Electronic Colloquium on Computational Complexity, Report No. 152 (2021)

Min-Entropic Optimality

Gal Arnon * Tomer Grossman *

Abstract

We introduce the notion of *Min-Entropic Optimality* thereby providing a framework for arguing that a given algorithm computes a function better than any other algorithm. An algorithm is $k(n)$ Min-Entropic Optimal if for every distribution D with min-entropy at least $k(n)$, its expected running time when its input is drawn from D is at most a multiplicative constant larger than the expected running time (also with respect to D) of any other algorithm that computes the same function. Min-Entropic Optimality is a relaxation of the well established notion of instance optimality (when $k(n) = 0$). Thereby, Min-Entropic Optimality provides a meaningful notion of optimality, even in scenarios where instance optimality is inherently impossible to achieve (for instance, in the super-linear regime).

We analyze basic properties of this notion and prove that for many values of $k(n)$ there exist functions that have Min-Entropic Optimal algorithms. We further show that some natural search problems, such as k-Sum, are unlikely to have optimal algorithms under this notion.

*Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100, Israel. Supported in part by grants from the Israel Science Foundation (no. 2686/20) and by the Simons Foundation Collaboration on the Theory of Algorithmic Fairness. Email: {gal.arnon,tomer.grossman}@weizmann.ac.il.

ISSN 1433-8092

Contents

1 Introduction

Given two algorithms, how do we decide which one of them is better? What does it mean for one algorithm to be "the best"? While worst-case complexity is the hallmark metric of theoretical computer science, it is not a very good measure for comparing algorithms. For example, if the worst-case inputs appear rarely in real-world scenarios, then this notion loses almost all meaning. Fagin, Lotem and Naor [FLN03] introduced the notion of instance optimality, a concrete and clear definition for what it means for an algorithm to be "the best" at computing some function. Roughly speaking, an algorithm A computing a function f is instance optimal if for every input x and every other algorithm B that computes f correctly on all inputs, A computes $f(x)$ in time comparable to B. If an algorithm is instance optimal, then it is unequivocally "the best".

Instance optimality is a very strong requirement since the optimal algorithm must be competitive with every algorithm on every input. Due to this, few problems can be shown to be instance optimal. In fact, any problem that requires super-linear time to compute in the worst-case cannot be instance optimal![1] Thus, this definition is only meaningful in the sub-linear regime.

In this paper we provide a relaxation of instance optimality that still captures a meaningful notion of an algorithm being "the best":

Definition 1.1 ($k(n)$-Min Entropic Optimal (Informal))**.** *A deterministic algorithm A computing a function f is* $k(n)$-Min Entropic Optimal *($k(n)$-ME Optimal) if there exists a constant α such that for any deterministic algorithm B that computes f and any input distribution D with min-entropy at least $k(n)$:*

$$\mathop{\mathbb{E}}_{x \leftarrow \mathcal{D}}[\mathsf{Time}_A(x)] \leq \alpha \mathop{\mathbb{E}}_{x \leftarrow \mathcal{D}}[\mathsf{Time}_B(x)]$$

Where $\mathsf{Time}_A(x)$ and $\mathsf{Time}_B(x)$ are the running times of A and B on input x respectively.

We say a function is $k(n)$-ME Optimal if it has a $k(n)$-ME Optimal algorithm computing it. Similarly, we say a function is not $k(n)$-ME Optimal ($k(n)$-ME Sub-Optimal) if there does not exist a $k(n)$-ME Optimal algorithm computing it.

In comparison to instance optimality, an ME Optimal algorithm is no longer required to be competitive with every algorithm on *every* input. Instead, an ME Optimal algorithm must be competitive with every other algorithm on every *distribution* of inputs with sufficiently high min-entropy. In fact, instance optimality can be seen as a special case of ME Optimality when $k(n) = 0$. On the other extreme, an algorithm is n-ME Optimal if it is the fastest possible algorithm (up to a constant) in expectation when inputs are drawn from the uniform distribution.

ME Optimality can be seen as a measure of how "stable" the optimality of an algorithm is. Suppose A is the fastest algorithm for computing some function that is possible in expectation when inputs are drawn from the uniform distribution. Now suppose that the input distribution has changed. It is possible that in this scenario A will fail utterly when compared with another algorithm B that "knows" the new distribution of inputs. However, if A is ME Optimal, then it is competitive with any other algorithm (provided the algorithm computes the function correctly

[1] Indeed, suppose that a function f has super-linear running time in the worst-case. For any algorithm A there exists x such that $A(x)$ runs in super-linear time. A competing algorithm B can have x and $f(x)$ hard-coded. On an input y it compares checks whether $y = x$. If so, it returns $f(x)$. Otherwise it computes $A(y)$. B computes $f(x)$ in linear time.

on all inputs, even ones that are not sampled by the distribution) *even if that algorithm knows the new input distribution* as long as that distribution has enough min-entropy.[2]

1.1 Our Contributions

Our main contribution in this paper is the introduction of the notion of ME Optimality which we believe to be an interesting and useful notion in the analysis of algorithms. See Section 2 for an exact definition of the notion and an in-depth discussion regarding the computation model used.

Basic Understanding of ME Optimality We prove some insightful properties of ME Optimality. For example we show that if a function is $k'(n)$-ME Optimal, then it is $k(n)$-ME Optimal for every $k(n) > k'(n)$. Additionally, we explore the connection between ME Optimality and worst case hardness.

Theorem 1.2 (Hardness Implies Sub-optimality (Informal))**.** *Any function with worst-case complexity $t(n) \in \omega(n)$ is not $o\left(\log\left(\frac{t(n)}{n}\right)\right)$-ME Optimal.*

ME-Optimality Hierarchy We further show that there exist both functions that are ME Optimal and ones that are not. In fact, we show an ME Optimality Hierarchy. This supports our assertion that ME Optimality is a meaningful and interesting notion.

Theorem 1.3. *(Informal) For every $k(n) \in \omega(\log(n))$ there exists a function that is $k(n)$-ME Optimal and is not $\frac{k(n)}{1+\epsilon}$-ME Optimal for any constant $\epsilon > 0$.*

Notice that these functions are ME Optimal but not instance optimal, showing that there are functions for which our framework is a meaningful generalization of instance optimality.

The functions implied by Theorem 1.3 are implied by a non-constructive argument. However, given standard hardness assumptions, such as the non-uniform exponential time hypothesis (non-uniform ETH), we obtain such functions explicitly, albeit with a larger gap.

(Sub-)Optimality of Natural Structured Problems. We further show that functions that have short witnesses and inhibit "structural symmetry" are likely not $k(n)$-ME Optimal for large values of $k(n)$. One example of such a problem is k-Sum.

Definition 1.4 (k-Sum problem)**.** *Let k be a constant. Given n integers in the range $[-d, d]$ for an integer d, find k numbers that sum up to zero over $\mathbb{Z}$. We say that the problem is* dense *if (roughly) $n^k \gg d$.*

Theorem 1.5. *[Sub-Optimality Of k-Sum (Informal)] Suppose that k-Sum with parameter $d(n)$ (in the dense regime) requires an expected running time of $\omega(k \log d(n) + k \log n)$ to compute when the input is chosen from the uniform distribution. Then it is not $(N - O(\log(N)))$-ME Optimal, where $N = \theta(n \cdot \log d(n))$ is the input size.*

[2] In this view, an algorithm is instance optimal if *for every* new distribution its expected running time given input from this distribution is comparable with any algorithm, even one that "knows" the distribution.

Thus, we show that an algorithm that is optimal on the uniform input distribution stops being optimal when the input distribution deviates from the uniform one by $O(\log N)$ bits of min entropy (where the uniform distribution to the problem has N bits of entropy). The best one could hope for are problems that are not $(N - O(1))$-ME Optimal, since all functions are N-ME Optimal.

Notice that our result is conditional on a lower-bound on the average-case running time required to compute k-Sum by a deterministic algorithm. Brakerski, Stephens-Davidowitz, and Vaikuntanathan [BSV20] show that under worst-case lattice assumptions, k-Sum is hard on average for (randomized) algorithms that run in time $d(n)^{o(1/k)}$. This average-case lower-bound is significantly stronger than is required for our result. Therefore under the hardness of specific worst-case lattice problems, k-Sum is not optimal for large values.

