_A(f \circ \pi),$$

where $\mathcal{A}_{\mathcal{P}}$ is the set of all Las Vegas algorithms for evaluating if f satisfies property $\mathcal{P}$, and π ranges over all permutations of $[n]$.

So far, we have considered only properties of functions of a given size n. Our definition of instance optimality is asymptotic in its nature and so we extend the definition of a property by allowing it to have functions of different sizes. Suppose that $\mathcal{P}$ is a property which contains graphs of all sizes $n \geq N$, for some constant N. We can then define a corresponding sequence of algorithms $\{A_n\}_{n \geq N}$, where A_n is responsible for graphs of size n.

Definition 2.2 (instance optimality)**.** *A sequence of properties $\mathcal{P} = \{\mathcal{P}_n\}_{n \in \mathbb{N}}$ invariant under a relabeling is* instance optimal *if there exist an absolute constants $c > 0$, and a sequence $\mathcal{A} = \{A_n\}_{n \in \mathbb{N}}$, where each A_n is a Las Vegas algorithm for $\mathcal{P}_n$, such that on every input $f\colon [n] \to [n]$, it holds that*

$$\mathit{Queries}_{A_n}^{\mathcal{P}_n}(f) \leq c \cdot \mathrm{RAC}(\mathcal{P}_n, f)$$

Next we present the analogous definition for being far from instance optimality.

[3] Note that a function can be viewed as an oriented graph where the outdegree is always equal to one, hence H can appear as a substructure in such a function if and only if the outdegrees are at most 1.

Definition 2.3 (ω-*far* from instance optimality)**.** *Let $\omega\colon \mathbb{N} \to \mathbb{N}$ denote a function that grows to infinity as $n \to \infty$. We say that a sequence of algorithms $\{A_n\}_{n\in\mathbb{N}}$ evaluating if a sequence of functions $\{f_n\}_{n\in\mathbb{N}}$ (where $f_n\colon [n] \to [n]$) satisfies a sequence of properties $\mathcal{P} = \{\mathcal{P}_n\}_{n\in\mathbb{N}}$ is* ω-far from instance optimal *if there exists a constant N where for all $n \geq \mathbb{N}$ it holds that:*

$$Queries_{A_n}^{\mathcal{P}_n}(f_n) \geq \omega(n) \cdot \mathrm{RAC}(\mathcal{P}_n, f_n).$$

We say that the sequence of properties $\{\mathcal{P}_n\}$ is ω-far from instance optimal *if any sequence of algorithms $\{\mathcal{A}_n\}_{n\in\mathbb{N}}$ evaluating it is ω-far from instance optimal.*

In particular, a property $\mathcal{P}$ (or more precisely a sequence $\{\mathcal{P}_n\}$ of properties of functions $f\colon [n] \to [n]$, for any n) is polynomially far from instance optimal, if it is ω-far for some $\omega(n) = n^{\Omega(1)}$ polynomial in n.

2.2 Graphs

A graph property $\mathcal{P}$ is a collection of graphs that is closed under isomorphism. That is, if $G = (V, E) \in \mathcal{P}$ and $\pi\colon V \to V$ is a permutation, then the graph $G^\pi = (V, E^\pi)$ where $(u, v) \in E$ if and only if $(\pi(u), \pi(v)) \in E^\pi$ satisfies $G^\pi \in \mathcal{P}$.

Here we consider the adjacency list query model. We assume that the vertex set V is given to us in advance. Given a single query, an algorithm can either (i) find the degree d_v of v, or (ii) find the i-th neighbor of v (in some arbitrary ordering). We note that other variants of the adjacency list model in the literature also allow pair queries, that is, given $u, v \in V$ the algorithm can ask whether there is an edge between u and v. Our results hold word for word also in this variant.

Definitions of instance optimality are analogous to Section 2.1, except here the unlabeled certificate is an isomorphism of the graph.

Definition 2.4 (Unlabeled certificate complexity)**.** *The* ***randomized unlabeled certificate*** *complexity of a graph property $\mathcal{P}$ is defined as follows.*

$$\mathrm{RAC}(\mathcal{P}, G) = \min_{A\in\mathcal{A}_\mathcal{P}} \max_{\pi\in\Gamma} Queries_A^{\mathcal{P}}(\pi(G)),$$

where Γ is the set of all permutations of the vertex set, and $\mathcal{A}_\mathcal{P}$ is the set of all Las Vegas randomized algorithms that always evaluate membership in $\mathcal{P}$ correctly.

Definition 2.5 (instance optimality)**.** *A sequence of graph properties $\mathcal{P} = \{\mathcal{P}_n\}_{n\in\mathbb{N}}$ is* instance optimal *if there exist a constant $c > 0$ and a sequence of Las Vegas randomized algorithms $\mathcal{A} = \{A_n\}_{n\in\mathbb{N}}$ for $\mathcal{P}$, such that on every input G on n vertices, it holds that*

$$Queries_{A_n}^{\mathcal{P}_n}(G) \leq c \cdot \mathrm{RAC}(\mathcal{P}_n, G)$$

Definition 2.6 (ω-far from instance optimality)**.** *Let $\omega\colon \mathbb{N} \to \mathbb{N}$ denote a function that grows to infinity as $n \to \infty$. A sequence of algorithms $\{A_n\}_{n\in\mathbb{N}}$ evaluating if a sequence of graphs $\{G_n\}_{n\in\mathbb{N}}$ (where G_n is a graph of order n) satisfies a sequence of properties $\mathcal{P} = \{\mathcal{P}_n\}_{n\in\mathbb{N}}$ is* ω-far from instance optimal *if there exists a constant N where for all $n \geq \mathbb{N}$ it holds that:*

$$Queries_{A_n}^{\mathcal{P}_n}(G_n) \geq \omega(n) \cdot \mathrm{RAC}(\mathcal{P}_n, G_n).$$

We say that the sequence of properties $\{\mathcal{P}_n\}$ is ω-far from instance optimal *if any sequence of algorithms $\{\mathcal{A}_n\}_{n\in\mathbb{N}}$ evaluating it is ω-far from instance optimal.*

We conclude this section with the definition of a claw graph.

