Variants of Pseudo-deterministic Algorithms and Duality in TFNP

by

Mohammad Hossein Ebtehaj

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2023

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

All the original results in this thesis are joint work with Shalev Ben-David. References are given for any result mentioned or proved in this thesis that is not original.

Abstract

We introduce a new notion of "faux-deterministic" algorithms for search problems in query complexity. Roughly, for a search problem $\mathcal{S}$, a faux-deterministic algorithm is a probability distribution $\mathcal{A}$ over deterministic algorithms $A \in \mathcal{A}$ such that no computationally bounded adversary making black-box queries to a sampled algorithm $A \sim \mathcal{A}$ can find an input x on which A fails to solve $\mathcal{S}$ ($(x, A(x)) \notin \mathcal{S}$). Faux-deterministic algorithms are a relaxation of *pseudo-deterministic* algorithms, which are randomized algorithms with the guarantee that for any given input x, the algorithm outputs a unique output y_x with high probability. Pseudo-deterministic algorithms are statistically indistinguishable from deterministic algorithms, while faux-deterministic algorithms relax this statistical indistinguishability to computational indistinguishability.

We prove that in the query model, every verifiable search problem that has a randomized algorithm also has a faux-deterministic algorithm. By considering the pseudo-deterministic lower bound of Goldwasser et al. [Gol+21], we immediately prove an exponential gap between pseudo-deterministic and faux-deterministic complexities in query complexity. We additionally show that our faux-deterministic algorithm is also secure against quantum adversaries that can make black-box queries in superposition.

We highlight two reasons to study faux-deterministic algorithms. First, for practical purposes, one can use a faux-deterministic algorithm instead of pseudo-deterministic algorithms in most cases where the latter is required. Second, since efficient faux-deterministic algorithms exist even when pseudo-deterministic ones do not, their existence demonstrates a barrier to proving pseudo-deterministic lower bounds: Lower bounds on pseudo-determinism must distinguish pseudo-determinism from faux-determinism.

Finally, changing our perspective to the adversaries' viewpoint, we introduce a notion of "dual problem" $\mathcal{S}^*$ for search problems $\mathcal{S}$. In the dual problem $\mathcal{S}^*$, the input is an algorithm A purporting to solve $\mathcal{S}$, and our goal is to find an adverse input x on which A fails to solve $\mathcal{S}$. We discuss several properties in the query and Turing machine model that show the new problem $\mathcal{S}^*$ is analogous to a dual for $\mathcal{S}$.

Acknowledgements

First and foremost, I would like to express my deepest appreciation to my supervisor, Shalev Ben-David, for his invaluable guidance and unwavering support. Thinking with Shalev is always a very enjoyable process, and his intuitions, ideas, and knowledge have greatly helped my research throughout my Master's studies. Moreover, Shalev has always been very understanding and supportive during my Master's studies, and I am honored to have him as my advisor. I also want to thank the reading committee, Eric Blais and Rafael Oliveira, for reading my thesis and their valuable insights, help, and time. Finally, a general thanks to everyone in the Algorithms and Complexity group at the University of Waterloo whom I have learned from and shared many enjoyable moments with.

Dedication

I would like to dedicate this thesis to my parents for their support and love throughout my life.

Table of Contents

List of Figures

Chapter 1

Introduction

The study of randomized algorithms and their power compared to deterministic algorithms has been a major and extensively researched area within complexity theory. Roughly, a randomized algorithm R takes an input x and executes a deterministic algorithm $R(x, r)$ for a randomly chosen string r. Further, for algorithm R to solve a problem $\mathcal{S}$, we require that for every possible valid input $x \in \mathsf{dom}(\mathcal{S})$, $R(x, r)$ outputs a correct answer for at least $\frac{2}{3}$ fraction of all possible random seeds. We can often reduce the error of randomized algorithms using folklore amplification techniques, where we execute R over several random seeds $r_1, \ldots, r_k$ and verify if any of the outputs $R(x, r_1), \ldots R(x, r_k)$ are correct. Therefore, a randomized algorithm can often be boosted to almost always output a correct answer.

In the Turing machine model, one can think of randomized algorithms as a method of overcoming uniformity difficulties, where a single Turing machine has to solve the problem on every input length. More specifically, any randomized Turing machine can be efficiently amplified to obtain an error of $\frac{1}{2^{\mathrm{poly}(n)}}$ on inputs of length n. Thus, such a randomized algorithm provides a (non-uniform) family of efficient deterministic circuits solving our problem on every input length [Adl78]. Additionally, under the existence of complexity-theoretic hardness assumptions, we can derandomize every randomized Turing machine to obtain deterministic Turing machines and essentially prove $\mathsf{P} = \mathsf{BPP}$ [Yao82; NW94; IW97; HÅs+99]. These observations suggest that randomized algorithms should not be significantly more powerful than deterministic algorithms in the Turing machine model. Nevertheless, our understanding of randomness remains very limited as no unconditional derandomization results are known currently, and for many important problems such as

prime generation[1] and polynomial identity testing[2] we only have randomized algorithms.

The distinction between the power of randomized and deterministic algorithms is even more prominent in certain non-uniform computational models such as query complexity. In query complexity, we are given query access to a string x of length n, and our goal is to find a corresponding output y within $\text{polylog}(n)$ many queries to indices of x. Within $\text{polylog}(n)$ many queries, we can usually amplify a randomized query algorithm to an error of at most $\frac{1}{2^{\text{polylog}(n)}}$, which is still a tremendous boost. However, given that there are $O(2^n)$ many inputs, this error is not small enough to obtain deterministic algorithms with only a $\text{polylog}(n)$ query overhead. Besides, we can easily prove exponential separations for very natural (promise) problems in query complexity, and therefore, randomized algorithms are provably crucial in such models.

As a result, a natural question is whether we can use efficient randomized algorithms instead whenever deterministic algorithms are unavailable. For decision problems, where the algorithm either outputs *yes* or *no*, having only randomized algorithms is not a major issue. This is mainly because, with the above amplifications, we can obtain a randomized algorithm that almost always gives the same *yes* or *no* output on any given input. Therefore, the boosted randomized algorithm will be practically not distinguishable from a deterministic algorithm. However, the situation is more complicated for a search problem $\mathcal{S}$, where an input x can have multiple outputs y for which $(x, y) \in \mathcal{S}$. We denote the set of all such correct outputs for an input x as $\mathcal{S}(x)$.

First, a correct randomized algorithm can output *any* of the valid certificate in the set $\mathcal{S}(x)$; therefore, randomized search algorithms might not remain *consistent* when called for multiple times. The inconsistency would especially become a problem in cryptographic and distributed settings where the consensus of different parties is of main concern.

Furthermore, in the query model, randomized algorithms have inherent differences for search and decision problems. For instance, for total decision problems, the deterministic and randomized query complexities are polynomially related to each other [Nis89]; i.e., we have $\mathsf{BPP}^{\mathsf{dt}} \subset \mathsf{P}^{\mathsf{dt}}$ in query complexity.[3] On the other hand, for total search problems, we can obtain exponential separations for randomized and deterministic complexities [Lás+91], which in contrast implies $\mathsf{TFNP}^{\mathsf{dt}} \cap \mathsf{FBPP}^{\mathsf{dt}} \not\subset \mathsf{FP}^{\mathsf{dt}}$.

Motivated by these distinctions between search and decision problems, Gat and Gold-

[1] Given an integer n, find a prime $n \leq p \leq 2n$.

[2] Given a high-degree non-zero polynomial $p \not\equiv 0$, output a certificate $p(x) \neq 0$.

[3] $\mathsf{BPP}^{\mathsf{dt}}$ and P^{dt} denote the set of decision problems for decision trees in query complexity that have efficient randomized and deterministic algorithms, respectively. $\mathsf{TFNP}^{\mathsf{dt}}$, $\mathsf{FBPP}^{\mathsf{dt}}$, and $\mathsf{FP}^{\mathsf{dt}}$ are similarly defined as analogous versions of TFNP, FBPP, and FP in query complexity.

wasser [GG11] introduced the concept of pseudo-deterministic algorithms, which are a subclass of randomized algorithms that behave similarly to deterministic algorithms in practice. In the following, we will first provide a brief overview of pseudo-deterministic algorithms and their power in both the Turing machine model and query complexity. Then, motivated by impossibility results in query complexity, we will relax the definition of pseudo-deterministic algorithms to introduce faux-deterministic algorithms and summarize our results for faux-deterministic algorithms in the query model. Eventually, we introduce a new notion of dual problems for search problems and briefly mention our motives for calling them dual problems.

1.1 An overview of pseudo-deterministic algorithms

A pseudo-deterministic algorithm for a search problem $\mathcal{S}$ is a randomized algorithm R that on each input x outputs a unique output y_x with high probability;

$$\Pr_r[R(x,r) = y_x] \geq \frac{2}{3}.$$

Pseudo-deterministic algorithms can leverage randomness to solve search problems more efficiently than deterministic algorithms. However, unlike general randomized algorithms, we can still amplify pseudo-deterministic algorithms to obtain a single output with high probability. Therefore, pseudo-deterministic algorithms can practically become indistinguishable from deterministic algorithms for search problems.[4] As a result, we obtain an intermediate class of search problems, psFP, that have pseudo-deterministic algorithms.

Clearly, pseudo-deterministic algorithms are more powerful than deterministic ones and weaker than randomized algorithms; therefore, we immediately get $\mathsf{FP} \subset \mathsf{psFP} \subset \mathsf{FBPP}$. In the Turing machine model, Gat and Goldwasser [GG11] proved another characterization of this class as $\mathsf{psFP} = \mathsf{FP}^{\mathsf{BPP}}$. In more detail, a search problem has a pseudo-deterministic algorithm if and only if it has a deterministic algorithm that can make queries to a randomized verifier. A simple yet intriguing corollary of this result is that $\mathsf{P} = \mathsf{BPP}$ if and only if $\mathsf{FP} = \mathsf{psFP}$. Generally speaking, pseudo-deterministic algorithms as the search problem counterpart of randomized algorithms for decision problems.

In the same paper [GG11], Gat and Goldwasser also proved the existence of pseudo-deterministic algorithms for problems such as polynomial identity testing and asked the

[4] Note that any randomized algorithm for decision problems is also a pseudo-deterministic algorithm as well.

question of whether such algorithms exist for prime generation problems. Towards the resolution of this problem, Oliveira and Santhanam [OS17] showed the existence of sub-exponential pseudo-deterministic algorithms for a prime generation that works on infinitely many input lengths. Most recently, Chen et al. [Che+23] improved the previous result to obtain a polynomial-time pseudo-deterministic algorithm in the same infinitely often regime. However, we do not yet have a pseudo-deterministic algorithm that works on every input length.

Besides the Turing model, there has been a lot of research on the existence or impossibility of pseudo-deterministic algorithms in different computational models such as query complexity [GGR13; Gol+21; CDM23]. Here, we consider the FIND1 problem in query complexity, which can be viewed as a complete problem for search problems having efficient verifiers and randomized algorithms.

Informal Definition (FIND1)**.** Given a binary string $x \in \{0,1\}^n$ of length n with Hamming weight at least $|x|_1 \geq \frac{n}{2}$, find an index i for which $x_i = 1$.

The deterministic query complexity of this problem is $\Theta(n)$ as in the worst-case, we may need to make $\frac{n}{2}$ queries to find an index i with $x_i = 1$. On the other hand, outputting random indices will give us a randomized decision tree of depth $O(1)$. Goldwasser et al. [Gol+21] proved an exponential gap between randomized and pseudo-deterministic complexity of FIND1 by proving a lower bound of $\Omega(n^{\frac{1}{2}})$ and conjectured this lower bound can be improved to $\Omega(n)$. Later, in [CDM23] a lower bound of $\Omega(n^{\frac{1}{3}})$ was achieved via a simpler proof. Nevertheless, it is still open whether we can obtain a better lower bound of $\Omega(n)$.

The consequence of the above lower bounds is that for important problems, such as FIND1, we do not have algorithms that are both indistinguishable from deterministic algorithms and are efficient. To overcome this problem, we introduce a relaxation of pseudo-deterministic algorithms and study their complexity for the class of problems having verifiers and randomized algorithms.

1.2 Faux-determinism

As previously mentioned, deterministic and pseudo-deterministic algorithms are (statistically) indistinguishable. Therefore, whenever deterministic algorithms are unavailable, we can execute an efficient pseudo-deterministic algorithm instead. An interesting observation is that usually, in such an application, the primary algorithm calling the pseudo-deterministic algorithm should also be efficient. Therefore, for practical purposes, it is

too restrictive to require unconditional indistinguishability; rather, any randomized algorithm that is computationally indistinguishable from a deterministic algorithm would be sufficient. Motivated by computational indistinguishability, we introduce a new notion of faux-deterministic algorithms as a relaxation of pseudo-deterministic algorithms. More specifically, we want a distribution of algorithms $\mathcal{A}$ such that no computationally bounded algorithm D making black-box queries to a random algorithm $A \sim \mathcal{A}$ can find an input on which A fails.

Informal Definition (Faux-determinism)**.** Let $\mathcal{S}$ be a search problem and $\mathcal{A}$ be a distribution of algorithms $A \in \mathcal{A}$. We say $\mathcal{A}$ is a *faux-deterministic* algorithm for $\mathcal{S}$ whenever any computationally bounded adversary D making black-box queries to outputs of a randomly chosen algorithm $\boldsymbol{A} \sim \mathcal{A}$ fails to distinguish $\boldsymbol{A}$ from a solution of $\mathcal{S}$; i.e.,

$$\Pr_{\boldsymbol{A} \sim \mathcal{A}}[x \leftarrow D[\boldsymbol{A}]; \boldsymbol{A}(x) \notin \mathcal{S}(x)] < \epsilon,$$

where ϵ is a negligible error parameter and $x \leftarrow D[\boldsymbol{A}]$ is a random variable corresponding to the output of algorithm D after querying outputs of $\boldsymbol{A}$. For an algorithm $A \in \mathcal{A}$, we call an input x for which $A(x) \notin \mathcal{S}(x)$ an *adverse* input for A.