1.2 Future Research Directions

While our work lays the foundation for the study of Min-Entropic Optimality, there remain many open problems. We list a couple here. For more concrete open problems see Section 5.

Natural ME Optimal Problems. In this work we have shown that there exist problems that are $k(n)$-ME Optimal (and not $(k(n)/2)$-ME Optimal). These problems are rather contrived, leaving open the question of finding a "natural" problem that is $o(n)$-ME Optimal.

Question 1.6. *Does there exist a "natural" problem that is $o(n)$-ME Optimal, but not instance optimal?*

We have shown that functions that are very hard in the worst-case cannot be optimal. A natural problem answering Question 1.6 should therefore be at most moderately hard. This rules out problems that are believed to require exponential time in the worst-case such as SAT, CLIQUE, and some cryptographic functions. Furthermore, we have shown that problems that inhibit certain structure and have short witnesses, also cannot resolve the question.

Thus, natural candidates for ME-optimal problems are ones which can be computed efficiently and have large witnesses. An example of one potential candidate is the edit-distance problem.

Alternative Models. There are many meaningful variants of ME Optimality that could be considered. For example, one could relax how competitive the algorithm has to be by changing α in Definition 1.1 to larger than a constant (e.g. $\alpha(n) = \text{polylog}(n)$).

Alternatively, one could consider other subsets of distributions, such as ones that are efficiently sampleable, or distributions with some Shannon entropy (rather than min-entropy). We leave it for future research to study other meaningful notions of optimality.

1.3 Related Work

There has been significant previous research done on different notions of optimality, and on the kinds of problems that we analyze in this paper.

1.3.1 Notions of Optimality

The notion of optimality has been studied in the past in numerous contexts, mostly under the setting of instance optimality. Fagin, Lotem and Naor [FLN03] coined the term instance optimal and provided an algorithm that is instance optimal for finding top k aggregate score in a database under the promise that each column in the database is sorted. Demaine, López-Ortiz and Munro [DLM00] gave instance optimal algorithms for finding intersections, unions, or differences of a collection of sorted sets. Baran and Demaine [BD04] discovered instance optimal algorithms for finding the nearest and furthest points on a curve. Grossman, Komargodski and Naor [GKN20] studied instance optimality in the decision tree model.

Since instance optimality is a very strong requirement, there are many contexts in which it cannot be meaningfully achieved. To deal with this, there are a number of works that have studied related weaker notions. Roughly stated, an algorithm is instance optimal for a function if on *every* input it is competitive with *every* other algorithm (that computes the function correctly) on that input. Notice that there are two "for-all" quantifiers: one on the inputs and one on the algorithms. Past works have relaxed this definition by only requiring that the algorithm be competitive against natural sub-classes of algorithms. This notion is sometimes referred to this as *Unlabeled* Instance Optimality.

Afshani, Barbay and Chan [ABC17] provided unlabeled instance optimal algorithms for finding the convex hull and set maxima. Valiant and Valiant [VV16] discovered an unlabeled instance optimal algorithm for finding an approximation of a distribution given independent samples from it (with the cost function being the number of samples). They later [VV17] gave an unlabeled instance optimal algorithm for the identity testing problem. This is the problem of, given the explicit description of a distribution, deciding whether a set of samples was drawn from it or from a distribution promised to be far from it. Grossman, Komargodski, and Naor [GKN20] studied unlabeled instance optimality in the query model. In particular, they show that the question of whether a function is unlabeled instance optimal can be strongly dependent on the class of algorithms with which an optimal algorithm needs to compete.

The two "for-all" quantifiers. Unlike these works, that relax instance optimality by relaxing the "for-all" quantifier over algorithms (that is, only require an optimal algorithm to compete against specific sub-classes of algorithms), in this work we relax the quantifier over inputs. To be ME Optimal an algorithm is still required to perform as well as any other algorithm that is always correct. However, rather than being required to perform optimally on every input, ME Optimality requires that it perform optimally on a certain class of input distributions.

On k-Sum. In this work, we analyze the optimality of k-Sum. The hardness of this problems has been analyzed thoroughly in prior work.

k-Sum has been conjectured to be hard (in the worst case) for (randomized) algorithms that run in time $n^{o(k)}$, and this is backed by the exponential time hypothesis [PW10]. Furthermore, solving k-Sum in running time $d(n)^{1-\epsilon}$ for any constant $\epsilon > 0$, would refute SETH [ABHS19]. For average case lower bounds, Brakerski et al. [BSV20] show that k-Sum in the dense regime is hard for algorithms running in time $d(n)^{o(1/k)}$ under worst-case lattice assumptions.

Average-Case Complexity. Min-Entropic Optimality can be seen as combining the concept of instance optimality with Levin's theory of average-case complexity [Lev86], where the complexity of an algorithm is measured as its expected running time when the input is chosen from a given distribution. Saying that an algorithm is n-ME Optimal is equivalent to, in Levin's framework, saying that it is the algorithm with the smallest average-case complexity when the input distribution is the uniform one.

1.4 Paper Organization

In Section 2 we formally define ME Optimality, and prove some basic properties regarding the definition. Section 3 is split into two parts: In Section 3.1 we show that worst-case hardness and (sub-)optimality are inherently linked. Next, in Section 3.2 we show that that there exist ME Optimal functions. In Section 4 we prove a lemma showing that problems with short witnesses which have structural symmetry do not have optimal algorithms thereby proving Theorem 1.5. In Section 5 we discuss problems we leave open and future research directions.

2 Introducing Min Entropic Optimality

In Section 2.1 we introduce the computational model used in this paper, Non-uniform RAM. Next, in Section 2.2, we formally define the notion of Min-Entropic Optimality and show some basic properties of it. Finally, in Section 2.3 we discuss why Non-uniform RAM is the model of choice for Min-Entropic Optimality.

2.1 The Computational Model

The computation model we use for this work is the "Non-uniform RAM" Model.[3] A non-uniform RAM machine is a Turing machine with two types of tapes: sequential access tapes and direct access tapes. A sequential access tape can only be accessed sequentially, whereas any index in a direct access tape can be accessed in $O(1)$ time. Running the machine on an input is as follows: The input x is placed on a direct access tape of the machine and an advice string a (which is determined only by $|x|$) is placed on a sequential access tape. The machine additionally has a random-access work tape. The machine then computes its output in the regular way using these tapes. Notice, crucially, that for a machine to read the i'th bit of its advice string it must take $\Omega(i)$ time whereas reading the i'th bit of any other tape takes time $O(1)$.

We henceforth simply refer to a Non-Uniform RAM machine as an algorithm. We would like to emphasize that although this is the model we chose to work with, all the results in this paper apply with any other natural model (such as Turing machines that take advice), up to small changes in parameters.

2.2 Min-Entropic Optimality

In this section we formally define $k(n)$-Min Entropic Optimality. We start with some notation.

Notation 2.1 (Running time of A on input x)**.** *Let A be a non-uniform RAM machine. For an input $x \in \{0,1\}^n$ we let $\mathsf{Time}_A(x)$ denote the running time of A on input x.*

Definition 2.2 (Min-Entropy)**.** *Let D be a distribution over $\{0,1\}^\ell$. The min-entropy of D is defined as $H_\infty(D) = -\log\{\max_{d\in\{0,1\}^\ell} \Pr[D = d]\}$.*

Below we define Min-Entropic Optimality:

[3] Non-uniform RAM is also sometimes referred to as "random-access Turing machine".

Definition 2.3 (Min-Entropic Optimality)**.** *A Non-Uniform RAM machine A is $k(n)$* Min-Entropic Optimal *($k(n)$-ME Optimal) for a sequence of functions $\mathcal{F} = \{f_n\}_{n\in\mathbb{N}}$ if:*

- *A correctly computes $\mathcal{F}$ on all inputs.*
- *There exists $n_0 \in \mathbb{N}$ and a universal constant $\alpha \in \mathbb{N}$ such that, for every $n > n_0$, every distribution $\mathcal{D}_n$ over $\{0,1\}^n$ with $H_\infty(\mathcal{D}_n) \geq k(n)$, and every algorithm B that correctly computes $\mathcal{F}$ on all inputs it is true that*

$$\mathop{\mathbb{E}}_{x\leftarrow\mathcal{D}_n}[\mathsf{Time}_A(x)] \leq \alpha \mathop{\mathbb{E}}_{x\leftarrow\mathcal{D}_n}[\mathsf{Time}_B(x)]$$

If A is not $k(n)$-ME Optimal we say that it is $k(n)$-ME Sub-Optimal.