Definition 2.7 (Claw)**.** *The* Claw graph, S_3, *is a* 3*-star. That is, a four vertex graph consisting of a single vertex, of degree three, which is connected to three vertices each with degree one.*

3 Collisions and Claws: Logarithmic Separation

In this section we formally present and analyze our construction proving Theorem 1.3 Item 3 and Theorem 1.4 Item 3: that detecting collisions (in functions $f\colon [n] \to [n]$) and claws (in graphs) is not instance optimal.

Theorem 3.1 (Theorem 1.4 Item 3 Reworded)**.** *The property $\mathcal{P}_{S_3}$ of containing a claw is $\Omega(\log(n))$-far from instance optimality.*

Theorem 3.2 (Theorem 1.3 Item 3 Reworded)**.** *Fix $a, b, c \in \mathbb{N}$. Let $H = H_{a,b,c}$ denote the oriented graph containing two paths of length a and b which collide in a vertex, followed by a path of length c. The function property $\mathcal{P}_H$ is $\Omega(\log(n))$-far from instance optimal.*

These two cases (i.e., claws in graphs and collisions in functions) are very similar and the proof that they are not instance optimal is almost identical. Thus, for the majority of the section we focus on the case of claws in graphs. At the end of the section we describe the minor adaptations required for the case of collisions in functions.

We start by presenting the construction for claws. In Section 3.1 we adapt the construction for collisions in functions. Here and in the rest of the paper, we do not try to optimize the constant terms. In particular, the constant $c = 1/10$ appearing in the exponent of the query complexity is somewhat arbitrary; the same construction essentially works for any $c < 1/2$ (and with some adaptations it can be made to work for larger values of the constant c).

Construction 3.3. *Consider the following process for generating a graph over the vertex set $[n]$, which starts with an empty graph and gradually adds edges to it.*

- *For each integer $\frac{1}{1000}\log n \le i \le \frac{1}{100}\log n$, pick $a_i = n/2.2^i$ uniformly random disjoint simple paths of length 2^i in the graph.*
- *Pick a uniformly random integer $\frac{1}{1000}\log n \le t \le \frac{1}{100}\log n$, which we consider as the "good" index. Pick a random collection $\mathcal{P}_t$ of $b_t = n^{9/10}/1.1^t$ of the paths of length 2^t. Apply to each path $P \in \mathcal{P}_t$ the following transformation: let u_P and v_P denote the two ends of the path. Now connect u_P to two isolated vertices, and v_P to two other isolated vertices. This turns P into a tree of size $2^t + 4$ built from a long path and two claws, one at each end of the path.*
- *All vertices that do not participate in any of the above structures remain isolated.*

We claim that an algorithm holding a certificate requires only $O(n^{1/10})$ queries to find a claw. Since t is known from the certificate, the strategy is simply to only try walks of length 2^t.

Lemma 3.4. $\mathbb{E}_{G\leftarrow\Delta}\,\mathrm{RAC}(\mathcal{P}, G) = O(n^{1/10})$.

Proof. We show the following stronger claim: For any $G \in \Delta$, $\mathrm{RAC}(\mathcal{P}, G) \in O(n^{1/10})$.

The algorithm repeats the following until a claw is found. It picks a point at random, and walks at an arbitrary direction for 2^t steps or unless the walk cannot continue anymore, i.e., due to reaching the end of a path. The correctness of the algorithm is immediate. We show that if the true function matches the certificate then the algorithm makes $O(n^{1/10})$ queries in expectation.

The probability that a random point will fall on a path of length 2^i is $\frac{a_i \cdot 2^i}{n} = \frac{1}{1.1^i}$. The number of queries made, if we land on a path of length 2^i is bounded by $\min(2^i, 2^t)$.

Thus the expected number of queries made by our algorithm every time it picks a point and walks until the path ends or until the walk reaches a length of 2^t is at most

$$\sum_{i=0}^{t} 2^i \cdot \frac{1}{1.1^i} + \sum_{i=t}^{\log n} 2^t \cdot \frac{1}{1.1^i} = O\left(\frac{2^t}{1.1^t}\right) \tag{3.1}$$

The same asymptotic bound on the expectation holds also if we condition on the event that the path on which we fell at a certain round did not contain a claw.

Let X_j denote the number of steps taken by the algorithm in the j-th attempt, and let E_j denote the event that a claw is found in attempt j. The above discussion implies the following:

Claim 3.5. $\mathbb{E}\left[X_j | \neg E_1 \wedge \neg E_2 \wedge \ldots \wedge \neg E_{j-1}\right] = O\left(\frac{2^t}{1.1^t}\right)$.

To complete the proof, it suffices to prove the following claim.

Claim 3.6. $\Pr(E_j | \neg E_1 \wedge \neg E_2 \wedge \ldots \wedge \neg E_{j-1}) = \Theta\left(\frac{2^t}{1.1^t} \cdot \frac{1}{n^{1/10}}\right)$.

To prove Claim 3.6, observe that E_j holds if and only if the random starting point chosen in attempt j belongs to one of the $n^{9/10}/1.1^t$ paths of length 2^t. There are $2^t \cdot n^{9/10}/1.1^t$ such points out of a total of n points, and the claim follows by dividing these last two quantities.

Finally, the proof of the lemma follows from these two claims using linearity of expectation and a standard analysis of geometric random variables. □

The main result of this section, given below, is a lower bound showing that algorithms without a certificate require a number of queries that is larger by a multiplicative logarithmic factor compared to the best algorithm with a certificate.

Theorem 3.7. *For any algorithm A (without a certificate), $\mathbb{E}_{G \leftarrow \Delta}\, Queries_A(G) = \Omega(n^{1/10} \log(n))$.*

To prove Theorem 3.7, we first revisit Construction 3.3, discussing an equivalent way to generate the same distribution that is more suitable for our analysis. This alternative construction has some offline components, that take place before the algorithm starts to run, and an online component, that reveals some of the randomness during the operation of the algorithm.