For the sake of simplicity, we assume each internal algorithm $A \in \mathcal{A}$ to be deterministic. However, we can also allow pseudo-deterministic algorithms as internal algorithms as well. In the latter case, any pseudo-deterministic algorithm will be a faux-deterministic algorithm consisting of a single algorithm.

Faux-deterministic algorithms can be particularly useful in query complexity, where pseudo-deterministic algorithms for natural problems such as FIND1 do not exist. Roughly, we will show that in the query model, faux-deterministic algorithms exist for a large class of search problems, including FIND1.

Further, we interpret the existence of faux-deterministic algorithms as an explanation of why proving better lower bounds for pseudo-deterministic algorithms might be challenging. In a general proof of the lower bound of m queries, one shows that for any pseudo-deterministic algorithm of low depth $d < m$, there exists a counter-example input x on which the algorithm fails. However, the proof might not give an efficient way of constructing the counter-example input x. Consequently, we call a lower bound proof constructive if we can efficiently generate a counter-example for any pseudo-deterministic algorithm of low depth $d < m$.

As mentioned earlier, a lower bound of $\Omega(n)$ has been conjectured for the pseudo-deterministic complexity of FIND1 [Gol+21]. In both pseudo-deterministic lower bounds of

$\Omega(n^{\frac{1}{2}})$ [Gol+21] and $\Omega(n^{\frac{1}{3}})$ [CDM23] general non-constructive methods in query complexity such as polynomial method [Bea+01] and polynomial relation of randomized and deterministic algorithms [Nis89] were used. These techniques generally incur quadratic and cubic losses over $\Omega(n)$, respectively. On the other hand, the existence of a faux-deterministic algorithm signifies constructive proofs may not exist as any such proof should be able to "distinguish" a pseudo-deterministic algorithm from a faux-deterministic one.

1.2.1 Our results

We prove that in the query model, every verifiable search problem that has an efficient randomized algorithm also has an efficient faux-deterministic algorithm.

Informal Theorem. *Let $\mathcal{S}$ be a search problem with a verifier and randomized decision trees of depths* polylog(n)*. Then, $\mathcal{S}$ has a faux-deterministic algorithm of depth* polylog(n) *where any adversary querying* poly(n) *many queries finds an adverse input with negligible probability.*

This immediately proves an exponential separation between pseudo-deterministic and faux-deterministic complexities as the FIND1 problem has both a randomized algorithm and a verifier.

The main idea for the construction is to create a decision tree on which we make interleaving adaptive and non-adaptive queries to the randomized algorithm and the verifier. Intuitively, the adaptive queries result in many possibilities for the adversary to check before finding an adverse input, while the non-adaptive queries prevent the adversary from learning the pattern of which inputs correspond to which possibilities. Therefore, the mix of adaptive and non-adaptive queries allows us to fool adversaries while maintaining a low depth of polylog(n). We will improve our results to prove that even a quantum algorithm cannot find an adverse input for our construction.

Informal Theorem. *The faux-deterministic algorithm in the above is also secure against quantum adversaries making* poly(n) *many queries in superposition.*

Clearly, having a randomized algorithm is a necessary condition for the existence of faux-deterministic algorithms. An interesting question is whether verifiers are also necessary for a faux-deterministic algorithm. We initiate a discussion about this question by considering a total variant of the FIND1 problem where on an input x with hamming weight $|x|_1 < \frac{n}{2}$, the algorithm can output any arbitrary index.

Informal Definition (FINDMEDIAN)**.** Given a binary string $x \in \{0,1\}^n$ of length n, output an index i with $x_i = 1$, if $|x|_1 \geq \frac{n}{2}$. Otherwise, any index $i \in [n]$ can be outputted.

By this definition, any randomized, deterministic, or pseudo-deterministic solution for FIND1 will also be a solution for FINDMEDIAN. Therefore, solving FIND1 is as hard as solving FINDMEDIAN. However, unlike the FIND1 problem, FINDMEDIAN is not verifiable. More specifically, we cannot verify whether $(x, i) \in$ FINDMEDIAN because this requires us to compute the hamming weight $|x|_1$. To emphasize the importance of verifiability, we obtain the following result.

Informal Theorem. FINDMEDIAN *problem does not have a faux-deterministic algorithm.*

Finally, we revisit the hardness of constructive proofs for pseudo-deterministic lower bounds as follows. Any pseudo-deterministic algorithm for FIND1 implements a function f computing $\mathcal{S}$. I.e.,

$$f : \mathcal{X}_n \to [n],$$
$$f(x) \in \mathcal{S}(x),$$

where $\mathcal{X}_n$ is the set of all strings $x \in \{0,1\}^n$ with hamming weight $|x|_1 \geq \frac{n}{2}$. A natural approach for a constructive pseudo-deterministic lower bound is to find a highly sensitive input for any function f computing $\mathcal{S}$ [Nis89]. More specifically, for a function f, an input x has block sensitivity at least k if there exist disjoint blocks of indices $B_1, \ldots, B_k \subset [n]$ such that $f(x) \neq f(x^{B_i})$.[5] In such a case, we say f has block sensitivity at least k and denote $\text{bs}(f) \geq k$.

Both lower bounds of [Gol+21; CDM23] prove that any function f computing FIND1 has high block sensitivity. We prove that finding such highly sensitive input is a hard task and is essentially non-constructive. [Nis89].

Informal Theorem. *Suppose an adversary is given query access to a function f computing $\mathcal{S}$. Therefore, f has has block sensitivity at least $\Omega(\sqrt{n})$. However, no adversary making polynomially many queries can find such a highly sensitive input x within polynomially many queries.*

[5]For a subset of indices $B \subset [n]$ and binary string x, x^B denotes the string obtained by altering x on indices $i \in B$.

1.3 Duality

Another approach for looking at a faux-deterministic algorithm is through the adversary's viewpoint. Specifically, for a hard search problem and a given deterministic algorithm A attempting to solve $\mathcal{S}$, the adversary is trying to solve yet another search problem: finding an adverse input x such that $A(x) \notin \mathcal{S}(x)$.

This motivates us to define a *dual* search problem consisting of pairs (A, x) where A is an efficient algorithm and $A(x) \notin \mathcal{S}(x)$. We denote this meta-complexity problem as $\mathcal{S}^*$ and call it the dual of our initial search problem $\mathcal{S}$. With this in mind, we can rephrase the existence of faux-deterministic algorithms as the dual problem being hard on average; in other words, a hard distribution exists for which $\mathcal{S}^*$ cannot be solved.

Generally, we can view the complexity of $\mathcal{S}^*$ as the complexity of finding a proof of hardness for $\mathcal{S}$. In other words, if $\mathcal{S}^*$ is easy, we have constructive proof that $\mathcal{S}$ is hard. Otherwise, $\mathcal{S}^*$ is hard, and the proof of hardness for $\mathcal{S}$ is non-constructive. Previously, under different terminologies, similar notions to the dual problem were introduced with a focus on decision problems and refuting a single efficient algorithm [Kab00]. In this regard, Gutfreund et al. [GST07] proved that under $\mathsf{P} \neq \mathsf{NP}$, for NP-complete problems and any polynomial-time algorithm A, we can find an adverse input of infinitely many inputs lengths for A in polynomial-time. Later, this result was improved by Asterias [Ats06] so that the dual algorithm only has black-box access to the algorithm A. Finally, Chen et al. [Che+22] generalized this result to other separations in complexity theory. Roughly speaking, these results imply that for hard separation (such as $\mathsf{P} \neq \mathsf{NP}$ or $\mathsf{P} \neq \mathsf{PSPACE}$), the dual problem is easy, and for easy separation (such as provable separations in query complexity), the dual problem is hard.

We study such meta-complexity duality not just for NP-complete problems but for arbitrary search problems (considering search instead of decision problems is natural since the dual problem is always a search problem). The term "duality" is motivated by several properties of these meta-complexity problems.

Informal Observation. In different computational models, some or all of the following are correct.

i if $\mathcal{S}$ is verifiable, then $\mathcal{S}^*$ is also verifiable;

ii if $\mathcal{S}$ is hard, then $\mathcal{S}^*$ is total;

iii if $\mathcal{S}$ reduces to $\mathcal{S}'$, then $\mathcal{S}'^*$ reduces to $\mathcal{S}^*$;

iv if $\mathcal{S}$ has a randomized algorithm then $\mathcal{S}^*$ is hard on average;

v if $\mathcal{S}$ is hard on average, then $\mathcal{S}^*$ has a randomized algorithm.

We will consider the query and Turing machine model and prove some of the above observations for each of them.

1.4 Organization

Chapter 2 briefly reviews the basic notations, definitions, and preliminaries used throughout the thesis. Most of the discussed topics covered can be found in the books [AB09] and [Gol08]. For a more thorough introduction to quantum computing, the standard textbook is [NC10]. The survey [Bd02] is a great introductory text for query complexity. For quantum query complexity, one can also refer to more recent surveys [Amb17; Aar21].

In Chapter 3, we formally introduce faux-deterministic algorithms and prove our results in the query model. Afterward, we discuss the dual problem for TFNP problems in both query and Turing machine models in Chapter 4. Finally, Chapter 5 concludes the thesis by discussing several open problems and outlining possible future research directions.

Chapter 2

Preliminaries

Notation For $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, ..., n\}$. By poly(n) and quasipoly(n), we denote the family of all functions growing at most polynomially ($f(n) = O(n^c)$) and quasi-polynomially ($f(n) = O(2^{\log^c n})$ for some constant c). We also define negl(n) to denote the family of all functions where the inverse grows faster than any polynomial ($f(n)^{-1} = \Omega(n^c)$ for every constant c). We may additionally use poly(n), quasipoly(n), and negl(n) to denote a function in these families.

For a finite alphabet Σ, and string $x \in \Sigma^n$, we define its *length* as $|x|_\Sigma := n$. Moreover, we define $\langle x \rangle \in \{0,1\}^{c.\lceil \log \Sigma \rceil \times n}$ to be the canonical *binary encoding* of the string $x \in \Sigma^n$ with alphabet Σ where c is a constant independent of Σ and n. For technical reasons, we also assume that the chosen encoding is *paddable* where for any $k \in \mathbb{N}$, the strings $\langle x \rangle 0^k$ is also decoded to the original string $x \in \Sigma^n$. For a set $X \subset \Sigma^*$, we define $\langle X \rangle := \{\langle x \rangle | x \in X\}$ to be the corresponding encoded set. We may use x (X) interchangeably with $\langle x \rangle$ ($\langle X \rangle$) whenever it is obvious from the context.

For a binary string $x \in \{0,1\}^n$, we also denote its length as $|x|_2 := |x|_{\{0,1\}} = n$ and its *Hamming weight* as $|x|_1 := \sum_{i=1}^n x_i$. We may omit the subscripts above whenever it is obvious from context. For index $i \in [n]$, the string x^i denotes the string obtained by negating the i-th bit of x. We extend this notation for x^B where $B \subset [n]$ is a subset of indices in a natural way.

Often, we denote distributions by calligraphic letters such as $\mathcal{T}$, $\mathcal{R}$, and $\mathcal{A}$, and random variables sampled from this distribution with corresponding bold letters $\boldsymbol{T} \sim \mathcal{T}$,$\boldsymbol{R} \sim \mathcal{T}$, $\boldsymbol{A} \sim \mathcal{A}$.

2.1 Quantum computing

We will use the standard *bra-ket* notation for quantum computing. For a *quantum register* X of n qubits, we denote *quantum states* as unit vectors

$$|\phi\rangle_X$$

in Hilbert space

$$\mathcal{H}_X = (\mathbb{C}^{2^n}, \langle|\rangle),$$

where $\langle|\rangle$ is the usual complex inner product. We use the notation $|x\rangle_X$ for a binary string $x \in \{0,1\}^n$ to denote the corresponding elementary vector having a one in the x entry. We denote a *multi-register* system as $X = X_1 \dots X_k$, where

$$\mathcal{H}_X = \mathcal{H}_{X_1} \otimes ... \otimes \mathcal{H}_{X_k}.$$

We may omit X whenever it is obvious from the context.

A *unitary transformation* U on register X is a linear transformation preserving l_2-norm;

$$\forall\, |\phi\rangle_X\,, \quad \|U\,|\phi\rangle_X\,\|_2 = \|\,|\phi\rangle_X\,\|_2.$$

With abuse of notation, we additionally use U to denote unitary transformations $U \otimes I$ and $I \otimes U$ on larger registers XY and YX, respectively.

A *measurement* corresponding to the *computational basis*

$$\{|x\rangle_X\,|x \in \{0,1\}^n\}$$

of register X on state

$$|\phi\rangle_{XY} = \sum_{x\in\{0,1\}^n} \alpha_x\,|x\rangle_X\,|\phi_x\rangle_Y$$

collapses the state to $|x\rangle_X\,|\phi_x\rangle_Y$ with probability α_x^2. The measurement of the second register would be defined similarly. We can also consider measurements in a more general setting of *projection operators*. Let $\mathcal{M} = \{\Pi_1, \dots, \Pi_k\}$ be a set of projection matrices, where $\sum_{i=1}^k \Pi_i = I$ and $\Pi_i = \Pi_i^2$ for $i \in [k]$. The outcome $i \in [k]$ of measurement $\mathcal{M}$ on state $|\phi\rangle$ occurs with probability $\|\Pi_i\,|\phi\rangle\,\|_2$, where the resulting state will become

$$\frac{\Pi_i\,|\phi\rangle}{\|\Pi_i\,|\phi\rangle\,\|_2}.$$

2.2 Search and decision problems

Usually, search problems are defined as relations $S \subset \{0,1\}^* \times \{0,1\}^*$ over all possible input lengths. For our purposes, though, we consider them as a family of relations over each input length.