Definition 2.4. *We say a function $\mathcal{F} = \{f_n\}_{n\in\mathbb{N}}$ is $k(n)$*-ME Optimal, *if there is an algorithm A evaluating $\mathcal{F}$ that is $k(n)$-ME Optimal. Similarly, we say that $\mathcal{F} = \{f_n\}_{n\in\mathbb{N}}$ is $k(n)$*-ME Sub-Optimal *if there is no algorithm A that is $k(n)$-ME Optimal for $\mathcal{F}$.*

Definition 2.3 generalizes immediately from functions to relations (i.e. search problems where there is more than one legal solution) over larger alphabets. In this case we replace the requirement on the algorithms A and B from needing to compute $\mathcal{F}$ correctly on all inputs to needing to return a valid solution for every input that has one and $\perp$ for every input that has no valid solutions.

Notice that $k(n)$-ME Sub-Optimality is an "infinitely-often" notion (i.e. for every algorithm A there are infinite input lengths n for which there exists an algorithm B that is significantly faster than it). In fact, in this paper, all of our sub-optimality results are stronger and show "almost-everywhere" sub-optimality. Thus, this distinction will not be important throughout this paper.

The following definition of Min Entropic Optimality is equivalent and, in practice, often easier to use.

Definition 2.5 (Alternative Definition of Min-Entropic Optimality)**.** *An algorithm A is $k(n)$* Min-Entropic Optimal *($k(n)$-ME Optimal) for a sequence of functions $\mathcal{F} = \{f_n\}_{n\in\mathbb{N}}$ if:*

- *A correctly computes $\mathcal{F}$ on all inputs.*
- *There exists $n_0 \in \mathbb{N}$ and a universal constant $\alpha \in \mathbb{N}$ such that, for every $n > n_0$, every set of inputs $S \subseteq \{0,1\}^n$ satisfying $|S| \geq 2^{k(n)}$, and every algorithm B that correctly computes $\mathcal{F}$ on all inputs, it is true that*

$$\mathop{\mathbb{E}}_{x\leftarrow S}[\mathsf{Time}_A(x)] \leq \alpha \mathop{\mathbb{E}}_{x\leftarrow S}[\mathsf{Time}_B(x)]$$

The equivalence of the two definitions follows from the following fact:

Fact 2.6 ([Vad12] Lemma 6.10)**.** *Let D be a distribution over $\{0,1\}^n$ with $H_\infty(D) \geq k$. Then $D = \sum_i p_i D_i$, where $0 \leq p_i \leq 1$, $\sum_i p_i = 1$, and $\{D_i\}_i$ are uniform distributions over a set of size 2^k in $\{0,1\}^n$.*

Lemma 2.7. *Definitions 2.3 and 2.5 are equivalent for any $k(n) \in \mathbb{N}$.*

Proof. Every function satisfying Definition 2.3 also satisfies Definition 2.5 since the uniform distribution over a set of size $2^{k(n)}$ has min-entropy $k(n)$. We now show that the other direction is also true. Let $\mathcal{F} = \{f_n\}_{n\in\mathbb{N}}$ be a function that satisfies Definition 2.5, and let A be its optimal algorithm. Now, let D be some distribution over $\{0,1\}^n$ with min-entropy $k(n)$. By Fact 2.6 we can write $D = \sum_i p_i D_i$ where $0 \leq p_i \leq 1$, $\sum_i p_i = 1$ and D_i are uniform distributions over sets of size $2^{k(n)}$ in $\{0,1\}^n$. Notice that by the linearity of expectation, for every algorithm M, $\mathbb{E}_{x\leftarrow D}[\mathsf{Time}_M(x)] = \sum_i p_i \, \mathbb{E}_{x\leftarrow D_i}[\mathsf{Time}_M(x)]$. Therefore for every B computing the same function as A:

$$\mathop{\mathbb{E}}_{x\leftarrow D}[\mathsf{Time}_A(x)] = \sum_i p_i \mathop{\mathbb{E}}_{x\leftarrow D_i}[\mathsf{Time}_A(x)] \leq \alpha \sum_i p_i \mathop{\mathbb{E}}_{x\leftarrow D_i}[\mathsf{Time}_B(x)] = \alpha \mathop{\mathbb{E}}_{x\leftarrow D}[\mathsf{Time}_B(x)]$$

Where the inequality is true because A is optimal in the sense of Definition 2.3, and because the distributions D_i are uniform over sets of size $2^{k(n)}$. □

ME Optimality can be seen as a generalization of instance optimality, defined below:

Definition 2.8 (Instance Optimality [FLN03])**.** *An algorithm A is* Instance Optimal *for a sequence of functions $\mathcal{F} = \{f_n\}_{n\in\mathbb{N}}$ if:*

- *A correctly computes $\mathcal{F}$ on all inputs.*
- *There exists $n_0 \in \mathbb{N}$ and a universal constant $\alpha \in \mathbb{N}$ such that, for every $n > n_0$, every input x, and every algorithm B that correctly computes $\mathcal{F}$ on all inputs, it is true that*

$$\mathsf{Time}_A(x) \leq \alpha \mathsf{Time}_B(x)$$

Below are a number of simple observations about ME Optimality:

Observation 2.9. *The following statements are true:*

1. *Every function is n-ME Optimal.*
2. *If a function is $k(n)$-ME Optimal it is also $k'(n)$-ME Optimal for all $k'(n) > k(n)$.*
3. *A function is 0-ME Optimal if and only if it is instance optimal.*

2.3 On Choosing a Computational Model for ME Optimality

The computer science literature is riddled with different computational models, each with its own pros and cons such as Turing machines, non-uniform circuits, non-deterministic machines, RAM machines, etc. At a high level, the goal of this paper is to propose a way to compare an algorithm against all possible other algorithms for the same function and say that it is "the best". It is therefore important to to define "algorithm" in a sensible and reasonable way to achieve our goals.

The notion of ME Optimality seems to necessitate considering non-uniform models of computation. This is because we want to compare an algorithm with "all other possible algorithms" and in particular those that hard-code information about the input distribution into their structure. This concept of hard-coding information is inherently non-uniform.

The Non-Uniform RAM model is an intuitive and natural notion. We believe that this model is best suited for comparing algorithms. Furthermore, algorithms are generally analyzed in the

RAM model, and thus we do not deviate from standard methods of analysis. It also covers our wish for the machine to be non-uniform. Furthermore, since the advice is written on a sequential tape, we have that while an algorithm could "hard-code" much information into its advice string, it will cost it to access this information. Notice that the model becomes trivial and uninteresting for time-analysis if we were to change the advice tape to being direct access - everything could be computed in $O(|x|)$ time by placing the entire truth table of the function being computed in the advice and simply "jumping" to the right spot and outputting the solution written there.

Given that we are in the non-uniform regime, standard (non-uniform) circuits may seem like a good model. But alas, we wish to compare algorithms on an input by input basis. With circuits, the complexity measure is generally the size of the circuit, which is dependent only on the length of the input, and not on the actual input.

We further note that our model is nearly equivalent to other natural non-uniform models. In particular this model is equivalent to Turing machine with advice with constant number of tapes up to polylog factors (see eg [Tou01]). [PF79] showed that every Turing machine can be simulated by an oblivious Turing machine with logarithmic overhead, and thus every Turing machine with running time S can be simulated by a circuit of size $S \log S$ (see e.g [AB09] Thm 6.6). A circuit of size S can be simulated by a (constant tape) Turing machine with running time $S \log S$: This is done by encoding the circuit as advice, and since a graph on e edges and v vertices requires $2e \log v$ bits to encode, the size of the advice will be $S \log S$ (see discussion under Definition 1.12 in [Gol08] for more detail). Given this advice, there exists a multi-tape Turing machine that simulates the encoded circuit that runs in time $\Theta(S \log S)$. Thus, in the worst case, all these models are equivalent up to polylog factors.