For what follows, let $B(n)$ denote the maximum possible number of (initially isolated) vertices that are added as neighbors of claws in the second part of Construction 3.3. Note that this number is maximized when t takes its minimal possible value, and satisfies $B(n) \leq n^{9/10}/1.1^{\log(n)/1000} = n^{9/10-\Omega(1)}$.

Unlike the original construction, here we think of the vertices of the constructed graph as having one of three colors: black, red, or blue. These colors shall help us keep track of the analysis. In essence (and roughly speaking), blue vertices lead to an immediate victory but they are very rare and unlikely to be found in less than $n^{1/10+\Omega(1)}$ queries; red vertices are not as rare: finding one of

these takes roughly $n^{1/10}$ queries, but $\Omega(\log n)$ red vertices are required to find a claw with constant probability; finally, all other vertices are black, and encountering a black vertex contributes very little to the probability of finding a claw.

Construction 3.8. *We start with an empty graph on n vertices colored black, and color $B(n)$ of the vertices in blue.*

We then construct, as in the first bullet of Construction 3.3, $n/2.2^i$ disjoint paths of length 2^i out of black vertices only, for every $\frac{1}{1000}\log n \leq i \leq \frac{1}{100}\log n$.

Next, for every i we pick a subset of $b_i = n^{9/10}/1.1^i$ paths out of those of length 2^i. We color the ends of these paths in red.

The last part of the construction happens online, while the algorithm runs. In each time step where the algorithm visits a black vertex, the construction remains unchanged. If the algorithm encounters a red vertex, then we reveal the randomness in the construction in the following way.

- *Let $I = \{\frac{1}{1000}\log n \leq i \leq \frac{1}{100}\log n$: there exists a path of length 2^i with a red end$\}$. Note that initially, I simply contains all values of i in the relevant range; however in the construction I will become smaller with time.*
- *Let $i \in I$ denote the unique integer satisfying that the currently visited red vertex lies on a path of length 2^i. We flip a coin with probability $1/|I|$. If the result is 'heads', we consider i as the "good" index and do the following: all red ends of paths of length 2^i are connected to (isolated) blue vertices, all vertices in the graph are recolored by black, and the construction of the graph is complete.*
- *If the result of the above flip is 'tails', we turn all red ends of paths of length 2^i to black, and remove i from I.*

Finally, if the algorithm encounters a blue vertex, we pick $i \in I$ uniformly at random to be the "good" length, and connect the red ends of paths of length 2^i to blue isolated vertices randomly. We then recolor all vertices in the graph to black and consider the construction complete.

It is straightforward to check that Construction 3.8 produces the exact same distribution over graphs as Construction 3.3, and furthermore it does not reuse randomness revealed by the algorithm in previous parts. Of particular interest is the following observation.

Observation 3.9. *Consider any point of time during the online phase of Construction 3.8, and let I be as the defined in the first bullet. Then for any $t \in I$, the probability that t will be the eventual "good" index, conditioning on all previous choices made during the construction, is $1/|I|$.*

We say that A *wins* if it either finds a claw or encounters a blue vertex. The following lemma is a strengthening of Theorem 3.7, and its proof immediately yields a proof for the theorem.

Lemma 3.10. *There exists $C > 0$ such that for any $n \in \mathbb{N}$, any algorithm without a certificate requires at least $Cn^{0.1}\log n$ queries to win with success probability $9/10$.*

From this lemma, it immediately follows that the expected winning time for the algorithm is $\Omega(n^{0.1}\log n)$, which in turn implies the theorem (by definition of winning). Indeed, any algorithm that finds a claw can immediately find a blue vertex and win, since each claw center has two blue neighbors. Thus, we devote the rest of this section to the proof of Lemma 3.10.

The next (easy) lemma states that encountering a blue vertex is a rare event which will not substantially impact our analysis.

Lemma 3.11. *There exists an absolute constant $\varepsilon > 0$ satisfying the following. For any algorithm A without a certificate, with high probability A does not query a blue vertex within $n^{\frac{1}{10}+\varepsilon}$ steps.*

Proof. If A has already won (found a claw), then no blue vertices remain and so the probability to encounter one is zero.

Otherwise, the only chance to encounter a blue vertex at time t is by sampling a vertex from the set of vertices V_t that we did not see so far (i.e., vertices that were not queried and were not revealed to be neighbors of queried vertices). Note that $|V_t| = n - O(t) \geq n/2$ for $t = o(n)$, and the probability that the sampled vertex is blue is bounded by $B(t)/|V_t| \leq 1/n^{0.1+\Omega(1)}$. The proof follows by a union bound. □

The following result shows that finding $\Omega(\log n)$ red vertices with constant probability requires $\Omega(n^{0.1} \log n)$ queries. To complete the proof, we shall see later that either finding a blue vertex or collecting at least logarithmically many red vertices is essential to win with constant probability.

Lemma 3.12. *There exists a constant $c > 0$ so that for all $n \in \mathbb{N}$, any algorithm A which makes $cn^{0.1} \log n$ queries will find less than $\frac{1}{1000} \log n$ red vertices in expectation.*

Proof. First, note that we may assume that no blue vertex is ever encountered during the process. If such a vertex is visited, then all red vertices are recolored to black immediately, and so the success probability is zero in this case.

We show that the lower bound applies even if we augment the algorithm with the following additional information revealing some of the randomness in the construction. Clearly, this immediately yields a lower bound for the general case, where the algorithm does not hold such information.

Assume that the algorithm knows in advance, for each vertex v in the graph, whether it is part of a path (i.e., not an initially isolated vertex at the start of the online phase). Further, if v is part of path I of length 2^i, the algorithm knows I. However, the algorithm does not know initially which of the paths have their end vertices colored red.