A *(search) problem* over *input and output alphabets* Σ and Γ is a family of relations $\mathcal{S} = (\mathcal{S}_n)_{n=1}^{\infty}$ with $\mathcal{S}_n \subset \Sigma^n \times \Gamma^*$, where n denotes the *input length*. Later, in the duality chapter, we generalize the definition of search problems where Σ, Γ, and input lengths can vary with the value of n.

For each $x \in \Sigma^n$, we define the set of corresponding *certificates* as

$$\mathcal{S}(x) := \mathcal{S}_n(x) = \{y \in \Gamma^* | (x,y) \in \mathcal{S}_n\},$$

the *domain* as

$$\mathsf{dom}(\mathcal{S}) := \bigcup_n \mathsf{dom}(\mathcal{S}_n),$$

and the *range* as

$$\mathsf{range}(\mathcal{S}) := \bigcup_n \mathsf{range}(\mathcal{S}_n).$$

The problem $\mathcal{S}$ is called a *Boolean problem* if $\mathsf{range}(\mathcal{S}) = \{0,1\}$, *single-output* if $|\mathcal{S}(x)| \leq 1$ for all $x \in \mathsf{dom}(\mathcal{S})$, and *total* if $\mathsf{dom}(\mathcal{S}) = \bigcup_n \Sigma^n$. A problem that is both single-output and Boolean is called a *decision* problem.

2.3 Decision trees and query complexity

Decision trees A *deterministic decision tree* T over finite input and output alphabets Σ and Γ and length $n \in \mathbb{N}$ is a rooted ordered tree where each internal node is labeled with a variable x_i, $i \in [n]$, and has $|\Sigma|$ children, each labeled as one of the possible values $x_i \in \Sigma$. Further, each leaf is labeled with a potential output in Γ^*.

For an input $x \in \Sigma^n$, the *execution* of the tree on x, denoted as $T[x]$, is recursively computed as follows. Beginning with the root, if the current node is a leaf, output its label. Otherwise, there exists an index $i \in [n]$ such that the current node is labeled with the value x_i. Query x_i and continue the computation from the corresponding child until a leaf is reached. The *(query) complexity* of T, denoted as $\mathrm{C}^{\mathsf{dt}}(T)$, is defined as its depth which is the maximum number of queries made to the worst-case input.

A *randomized decision tree* $\mathcal{T}$ over alphabets Σ and Γ and length $n \in \mathbb{N}$ is a probability distribution over a finite set of deterministic decision trees. For an input $x \in \Sigma^n$, the execution of the tree $\mathcal{T}[x]$ is a random variable in Γ^* evaluated by sampling a tree $\boldsymbol{T} \sim \mathcal{T}$ and computing $\boldsymbol{T}[x]$. Similarly, we define complexity of $\mathcal{T}$, $\mathrm{C}^{\mathsf{dt}}(\mathcal{T})$, as the depth of the deepest tree in $\mathcal{T}$; i.e.,

$$\mathrm{C}^{\mathsf{dt}}(\mathcal{T}) := \max_{T \in \mathcal{T}} \mathrm{C}^{\mathsf{dt}}(T).$$

To define the quantum version of decision trees, we take another approach as we want to be able to query in superposition. For simplicity, we assume that input and output alphabets Σ and Γ are encoded using binary fields $\mathbb{F}_{2^{\lceil \log |\Sigma| \rceil}}$ and $\mathbb{F}_{2^{\lceil \log |\Gamma| \rceil}}$. For a string $x \in \Sigma^n$ we define the *oracle unitary* U^x as follows:

$$\forall i \in [n], y \in \Sigma, \qquad U^x \left|i\right\rangle_I \left|y\right\rangle_Q := \left|i\right\rangle_I \left|x_i \oplus y\right\rangle_Q ,$$

where I and Q are *index* and *query* registers, respectively. Now, a *quantum decision tree* is a sequence of unitaries

$$\mathcal{Q} = (U_0, U_1, \ldots, U_d)$$

that act on a multi-register system

$$\left|\phi_I\right\rangle_I \left|\phi_Q\right\rangle_Q \left|\phi_W\right\rangle_W \left|\phi_O\right\rangle_O$$

consisting of *index, query, work, and output* registers of sizes $\lceil \log n \rceil$, $\lceil \log |\Sigma| \rceil$, m, and $k\lceil \log |\Gamma| \rceil$, respectively. We define the query complexity of $\mathcal{Q}$ as $\mathrm{C}^{\mathsf{dt}}(\mathcal{Q}) := d$ and the execution $\mathcal{Q}[x]$ as the measurement outcome of the output register in the final state

$$U_d U^x \ldots U^x U_0 \left|0\right\rangle_I \left|0\right\rangle_Q \left|0\right\rangle_W \left|0\right\rangle_O .$$

Remark. Note that as with the quantum case, the deterministic and randomized decision trees can also be described as circuits with access to query gates. Consequently, we can define the query complexity of these circuits as the number of times a query gate has been used. ▷

For a family of deterministic, randomized, or quantum decision trees

$$A \in \{(T_n), (\mathcal{T}_n), (\mathcal{Q}_n), \}$$

we define the depth of the family as the sequence

$$\mathrm{C}^{\mathsf{dt}}(A)_n := \mathrm{C}^{\mathsf{dt}}(A_n).$$

Query algorithms Let $\mathcal{S} = (\mathcal{S}_n)_{n=1}^{\infty}$ be a search problem.

- A family of deterministic decision trees $T = (T_n)_{n=1}^{\infty}$ is a *deterministic algorithm* for problem $\mathcal{S}$, if for all $n \in \mathbb{N}$ and $x \in \Sigma^n$,

$$T_n[x] \in \mathcal{S}_n(x).$$

- A family of randomized decision trees $\mathcal{T} = (\mathcal{T}_n)_{n=1}^{\infty}$ is a *randomized algorithm* for the search problem $\mathcal{S}$ with error ε, if for all $n \in \mathbb{N}$ and $x \in \Sigma^n$,

$$\Pr_{\boldsymbol{T_n} \sim \mathcal{T}_n}[\boldsymbol{T_n}[x] \in \mathcal{S}_n(x)] \geq 1 - \varepsilon.$$

- A family of randomized decision trees $\mathcal{T} = (\mathcal{T}_n)$ is a *pseudo-deterministic algorithm* for the search problem $\mathcal{S}$ with error ε, if for all $n \in \mathbb{N}$ and for all $x \in \Sigma^n$, there exists a unique $y_x \in \mathcal{S}_n(x)$ such that,

$$\Pr_{\boldsymbol{T} \sim \mathcal{T}_n}[\boldsymbol{T}[x] = y_x] \geq 1 - \varepsilon.$$

- A family of quantum decision trees $\mathcal{Q} = (\mathcal{Q}_n)$ is a *quantum algorithm* for the search problem $\mathcal{S}_n$ with error ε, if for all $n \in \mathbb{N}$,

$$\Pr[\mathcal{Q}_n[x] \in \mathcal{S}_n(x)] \geq 1 - \varepsilon.$$

  Similar to the randomized case, we can define pseudo-deterministic quantum decision trees similarly.

- For a search problem $\mathcal{S}$, we define the canonical decision problem $f_{\mathcal{S}}$ to be such that for all $n \in \mathbb{N}$, $x \in \mathsf{dom}(\mathcal{S}_n)$, and $y \in \mathsf{range}(\mathcal{S}_n)$,

$$f_{\mathcal{S}}(x, y) := \begin{cases} 1 & y \in \mathcal{S}(x), \\ 0 & \text{otherwise.} \end{cases}$$

  A family of decision trees, $T = (T_n)$, is a *deterministic verifier* for $\mathcal{S}$ if for all $n \in \mathbb{N}$, $x \in \mathsf{dom}(\mathcal{S}_n)$, and $y \in \mathsf{range}(\mathcal{S}_n)$,

$$T_n(x, y) = f(x, y).$$

Query complexity For the search problem $\mathcal{S}$, we define *query (decision tree) complexities* $\mathbb{D}^{\mathsf{dt}}(\mathcal{S})$, $\mathbb{R}^{\mathsf{dt}}(\mathcal{S})$, $\mathbb{PS}^{\mathsf{dt}}(\mathcal{S})$, $\mathbb{Q}^{\mathsf{dt}}(\mathcal{S})$, and $\mathbb{V}^{\mathsf{dt}}(\mathcal{S})$ as the minimum complexity of an deterministic, randomized, pseudo-deterministic, quantum, and verification algorithms for $\mathcal{S}$, respectively as

$$\begin{aligned}
\mathbb{D}^{\mathsf{dt}}(\mathcal{S}) &:= \min\{\mathrm{C}^{\mathsf{dt}}(T) | T \text{ deterministically computes } S\};\\
\mathbb{R}^{\mathsf{dt}}_{\varepsilon}(\mathcal{S}) &:= \min\{\mathrm{C}^{\mathsf{dt}}(\mathcal{T}) | \mathcal{T} \text{ computes } \mathcal{S} \text{ with error } \varepsilon\};\\
\mathbb{PS}^{\mathsf{dt}}_{\varepsilon}(\mathcal{S}) &:= \min\{\mathrm{C}^{\mathsf{dt}}(\mathcal{T}) | \mathcal{T} \text{ pseudo-deterministically computes } \mathcal{S} \text{ with error } \varepsilon\};\\
\mathbb{Q}^{\mathsf{dt}}_{\varepsilon}(\mathcal{S}) &:= \min\{\mathrm{C}^{\mathsf{dt}}(\mathcal{Q}) | \mathcal{Q} \text{ computes } \mathcal{S} \text{ with error } \varepsilon\};\\
\mathbb{V}^{\mathsf{dt}}(\mathcal{S}) &:= \min\{\mathrm{C}^{\mathsf{dt}}(T) | T \text{ deterministically computes } f_{\mathcal{S}}\}.
\end{aligned}$$

Whenever the error parameter ε is omitted, we assume $\varepsilon = \frac{1}{3}$.

2.4 Turing machine model

In the Turing machine model, it is more convenient to work with binary inputs and outputs for a search problem $\mathcal{S}$. In such scenarios, with abuse of notation, we may use strings $x \in \mathsf{dom}(\mathcal{S})$ and $y \in \mathsf{range}(\mathcal{S})$ interchangeably with their binary encoding $\langle x \rangle$ and $\langle y \rangle \in \{0,1\}^*$ to denote the inputs and outputs of the search problem as Boolean strings and view the search problem as a relation

$$\mathcal{S} \subset \langle \mathsf{dom}(\mathcal{S}) \rangle \times \langle \mathsf{range}(\mathcal{S}) \rangle \subset \{0,1\}^* \times \{0,1\}^* .$$

We use the read-work-write multi-tape Turing machine (TM) model defined in [AB09] and denote the *run-time* of a Turing machine M on input x as $\mathsf{TIME}(M, x)$.

Let $\mathcal{S}$ be a search problem, and $T : \mathbb{N} \to \mathbb{N}$ be a time-bound[1] function.

- We say Turing machine M computes $\mathcal{S}$ *deterministically* if $M[x] \in \mathcal{S}(x)$ for all $x \in \mathsf{dom}(\mathcal{S})$. We say M computes $\mathcal{S}$ in time T if we also have $\mathsf{TIME}(M, x) \leq T(|x|_2)$ for all $x \in \mathsf{dom}(\mathcal{S})$.
- We say Turing machine M is a *verifier* for $\mathcal{S}$ if $(x, y) \in \mathcal{S}$ if and only if $M[x, y] = 1$, and M verifies $\mathcal{S}$ in time T if we additionally have

  $$\mathsf{TIME}(V, (x, y)) \leq T(|x|_2),$$

  for all pairs $(x, y) \in \mathcal{S}$.

[1] We often require T to be a *time-constructable* function. However, almost all of the non-artificial functions satisfy time-constructability. Therefore, we omit further details on this condition.

- We say M is a *probabilistic Turing machine (PTM)* computing $\mathcal{S}$ with error ϵ if there exists a polynomial p such that for all $x \in \mathsf{dom}(\mathcal{S})$,

$$\Pr_{r \in \{0,1\}^{p(|x|)}}[M[x,r] \notin \mathcal{S}(x)] \leq \epsilon.$$

  A probabilistic Turing machine M runs in time T if

$$\mathsf{TIME}(M,(x,r)) \leq T(|x|_2),$$

  for all $x \in \mathsf{dom}(\mathcal{S})$ and $r \in \{0,1\}^{T(|x|_2)}$. Whenever ϵ is omitted, we assume $\epsilon = \frac{1}{3}$.

Complexity classes in Turing machine model Having the definition of the Turing machine and what it means to solve a search problem, we can define the classical complexity classes FP, FBPP, and FNP as the class of search problems having polynomial-time deterministic, randomized, and verification algorithms. Note that the above classes contain both total and promise search problems. We define the class TFNP to consist of total search problems in FNP. Additionally, we define decision classes P, NP, BPP as the class of total decision search problems in FP, FNP, and FBPP.

Chapter 3

Faux-determinism in Query Model

In this chapter, we formally introduce the notion of *faux-deterministic* algorithms in the query model. To do so, we start by defining an *adverse* input as an input on which the given algorithm T fails to produce a correct answer for the search problem $\mathcal{S}$.