3 Do Min-Entropic Optimal Functions Exist?

When defining a new definition, such as ME-Optimality, two immediate questions come up:

1. Do there exist sub-optimal functions?

2. Do there exist optimal functions?

We answer both questions in the affirmative, for a wide range of values $k(n)$.

3.1 Hard Functions are Sub-Optimal

We start by showing that hard functions (in the worst case) are not optimal.

Lemma 3.1. *Let $f : \{0,1\}^* \to \{0,1\}$ be a function with worst-case complexity $t(n) \in \omega(n)$. Then f is $\log(h(n))$-ME Sub-optimal for any $h \in o\left(\frac{t(n)}{n}\right)$.*

We prove this by noting that any algorithm computing a function that is hard on the worst-case must have some large "hard set" on which its running time is high. If we then restrict the input to only this set, its expected running time must be high. On the other hand a competing algorithm can have this specific set "hard-coded" into its description thus allowing it to have small expected running time on the same set.

Proof. We first show that any algorithm that evaluates f must have at least $\Omega(t(n)/n)$ instances with running time $\Omega(t(n))$. Suppose towards contradiction that there exists a deterministic algorithm A that evaluates f correctly, and for which the set S of inputs of length n on which A runs in time $\Omega(t(n))$ is such that $|S| \in o(t(n)/n)$. We will use this set S to construct an algorithm B for computing f with worst-case running time smaller than $o(t(n))$, thus contradicting the assumption on f. Consider the algorithm B, that receives as advice the set S of all the hard instances for A along with their solutions. This advice is of length $O(n|S|)$. On input $x \in \{0,1\}^n$ the algorithm searches whether $x \in S$. If so, it returns whatever result is written in the advice. Otherwise it returns $A(x)$. For every $x \in S$, the running time of B on x is $O(n|S| + n)$ since B must read the input x, and compare it to the strings in its advice until it finds to correct solution. The worst case running time of B on inputs $x \in \{0,1\}^n \backslash S$ is $O(n|S| + n) + o(t(n))$. This is because it first reads the input and scans its entire advice, taking time $O(n + n|S|) \in O(n|S|)$ and then executes $A(x)$. Since $x \notin S$, by the definition of S we have that $A(x)$ runs in time $o(t(n))$. Thus the worst-case running time of B is $O(n|S|) + o(t(n))$. Recalling that $|S| \in o(t(n)/n)$ we get that this is $o(t(n))$ contradicting the assumption that f has worst case running time $\Omega(t(n))$.

Let $h \in o(t(n)/n)$. We now show that f is $\log(h(n))$-ME Sub-optimal. Fix an algorithm A that computes f correctly. Let $S = \{x | \mathsf{Time}_A(x) \in \Omega(t(n))\}$. As shown above, $|S| \in \Omega(t(n)/n)$. Let T be a subset of S such that $|T| = h(n)$. By definition:

$$\mathop{\mathbb{E}}_{x \leftarrow T}[\mathsf{Time}_A(x)] \in \Omega(t(n))$$

We define B similarly to before, but now with respect to the set T. That is, B receives all the elements of T and their solutions as its advice. On an input, it first checks whether a solution exists in the advice. If one does it answers with whatever the advice says and otherwise runs A on the input and returns whatever A returned. We have that

$$\mathop{\mathbb{E}}_{x \leftarrow T}[\mathsf{Time}_B(x)] \in O(n|T|) \equiv O(n \cdot h(n))$$

Since $h \in o(t(n)/n)$, this implies that $\mathbb{E}_{x \leftarrow T}[\mathsf{Time}_B(x)] \in o(t(n))$. And thus for every algorithm A, there exists a set T, such that $|T| = h(n)$ and algorithm B that outperforms A on T, meaning that f is $\log(h(n))$-ME Sub-optimal.

□

Remark 3.2. *There exist concrete functions which are hypothesised to be hard on the worst-case. 3-SAT is one such example under the non-uniform exponential time hypothesis. Therefore if non-uniform ETH is true, then 3-SAT is $(n - o(n))$-ME Sub-optimal.*

Later on, in Section 4 we argue that natural problems such as $\sum$ are sub-optimal for much larger values of $k(n)$ than implied by Lemma 3.1.

3.2 A Hierarchy of ME Optimal Functions

Next we explore the question asked in the title of the section:

Does there exist a function that is $o(n)$-ME Optimal, but is not Instance Optimal?

We answer the above question positively. Specifically we show a hierarchy of problems which are $k(n)$-ME Optimal but not $g(n)$-ME Optimal with $g(n) \ll k(n)$ for all sufficiently large $k(n)$. See theorem 3.3 for the precise value of $g(n)$ and precise requirements for $k(n)$.

We describe our construction below. In the following we will always use the alternative, equivalent, definition of ME Optimality using sets (Definition 2.5).

Our construction combines the XOR function, which is instance optimal with linear complexity, with a problem that is hard to compute in the worst case. Let g be a function that has worst-case running time of $2^{n/2}$. Such a function exists due to the non-uniform hierarchy (e.g. [AB09] Thm 6.17).[4] We combine g with the XOR function as follows: If the first $n - k(n)$ bits are all equal to 0, then output the value of g on the remaining bits. Otherwise, compute the XOR of all the bits.

We call an input Type (1) if it does not begin with $n - k(n)$ zeroes (i.e. the XOR is computed). We can show that computing the answer to Type (1) inputs requires time $\Theta(n)$. We refer to inputs that do begin with $n - k(n)$ zeroes as Type (2) inputs. Such inputs require $2^{k(n)/2}$ running time in the worst-case. The number of Type (2) inputs is exactly $2^{k(n)}$, and thus the number of Type (1) inputs greatly outnumbers the number of Type (2) inputs.

To show optimality, consider a set of inputs of size $2^{2k(n)}$. Since there can be at most $2^{k(n)}$ inputs of Type (2) in the set, the vast majority of the inputs in the set (at least $2^{2k(n)} - 2^{k(n)}$) must be of Type (1). Recall that ME Optimality considers the expected running time on an input drawn from the set. The sheer number of Type (1) inputs "drowns out" the running time of the Type (2) inputs. As Type (1) inputs require running time that is at least linear to compute, this results in the fact that every algorithm requires expected time $\Omega(n)$ on every set of this size. Moreover, consider the naive algorithm that on input x either computes its XOR (if it is Type (1)) and if it Type (2) computes $g(x)$ by storing and looking up its entire truth table. This algorithm has expected running time $O(n)$ when the input is chosen uniformly at random from any set of inputs of size $2^{2k(n)}$ (since it takes time $O(n)$ for the vast majority of the inputs). Thus this algorithm is $2k(n)$-ME Optimal.

To show sub-optimality, we consider the fact that due to the Type (2) inputs, our function has worst-case running time of $2^{k(n)/2}$. Lemma 3.1 then implies that our function is $\log(h)$-ME Sub-optimal, for any $h \in o\left(\frac{2^{k(n)/2}}{n}\right)$.[5]

As previously mentioned, our construction uses internally a function that is hard to compute in the worst-case. Such functions exist unconditionally by the non-uniform hierarchy theorem for worst-case running time (converted to our setting of non-uniform RAM machines). While this is unconditionally true, it is non-constructive. To get an explicit function we can replace the non-uniform hierarchy function by a concrete problem given computational assumptions. Examples of natural functions for this purpose include 3-SAT, which is exponentially hard in the worst case under the (non-uniform) exponential time hypothesis. Another useful example is that of sub-exponentially secure one-way functions.

We remark that our construction does not require exponential hardness, but the easier the function, the larger the gaps in the hierarchy, and the starting point of the hierarchy is larger.

Theorem 3.3. *For every $k(n) \in \omega(\log(n))$ there exists a function that is $k(n)$-ME Optimal and $\log(h)$ sub-optimal for any $h \in o\left(2^{k(n)/c}\right)$ for any constant $c > 1$.*

[4] The value $2^{n/2}$ is rather arbitrary and is simply used here to make this overview simpler.