We denote by $\mathcal{I}_i$ the set of all paths of length 2^i, and let $V_i = \bigcup_{I \in \mathcal{I}_i} V(I)$ denote the set of all vertices in these paths. Note that $V_i \cap V_j = \emptyset$ for $i \neq j$. Further let $\mathcal{R}_i \subseteq \mathcal{I}_i$ denote the set of all paths in $\mathcal{I}_i$ whose end vertices are red, and let $\mathcal{B}_i = \mathcal{I}_i \setminus \mathcal{R}_i$ denote the set of all other paths (i.e., all paths with black ends) in $\mathcal{I}_i$. Note that $\mathcal{R}_i$ and $\mathcal{B}_i$ are initially unknown to the algorithm.

The main technical claim of the proof is as follows.

Claim 3.13. *There exists an absolute constant $\alpha > 0$ satisfying the following for any fixed $\kappa > 0$. For all $n \in \mathbb{N}$, the probability that the algorithm finds a red vertex in V_i within its first $\kappa n^{0.1}$ queries in V_i is bounded by $\alpha\kappa$. This is true regardless of which queries were made outside of V_i.*

Proof. Let E denote the event that a red vertex is found within the first $\kappa n^{0.1}$ queries in V_i. The statement of the claim is that $\Pr(E) = O(\kappa)$ where the hidden term in the $O(\cdot)$ expression is an absolute constant (independent of n).

Consider two events, E_{long} and E_{short}, defined as follows. E_{long} is defined like E, with the additional requirement that the red vertex is found in a path in which the algorithm made at least 2^{i-2} queries (i.e., at least half of the path length). The complementary event $E_{\text{short}} = E \setminus E_{\text{long}}$ is the event that the red vertex is found in a path to which less than 2^{i-2} queries were made.

Note, first, that there are more than $n^{0.9}$ paths of length 2^i, and that we are interested here in the domain where at most $O(n^{0.1})$ queries are being made within V_i. Thus, at any time along the

process, the probability that a certain path $I \in \mathcal{I}_i$ whose endpoints were not visited yet satisfies $I \in \mathcal{R}_i$ (i.e., has red endpoints) is at most $(1+o(1))p$, where

$$p = \frac{n^{9/10}/1.1^i}{n/2.2^i} = \frac{2^i}{n^{0.1}}$$

was the a priori probability (before the process started) that $I \in \mathcal{R}_i$.

Since $E = E_{\text{long}} \cup E_{\text{short}}$, to prove the claim it suffices to show separately that $\Pr(E_{\text{long}}) = O(\kappa)$ and $\Pr(E_{\text{short}}) = O(\kappa)$.

We first bound $\Pr(E_{\text{long}})$. Let $\mathcal{L}_i$ denote the set of paths $I \in \mathcal{I}_i$ satisfying that (i) at least 2^{i-2} of the vertices in I were visited during the first $\kappa n^{0.1}$ queries, and (ii) one of the endpoints of I was visited during that time. Note that

$$|\mathcal{L}_i| \leq \frac{\kappa n^{0.1}}{2^{i-2}}.$$

By the above, for each $I \in \mathcal{L}_i$, the probability that $I \in \mathcal{R}_i$ (even conditioned on all previous queries made by the algorithm) is bounded by $(1+o(1))p$. Taking a union bound, we have that

$$\Pr(E_{\text{long}}) \leq \sum_{I \in \mathcal{L}_i} \Pr(I \in \mathcal{R}_i) \leq |\mathcal{L}_i| \cdot (1+o(1))p \leq (1+o(1)) \cdot \frac{\kappa n^{0.1}}{2^{i-2}} \cdot \frac{2^i}{n^{0.1}} = O(\kappa).$$

To bound $\Pr(E_{\text{short}})$, consider any $I \in \mathcal{I}_i \setminus \mathcal{L}_i$ where none of the endpoints of I have been revealed (if an endpoint was revealed to be black then $I \notin \mathcal{R}_i$ and the probability to find a red vertex in I is zero). Let $Q \subset I$ denote the set of already queried vertices in I. Recall that $|Q| < 2^{i-2}$. We claim that conditioned on all the information visible to the algorithm at this point, any vertex in $I \setminus Q$ (including all neighbors of Q) has probability at most $\frac{4}{2^i}$ to be an endpoint of I.

The proof of this claim relies on a symmetry-based perspective. Instead of thinking of I as a path on 2^i vertices, one can view it as a cycle on 2^i nodes where all edges are initially considered unseen. When one of the endpoints of an unseen edge e is queried, we flip a coin with probability $1/r$, where r is the number of unseen edges at this point, and if the result is 'heads' we break the cycle into a path by removing e from the cycle. Now to prove our claim, as long as less than 2^{i-2} vertices in I were queried, the number of unseen edges is bigger than 2^{i-1}, and so the probability of each specific node to belong to the edge that will break the cycle is less than $2/2^{i-1} = 4/2^i$.

Thus, the (conditional) probability that any particular query in a path $I \in \mathcal{I}_i \setminus \mathcal{L}_i$ will reveal a red vertex is at most

$$\frac{4}{2^i} \cdot (1+o(1)) \cdot \frac{2^i}{n^{0.1}} = O\left(\frac{1}{n^{0.1}}\right).$$

This is true for any of our $\kappa n^{0.1}$ queries in V_i. By a union bound, $\Pr(E_{\text{short}}) = O(\kappa)$. □

The proof of the lemma given Claim 3.13 follows from a simple linearity of expectation argument. For each $\frac{1}{1000}\log n \leq i \leq \frac{1}{100}\log n$, Let I_i denote the indicator random variable of whether a red vertex was found in V_i, and let q_i denote the number of queries made in V_i in the first $cn^{0.1}\log n$ rounds of the process. It follows from the claim that $\mathbb{E}[I_i] \leq \alpha q_i / n^{0.1}$, and so

$$\mathbb{E}[\#\text{ red vertices encountered}] \leq \sum_{\frac{1}{1000}\log n \leq i \leq \frac{1}{100}\log n} \alpha \frac{q_i}{n^{0.1}} \leq \frac{\alpha}{n^{0.1}} cn^{0.1}\log n \leq \alpha c \log n,$$

which completes the proof by taking c small enough as a function of α. □

We now have all the ingredients to complete the proof of Lemma 3.10.