Definition 3.1 (Adverse input)**.** For a search problem $\mathcal{S}$ and a decision tree T attempting to solve $\mathcal{S}$ on inputs of length n, we say $x \in \mathsf{dom}(\mathcal{S}_n)$ is an *adverse input* for T whenever $T[x] \notin \mathcal{S}_n(x)$. ▷

Remark. The adverse input should be inside the promise $\mathsf{dom}(\mathcal{S}_n)$ since T is only attempting to solve inputs inside the promise of $\mathcal{S}$. ▷

Afterward, we can formally define a faux-deterministic algorithm as a distribution $\mathcal{T}$ over decision trees for a search problem $\mathcal{S}$. We further require that a randomly chosen decision tree $T \sim \mathcal{T}$ fools any computationally bounded adversary A attempting to find an adverse input for T. Formally, we proceed as follows.

Definition 3.2 ((Classical) faux-determinism in query model)**.** Let $\mathcal{S}$ be a search problem, $q = q(n) \in \mathbb{N}$ be a query upper bound, and $\varepsilon = \varepsilon(n) \geq 0$ be an error parameter. We say distribution $\mathcal{T} = (\mathcal{T}_n)$ of decision trees is a (q, ε)-*faux-deterministic algorithm* for $\mathcal{S}$ if for all $n \in \mathbb{N}$, any adversary A_n that makes less than $q(n)$ queries to the output of a random decision tree $\boldsymbol{T} \sim \mathcal{T}_n$ finds an adverse input with probability at most $\varepsilon(n)$:

$$\Pr_{\boldsymbol{T} \sim \mathcal{T}_n}[x \leftarrow A_n[\boldsymbol{T}]; \boldsymbol{T}[x] \notin \mathcal{S}(x)] \leq \varepsilon(n).$$

Similar to randomized decision trees, the complexity of a faux-deterministic algorithm is defined as its worst-case complexity over all the possible trees in the distribution:

$$\mathrm{C}^{\mathsf{dt}}(\mathcal{T}) := \max_{T \in \mathcal{T}} \mathrm{C}^{\mathsf{dt}}(T).$$

▷

Remark. Each decision tree T for the search problem $\mathcal{S}$ denotes an input of size $[n]^{2^{n-1}}$. Thus, the adversary A_n is a decision tree that works on this larger input size. ▷

Remark. In the above definition, the existence of deterministic and randomized adversaries is equivalent. If a randomized adversary succeeds with probability $\varepsilon(n)$, we can choose the best deterministic algorithm for each input length n to obtain a deterministic adversary that succeeds with probability at least $\varepsilon(n)$ as well. Therefore, without loss of generality, we only consider deterministic adversaries. ▷

In the above definition, the input size for the adversary is $O(2^n)$. Therefore, any efficient adversary - in terms of input size - needs to make at most $\text{polylog}(2^n) = \text{poly}(n)$ many queries. Therefore, we are interested to know if we can find efficient faux-deterministic algorithms where any attacker querying polynomially many inputs succeeds to find an adverse input with only a negligible probability. In the terminology of our definition, this means whether $(\text{poly}(n), \text{negl}(n))$-faux-deterministic algorithms of complexity $\text{polylog}(n)$ exist.

In the first section, we formally define the previously mentioned FIND1 [Gol+21] problem. Then, we construct a faux-deterministic algorithm for FIND1 by establishing classical lower bounds on the query complexity and error probability for any adversary attacking our algorithm. Additionally, using the hybrid method in quantum query complexity [Ben+97] we adjust the query and error lower bounds to obtain slightly worse bounds for adversaries making quantum queries. Finally, we investigate the optimality of our lower bounds and present two open problems regarding faux-deterministic algorithms for FIND1 problem.

Having faux-deterministic algorithms for the FIND1 problem, we further extend our results to a larger class of search problems in the second section. Mainly, we use the reduction of [Gol+21] to obtain faux-deterministic algorithms for search problems that have efficient verifiers and randomized algorithms.

Naturally, we ask whether verification assumptions of these results can be relaxed to prove the existence of faux-deterministic algorithms for larger classes of search problems. We introduce the FINDMEDIAN problem to suggest a negative answer for the previous question. Finally, we conclude by considering the hardness of constructing pseudo-deterministic lower bounds for the FIND1 problem, and the relation of constructive lower bounds with faux-deterministic algorithms.

3.1 Faux-deterministic construction for the FIND1 problem

Before demonstrating our results, we first give some intuition and motivation behind our construction for the FIND1 problem as follows. For input length n, the inputs for $\mathsf{FIND1}_n$ are the set of strings $x \in \{0,1\}^n$ with large hamming weight $|x|_1 \geq \frac{n}{2}$. Further, our goal is to find an index i for which $x_i = 1$.

Definition 3.3 (FIND1)**.** For $n \in \mathbb{N}$, let $\mathsf{FIND1}_n$ be defined as

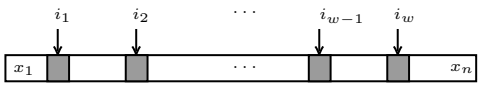
$$\mathsf{FIND1}_n := \Big\{(x,i) \in \{0,1\}^n \times \{0,1\}^{\lceil \log n \rceil} \Big|\ |x|_1 \geq \frac{n}{2} \text{ and } x_i = 1\Big\}.$$

We define the search problem FIND1 over binary alphabets to be the sequence

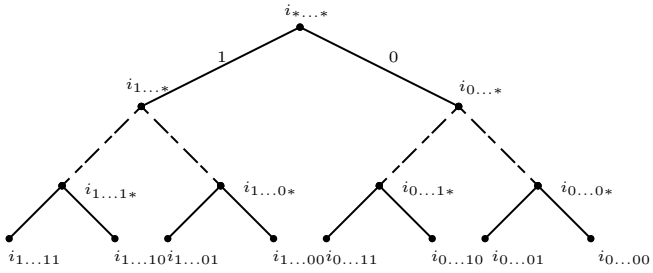
$$\mathsf{FIND1} := (\mathsf{FIND1}_n)_{n=1}^{\infty}. \qquad \triangleright$$

As previously noted, the deterministic query complexity of FIND1 problem is $\Omega(n)$. As a result, for any given tree of depth $d = o(n)$ there are many inputs on which the tree fails. Thus, a single tree of depth $\text{polylog}(n)$ cannot be effective against an adversary querying it, as the adversary can hard-wire an adverse input for each input length n. Therefore, we need to construct a randomized algorithm to have any chance against adversaries.

A non-adaptive randomized idea: Since the inputs have large hamming weights $|x|_1 \geq \frac{n}{2}$, we have a very simple and efficient randomized algorithm for the FIND1 problem that queries two randomly chosen indices $i, j \in [n]$, and outputs i and j by verifying whether $x_i = 1$ or $x_j = 1$. Consequently, with probability at least $\frac{3}{4}$ either $x_i = 1$ or $x_j = 1$. We can easily boost this non-adaptive algorithm to obtain a randomized algorithm $\mathcal{T} = (\mathcal{T}_n)$ of complexity $w = \text{polylog}(n)$ that queries random indices $i_1, \ldots, i_w$, and outputs the first index i_j for which $x_{i_j} = 1$. As a result, we have a (non-adaptive) algorithm computing FIND1 with error at most $\frac{1}{2^{\text{polylog}\, n}}$ (figure (3.1)). A naive idea for a faux-deterministic algorithm would be to choose a random tree $\boldsymbol{T} \sim \mathcal{T}_n$ and hope such a tree can fool adversaries. Nevertheless, an adversary can still find an adverse input with $\text{polylog}(n)$ many queries.



(a) A non-adaptive algorithm for FIND1



(b) An adaptive algorithm for FIND1

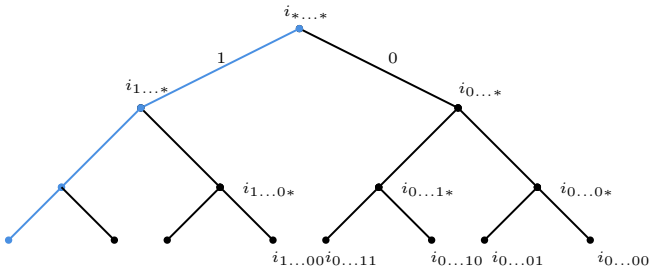
Figure 3.1: In the non-adaptive algorithm, we query a set of uniformly chosen indices $B = \{i_1, \ldots, i_w\}$ and output the first one found in x. In the adaptive algorithm, we choose an independant random index in each of the poossible branches.

Attack sketch for a non-adaptive randomized algorithm

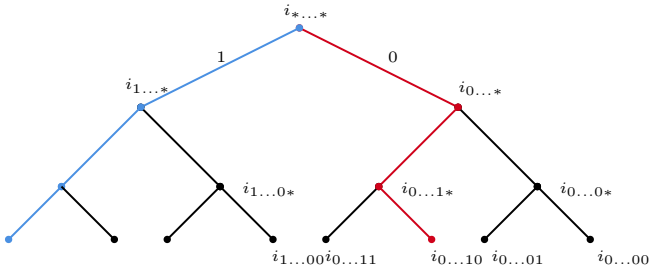
The adversary starts with the string $x(1) = 1^n$ and queries x from $\boldsymbol{T}$. Consequently, $\boldsymbol{T}$ reveals the first randomly chosen index $i_1 = \boldsymbol{T}[x]$. Next, the adversary obtains the second string $x(2) = x(1)^{i_1}$ by making the index i_1 zero. Now, the query $\boldsymbol{T}[x(2)]$ has to reveal i_2 as $x(2)_{i_1} = 0$.
Since the queries $i_1, \ldots, i_w$ are made independently, the adversary can continue this process to obtain inputs $x(3), \ldots, x(w)$ by identifying and modifying the outputted indices $i_2, \ldots, i_w$. Finally, in the input $x(w)$, all the indices $i_1, \ldots, i_w$ queried by the tree are modified to have value $x_{i_j} = 0$ for $j \in [w]$. Therefore, $x(w)$ is an adverse input for $\boldsymbol{T}$. Clearly, this algorithm only queries $w = \text{polylog}(n)$ many inputs.

From this attack, we conclude that non-adaptive queries can easily reveal the structure of the tree. Consequently, the next natural idea would be to make the amplification process adaptive.

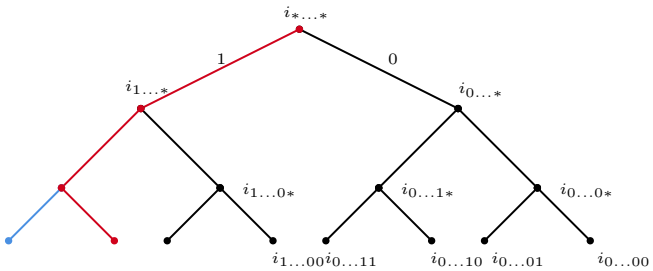
An adaptive randomized idea: Let $b_1 \ldots b_k *^{d-k}$ denote a string of length d that starts with binary string $b_1 \ldots b_k \in \{0,1\}^k$ and ends with $*$ symbols so that it has length d. Our randomized algorithm is a binary decision tree where strings in $\{0,1\}^k *^{d-k}$ denote the label of nodes at depth k. For each such internal label $b_1 \ldots b_k *^{d-k}$, we assign a uniformly independent randomly chosen index $i_{b_1 \ldots b_k *^{d-k}}$. Then, our distribution produces a random tree $\boldsymbol{T'} \sim \mathcal{T'}_n$.



(a) Initial path traversed for input x (blue).



(b) Path generated by changing an early queried index (red).



(c) Path generated by changing a later queried index (red).

Figure 3.2: When changing early queried indices, the whole path of the next queried indices changes. However, changing a later queried index does not change the path of the tree significantly.

We start by querying the root index i_{*^d} to obtain $b_1 = x_{i_{*^d}}$. Note that b_1 denotes whether the first query has outputted a 1-index or a 0-index. Next, based on the value of b_1 (either 0 or 1), we proceed to query either the index of right node $i_{0*^{d-1}}$ or left node $i_{1*^{d-1}}$. We can continue this adaptive query process until we reach a leaf that has a label $b_1 \dots b_d$. Eventually, we output the last querieed index j for which $x_j = 1$ (figure (3.1)).

Now, to find an adverse input for $\boldsymbol{T'}$, the adversary has to find an input x that follows the all-zero branch $(i_{*^d}, i_{0*^{d-1}}, \dots, i_{0^d})$. This is because any other possible branch find at least one index j with $x_j = 1$, and therefore, it leads to a correct output. Thus, the previous attack would not work as zeroing an index can change the path of subsequent queries, and the adversary may need to check many of the possible branches before finding the all-zero branch. On the other hand, the adversary can still succeed by "climbing" the tree and skipping many of the branches with each step. We explain this attack in the following sketch.

Attack sketch for adaptive randomized algorithm

Consider an input x for which the tree has traversed the path of queried indices $(j^1, \dots, j^{d-1}, j^d)$. If the adversary queries input $y = x^{j^1}$, with high probability, all subsequent queries are changed to obtain another path of indices $(j^1, k^2, \dots, k^d)$, where the new queried indices positions $k^2, \dots, k^d$ are independent of $j^2, \dots, j^d$. Thus, the positions of many of the queried indices in y will differ from x with high probability. However, if the adversary switches a later index j^{k-1}, the newly obtained path becomes $(j^1, \dots, j^{k-1}, k^d)$ where only one new query position k^d is independent of j^k. As a result, only a few of the queried indices in y will differ from x (see figure (3.2)).
Furthermore, if an index j is queried in y, the outputs of $\boldsymbol{T'}[y^j]$ and $\boldsymbol{T'}[y]$ will differ with high probability, as the path of subsequent queries changes. Therefore, for any string y we can query $\boldsymbol{T'}[y^j]$ for $j \in [n]$ to learn the queried indices in y. As a result, the adversary can learn the difference of queried indices between x and x^j, and learn for which j many of the queried indices will change. In this way, the adversary can distinguish the initial queries from later ones, manipulate the path from the top, and effectively skip most of the intermediate branches to find the all-zero path.