[5] The actual proof is slightly more involved so as to achieve better parameters. Specifically we achieve $h \in o\left(\frac{2^{k(n)/2}}{k(n)}\right)$.

Our result above is unconditional but non-constructive. Given some computational assumptions we can turn the theorem above to give us an explicit function. [6]

Theorem 3.4. *Assuming non-uniform ETH, there exists an explicit function, f, where for every $k(n) \in \omega(\log(n))$ f is $k(n)$-ME Optimal and $\log(h)$-ME Sub-Optimal for any $h \in o\left(2^{k(n)/c}\right)$ for some constant $c > 1$.*

Both the above theorems follow from Lemma 3.5.

Proofs Of Theorems 3.3 and 3.4. We start by proving theorem 3.3. By the non-uniform hierarchy theorem ([AB09] Thm 6.17) there exists a function that requires circuits of size $\frac{2^n}{200n}$.[7] Since the cost of simulating a circuit given a RAM with advice running in time t is $t\text{polylog}(t)$, there also exists functions that requires RAM with advice with running time $2^{n/c}$ for every $c > 1$ to evaluate in the worst-case. Plugging this into Lemma 3.5 with $k'(n) = k(n)/2$ as the optimality parameter we get theorem 3.3.

The proof of theorem 3.4 is identical to that of theorem 3.3 except that we use 3-SAT as an explicit hard function. Since we assume non-uniform ETH, 3-SAT takes $2^{\delta n}$ time to decide in the worst-case for some constant $\delta > 0$. Since the cost of simulating a circuit given a RAM with advice running in time t is $t\text{polylog}(t)$, then assuming non-uniform ETH solving 3SAT requires time $2^{\delta' n}$ for a RAM with advice machine. Plugging this into Lemma 3.5 we get theorem 3.4.

□

Lemma 3.5. *Let $g : \{0,1\}^* \to \{0,1\}$ be a function whose worst-case running time on inputs of length n is $t(n)$. Then for every k such that $t(k(n)) \in \omega(n)$ the following function $f : \{0,1\}^* \to \{0,1\}$ is $2k(n)$-ME Optimal[8] and $\log(h)$-ME Sub-optimal for any $h \in o\left(\frac{t(k(n))}{k(n)}\right)$.*

On input $x \in \{0,1\}^n$:

(a) If $x = (0^{n-k(n)}, y)$ then output $g(y)$ (i.e. if the first $n - k(n)$ bits of x are all 0 then output g applied to its $k(n)$ remaining bits).

(b) Else, output the XOR of all the bits of x: $\oplus_{i=1}^{n} x_i$.

Proof Of Lemma 3.5. In the following, we show that f is $\log(h)$-ME Sub-optimal for any $h \in o(t(n)/n)$ and $2k(n))$-ME Optimal in two parts. Claim 3.6 shows that f is $\log(h)$-ME Sub-optimal. In Claim 3.7 we show that it is $2k(n)$-ME Optimal.

Claim 3.6. *f is $\log(h(n))$-ME Sub-optimal for any $h \in o\left(\frac{t(k(n))}{k(n)}\right)$*

Proof. Fix an algorithm A. Consider inputs that begin with $n - k(n)$ zeroes. The function has worst-case running time of $t(k(n))$ due to the assumption of g. Identically to Lemma 3.1, there must be $\Omega\left(\frac{t(k(n))}{k(n)}\right)$ instances that take A running time $\Omega(t(k(n)))$. Thus pick a set T, satisfying $|T| \in o\left(\frac{t(k(n))}{k(n)}\right)$ such that $\mathsf{Time}_A(x) \in \Omega(t(k(n)))$ for each $x \in T$. Let $a_{k(n)}$ be the advice containing

[6] Note that the suboptimality result in the non-constructive, non-conditional result in theorem 3.3 is for *any* constant $c > 1$, while the constructive, and conditional supoptimality result as in theorem 3.4 is for *some* constant $c > 1$.

[7] Due to [Lup70] there are no functions that require much more than $\frac{2^n}{200n}$ time to compute.

[8] The precise value is $(2 - o(1))k(n)$.

the truth table of g for inputs in T (only the last $k(n)$ bits). Let B be the algorithm that checks that the input begins with all 0's, if it does, it checks if x is in the advice. If x is in the advice, it outputs what the advice says and otherwise it does the same as A.

$$\mathop{\mathbb{E}}_{x\leftarrow T}[\mathsf{Time}_B(x)] = n + (k(n)+1)|T| \in o\left(n + k(n)\frac{t(k(n))}{k(n)}\right)$$

Since $t(k(n)) \in \omega(n)$, this implies that $\mathbb{E}_{x\leftarrow T}[\mathsf{Time}_B(x)] \in o(t(k(n)))$.

□

Claim 3.7. *The function f is $2k(n)$-ME Optimal.*

Proof. By Lemma 2.7, in order to show that a function in optimal, it suffices to propose an algorithm A that correctly computes f on all inputs such that for every B that also computes f correctly and every set $S \subseteq \{0,1\}^n$ of size $2^{2k(n)}$:

$$\mathop{\mathbb{E}}_{x\leftarrow S}[\mathsf{Time}_A(x)] \in O(\mathop{\mathbb{E}}_{x\leftarrow S}[\mathsf{Time}_B(x)])$$

We show this in two stages. First we show an algorithm A such that $\mathbb{E}_{x\leftarrow S}[\mathsf{Time}_A(x)] \in O(n)$ for every set S of size $2^{2k(n)}$. Then we conclude the proof by showing that for every algorithm B, that computes f correctly on every input, it must be that $\mathbb{E}_{x\leftarrow S}[\mathsf{Time}_B(x)] \in \Omega(n)$ for every set S of size $2^{2k(n)}$.

The optimal algorithm A is described below:

- The algorithm has as advice the entire truth table of g for inputs of size $k(n)$.
- On input $x \in \{0,1\}^n$:
  1. If $x = (0^{n-k(n)}, y)$ return $f(x) = g(y)$ by returning the entry for y in the truth table of g.
  2. Otherwise read the entire input and output its XOR.

It is immediate that A computes f correctly on every input. We need to show that its expected running time is not too high. Fix some set $S \subseteq \{0,1\}^n$ of size $2^{2k(n)}$. We partition S into two parts. The first which we denote by S_1 are all values x that begin with $n-k(n)$ zeroes, and hence their value is computed by line (1) of the definition of A. S_2 are all the remaining values of S (these values are computed by line (2) of A). Notice that A computes every element in S_1 in time $O(n2^{k(n)})$: It reads the entire input and then goes over all of its advice (which has size $n2^{k(n)}$) until it reaches the correct output. Every element in S_2 is computed in $O(n)$ time. Thus, for a constant $c \in \mathbb{N}$ we have:

$$\begin{aligned}\mathop{\mathbb{E}}_{x\leftarrow S}[\mathsf{Time}_A(x)] &= 2^{-2k(n)} \cdot \left(\sum_{x\in S_1} \mathsf{Time}_A(x) + \sum_{x\in S_2} \mathsf{Time}_A(x)\right) \\ &\leq 2^{-2k(n)} \cdot c \cdot \left(n2^{k(n)}|S_1| + n|S_2|\right) \qquad (3.1) \\ &\leq 2^{-2k(n)} \cdot c \cdot \left(n2^{k(n)} \cdot 2^{k(n)} + n \cdot 2^{2k(n)}\right) \qquad (3.2) \\ &= 2cn\end{aligned}$$

Where the constant c that appears in eq. (3.1) comes from the fact that computing elements in S_1 takes time $O(n2^{k(n)})$ rather than exactly this value. Equation (3.2) is true since $|S_1| \leq 2^{k(n)}$ (there are only $2^{k(n)}$ values in $\{0,1\}^n$ that begin with $n-k(n)$ zeroes), and $|S_2| \leq 2^{-2k(n)}$ (since $|S_2| \leq |S| = 2^{2k(n)}$). Hence, we have that on every set S of size $2^{2k(n)}$, $\mathbb{E}_{x \leftarrow S}[\mathsf{Time}_A(x)] \in O(n)$ as claimed.