Proof of Lemma 3.10. The algorithm (without certificate) wins if and only if it either encounters a blue vertex, or it encounters a red vertex and the subsequent coin flip results in 'heads'. By Lemmas 3.11 and 3.12 and a union bound, there exists a small enough absolute constant $C > 0$ so that the probability that either of these events happen in the first $Cn^{1/10} \log n$ rounds is bounded by 9/10. The proof follows. □

3.1 From Claws to Collisions

We now briefly define a similar construction aimed at showing the $\Theta(\log n)$ separation for collisions in functions. The same construction and proof also apply if instead of a collision we consider an "extended collision" $H_{a,b,c}$ as defined in Theorem 3.2.

Construction 3.14. *Consider the following process for generating a function $f\colon [n] \to [n]$:*

- *For each integer $\frac{1}{1000} \log n \le i \le \frac{1}{100} \log n$, add $a_i = n/2.2^i$ disjoint paths of length 2^i in the function.*
- *Pick a uniformly random integer $\frac{1}{1000} \log n \le t \le \frac{1}{100} \log n$. Pick a collection $\mathcal{P}_t$ of $b_t = n^{9/10}/1.1^t$ of the paths of length 2^t. For each such path P, let u and v denote the first and last element in the path; set $f(v)$ to be an arbitrary value in $P \setminus \{u, v\}$, which creates a collision. Close all other paths (of all lengths) into cycles: specifically, using the same notation, set $f(v) = u$.*
- *For all values $x \in [n]$ that do not participate in any of the above paths, set $f(x) = x$ to be a fixed point.*[4]

As is the case with claws in graphs, an algorithm holding a certificate would know the value of t, and make walks of length 2^t until finding a collision, with a query complexity of $O(n^{1/10})$. Meanwhile, to show the lower bound for an algorithm without a certificate, we use a coloring scheme with only two colors – red and black – where elements that are ends of paths which serve as "candidates" for a collision are marked red, and all other elements are marked black. Similarly to the above, in order to find a collision with constant probability, the algorithm needs to find $\Omega(\log n)$ red elements with constant probability, which requires $\Omega(n^{1/10} \log n)$ queries.

4 Function Properties Far from Instance Optimal

In this section we study the instance optimality of properties of functions $f\colon [n] \to [n]$. As shown in the previous section, the property of containing a collision is $\Theta(\log n)$-far from instance optimal. We show that any pattern with either at least two collisions or at least one fixed point is *polynomially* far from instance optimality. This is summarized in the theorem below.

Theorem 4.1. *Let H be any constant-size oriented graph (possibly with self-edges) where each node has out-degree at most one. Suppose further that H either contains (i) a fixed point (i.e., an*

[4] We note that the construction can be easily modified to not include fixed points: simply use the remaining values to close cycles of length 2 or 3 instead of fixed points, which are essentially cycles of length 1.

edge from a node to itself) or (ii) at least two nodes with in-degree at least two, or (iii) at least one node with in-degree at least three.

The function property $\mathcal{P}_H$ of containing H as a substructure is $\tilde{\Omega}(n^{1/4})$-far from being instance optimal.

The rest of this section is devoted to the proof of the theorem. We start with the construction used to prove the theorem.

Construction 4.2. *Let H be an oriented graph satisfying the conditions of Theorem 4.1. Define the* entry vertices *of H to be those vertices with in-degree 0 in H (in the special case where H is a single fixed point, define its single vertex as the entry point). Let T be the total number of entry vertices in H.*

Define an input distribution Δ as follows: first, pick $\alpha\frac{n}{\log n}$ vertices uniformly at random and split them into $N = \alpha n^{1/4}/\log(n)$ disjoint cycles of length $n^{3/4}$, for a small absolute constant $\alpha > 0$. Denote these cycles by $C_1, ..., C_N$. For each cycle C_i we associate a unique prime number, p_i, where all p_i's are in the range $(n^{1/4}/4, n^{1/4}/2)$ for an appropriate value of c. Note that this is possible due to well-known results on the density of prime numbers. We say that all points contained in the union $\bigcup_{i=1}^{N} C_i$ are of type 1*.*

For each cycle C_i, we add paths of length $\alpha n^{1/4}$ entering it, where the distance (in C_i) between the entry points of every two adjacent paths entering the cycles is exactly p_i. Since each cycle C_i has a length of $n^{3/4}$, it has $\Theta(\sqrt{n})$ paths entering it, each of length $n^{1/4}$. We say that all points participating in these paths are of type 2*.*

Lastly, a collection $x_1, \ldots, x_T$ of exactly T points from $\bigcup_{i=1}^{N} C_i$ is picked uniformly at random conditioned on the event that no two of these points come from the same cycle. Denote the latter event by E and note that $\Pr(E) = 1 - o(1)$. For each x_k, let C_{i_k} denote the cycle containing it, and turn this cycle into a path ending at x_k by removing the outgoing edge from x_k. Finally, insert a copy of H using all $x_1, \ldots, x_T$ as entry points.

To complete the function into one that has size n, partition all remaining (unused) points into disjoint cycles of length $n^{3/4}$ each.

Crucially, an algorithm with a certificate knows the indices $i_1, \ldots, i_T$ (and the corresponding primes $p_{i_1}, \ldots, p_{i_T}$). Given these primes, it turns out that the algorithm is able to find the relevant cycles, walk on them until finding the entry points, and building the full H-copy, all using $O(n^{3/4})$ queries. In contrast, for an algorithm without the certificate, the entry points are distributed uniformly over the union of all cycles, and thus a lower bound of $\tilde{\Omega}(n)$ can be shown.

Lemma 4.3. $\mathbb{E}_{f\leftarrow\Delta}\, RAC(\mathcal{P}_H, f) = O(n^{3/4})$.