Consequently, if the tree is “too” adaptive, the adversary can again learn the queries made in the tree. To overcome these problems, we combine both adaptive and non-adaptive queries in the following construction.

3.1.1 Mixing adaptive and non-adaptive queries: tree of $\vee$s

Let w and d be the width and depth parameters. Our construction is a distribution over a set of deterministic trees, where each tree consists of an adaptive binary tree with depth d and each internal node makes non-adaptive queries to w many randomly chosen indices denoted by the block of indices B where $|B| = w$. Next, if any of the indices $i \in B$ is a 1-index ($x_i = 1$), the evaluation of the node is $B(x) = 1$, and we move to the left child. Otherwise, the logical OR ($\vee$) of indices in B equals $B(x) = 0$, and consequently, we move to the right child (see figure (3.3)). Formally, we proceed as follows.

Definition 3.4. Suppose input length $n \in \mathbb{N}$ is given. We define strings $z \in \{0,1\}^{d-k}*^k$ with length d to denote nodes of a binary tree where,

- the root is denoted as string $*^d$;
- the internal nodes at depth k are written as strings

  $$b_1 \dots b_k *^{d-k},$$

  where

  $$b_1, \dots, b_k \in \{0,1\};$$

- and strings $\{0,1\}^d$ denote the 2^d possible leaves of our tree.

For each internal node

$$z = b_1 \dots b_k *^{d-k}$$

with $k < d$, we assign a uniformly independent random blocks of indices

$$\boldsymbol{B}_z \subset [n]$$

of size $|\boldsymbol{B}_z| = w$. On input $x \in \{0,1\}^n$, the algorithm starts at root node $z = *^d$ and recursively proceeds as described below. For each input length n, our distribution $\mathcal{T}_n$ would be a uniform distribution over all the possible trees in the following.

Tree of $\vee$s

On an internal node $z = b_1 \dots b_k *^{d-k}$, we query x for each index $i \in \boldsymbol{B}_z$ and compute

$$b_{k+1} = \boldsymbol{B}_z(x) := \bigvee_{i \in \boldsymbol{B}_z} x_i,$$

and move to the next node $b_1 \dots b_k b_{k+1} *^{d-k-1}$. More specifically,

- If for any of indices $i \in \boldsymbol{B}_z$ we find $x_i = 1$, we move to the corresponding node $b_1 \dots b_k 1 *^{d-k-1}$ because $b_{k+1} = 1$.
- However, if for all $i \in \boldsymbol{B}_z$ we have $x_i = 0$, we move to the node $z0*^{r-1}$ as $b_{k+1} = 0$.

Eventually, we arrive at a leaf $l \in \{0,1\}^d$. If $l = 0^n$, we have not found any index with value 1 in the previously queried blocks. Therefore, we output $\perp$ as an incorrect answer. Otherwise, a unique integer $k < d$ and binary string $b_1 \dots b_k$ exist such that

$$l = b_1 \dots b_k 10^{d-k-1}.$$

Therefore, the internal node $z = b_1 \dots b_k *^{d-k}$ denotes the last node on which the tree has found a 1 on input x. Intuitively, this is the last block on which the execution of the tree goes to the right node. Finally, the algorithm outputs the smallest index $\boldsymbol{i} \in \boldsymbol{B}_z$ with $x_i = 1$.

▷

Before proceeding to prove the effectiveness of $\mathcal{T}_n$ against adversaries, we first define a few notations and prove two important lemmas for our distribution.

3.1.2 Auxiliary definitions and lemmas

Definition 3.5. For each tree $T \in \mathcal{T}_n$ and valid input $x \in \{0,1\}^n$ with $|x|_1 \geq \frac{n}{2}$, we define $l_T(x)$ to be the *string label* corresponding to the leaf obtained by computation of $T[x]$. Thus, the tree T fails to find a 1-index in an input x if and only if $l_T(x) = 0^d$. ▷

Definition 3.6. Additionally, we define an *ordering* $1 \preceq * \preceq 0$ and extend this order to a linear order for all of the internal and leaf labels $z \in \{0,1\}^r *^{d-r}$ with $0 \leq r \leq d$ by

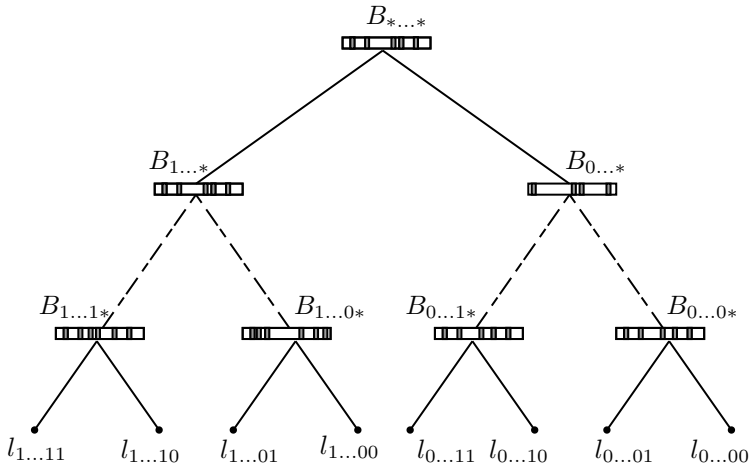


Figure 3.3: An alternating adaptive and non-adaptive algorithm. In each node z, a subset B_z of indices are queried non-adaptively, and based on the value of $\bigvee_{i \in B_z} x_i$, the next subsets and queries are chosen.

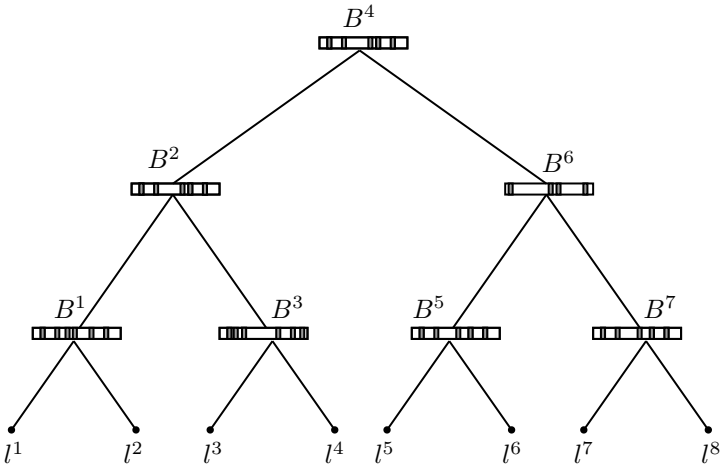


Figure 3.4: An example of the ordered leaves and blocks in a tree of $\vee$s of depth three. Whenever an input reaches a leaf l^k, the algorithm outputs an index from the set B^k which is the last block on which an index 1 has been found.

alphabetic rule (see figure (3.4)). This ordering is very similar to the ordering used in a binary search tree with the difference that it is reversed. More specifically,

$$1^d \prec 1^{d-1}* \prec 1^{d-1}0 \prec \cdots \prec 0^{d-1}1 \prec 0^{d-1}* \prec 0^d.$$

For the sake of conciseness, we also define l^k to be the *label of the k-th largest leaf* for $k \in [2^d]$; In other words, $l^k := (2^d - k)_b$, where $(\cdot)_b$ denotes the binary representation. For instance, we have $l^1 = 1^d, l^2 = 1^{d-1}0, \ldots,$ and $l^{2^d} = 0^d$.

Finally, for each label $z \in \{0,1\}^r *^{d-r}$ with $0 \le r \le d$, we define the *predecessor* and *successor* of z in this order to be z^- and z^+ whenever they exist. ▷

In the following lemma, we relate the leaf $l_T(x)$ obtained by execution T on x and the corresponding block B_z from which the tree outputs $T[x] \in B_z$. In short, if the tree reaches the k-th leaf $l_T(x) = l^k$, the tree outputs an index from the immediately after block $B^k = B_{(l^k)^+}$ in our ordering (see figure (3.4).)

Lemma 3.7. *Let $T \in \mathcal{T}$ be a tree from our construction and x be an arbitrary input.*

i. For every $k \in [2^d - 1]$ and any two consecutive leaf labels $l^{k+1} \succ l^k$, there exists exactly one internal label $z = (l^{k+1})^- = (l^k)^+$ such that

$$l^{k+1} \succ z \succ l^k.$$

Conversely, each internal node z is between two consecutive leaves

$$z^+ \succ z \succ z^-.$$

We denote the k-th largest internal block *as $B^k := B_{(l_k)^+}$.*

ii. If $l_T(x) = l^k$, the algorithm outputs an index from the block B^k.

Proof. For (i), suppose $l^k = b_1 \ldots b_i 10^{d-i-1}$ for some i. Then $l^{k+1} = b_1 \ldots b_i 01^{d-i-1}$, and the only possible string z between l^k and l^{k+1} is $b_1 \ldots b_i *^{d-i}$. Therefore, $z = (l^{k+1})^- = (l^k)^+$. Conversely, each internal node z is between two consecutive leaves z^- and z^+.

To see (ii), let $l_T(x) = l^k = b_1 \ldots b_i 10^{d-i-1}$ for some i. Then, $(l^k)^- = b_1 \ldots b_i *^{d-i}$, which is the last block in the path of $l_T(x)$ that has value $B_{(l^k)^-}(x) = 1$. By definition (3.4), the output of the tree on x comes exactly from this block. Since we defined $B^k = B_{(l^k)^-}$, the algorithm outputs an index from B^k. □

Next, we prove the central lemma for security against adversaries. Note that the main goal of the adversary is to find an input for which $l_T(x) = 0^d$. Knowing the first k blocks $B^1, \ldots, B^k$, the adversary can choose a string x to ensure $x_i = 0$ for every $i \in \bigcup_{j=1}^{k} \boldsymbol{B}^i$. Therefore,

$$\boldsymbol{B}^1(x) = \cdots = \boldsymbol{B}^{k-1}(x) = 0.$$

As a result, the tree cannot output an index from the blocks $\{\boldsymbol{B}^1, \ldots, \boldsymbol{B}^{k-1}\}$. Thus, by the previous lemma, $l_{\boldsymbol{T}}(x) \succeq l^k$. Nevertheless, we claim that if the adversary has no information about the remaining blocks $\boldsymbol{B}^{k+1}, \ldots, \boldsymbol{B}^{2^d}$, then with high probability any such input x has $l_{\boldsymbol{T}}(x) \preceq l^k$.

Lemma 3.8. *Let $x \in \{0,1\}^n$ where $|x|_1 \geq \frac{n}{2}$ be a valid input and $k \in [2^d]$ be a fixed index. Then,*

$$\Pr_{\boldsymbol{T} \sim \mathcal{T}_n} \left[l_{\boldsymbol{T}}(x) \preceq l^k \middle| \boldsymbol{B}^1 = B^1, \ldots, \boldsymbol{B}^{k-1} = B^{k-1} \right] \geq (1 - \frac{1}{2^w})^d.$$

Proof of lemma (3.8). Let l^k be the k leaf as before, and $y \in \{0,1\}^r *^{d-r}$ with $r \in [d]$ be an internal label. We say y is a *1-prefix* of l^k if we can write l^k and y as

$$l^k = \hat{y} 1 \hat{z} \qquad y = \hat{y} * *^{|\hat{z}|}.$$

In the same fashion, we can also define 0-*prefixes*.

Consequently, an input x reaches a leaf $l_{\boldsymbol{T}}(x) = l^k$ if and only if all of the 1-prefixes of l such as y we have $\boldsymbol{B}_y(x) = 1$ and for all of the 0-prefixes such as y' we have $\boldsymbol{B}_{y'}(x) = 0$. We can weaken this fact to obtain the following claim.