We now turn towards showing that for every algorithm B that correctly computes f on every input and every set $S \subseteq \{0,1\}^n$ of size $2^{2k(n)}$ it must be that $\mathbb{E}_{x \leftarrow S}[\mathsf{Time}_B(x)] \in \Omega(n)$. Intuitively this is due to two facts: (1) For any set S of size $2^{2k(n)}$ the vast majority of $x \in S$ will *not* begin with $n-k(n)$ zeroes since there are only $2^{k(n)}$ such strings. Hence the *expected* running time of the algorithm will be dominated by the running time of the elements that do not begin with zeroes. Fact (2) is that for any string x that does not begin with zeroes, one must read all of x in order to compute $f(x) = \oplus_{i=1}^{n} x_i$, and so computing the correct result on these inputs takes linear time.

We begin by showing that every B that correctly computes f must run in time at least $n-2$ on every x that does not begin with $n-k(n)$ zeroes. Fix such an x. Suppose towards contradiction that B runs in time smaller than $n-2$ when given x as input. We show that there necessarily exists a different string x' on which B errs. Since B runs in time less than $n-2$, there exist two distinct indices such that B does not read x at these locations. We show that there exists a new string x' such that (a) x' differs from x only in bits that B does not read (b) Bitwise-XOR$(x') =$ $1 -$ Bitwise-XOR(x) and (c) x' does not begin with $n-k(n)$ zeroes. This together implies that $f(x) = 1 - f(x')$, but that $B(x) = B(x')$ in contradiction to the assumption that B computes f correctly on all inputs. Let i and j be the indices of x that B does not read and assume $i < j$. Define x' as the string that is equal to x except that:

1. If $x_i = 0$ then set $x'_i = 1$.
2. Otherwise set $x'_j = 1 - x_j$

Requirement (a) is true since x' is equal to x except potentially for the bits in locations i and j which are by definition not read by B. Requirement (b) holds since x' is exactly x except with either (and not both) x_i or x_j flipped. Next for requirement (c):

1. If $x_i = 0$ then it is impossible that flipping this bit to 1 causes x' to begin with all zeroes.
2. Otherwise $x_i = 1$:
   (a) If $j > n - k(n)$: Requirement (c) is met since by assumption x does not begin with $n-k(n)$ zeroes, and we are not changing the prefix of x.
   (b) Otherwise, it must be that $i < n-k(n)$ (since $i < j$ and $j \leq n-k(n)$). Since $x_i = 1$ and x_i remains unchanged, the prefix does not begin with all 0s.

Thus for inputs that do not begin with $n-k(n)$ zeroes, B must run in time at least $n-2$. We now use this to bound the expected running time of any algorithm on any large enough set. Fix some algorithm B that correctly computes f on inputs of length n and fix some set $S \subseteq \{0,1\}^n$ of size $2^{2k(n)}$. As before, let $S_1 \subseteq S$ be the set of all elements that begin with $n-k(n)$ zeroes and

$S_2 = S \backslash S_1$ be the rest of the elements of S. Then:

$$\begin{aligned}
\mathop{\mathbb{E}}_{x \leftarrow S}[\mathsf{Time}_B(x)] &= 2^{-2k(n)} \cdot \left(\sum_{x \in S_1} \mathsf{Time}_B(x) + \sum_{x \in S_2} \mathsf{Time}_B(x) \right) \\
&\geq 2^{-2k(n)} \cdot \sum_{x \in S_2} \mathsf{Time}_B(x) \\
&\geq 2^{-2k(n)} \cdot |S_2| \cdot (n-2) \qquad (3.3) \\
&\geq 2^{-2k(n)} \cdot \left(2^{2k(n)} - 2^{k(n)} \right) \cdot (n-2) \qquad (3.4) \\
&= (n-2) \left(1 - 2^{-k(n)} \right) \\
&\in \Omega(n) \qquad (3.5)
\end{aligned}$$

Where eq. (3.3) is true because, as previously established, every element in S_2 takes B time at least $n-2$. Equation (3.4) is correct because $S_2 = S \backslash S_1$, $|S| = 2^{2k(n)}$ and $|S_1| \leq 2^{k(n)}$. Finally, eq. (3.5) is true since $k(n) \geq 1$. □

□

4 Stronger Sub-optimality For Structured Relations

In Section 3, we showed that the worst case hardness of problems is inherently tied to the sub-optimality of these problems. In section 3.1 we showed that the hardness of a function directly gives parameters for $k(n)$ for which the function is sub-optimal, and in section 3.2 we showed a hierarchy where as the worst case hardness of the function increased, the value where the function switches from being sub-optimal to being optimal also increased. In this section we show that optimality isn't always directly related to the worst case hardness of the function. We show that some natural search problems that are solvable in polynomial time are sub-optimal for large values of $k(n)$. Informally, these are problems that have some "structural symmetry" and have short witnesses. In the following our results will be described in the more general language of relations, rather than functions. That is, the problem is a relation $R \subseteq \{0,1\}^n \times \{0,1\}^m$ and "solving" the relation amounts to, on input x finding some w such that $(x, w) \in R$.

On a very high level, if a relation has short witnesses, then there must be many inputs that share the same witness. Due to the symmetry of the relation at hand, each witness will have the same number of inputs for which it is a witness. Therefore an algorithm that "knows" that the input will come from a set of inputs that all share the same witness, w, can compute the function on these inputs in roughly the verification time.

In more detail, consider a relation $R \subseteq \{0,1\}^n \times \{0,1\}^m$ for $m \ll n$. Suppose that finding a solution for x takes time at least t in expectation (over the uniform distribution) and that there exists an algorithm V that runs in time $t' \ll t$ such that $V(x, w) = 1$ if and only if $(x, w) \in R$ (i.e the complexity of verifying a solution given any witness is much smaller than the complexity of finding the witness). We now partition the inputs into sets, where all inputs in a given set share a witness. Since we have a set for each witness the total number of sets we have will be 2^m. Due to the assumed properties of R, all our sets will be of nearly identical size. Thus all the inputs in the relation will split evenly among all the 2^m sets. By a simple averaging argument, this implies that there exists some w such that the expected running time of A when inputs are sampled from the set corresponding to instances with the witness w is at least t. Now consider the algorithm B that on input x computes $V(x, w)$ and returns w if V returned TRUE. Otherwise, it finds a solution to R by going over every possible other witness and verifying until V finally returns TRUE. Then B runs in time $t' \ll t$ on every input in the set of instances that correspond with w. B therefore "beats" the running time of any A on this set.

We begin by giving some notation that will allow us to describe our results. This includes simple notation about relations and notation which will allow us to discuss the expected running time of algorithms on the subset of inputs to our problems that have witnesses in the relation.

Notation 4.1. *Let $R \subseteq \Sigma_1 \times \Sigma_2$ be a relation.*

- $R_{|X} = \{x | \exists w \text{ where } (x, w) \in R\}$.
- $T^R_{avg} = \min_A \mathbb{E}_{x \leftarrow R_{|X}}[\mathsf{Time}_A(x)]$. *That is, the expected running time of the fastest algorithm that computes R, where the the input is chosen uniformly at random from elements in the relation.*

All our results in this section will only apply for certain types of search problems, in particular, symmetric (d, α) index-witness relations. Such search problems are common, and include many central problems to the field of fine grained complexity such as k-Sum, k-clique and the orthogonal vectors problem.

Definition 4.2 ((Σ, α) Index-Witness Relation)**.** *A relation R is a (Σ, α)* index-witness *relation if $R \subseteq \Sigma^n \times \{0, 1\}^{\alpha \log(n)}$ and for every $((x_1, \ldots, x_n), (i_1, \ldots, i_\alpha)) \in R$ and every $j \neq k$, $i_j \neq i_k$.*

Definition 4.3 (Symmetric (Σ, α) Index-Witness Relation)**.** *Let R be a (Σ, α) index-witness relation. R is* symmetric *if for every $((x_1, \ldots, x_n), (i_1, \ldots, i_\alpha)) \in R$ and every permutation π over $[n]$, $((x_{\pi(1)}, \ldots, x_{\pi(n)}), (\pi(i_1), \ldots, \pi(i_\alpha))) \in R$.*

We now give the main lemma of this section, saying that symmetric index-witness relations that have specific properties are ME Sub-Optimal.