Lemma 4.4. *For any algorithm A (without a certificate), $\mathbb{E}_{f\leftarrow\Delta}\, Queries_A(f) = \Omega(n/\log n)$.*

Proof Of Lemma 4.3. We show the following stronger claim: For any $f \in \Delta$, $\mathcal{RAC}(\mathcal{P}_H, f) = O(n^{3/4})$. We provide an algorithm to find all T entry points with $\Omega(1)$ success probability using $O(n^{3/4})$ queries. Once all T entry points are found, completing the copy of H is trivial. The algorithm does the following for a large enough constant C.

- **Phase 1: sampling short paths.** Pick $Cn^{1/2}$ vertices uniformly at random and follow the path stemming from each point for $Cn^{1/4}$ steps.

- **Phase 2: sampling long paths.** Pick $Cn^{1/4}$ points uniformly at random and follow the path stemming from each point for $Cn^{1/2}$ steps.
- **Phase 3: collection intersections and closing cycles.** Consider any walk W generated in step 2, for which there are three walks generated in step 1 which intersect W at points y_1, y_2, y_3 (for this matter, the intersection point of two walks is defined as the first point in which they intersect). If the distance between all of these intersection points is a multiple of p_{i_k} for some $k \in [T]$, we follow W as long as possible (until the walk closes a cycle or reaches a fixed point; the latter means, conditioned on the certificate being correct, that an entry point was found).

First note that $\Omega(n)$ of the points are of type 2. Thus in Phase 1, with high probability at least $\sqrt{n}$ of the points picked will be of type 2. Since each such point is followed for a length of $n^{1/4}$, the algorithm must query a point that is on the intersection of a path and a cycle. Each of the cycles that contain an entry point will with high probability have $n^{1/4}$ points queried by our algorithm on them. Additionally, with constant probability arbitrarily close to one (for C large enough), Phase 2 will pick at least one point in each of the cycles $C_{i_1}, \ldots, C_{i_T}$ containing an entry point, and subsequently make a walk of length $C\sqrt{n}$ on each of these cycles.

We conclude that with probability $\Omega(1)$, the following holds for all cycles $k \in [T]$: the path generated within C_{i_k} in Phase 2 will intersect at least two paths generated in Phase 1. Thus, Phase 3 will ensure that the rest of C_{i_k} will be queried, until reaching the entry point. Given all query points, H will be queried, as needed. Thus, the success probability of a single iteration of the algorithm is $\Omega(1)$.

It remains to bound the expected query complexity of a single iteration of the algorithm. Clearly, Phases 1 and 2 always take $O(n^{3/4})$ queries. Phase 3 takes $O(n^{3/4})$ queries when restricted to cycles that contain an entry point, since there are only $O(1)$ such cycles. What about other cycles?

With high probability, the following holds for all cycles C' not containing an entry point: at most $O(n^{1/4})$ of the short paths from Phase 1 and at most $O(\log^{1.1} n)$ of the long paths from Phase 2 will intersect C'. It follows that for with high probability, simultaneously for all such cycles, the number of short-long intersections is $O(\log^2 n)$. Condition on this high probability event.

Let C' be any cycle not containing an entry point, and let p' be the unique prime associated with this cycle. Also let $p \in \{p_{i_1}, \ldots, p_{i_k}\}$ be any prime associated with one of the entry point cycles, and let $y_1, \ldots, y_m$ denote all intersection points found in C'. Note that all distance between y_i and y_j are divisible by p', but the probability that each such distance is divisible by p is $O(1/p) = O(1/n^{1/4})$. Thus, the probability that there are three intersection points y_i, y_j, y_l whose distances are all divisible by p is $O(\log^4 n/\sqrt{n})$. Phase 3 will query the whole cycle C' only if this event holds.

Finally, taking a union bound over all values of $p \in \{p_{i_1}, \ldots, p_{i_k}\}$ and all possible cycles C', we conclude that the expected number of queries made in Phase 3 on incorrect cycles C' (which do not contain entry points) is of order $O(n^{3/4} \cdot \log^4 n/n^{1/4}) = o(n^{3/4})$.

We have seen that a single iteration of the algorithm above succeeds with $\Omega(1)$ probability and has expected query complexity $O(n^{3/4})$. The proof follows by linearity of expectation and standard bounds on geometric random variables. □

Proof of Lemma 4.4. Consider the construction right after all type 1 points are determined, and suppose that the algorithm is given the following information "for free": for each cycle C_i, the algorithm knows exactly which points belong to C_i as well as their order along the cycle. Note that the algorithm does not know a priori the indices $i_1, ..., i_T$ of the cycles which will be turned

into a path (which ends in an entry point to the H-copy). It also does not receive any additional information about type 2 points, aside from the information detailed above.

We say that a point in $[n]$ is *critical* if it participates in the H-copy. Note that an algorithm must either query a critical point or make $\Omega(n)$ total queries in order to be correct with constant probability. Thus we bound the number of queries an algorithm makes until it queries a critical point. Once such a point is queried, the algorithm stops running.

Let $C = \bigcup_{i=1}^{N} C_i$ denote the set of all points lying in cycles, and let $B = [n] \setminus C$ denote the set of all other points. With some abuse of notation let H be the (unknown) set of critical points. We note that for any $x \in C$, the probability that $x \in H$ is completely independent of the queries that the algorithm makes in B. The same is true for $x \in B$ with respect to queries made in C.

Consider first queries that the algorithm makes in B. Because of the above independence, the probability that after i queries were made in B, the next query will reveal a critical point is bounded by $|H|/(|B| - i) = \Theta(1/n)$. Thus, any algorithm that makes $o(n)$ queries has $o(1)$ probability to query a critical point in B.