Claim 3.9. *If for every 1-prefix y of l^k we have $\boldsymbol{B}_y(x) = 1$, then $l_{\boldsymbol{T}}(x) \preceq l^k$.*

We postpone the proof of claim (3.9) until after proving the lemma. For the lemma, we prove that with high probability for any l^k the conditions of the claim hold. Let y be an arbitrary 1-prefix of l^k where

$$l^k = \hat{y} 1 \hat{z} \qquad y = \hat{y} * *^{|\hat{z}|}.$$

By the defined linear order, we have $y \succ l^k$ since $* \succ 1$. Therefore, by the natural ordering of the leaves and lemma (3.7) we have

$$y \succ l^k \succ (l^k)^- = (l^{k-1})^+ \succ (l^{k-2})^+ \succ \cdots \succ (l^1)^+.$$

Additionally,

$$\boldsymbol{B}^1 = \boldsymbol{B}_{(l^1)^+}, \ldots, \boldsymbol{B}^{k-1} = \boldsymbol{B}_{(l^{k-1})^+}.$$

Therefore, $\boldsymbol{B}_y \notin \{\boldsymbol{B}^1, \ldots, \boldsymbol{B}^{k-1}\}$, and the random set $\boldsymbol{B}_y$ is independent of conditioned sets $\boldsymbol{B}^1, \ldots, \boldsymbol{B}^{k-1}$. Hence, for all $x \in \{0,1\}^n$ with $|x|_1 \geq \frac{n}{2}$,

$$\begin{aligned}\Pr_{\boldsymbol{T}\sim\mathcal{T}_n}\left[\boldsymbol{B}_y(x) = 1 \middle| \boldsymbol{B}_{1^n} = B^1, \ldots, \boldsymbol{B}^{k-1} = B^{k-1}\right] &= \Pr_{\boldsymbol{T}\sim\mathcal{T}_n}\left[\boldsymbol{B}_y(x) = 1\right] \\ &\geq 1 - \frac{1}{2^w},\end{aligned}$$

where in the last step we used the fact that $|x|_1 \geq \frac{n}{2}$. Thus, the probability of missing every 1-index is at most $\frac{1}{2^w}$. To complete the proof, let $y_1, \ldots, y_r$ be all the 1-prefixes of l^k. Note that since $l_k \in \{0,1\}^d$ we have $r \leq d$. Since all the sets $\boldsymbol{B}_{y_1}, \ldots, \boldsymbol{B}_{y_r}$ are independent of each other and $\boldsymbol{B}^1, \ldots, \boldsymbol{B}^{k-1}$, we can write

$$\begin{aligned}&\Pr_{\boldsymbol{T}\sim\mathcal{T}_n}\left[l_{\boldsymbol{T}}(x) \preceq l^k \middle| \boldsymbol{B}^1 = B^1, \ldots, \boldsymbol{B}^{k-1} = B^{k-1}\right] \\ \geq &\Pr_{\boldsymbol{T}\sim\mathcal{T}_n}\left[\boldsymbol{B}_{y_1}(x) = 1, \ldots, \boldsymbol{B}_{y_r}(x) = 1 \middle| \boldsymbol{B}^1 = B^1, \ldots, \boldsymbol{B}^{k-1} = B^{k-1}\right] \\ = &\Pr_{\boldsymbol{T}\sim\mathcal{T}_n}\left[\boldsymbol{B}_{y_1}(x) = 1, \ldots, \boldsymbol{B}_{y_r}(x) = 1\right] \\ = &\Pr_{\boldsymbol{T}\sim\mathcal{T}_n}\left[\boldsymbol{B}_{y_1}(x) = 1\right] \ldots \Pr_{\boldsymbol{T}\sim\mathcal{T}_n}\left[\boldsymbol{B}_{y_r}(x) = 1\right] \\ \geq &(1 - \frac{1}{2^w})^r \\ \geq &(1 - \frac{1}{2^w})^d.\end{aligned}$$

□

Proof of claim (3.9). For the sake of contradiction, suppose $l_{\boldsymbol{T}}(x) \succ l^k$, and let $\hat{y}$ be the largest common prefix of l^k and $l_{\boldsymbol{T}}(x)$ with length m. Since $l_{\boldsymbol{T}}(x) \neq l^k$, we know that l^k and $l_{\boldsymbol{T}}(x)$ differ in at least one index. Additionally, we have $l_{\boldsymbol{T}}(x) \succ l^k$. Therefore, we can write l^k and $l_{\boldsymbol{T}}(x)$ as

$$l^k = \hat{y}1\hat{z} \qquad l_{\boldsymbol{T}}(x) = \hat{y}0\hat{z}'.$$

Now consider the 1-prefix $y = \hat{y}*^{d-m}$ of l^k; since $\hat{y}$ is also a prefix of $l_{\boldsymbol{T}}(x)$, the execution of tree $\boldsymbol{T}$ on x must have visited the internal block $\boldsymbol{B}_y$. Also, since $l_T(x)$ extends $\hat{y}$ by a 0 symbol, the subset $\boldsymbol{B}_y$ must have not found a 1-index in the input x. Therefore, $\boldsymbol{B}_y(x) = 0$, which is a contradiction to the assumption of claim. □

3.2 Security against classical adversaries for FIND1 problem

We can now state our main result in the following theorem.

Theorem 3.10. *Let $\mathcal{T} = (\mathcal{T}_n)$ be the distribution in definition (3.4) with width and depth parameters w and d. Then, $\mathcal{T}_n$ is a $(2^d, \frac{d2^d}{2^w})$-faux deterministic algorithm of complexity wd for the* FIND1 *problem.*

Immediately, we can obtain the following corollary using a single variable.

Corollary 3.11. *The* FIND1 *problem has a $(2^{\frac{\sqrt{t}}{4}}, 2^{-\frac{\sqrt{t}}{4}})$-faux-deterministic algorithm of total depth t. Specifically, for any $\varepsilon > 0$ and $t = \log^{2+\varepsilon}(n)$, we have a $(\text{quasipoly}(n), \text{negl}(n))$-faux-deterministic algorithm of depth $\text{polylog}(n)$.*

Proof of corollary (3.11). Letting $w = \frac{3}{4}\sqrt{t}$ and $d = \frac{1}{4}\sqrt{t}$ we obtain a tree with totap depth $\frac{3}{16}t < t$. By theorem (3.10), any attacker with less than $2^{\frac{1}{4}\sqrt{t}}$ queries will succeed at finding an invalid output with probability at most,

$$\frac{d2^d}{2^w} \leq 2^{2d-w} = 2^{-\frac{1}{4}\sqrt{t}}.$$

For the second part, we have $2^{\frac{\sqrt{\log^{2+\varepsilon}(n)}}{4}} = \text{quasipoly}(n)$ and $2^{-\frac{\sqrt{\log^{2+\varepsilon}(n)}}{4}} = \text{negl}(n)$. □

Proof of theorem (3.10). Clearly, the tree has query complexity $\mathrm{C}^{\mathsf{dt}}(\mathcal{T}_n) = wd$. Intuitively, an arbitrary adversary $\mathcal{A}$ queries an initial string x and with high probability $((1 - \frac{1}{2^w})^d)$ we have

$$\boldsymbol{B}_{*^d}(x) = \cdots = \boldsymbol{B}_{1^{d-1}*} = 1.$$

Thus,

$$l_{\boldsymbol{T}}(x) = 1^d = l^1,$$

and the tree outputs an index from the block $\boldsymbol{B}^1 = \boldsymbol{B}_{1^{d-1}*}$. Now, with high probability, changing any arbitrary index does not change the current branch leading to l^1, as each of the above blocks finds a few indices with a value 1 with a high probability. Therefore, the natural strategy for $\mathcal{A}$ is to alter the outputted indices until it finds all of the indices queried in $\boldsymbol{B}^1$.

Suppose after some steps, the adversary has come to know the indices queried in the blocks $\boldsymbol{B}^1, \ldots, \boldsymbol{B}^{k-1}$. By lemma 3.8, any queried input x will reach to a leaf $l_{\boldsymbol{T}(x)} \preceq l^k$ with

high probability. Therefore, the adversary's only chance would be to continue finding each of the next blocks $\boldsymbol{B}^{k+1}, \ldots, \boldsymbol{B}^{2^d}$ to finally find an input that reaches the all zero path. we will formalize this observation as follows.

Suppose an adversary $\mathcal{A}$ is given a random tree $\boldsymbol{T} \sim \mathcal{T}_n$, and queries q inputs

$$x_{(1)}, x_{(2)}, \ldots, x_{(q)}$$

with $q \leq 2^d - 1$. Without loss of generality, we assume that the adversary outputs the string $x_{(q)}$ as its answer. Also, for the sake of simplicity, we assume that oracle returns additional information as well. More specifically, when querying $x_{(k)}$ with $l_T(x_{(k)}) = l^j$ the oracle returns

- all of the blocks $\boldsymbol{B}^1, \ldots, \boldsymbol{B}^j$,
- and all of the first k blocks $\boldsymbol{B}^1, \ldots \boldsymbol{B}^k$.

We can now lower bound the probability that $l_{\boldsymbol{T}}(x_{(q)}) \neq 0^n$ and also the probability p that the adversary fails as follows.

$$\begin{aligned}
p &:= \Pr_{\boldsymbol{T} \sim \mathcal{T}_n}\left[l_{\boldsymbol{T}}(x_{(q)}) \neq 0^n\right] \\
&\geq \Pr_{\boldsymbol{T} \sim \mathcal{T}_n}\left[l_{\boldsymbol{T}}(x_{(1)}) \preceq l^1, ..., l_{\boldsymbol{T}}(x_{(q)}) \preceq l^q\right] \\
&= \prod_{k=1}^{q} \Pr_{\boldsymbol{T} \sim \mathcal{T}_n}\left[l_{\boldsymbol{T}}(x_{(k)}) \preceq l^k \middle| l_{\boldsymbol{T}}(x_{(1)}) \preceq l^1, \ldots, l_{\boldsymbol{T}}(x_{(k-1)}) \preceq l^{k-1}\right] \\
&= \prod_{k=1}^{q} p_k,
\end{aligned}$$

where p_k is the probability that the k-th query reaches to at most the k-th leaf conditioned on the assumption that all of the previous $k-1$ queries have resulted in a leaf larger than l^k. Therefore, each of these queries has revealed at most the first $k-1$ blocks $\boldsymbol{B}^1, \ldots, \boldsymbol{B}_{k-1}$. Additionally, we assumed that after $k-1$ queries, the oracle returns all of the first $k-1$ blocks. Hence, x_k depends only on the first k blocks and we can write

$$p_k = \Pr_{\boldsymbol{T} \sim \mathcal{T}_n}\left[l_{\boldsymbol{T}}(x_k) \geq l^k \middle| \boldsymbol{B^1} = B^1, \ldots, \boldsymbol{B^{k-1}} = B^{k-1}\right] \geq (1 - \frac{1}{2^w})^d,$$

where we have used the lemma (3.8) in the last inequality for $k \in [q]$. Therefore,

$$p = \prod_{k=1}^{q} p_k \geq \prod_{k=1}^{q} (1 - \frac{1}{2^w})^d \geq (1 - \frac{1}{2^w})^{dq} \geq 1 - \frac{dq}{2^w} \geq 1 - \frac{d2^d}{2^w}. \qquad \square$$

3.3 Quantum secure faux-deterministic algorithms

Eventually, we aim to adjust the classical result of theorem (3.10) with the expected quadratic loss to obtain security against quantum adversaries as well. However, we need to concretely define a quantum adversary and quantum-secure faux-deterministic algorithms before proceeding.

Let T be a decision tree attempting to solve the search problem $\mathcal{S}$ on inputs of length n. As before, the goal of the adversary is to find an adverse input $x \in \mathsf{dom}(\mathcal{S}_n)$ where $T[x] \notin \mathcal{S}_n(x)$. Additionally, we want the adversary to query outputs of T in superposition using a unitary U^T that on state $|x\rangle |0\rangle$ outputs $|x\rangle |T[x] + y\rangle$.

Definition 3.12. Let T be decision tree taking inputs in $\mathsf{dom}(\mathcal{S}_n) \subset \Sigma^n$ and outputting answers in Γ^m. Additionally, let I and Q be quantum input (index) and query registers consisting of $n \log |\Sigma|$ and $m \log |\Gamma|$ qubits, respectively. We define the query unitary U^T for T on these registers to be

$$U^T |x\rangle_I |q\rangle_Q = \begin{cases} |x\rangle_I |T[x] \oplus q\rangle_Q & x \in \mathsf{dom}(\mathcal{S}_n), \\ |x\rangle_I |q\rangle_Q & \text{otherwise.} \end{cases}$$

▷

Remark. The tree is only attempting to work on inputs $x \in \mathsf{dom}(\mathcal{S}_n)$. Therefore, querying inputs outside of the promise should not reveal any information about the tree. ▷

We can continue to define a quantum adversary for a decision tree T and search problem $\mathcal{S}$ using the unitary U^T. Informally, the adversary $\mathcal{Q} = (U_1, \ldots, U_q)$ starts with an initial state $|\psi_0\rangle$ and applies q queries to obtain the final state,

$$|\psi_q\rangle = U_q \boldsymbol{U^T} \ldots \boldsymbol{U^T} U_1 \boldsymbol{U^T} |\psi_0\rangle$$

and output an answer based on the measurement of this $|\psi_q\rangle$.

Definition 3.13. Suppose $\mathcal{S}$ is a search problem over input and output alphabets Σ and Γ and input length $n \in \mathbb{N}$. We define registers (I, Q, W, O) to be index (input), query, work, and output registers consisting of $n \log |\Sigma|$, $m \log |\Gamma|$, r, and $n \log |\Sigma|$ qubits, respectively. A q-query quantum adversary $\mathcal{Q} = (U_1, \ldots, U_q)$ for T prepares an initial state,

$$|\psi_0\rangle = \sum_{x,q,w,y} \alpha_{x,q,w,y} |x\rangle_I |q\rangle_Q |w\rangle_W |y\rangle_O ,$$

and in each step, it applies unitaries $U^{\boldsymbol{T}}$ and U_{t+1} to obtain the next state,

$$|\psi_{t+1}\rangle = U_{t+1}U^{T}|\psi_t\rangle .$$

Next, it measures the last state $|\psi_q\rangle$ to obtain $|x\rangle_I |q\rangle_Q |w\rangle_W |y\rangle_O$ for some x, q, w, and y, and finally outputs $\mathcal{Q}[T] := y$. ▷

Having defined the above basic notions, we can now proceed to generalize the notion of faux-deterministic algorithms for quantum adversaries as well. Similar to the classical definition, we want any adversary making at most q queries fails with probability at least $1-\varepsilon$. However, unlike the classical setting, we also need to consider the randomness inside the measurements of a quantum adversary as well.