Lemma 4.4. *Let R be a symmetric (Σ, α) index-witness relation. Suppose that*

- *there exists a verifier V (i.e a function that given (x, w) returns TRUE if and only if $(x, w) \in R$) such that for every $(x, w) \in R$, the running time of $V(x, w)$ is in $o(T^R_{avg})$.*
- $|R_{|X}| > 8\binom{n}{\alpha} \log \binom{n}{\alpha}$.

Then R is $\log\left(\left\lfloor \frac{|R_{|X}|}{2n^\alpha} \right\rfloor\right)$-ME Sub-optimal.

Remark 4.5. *Even if the relation isn't symmetric if the input can be divided into sets where each set has the same witness the function will be sub-optimal for the size of the smallest set.*

Proof. Fix $n \in N$. First note that there are at most $z = \binom{n}{\alpha}$ different witnesses for R. This is because R is a (Σ, α) index-witness relation, and so for every $((x_1, \ldots, x_n), (i_1, \ldots, i_\alpha)) \in R$ and for every $j \neq k$, $i_j \neq i_k$. Let $w_1, \ldots, w_z$ be all the possible witnesses. We partition all the elements in $R_{|X}$ into sets $S_1, ..., S_z$ such that for every $x \in S_i$, $(x, w_i) \in R$.

Next we show that if R is a (Σ, α)-index relation then there exists a partition of the sets where each set is of size at least $|R_{|X}|/2$. On a high level this is due to the fact that our relation is symmetric.

We prove this using the probabilistic method: We show a randomized algorithm that with non-zero probability partitions all the input elements into the sets $S_1, \ldots, S_z$ such that for each γ, $|S_\gamma| > \frac{|R_{|X}|}{2z}$ (recall that $z = \binom{n}{\alpha}$). This implies that such a partition exists. The algorithm is as follows: For every $x \in R_{|X}$ let T_x be the set of indices such that for every $\gamma \in T_x$, $(x, w_\gamma) \in R$. Choose γ uniformly from T_x and place x in set S_γ.

We now show that for every $\gamma \in [z]$ the expected size of S_γ is $\frac{|R_{|X}|}{z}$. Fix γ and $\gamma' \neq \gamma$. Let $\gamma' \neq \gamma$, $w_\gamma = (i_1, \ldots, i_\alpha)$ and $w_{\gamma'} = (i'_1, \ldots, i'_\alpha)$. Let π be a permutation such that $w_{\gamma'} = (\pi(i_1), \ldots, \pi(i_\alpha))$. Such a permutation exists since there are no indices $j \neq k$ such that $i_j = i_k$ and none such that $i'_j = i'_k$. Let $x = (x_1, \ldots, x_n)$ be an element and T_x be all of the witnesses of x as before and suppose that $\gamma \in T_x$. Then since the relation is symmetric for $x' = (x_{\pi(1)}, \ldots, x_{\pi(n)})$ we have that $\gamma' \in T_{x'}$. Moreover, $|T_{x'}| = |T_x|$ by permuting all of the witnesses by π. Hence, the probability that x will be sent to S_γ is identical to the probability that x' will be sent to $S_{\gamma'}$. Since this is true for any γ, γ' and x, the expected size of S_γ is equal to the expected size of $S_{\gamma'}$ for every γ, γ'. Thus by symmetry, the expectation is equal to the number of elements divided by the number of possible sets, i.e. $\frac{|R_{|X}|}{z}$.

Fix $\gamma \in [z]$. Notice that the choice of whether some input x is in S_γ is independent of the choice on the location of any other input. We will use Hoeffding inequality to bound the probability that $|S_\gamma| < \frac{\mu}{2}$ where $\mu = \frac{|R_{|X}|}{z}$ is the expected number of inputs that will be in S_γ:

$$\Pr\left[|S_\gamma| \leq (1-\epsilon)\mu\right] \leq \exp\{-\epsilon^2 \mu/2\}$$

Setting $\epsilon = 1/2$, we get:

$$\Pr\left[|S_\gamma| \leq \mu/2\right] \leq \exp\{-\mu/8\}$$

We now apply the union bound over all z choices of γ to get that:

$$\Pr\left[\exists \gamma : \ |S_\gamma| \leq \mu/2\right] \leq z \exp\{-\mu/8\}$$

Noting that $\mu = \frac{|R_{|X}|}{z}$ and that $|R_{|X}| > 8z \log z$ we have that:

$$\Pr\left[\forall \gamma : \ |S_\gamma| > \frac{|R_{|X}|}{2z}\right] > 0$$

Recalling that $z = \binom{n}{\alpha}$, the size of the every S_γ is then at least $\left\lfloor \frac{|R_{|X}|}{2\binom{n}{\alpha}} \right\rfloor \geq \left\lfloor \frac{|R_{|X}|}{2n^\alpha} \right\rfloor$ that is the total number of elements divided evenly into all the sets.

Fix any algorithm A. By the probabilistic method, there must exist a set S_γ where $\mathbb{E}_{x \leftarrow S_\gamma}[\mathsf{Time}_A(x)] \geq T^R_{\text{Avg}}$.

Define B to be the algorithm that has as advice w_γ, and recall that w_γ is a witness for every $x \in S_\gamma$. The algorithm checks if indeed w_i is a witness to the input by running $V(x, w_i)$. If indeed $(x, w_\gamma) \in R$, B outputs w_γ, and otherwise it computes the function using brute force (i.e. it stores the entire truth table as advice after the witness).

Notice that:

$$\mathop{\mathbb{E}}_{x \leftarrow S_\gamma}[\mathsf{Time}_B(x)] \leq \max_x \max_w V(x, w) \in o(T^R_{\text{Avg}})$$

Where the last part of the above equation is due to the assumption in the lemma statement. Recall that the expected running time of A, when inputs are drawn uniformly from the set S_γ, is T^R_{Avg}. Thus we have shown that algorithm B runs in time that is significantly faster than A when inputs are drawn uniformly from the set S_γ. The relation is therefore $\log\left(\left\lfloor\frac{|R_{|X}|}{2n^\alpha}\right\rfloor\right)$-ME Sub-optimal. □

While the lemma above seems to require much of the relation at hand, there are natural problems that have this property. One such example is the k-Sum problem defined below:

Definition 4.6 (k-Sum Problem)**.** *The k-Sum problem with parameter an integer d, k-Sum$_d$ is: Given n integers in $[-d, d]$ find k numbers that sum up to zero over $\mathbb{Z}$. If no such k-tuple exists, output $\perp$.*

Remark 4.7. *Another example of a search problem that satisfies the conditions of Lemma 4.4 is the orthogonal vectors problem. However, $|R_{|x|}|$ for the orthogonal vectors problem is smaller than for k-Sum, resulting in a weaker suboptimality result, and therefore we focus on k-Sum.*

Our next result is using Lemma 4.4 to show that k-Sum is $(N - \log(N))$-ME suboptimal.

Theorem 4.8. *Let k be a constant and d be such that $\frac{1}{4k+2} \cdot \left\lfloor\frac{n}{k(20d+10)^{1/k}}\right\rfloor > 1$. Suppose that for every deterministic A that correctly computes k-Sum$_d$ on every input:*

$$\mathbb{E}_{(x_1,\ldots,x_n)\leftarrow[-d,d]^n}[\mathsf{Time}_A(x_1,\ldots,x_n)] \in \omega(k(\log d + \log n))$$

Then k-Sum$_d$ is $(n\log(2d+1) - O(\log n))$-ME Sub-Optimal.