To analyze the situation in C, observe that the event E defined during the construction satisfies $\Pr(E) = 1 - 1/\Theta(n^{1/4})$. Thus from elementary probabilistic considerations it will suffice to prove the following claim: let $x_1, \ldots, x_k$ be a uniformly random collection of k disjoint points in C without the condition that they come from different cycles (i.e., without conditioning on the event E). Then the expected number of queries required for any algorithm (without a certificate) to query at least one of these points is $\Omega(n/\log n)$. However, the proof of this claim is essentially trivial, following from standard estimates for sampling without replacement: in our setting, we have $|C| = \Theta(n/\log n)$ red balls and T green balls in an urn, and the quantity we need to output is the expected amount of time until a green ball is sampled (without replacement). This quantity is of order $\Theta(n/\log n)$. □

5 Instance Optimality in Graphs

We consider whether questions of the type: "Does G contains a subgraph H" are instance optimal. We begin by defining a k-star.

Definition 5.1 (k-star)**.** *The k-star S_k is a graph with $k+1$ vertices: k vertices of degree 1, all connected to a vertex of degree k.*

For example, a 1-star is a graph consisting of 2 vertices, connected by an edge. A 2-star (or a wedge) contains 2 vertices of degree 1 connected to a third vertex of degree 2. A 3-star is a claw.

In Section 3, we have seen that claws are not instance optimal, by showing a $\Omega(\log n)$-separation between an algorithm with a certificate and one without a certificate in some case. What about other choices of H? It turns out that if H is a 1-star or 2-star then finding H is instance optimal. If H is any other graph, then finding H is polynomially far from instance optimal.

Theorem 5.2 (Theorem 1.4 repeated)**.** *Let H be any fixed graph and consider the property $\mathcal{P}_H$ of containing a copy of H as a subgraph. Then $\mathcal{P}_H$ is:*

- *instance optimal if H is an edge or a wedge (path with two edges);*
- *$n^{\Omega(1)}$-far from instance optimal if H is any graph other than an edge, a wedge, or a claw; and*
- *$\Omega(\log(n))$-far from instance optimal when H is a claw.*

The theorem follows from the lemmas below, together with the results of Section 3. Lemma 5.3 proves the first item of the theorem. Lemma 5.4 proves that for every H that is not a k-star, finding H is not instance optimal. Lemma 5.5 proves the separation for k-stars when $k \geq 4$.

Lemma 5.3. *$\mathcal{P}_H$ is instance optimal when H is a 1-star (edge) or a 2-star (wedge).*

Lemma 5.4. *For all graphs H that are not a k-star, there exists a distribution, Δ, such that $\mathbb{E}_{G \leftarrow \Delta}$ Queries$_A(G) = \Omega(n)$ for any algorithm A (without a certificate) determining membership in $\mathcal{P}_H$, whereas $\mathrm{RAC}(\mathcal{P}_H, \Delta) = O(n^{1/2} \log n)$.*

Lemma 5.5. *Let $H = S_k$ for $k \geq 4$. There exists a distribution Δ such that for any algorithm A (without a certificate) determining membership in $\mathcal{P}_H$, $\mathbb{E}_{G \leftarrow \Delta}$ Queries$_A(G) = \Omega(n)$, while $\mathrm{RAC}(\mathcal{P}_H, \Delta) = O(n^{1/2})$.*

Proof Of Lemma 5.3. When H is a single edge, it is immediate that the algorithm which repeatedly picks a random vertex and checks whether it has neighbors is instance optimal. When H is a wedge, consider the following algorithm: pick a random vertex u. If u has two neighbors, then a wedge was found. If u has one neighbor, v, then check if v has a neighbor. It is not hard to verify that the algorithm is instance optimal. □

Proof of Lemma 5.4. For a given graph H we define an input distribution Δ as follows. Pick $\sqrt{n}$ vertices arbitrarily. These vertices are the center of disjoint k-stars. Each remaining vertex is connected to exactly one of the star centers such that each star has a unique degree in $[\frac{\sqrt{n}}{4}, \frac{3\sqrt{n}}{2}]$. Note that in this construction each vertex is either a star center, or a vertex of degree 1. Next we pick $|H|$ degree-1 vertices uniformly at random, such that no two vertices that are chosen belong to the same star. These vertices are then connected to form a clique, and thus in particular contain H. Denote this set of vertices by S.

Our result follows from the following two claims:

Claim 5.6. $\mathrm{RAC}(\mathcal{P}_H, \Delta) = O(n^{1/2} \log n)$.

Claim 5.7. *For any algorithm A (without a certificate) for $\mathcal{P}_H$, $\mathbb{E}_{G \leftarrow \Delta}$ Queries$_A(G) = \Omega(n)$.*

Proof Of Claim 5.6. The algorithm starts by finding the center of each star and its degree as follows:

$A \leftarrow \emptyset$
for $i = 1, ..., \Theta(\sqrt{n} \log n)$ **do**
  Pick an arbitrary vertex $v \in V$, and query the degree of v.
  if $\deg(v) = 1$ **then**
    Let u be the neighbor of v. u is a center of a star. If $u \notin A$, add u to A.
  end if
end for

The probability of any star center to be found in any particular step is $\Theta(1/\sqrt{n})$, so after $\Theta(\sqrt{n} \log n)$ rounds of the above process, A will contain all star centers.

Next, we query the degrees of all star centers. This takes $O(\sqrt{n})$ queries.