Definition 3.14 (quantum-secure faux-determinism in query model)**.** Let $\mathcal{S}$ be a search problem, $q = q(n) \in \mathbb{N}$ be query upper bound, and $\varepsilon = \varepsilon(n) \geq 0$ be an error parameter. We say distribution $\mathcal{T} = (\mathcal{T}_n)$ of decision trees is a *quantum secure* (q, ε)*-faux-deterministic* algorithm for $\mathcal{S}$ if, for all $n \in \mathbb{N}$, and any quantum adversary $\mathcal{Q}_n$ making less than $q(n)$ quantum queries we have

$$\Pr_{\boldsymbol{T}\sim\mathcal{T}_n,\mathcal{Q}}\Big[\boldsymbol{x} \leftarrow \mathcal{Q}[\boldsymbol{T}]; \boldsymbol{T}[\boldsymbol{x}] \notin \mathcal{S}(\boldsymbol{x})\Big] \leq \varepsilon(n). \qquad \triangleright$$

Remark. We can also define the quantum-secure faux-deterministic algorithm with three parameters (q, δ, ϵ) to have

$$\Pr_{\boldsymbol{T}\sim\mathcal{T}_n}\Big[p_{\mathcal{Q}}(\boldsymbol{T}) \geq \delta(n)\Big] \leq \varepsilon(n),$$

where $p_{\mathcal{Q}}(\boldsymbol{T})$ is the probability that algorithm $\mathcal{Q}$ fails on $\boldsymbol{T}$ over the internal randomness of $\mathcal{Q}$

$$p_Q(\boldsymbol{T}) = \Pr_{\mathcal{Q}}\big[\boldsymbol{x} \leftarrow \mathcal{Q}[\boldsymbol{T}]; \boldsymbol{T}[\boldsymbol{x}] \notin \mathcal{S}(\boldsymbol{x})\big].$$

The alternative definition is equivalent to saying that the adversary fails with a probability of at least $(1-\delta)$ for at least $(1-\varepsilon)$ fraction of the trees. Ultimately, a (q, δ, ϵ)-secure algorithm with the alternative definition is a $(q, \delta+\epsilon)$-secure algorithm with respect to the

original definition. I.e.,

$$\begin{aligned}
&\Pr_{\boldsymbol{T}\sim\mathcal{T}_n,\mathcal{Q}}\left[\boldsymbol{x}\leftarrow\mathcal{Q}[\boldsymbol{T}];\boldsymbol{T}[\boldsymbol{x}]\notin\mathcal{S}(\boldsymbol{x})\right]\\
=&\Pr_{\boldsymbol{T}\sim\mathcal{T}_n,\mathcal{Q}}\left[\boldsymbol{x}\leftarrow\mathcal{Q}[\boldsymbol{T}];\boldsymbol{T}[\boldsymbol{x}]\notin\mathcal{S}(\boldsymbol{x})\middle|p_Q(\boldsymbol{T})\geq\delta\right]\Pr_{\boldsymbol{T}\sim\mathcal{T}_n,\mathcal{Q}}\left[p_Q(\boldsymbol{T})\geq\delta\right]\\
+&\Pr_{\boldsymbol{T}\sim\mathcal{T}_n,\mathcal{Q}}\left[\boldsymbol{x}\leftarrow\mathcal{Q}[\boldsymbol{T}];\boldsymbol{T}[\boldsymbol{x}]\notin\mathcal{S}(\boldsymbol{x})\middle|p_Q(\boldsymbol{T})\leq\delta\right]\Pr_{\boldsymbol{T}\sim\mathcal{T}_n,\mathcal{Q}}\left[p_Q(\boldsymbol{T})\leq\delta\right]\\
\leq&\epsilon+\delta.
\end{aligned}$$

Conversely, (q,ϵ) security in the original definition translates into an $(q,\delta,\frac{\epsilon}{\delta})$ security of the alternative definition as well.

$$\begin{aligned}
\Pr_{\boldsymbol{T}}\left[\Pr_{\mathcal{Q}}\left[\boldsymbol{x}\leftarrow\mathcal{Q}[\boldsymbol{T}];\boldsymbol{T}[\boldsymbol{x}]\notin\mathsf{FIND1}(x)\right]\geq\delta\right]\leq&\frac{\mathbb{E}_{\boldsymbol{T}}[\Pr_{\mathcal{Q}}[\boldsymbol{x}\leftarrow\mathcal{Q}[\boldsymbol{T}];\boldsymbol{T}[\boldsymbol{x}]\notin\mathsf{FIND1}(\boldsymbol{x})]}{\delta}\\
\leq&\frac{\varepsilon}{\delta}.
\end{aligned}$$

▷

3.4 Security against quantum adversaries for FIND1

We prove the adjusted theorem for security against quantum adversaries with a similar idea to hybrid method first introduced in [Ben+97].

Theorem 3.15. *Let w and d be the width and depth parameters as before. The distribution $\mathcal{T}=(\mathcal{T}_n)$ in definition (3.4) is a quantum-secure $(2^d, 2(2^d\sqrt{d}+\frac{1}{2})^{\frac{2}{3}}2^{-\frac{w}{3}})$-faux deterministic algorithm for* FIND1*.*

Similar to the classical case, we can obtain a simpler form with one variable as below. Note that for the quantum security, we have the standard Grover type quadratic loss [Gro96] where the parameter $2^{\frac{\sqrt{t}}{4}}$ is adjusted to

$$2^{\frac{\sqrt{t}}{8}}=\sqrt{2^{\frac{\sqrt{t}}{4}}}.$$

Corollary 3.16. *The* FIND1 *problem has a quantum-secure $(2^{\frac{\sqrt{t}}{8}}, 2^{-\frac{\sqrt{t}}{8}})$-faux-deterministic algorithm of total depth t. Specifically, for any $\varepsilon>0$ and $t=\log^{2+\varepsilon}(n)$, we have a quantum-secure $(\mathrm{quasipoly}(n),\mathrm{negl}(n))$-faux-deterministic algorithm of depth $\mathrm{polylog}(n)$.*

Proof of corollary (3.16). For large enough t, letting $w = \frac{7}{8}\sqrt{t}$ and $d = \frac{1}{8}\sqrt{t}$ in theorem (3.15), any attacker with less than $2^{\frac{1}{8}\sqrt{t}}$ queries will succeed at finding an invalid output with probability at most

$$2(2^d\sqrt{d} + \frac{1}{2})^{\frac{2}{3}} 2^{-\frac{w}{3}} \leq 2^{\frac{4d}{3}} w^{-\frac{w}{3}} = 2^{\frac{4}{24}\sqrt{t} - \frac{7}{24}\sqrt{t}} = 2^{-\frac{1}{8}\sqrt{t}}. \qquad \square$$

Proof of theorem (3.15). Let $\mathcal{Q} = (U_1, \ldots, U_q)$ be a quantum algorithm that makes q queries to the randomly chosen decision tree $\boldsymbol{T}$. The main idea is to construct a series of quantum algorithms $\mathcal{Q}^0 = \mathcal{Q}$, $\mathcal{Q}^1$, ..., and $\mathcal{Q}^q = \mathcal{Q}'$ such that for $k \in [q-1]$,

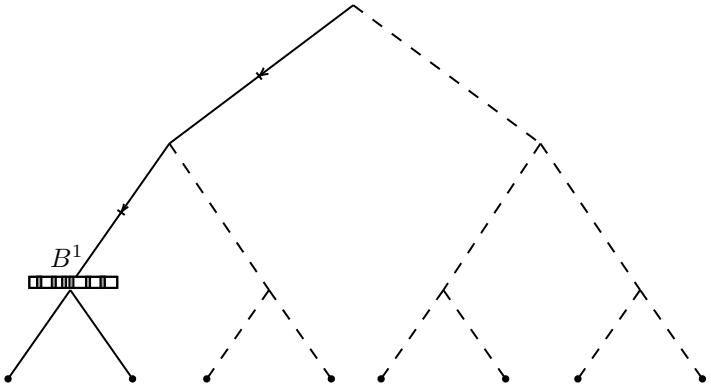
- $\mathcal{Q}^{k+1}$ has access to the first k block $\boldsymbol{B}^1, \ldots, \boldsymbol{B}^k$;
- $\mathcal{Q}^{k+1}$ replaces the first query $U^{\boldsymbol{T}}$ of $\mathcal{Q}^k$ with a simulated query $U^{\boldsymbol{T^{k+1}}}$;
- $\mathcal{Q}^{k+1}$ makes $q-k$ quantum queries to $U^{\boldsymbol{T}}$ in total.

Then, we make the following steps to prove $\mathcal{Q}$ fails with high probability.

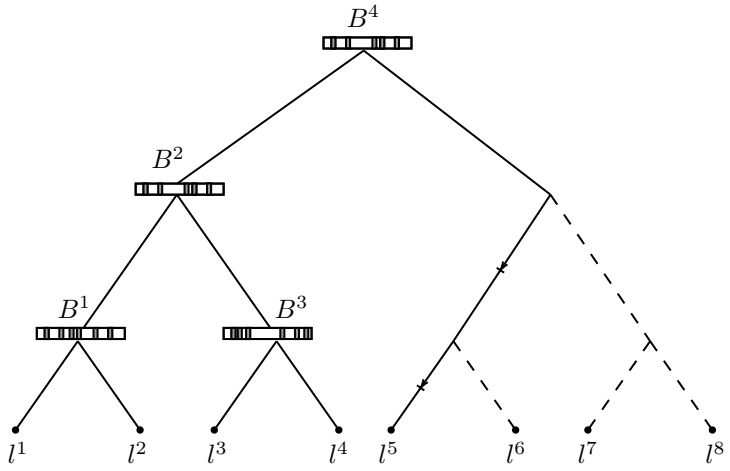
1. We prove that the distance between the pre-measurement final states of $\mathcal{Q}^{k+1}$ and $\mathcal{Q}^k$ will be very small with high probability over the choice of trees.
2. Consequently, we will also bound the distance between the pre-measurement of $\mathcal{Q} = \mathcal{Q}^0$ and $\mathcal{Q}' = \mathcal{Q}^q$.
3. We use the above to prove the success probability of $\mathcal{Q}$ will be close to the success probability of $\mathcal{Q}'$;
4. Finally, we prove $\mathcal{Q}'$ fails with high probability as it does not make any actual queries to the tree $\boldsymbol{T}$.

Consequently, $\mathcal{Q}$ will also fail with high probability.

We start with the definition $\mathcal{Q}^1$ as an illustration. Suppose $\mathcal{Q}^1$ knows the set $\boldsymbol{B}^1$. At the start, $\mathcal{Q}^1$ prepares the same state $|\psi_0\rangle$ as in $\mathcal{Q}$, and simulates the first query to tree $U^{\boldsymbol{T}}$ with another unitary $U^{\boldsymbol{T^1}}$, where the tree $\boldsymbol{T^1}$ is an approximation of the original tree $\boldsymbol{T}$. More specifically, to compute $\boldsymbol{T^1}[x]$ on input x, we start at the root of $\boldsymbol{T}$ and answer with 1 on all of the unknown intermediate nodes until we reach the left-most node $\boldsymbol{B^1}$ (See figure (3.2b)). Now, similar to $\boldsymbol{T}$, we output the first found index $i \in \boldsymbol{B^1}$ with $x_i = 1$ whenever such i exists. Otherwise, we output a random index $j \in [n]$. Note that for any



(a) The simulated tree $\boldsymbol{T^1}$ from $\boldsymbol{T}$ with the knowledge of $\boldsymbol{B}^1$.



(b) The simulated tree $\boldsymbol{T^4}$ from $\boldsymbol{T}$ with the knowledge of $\boldsymbol{B}^1$, $\boldsymbol{B}^2$, $\boldsymbol{B}^3$, and $\boldsymbol{B}^4$.

Figure 3.5: In both of the simulated trees $\boldsymbol{T^1}$ and $\boldsymbol{T^4}$ the quantum algorithm simulating them only computes the nodes of the tree for which the blocks are known. For instance, for any input x with $l_{\boldsymbol{T}}(x) \geq l^4$ the computed values of $\boldsymbol{T}$ and $\boldsymbol{T^4}$ are equal; $\boldsymbol{T}[x] = \boldsymbol{T^4}[x]$.

input x with $l_T(x) = l^1$ we have $\boldsymbol{T}[x] = \boldsymbol{T^1}(x)$. Therefore, in the new tree $\boldsymbol{T^1}$, we assume that all inputs $|x\rangle_I$ in $|\psi_0\rangle$ with non-zero amplitudes have $l_{\boldsymbol{T}}(x) \geq l^1$.

We can now define the approximated unitary using our knowledge of the block $\boldsymbol{B}^1$.

$$U^{\boldsymbol{T^1}} |x\rangle_I |i\rangle_O = \begin{cases} |x\rangle_I |\boldsymbol{T^1}[x] \oplus i\rangle_O & |x|_1 \geq \frac{n}{2}, \\ |x\rangle_I |i\rangle_O & \text{otherwise.} \end{cases}$$

By approximating the first query, $\mathcal{Q}^1$ obtains the state,

$$\left|\psi_1^1\right\rangle := U_1 U^{\boldsymbol{T^1}} |\psi_1\rangle .$$

Afterward, $\mathcal{Q}^1$ proceeds with the regular queries to unitary $U^{\boldsymbol{T}}$, obtains the states

$$\left|\psi_{t+1}^1\right\rangle = U_{t+1} U^{\boldsymbol{T}} \left|\psi_t^1\right\rangle .$$

for $t > 1$. Finally, $\mathcal{Q}^1$ measures the last state $\left|\psi_q^1\right\rangle$ to obtain a corresponding output similar to $\mathcal{Q}^0$. To obtain the remaining algorithms, we proceed similarly.

Constructing $\mathcal{Q}^{k+1}$ from $\mathcal{Q}^k$.