Proof. Let Σ be the set of integers in the range $[-d, d]$. Notice that $\log|\Sigma| \in \Theta(\log d)$. We show the following:

1. k-Sum$_d$ is a symmetric (Σ, k) index-witness relation.
2. k-Sum$_d$ can be verified in time $O(k(\log d + \log n))$.
3. The number of inputs to k-Sum$_d$ ($R_{|X}$ when $R = k$-Sum$_d$) that have a solution is at least $(2d+1)^{n-1}$. Note that this is larger than $8\binom{n}{\alpha}\log\binom{n}{\alpha} = 8\binom{n}{k}\log\binom{n}{k}$ as required.
4. Assuming that k-Sum$_d$ takes time $\omega(k\log d)$ to compute in expectation on the uniform distribution, $T_{avg}^{k\text{-Sum}_d} = \omega(k\log d)$. Due to our restriction on d and k, $k\log d \in o(n)$. Therefore a gap between verification time and average-case running time is possible.

Putting all of these together with Lemma 4.4 we get that k-Sum$_d$ is sub-optimal for:

$$\log\left(\left\lfloor\frac{(2d+1)^{n-1}}{2n^2}\right\rfloor\right) \geq n\log(2d+1) - O(\log n)$$

In order to prove the above we will use a lemma from [BSV20] showing that for small values of d, random instances of k-Sum$_d$ have solutions with high probability:

Lemma 4.9 ([BSV20], Lemma 3.4)**.** *If $x_1, \ldots, x_n$ are uniformly sampled integers from $[-d, d]$, and E_k is the event that there exist distinct indices $i_1, \ldots, i_k$ such that $x_{i_1} + \cdots + x_{i_k} = 0$ then*

$$\Pr[E_k] \geq 1 - e^{-\alpha}$$

Where $\alpha = \frac{1}{4k+2} \cdot \left\lfloor \frac{n}{k(20d+10)^{1/k}} \right\rfloor$.

We now prove the above items one by one:

1. **Relation Structure:** The input to k-Sum$_d$ consists of n integers of size d, $x_1, \ldots, x_n$. A witness for k-Sum$_d$ is k distinct indices $i_1, \ldots, i_{i_k} \in [n]$ such that the integers $x_{i_1}, \ldots, x_{i_k}$ sum to zero. Therefore k-Sum$_d$ is a (Σ, k) index-witness relation. It is symmetric since for every permutation π over $[n]$, if the numbers $x_{i_1}, \ldots, x_{i_k}$ sum to zero, then given the input $x_{\pi(1)}, \ldots, x_{\pi(n)}$, the numbers in indices $\pi(i_1), \ldots, \pi(i_k)$ sum to zero.

2. **Verification Time:** Given an instance $(x_1, \ldots, x_n)$ and a witness $(i_1, \ldots, i_{i_k})$ verification can be done in time $O(k(\log d + \log n))$: The verifier simply needs to sum all of the numbers in the locations specified by $(i_1, \ldots, i_{i_k})$. These k numbers are of size $O(\log d)$, and so this computation is easily achieved in time $O(k \log d)$.

3. **Relation Size:** We give a lower bound for $|R_{|X}|$, the number of inputs that have at least one k-tuple of numbers that sum to 0. By assumption,

$$\frac{1}{4k+2} \cdot \left\lfloor \frac{n}{k(20d+10)^{1/k}} \right\rfloor > 1$$

   Notice that the total number of possible inputs to k-Sum$_d$ is $(2d+1)^n$, since each of the n numbers is in the range $[-d, d]$. Thus by Lemma 4.9 the total number of inputs in the relation (i.e. that have solutions) is at least:

$$|R_{|X}| \geq (2d+1)^n (1 - e^{-1}) > (2d+1)^{n-1}$$

4. **Average-Case Running Time:** Assume that k-Sum$_d$ takes time $\omega(k \log d)$ to compute in expectation on the uniform distribution. We find a lower bound on $T_{avg}^{k\text{-Sum}_d}$, the average running time of finding a k-tuple of numbers that sum to 0 when the input is drawn from the set of inputs that have a solution. As previously analysed, the probability of sampling a random input with no k-Sum is upper-bounded by e^{-1}. Hence, if k-Sum$_d$ takes time $t \in \omega(k \log d)$ to compute in expectation on the uniform distribution, then $T_{avg}^{k\text{-Sum}_d}$ is at least $\frac{t}{1-e^{-1}} \geq \frac{t}{2} \in \omega(k \log d)$.

□

5 Open Problems

This work raises several open problems, and research directions.

We leave open the question of finding a natural problem that is $o(n)$-ME Sub-Optimal but not instance optimal.

Question 5.1 (Question 1.6 Repeated)**.** *Does there exist a "natural" problem that is $o(n)$-ME Optimal, but not instance optimal?*

The $o(n)$-ME Optimal functions we have been able to construct have linear average-case running time on the uniform input distribution. This is not a coincidence: It is due to the fact that the function constructed in Lemma 3.5 is the XOR function for almost all inputs. This leads to the following question:

Question 5.2. *Does there exist a function that is $o(n)$-ME Optimal which requires average-case running time (on the uniform input distribution) that is $\omega(n)$?*

ME Optimality is relevant in contexts where other complexity measures are used, such as many sub-linear models. Our construction of ME Optimal functions uses heavily the fact that there exists (very) a hard function. It remains open whether there exist problems that are $o(n)$-ME Optimal but not instance optimal in the sub-linear world where everything can be solved in linear time.

Question 5.3. *Does there exist a function that is $o(n)$-ME Optimal and not instance optimal in the decision tree model?*

Our hierarchy only applies when $k(n) \in \omega(\log(n))$. Raising the question if there exists a hierarchy for smaller values of $k(n)$. Finding a language that is optimal for $k(n) \in O(\log n)$ would be a useful first step in this direction.

Question 5.4. *Is there a function that is $O(\log(n))$-ME Optimal but not instance optimal?*

We have shown that "symmetric functions" that have short witnesses are sub-optimal, for much larger values than implied by Lemma 3.1. However there remains many natural functions that don't have short witnesses, leaving open if these relations are ME Optimal for $k(n) \in \omega(\log(n))$

Question 5.5. *For what parameters of $k(n)$ is edit distance ME Optimal?*

Our results in Section 4 are for search problems. There is a known equivalence between search problems and decision problems for average-case complexity [BCGL92]. This raises the question whether such an equivalence is true for Min-Entropic Optimality.

Question 5.6. *Suppose a search problem is $k(n)$-ME Optimal. Does this imply that the corresponding decision problem is also $k(n)$-ME Optimal?*

Acknowledgements

We would like to thank Tamer Mour and Moni Naor for helpful discussions and suggestions on the presentation of our results. We would also like to thank Ofer Grossman and Roei Tell for discussions about our notions and encouraging us to explore these ideas.

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach.* Cambridge University Press, 2009.

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

[ABHS19] Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. Seth-based lower bounds for subset sum and bicriteria path. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 41–57. SIAM, 2019.

[BCGL92] Shai Ben-David, Benny Chor, Oded Goldreich, and Michael Luby. On the theory of average case complexity. *J. Comput. Syst. Sci.*, 44(2):193–219, 1992.

[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.

[BSV20] Zvika Brakerski, Noah Stephens-Davidowitz, and Vinod Vaikuntanathan. On the hardness of average-case k-sum. *CoRR*, abs/2010.08821, 2020.

[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, January 9-11, 2000, San Francisco, CA, USA*, 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 2020, January 12-14, 2020, Seattle, Washington, USA*, pages 56:1–56:38, 2020.

[Gol08] Oded Goldreich. *Computational complexity - a conceptual perspective.* Cambridge University Press, 2008.

[Lev86] Leonid A. Levin. Average case complete problems. *SIAM J. Comput.*, 15(1):285–286, 1986.

[Lup70] O. B. Lupanov. On a method of circuit synthesis. *Journal of Symbolic Logic*, 35(4):593–594, 1970.

[PF79] Nicholas Pippenger and Michael J. Fischer. Relations among complexity measures. *J. ACM*, 26(2):361–381, 1979.

[PW10] Mihai Patrascu and Ryan Williams. On the possibility of faster SAT algorithms. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1065–1075. SIAM, 2010.

[Tou01] Iannis Tourlakis. Time-space tradeoffs for SAT on nonuniform machines. *J. Comput. Syst. Sci.*, 63(2):268–287, 2001.

[Vad12] S.P. Vadhan. *Pseudorandomness.* Foundations and Trends(r) in T. Now Publishers, 2012.

[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.