Recall that the degree of each star is unique, and thus the algorithm knows from the unlabeled certificate the degrees of the star centers that have a neighbor in S; call these star centers "good". Finally, we query all neighbors of all good star centers at a cost of $O(\sqrt{n})$ queries. This reveals the clique containing H. □

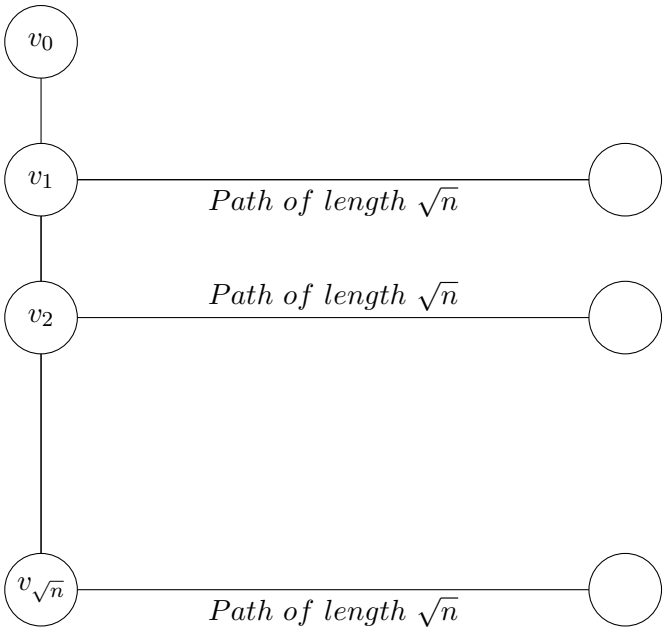

Figure 2: Construction for Lemma 5.5

Proof of Claim 5.7. The proof is very similar to that of Lemma 4.4 and we sketch it briefly.

Suppose the algorithm is given for free all the stars, but is *not* given S. In particular, for every star center v, the algorithm knows the set of all neighbors of v. The algorithm is required to query at least one node in S.

Suppose that we pick $|H|$ vertices uniformly at random among all degree-1 vertices. Let E denote the event that no two of them belong to the same star. Then $\Pr(E) = 1 - O(1/\sqrt{n})$. Similarly to Lemma 4.4, it suffices to prove the following: if we choose a set S of $|H|$ points from all degree-1 vertices without conditioning on E, then the expected number of queries required to find one vertex in S is $\Omega(n)$. As in Lemma 4.4, this follows from a standard analysis of a sampling with replacement setting where there are $O(1)$ green balls and $O(n)$ red balls, and we are interested in the number of balls that one needs to sample in expectation to get one green ball. □

The proof of the lemma follows. □

Proof of Lemma 5.5. We describe the construction for the property of containing a k-star for $k \geq 4$. Define an input distribution Δ as follows. Pick $\sqrt{n}$ vertices randomly, and connect them to form a path. Next randomly split the remaining vertices into $\sqrt{n}$ disjoint subsets each of size $\sqrt{n}-1$ or $\sqrt{n}-2$, $P_1, ..., P_{\sqrt{n}}$. Order the vertices in each P_i randomly, and connect them in this order to form a path. Lastly, connect a new vertex v_i to the first vertex in P_i for each i, and connect another vertex v_0 to v_1. See illustration in Figure 2.

The last phase of the construction is to pick a random vertex u. We then connect u to k additional vertices. Denote this set of vertices by S. As we show, the construction establishes the following:

Claim 5.8. $\mathrm{RAC}(\mathcal{P}_{S_k}, \Delta) = O(n^{1/2})$.

Claim 5.9. *For any algorithm A (without a certificate) for $\mathcal{P}_{S_k}$, $\mathbb{E}_{G \leftarrow \Delta}$ Queries$_A(G) = \Omega(n)$.*

Proof of Claim 5.8. The algorithm with a certificate knows the index k for which P_k contains the star center. It first finds $v_0, v_1, \ldots, v_{\sqrt{n}}$ in the following way. Pick a random starting point v, and walk until encountering either a degree-1 vertex or a degree 3 (or more) vertex. In the former case, we can walk on the other direction from v until reaching a degree 3 or more vertex. It is easy to check with $O(1)$ queries if the encountered vertex u is part of a k-star; assume henceforth that this is not the case. Thus, $u = v_i$ for some $1 \leq i \leq \sqrt{n} - 1$ (note that v_0 and $v_{\sqrt{n}}$ have degree 1 and 2 respectively, not 3). From here we can locate all vertices v_i for $2 \leq i \leq \sqrt{n} - 1$ (in this order or the reverse order) by checking, for each of its neighbors, whether it is degree 3 (or degree 1, which marks that we found v_0).

Given $v_0, \ldots, v_{\sqrt{n}}$ and the value k for which the star center is in P_k, the algorithm simply proceeds by querying all vertices in P_k (via a walk from v_k), including the star center. □

Proof of Claim 5.9. Suppose the algorithm is given, for free, all the paths, but is not given any information about the star center. The proof follows immediately since the star center was chosen uniformly at random among all vertices in the graph. □

The proof of the lemma follows from the above two claims. □

References

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

[AG21] Gal Arnon and Tomer Grossman. Min-entropic optimality. *Electron. Colloquium Comput. Complex.*, TR21-152, 2021.

[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, 2004.

[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 (SODA)*, pages 743–752, 2000.

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

[GKN20] Tomer Grossman, Ilan Komargodski, and Moni Naor. Instance complexity and unlabeled certificates in the decision tree model. In *11th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 56:1–56:38, 2020.

[GRS11] Mira Gonen, Dana Ron, and Yuval Shavitt. Counting stars and other small subgraphs in sublinear-time. *SIAM Journal on Discrete Mathematics*, 25(3):1365–1411, 2011.

[HO20] Yi Hao and Alon Orlitsky. Data amplification: Instance-optimal property estimation. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, 2020.

[HOSW18] Yi Hao, Alon Orlitsky, Ananda T. Suresh, and Yihong Wu. Data amplification: A unified and competitive approach to property estimation. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NeurIPS)*, page 8848–8857, 2018.

[HWZ21] Bernhard Haeupler, David Wajc, and Goran Zuzic. Universally-optimal distributed algorithms for known topologies. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, page 1166–1179, 2021.

[KNY18] Ilan Komargodski, Moni Naor, and Eylon Yogev. Collision resistant hashing for paranoids: Dealing with multiple collisions. In *EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 162–194, 2018.

[MP91] Nimrod Megiddo and Christos H. Papadimitriou. On total functions, existence theorems and computational complexity. *Theor. Comput. Sci.*, 81(2):317–324, 1991.

[Pap94] Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. Syst. Sci.*, 48(3):498–532, 1994.

[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, 2016.

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

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il