Suppose $\mathcal{Q}^k$ has been created, and $\mathcal{Q}^{k+1}$ has access to the first $k+1$ first blocks, $\boldsymbol{B}^1, \ldots, \boldsymbol{B}^k$. We define $\boldsymbol{T^{k+1}}[x]$ to simulate the tree $\boldsymbol{T}$, where on unknown blocks we answer with 1, and on the known blocks we answer similar to $\boldsymbol{T}$ by the knowledge of $\boldsymbol{B}^1, \ldots, \boldsymbol{B}^k$. Finally, we output a 1 from the last known block with $\boldsymbol{B}^i(x) = 1$ if any such i exists. Otherwise, we output a random index (see figure (3.5)).
Next, we replace the first real query $U^{\boldsymbol{T}}$ in $\mathcal{Q}^k$ with a simulated query $U^{\boldsymbol{T^{k+1}}}$ where for $x \in \{0,1\}^n$ we define

$$U^{\boldsymbol{T^{k+1}}} |x\rangle_I |i\rangle_O = \begin{cases} |x\rangle_I \left|\boldsymbol{T^{k+1}}[x] \oplus i\right\rangle_O & |x|_1 \geq \frac{n}{2}, \\ |x\rangle_I |i\rangle_O & \text{otherwise.} \end{cases}$$

Therefore, $\mathcal{Q}^{k+1}$ obtains,

$$\left|\psi_{k+1}^{k+1}\right\rangle := U_{k+1} U^{\boldsymbol{T^{k+1}}} \left|\psi_k^k\right\rangle ,$$

and for $t \geq k+1$ it continues to query regularly as

$$\left|\psi_{t+1}^{k+1}\right\rangle = U_{t+1} U^{\boldsymbol{T}} \left|\psi_t^{k+1}\right\rangle .$$

Finally, $\mathcal{Q}^{k+1}$ measures the state $\left|\psi_q^{k+1}\right\rangle$ to output a string x.

1. **Bounding the distance of $\mathcal{Q}^k$ and $\mathcal{Q}^{k+1}$:** Let l^k be the k-th leaf in the tree as in the proof of theorem (3.10). For any valid input x with $|x|_1 \geq \frac{n}{2}$, if $l_{\boldsymbol{T}}(x) \preceq l^k$, the tree only visits the blocks $\boldsymbol{B}^1, \ldots, \boldsymbol{B}^k$ and outputs from one of these blocks. Thus, for any such input x,

$$l_T(x) \preceq l^k \implies \boldsymbol{T}^{k+1}[x] = \boldsymbol{T}[x].$$

Therefore, $U^{\boldsymbol{T}}[x] = U^{\boldsymbol{T^{k+1}}[x]}$ unless $l_{\boldsymbol{T}}(x) \succ l^k$. Furthermore, $\mathcal{Q}^k$ does not have any information about $\boldsymbol{B}^{k+1}, \ldots, \boldsymbol{B}^{2^d}$ before its first real queries. Consequently, employing lemma (3.8) we can prove that $l_T(x) \succeq l^k$ happens with high probability for every input $|x\rangle$ in the query of $\mathcal{Q}^k$. We conclude that the states in consecutive algorithms remain close to each other.

Lemma 3.17. *For $k \in [q]$, $\mathbb{E}_{\boldsymbol{T}}\left[\left\| \left|\psi_q^{k+1}\right\rangle - \left|\psi_q^k\right\rangle \right\|\right] \leq \sqrt{\frac{4d}{2^w}}$.*

Proof. See appendix (A). □

2. **Bounding the distance between $\mathcal{Q}$ and $\mathcal{Q}'$:** Let $|\psi\rangle = \left|\psi_q^0\right\rangle$ and $|\psi'\rangle = \left|\psi_q^q\right\rangle$ be the final state before measurement when running the algorithms $\mathcal{Q}$ and $\mathcal{Q}'$, respectively. We can further bound the distance between these two states as

$$\begin{aligned}
\mathop{\mathbb{E}}_{\boldsymbol{T}}\left[\left\| |\psi'\rangle - |\psi\rangle \right\|\right] &= \mathop{\mathbb{E}}_{\boldsymbol{T}}\left[\left\| \sum_{k=1}^{q} \left|\psi_q^k\right\rangle - \left|\psi_q^{k-1}\right\rangle \right\|\right] \\
&\leq \mathop{\mathbb{E}}_{\boldsymbol{T}}\left[\sum_{k=1}^{q} \left\| \left|\psi_q^k\right\rangle - \left|\psi_q^{k-1}\right\rangle \right\|\right] \\
&= \sum_{k=1}^{q} \mathop{\mathbb{E}}_{\boldsymbol{T}}\left[\left\| \left|\psi_q^k\right\rangle - \left|\psi_q^{k-1}\right\rangle \right\|\right] \\
&\leq q\sqrt{\frac{4d}{2^w}}, \qquad (3.1)
\end{aligned}$$

where the second line is due to the triangle inequality, the third line is due to linearity of expectation, and the fourth line is by lemma (3.17).

3. **Relating success probabilities of $\mathcal{Q}$ and $\mathcal{Q}'$:** To compare the probability of success for $\mathcal{Q}$ and $\mathcal{Q}'$, we define projection $\Pi_1(T)$ to be the projection to states having input registers $|x\rangle_I$, where x is an adverse input for T. I.e., valid inputs $|x|_1 \geq \frac{n}{2}$ on which the tree T fails to find a 1-index. Similarly, we can define $\Pi_0(T)$ to be the projection to input registers on which the tree T succeeds or x is invalid.

Consequently, $\Pi_0(T)+\Pi_1(T) = I$, and probability of success for $\mathcal{Q}$ and $\mathcal{Q}'$ is computed as

$$p(T) := \|\Pi_1(T)\,|\psi\rangle\,\|^2 = \Pr_{\mathcal{Q}}\left[\boldsymbol{x} \leftarrow \mathcal{Q}[T]; |x| \geq \frac{n}{2} \text{ and } T[\boldsymbol{x}] \notin \mathsf{FIND1}(\boldsymbol{x})\right],$$
$$p'(T) := \|\Pi_1(T)\,|\psi'\rangle\,\|^2 = \Pr_{\mathcal{Q}'}\left[\boldsymbol{x} \leftarrow \mathcal{Q}'[T]; |x| \geq \frac{n}{2} \text{ and } T[\boldsymbol{x}] \notin \mathsf{FIND1}(\boldsymbol{x})\right].$$

Hence, for any tree $T \in \mathcal{T}_n$,

$$\begin{aligned}\sqrt{p(T)} &= \left\|\Pi_1\,|\psi\rangle\,\right\| \\ &= \left\|\Pi_1\,|\psi\rangle - \Pi_1\,|\psi'\rangle + \Pi_1\,|\psi'\rangle\,\right\| \\ &\leq \left\|\Pi_1\big(\,|\psi\rangle - |\psi'\rangle\,\big)\right\| + \left\|\Pi_1\,|\psi'\rangle\,\right\| \\ &\leq \left\|\,|\psi\rangle - |\psi'\rangle\,\right\| + \sqrt{p'(T)},\end{aligned}$$

where we have used triangle inequality in the third line and the fact that all eigenvalues of Π_1 have a norm at most 1 in the last line. Getting expectations from both sides, we obtain

$$\mathop{\mathbb{E}}_{\boldsymbol{T}}\left[\sqrt{p(\boldsymbol{T})}\right] \leq \mathop{\mathbb{E}}_{\boldsymbol{T}}\left[\left\|\,|\psi\rangle - |\psi'\rangle\,\right\|\right] + \mathop{\mathbb{E}}_{\boldsymbol{T}}\left[\sqrt{p'(\boldsymbol{T})}\right].$$

4. **Bounding the success probability:** By eq. (3.1), the first expectation in the above is upper bounded by $q\sqrt{\frac{4d}{2^w}}$. To upper bound $\mathbb{E}_T\left[\sqrt{p'(T)}\right]$, note that $\mathcal{Q}'$ depends only on the first q blocks, with $q < 2^d - 1$. As a result, last block $\boldsymbol{B}^{2^d}$ is independent from the output of $\mathcal{Q}'$. Therefore, we can easily show the following lemma.

   Lemma 3.18. $\mathbb{E}_{\boldsymbol{T}}\left[\sqrt{p'(\boldsymbol{T})}\right] \leq \sqrt{\frac{1}{2^w}}$.

   Proof. See appendix (A). □

   Therefore, we can resume upper bounding the expectation as

$$\mathop{\mathbb{E}}_{\boldsymbol{T}}\left[\sqrt{p(\boldsymbol{T})}\right] \leq q\sqrt{\frac{4d}{2^w}} + \sqrt{\frac{1}{2^w}} = (2q\sqrt{d}+1)2^{-\frac{w}{2}}.$$

   Using Markov's inequality [Ros98; AS16], we have

$$\Pr_{\boldsymbol{T}}[p(\boldsymbol{T}) \geq \delta^2] \leq \frac{\mathbb{E}_{\boldsymbol{T}}\left[\sqrt{p(\boldsymbol{T})}\right]}{\delta} \leq \frac{(2q\sqrt{d}+1)2^{-\frac{w}{2}}}{\delta},$$

and therefore,

$$\begin{aligned}
p :=& \Pr_{\boldsymbol{T},\mathcal{Q}}\left[\boldsymbol{x} \leftarrow \mathcal{Q}[\boldsymbol{T}]; \boldsymbol{T}[\boldsymbol{x}] \notin \mathsf{FIND1}\right] \\
=& \Pr_{\boldsymbol{T},\mathcal{Q}}\left[\boldsymbol{x} \leftarrow \mathcal{Q}[\boldsymbol{T}]; \boldsymbol{T}[\boldsymbol{x}] \notin \mathsf{FIND1} \middle| p(\boldsymbol{T}) \geq \delta^2\right] \Pr_{\boldsymbol{T},\mathcal{Q}}\left[p(\boldsymbol{T}) \geq \delta^2\right] \\
&+ \Pr_{\boldsymbol{T},\mathcal{Q}}\left[\boldsymbol{x} \leftarrow \mathcal{Q}[\boldsymbol{T}]; \boldsymbol{T}[\boldsymbol{x}] \notin \mathsf{FIND1} \middle| p(\boldsymbol{T}) \leq \delta^2\right] \Pr_{\boldsymbol{T},\mathcal{Q}}\left[p(\boldsymbol{T}) \leq \delta^2\right] \\
\leq& \Pr_{\boldsymbol{T}}\left[p(\boldsymbol{T}) \geq \delta^2\right] + \Pr_{\boldsymbol{T},\mathcal{Q}}\left[\boldsymbol{x} \leftarrow \mathcal{Q}[\boldsymbol{T}]; \boldsymbol{T}[\boldsymbol{x}] \notin \mathsf{FIND1} \middle| p(\boldsymbol{T}) \leq \delta^2\right] \\
\leq& \frac{(2q\sqrt{d}+1)2^{-\frac{w}{2}}}{\delta} + \delta^2.
\end{aligned}$$

Minimizing the righ hand side for δ, we have $2\delta = \frac{(2q\sqrt{d}+1)2^{-\frac{w}{2}}}{\delta^2}$ and thus, $\delta = (q\sqrt{d} + \frac{1}{2})^{\frac{1}{3}}2^{-\frac{w}{6}}$. Finally, since $q \leq 2^d - 1$

$$p \leq 2(2^d\sqrt{d} + \frac{1}{2})^{\frac{2}{3}}2^{-\frac{w}{3}}. \qquad \square$$

3.5 Upper bounds on breaking faux-deterministic algorithms for FIND1

Thus far, we have constructed $(2^{\sqrt{\frac{t}{4}}}, 2^{-\sqrt{\frac{t}{4}}})$ and $(2^{\sqrt{\frac{t}{8}}}, 2^{-\sqrt{\frac{t}{8}}})$ faux-deterministic algorithms of depth t for classical and quantum adversaries, which proves an exponential gap between the complexity of our construction t and our lower bounds, $2^{O(t^{\frac{1}{2}})}$. From the viewpoint of the adversary, there exists 2^{n-1} many different possible inputs in $\{0,1\}^n$ that have hamming weight at least $\frac{n}{2}$, and the given input length for the adversary is $N = 2^{n-1}$. As a result, an efficient adversary in terms of its input size needs to make at most $\text{polylog}(O(2^n)) = \text{poly}(n)$. Therefore, for $t = \text{polylog}(n)$, an efficient adversary has to make at least $\text{quasipoly}(n)$ many queries to have a non-negligible chance of finding an adverse input, and our lower bounds are secure against any efficient adversaries.

On the other hand, in terms of query complexity, the $\text{quasipoly}(n)$ lower bound is still very far from the adversary's input size of $O(2^n)$. Therefore, a natural question is whether such a lower bound is achievable and, if not, how tight our lower bounds are. In the following proposition, we give a negative answer to the first question by proving that with

at most $O(2^t)$ many queries we can find an adverse input with constant probability for any given initial distribution. Consequently, for $t = \text{polylog}(n)$, we get a quasipoly(n) upper bound, which roughly matches our lower bound.

Proposition 3.19. *Let $\mathcal{T}_n$ be any distribution of decision trees with total depth $t \leq \frac{\sqrt{n}}{2}$ attempting to solve* FIND1 *on inputs of length n. Then, for $k \in \mathbb{N}$, $\mathcal{T}_n$ cannot be $(k, 1 - e^{\frac{k}{2^{2t+1}}})$-faux-deterministic. Specifically, there exists an adversary that makes $k = 2^{t+1}$ many queries and finds an adverse input for $\frac{1}{2}$ fraction of the trees in the distribution.*

Proof. Let T be any fixed decision tree of depth $t \leq \frac{\sqrt{n}}{2}$. We define $I := \{i_1, \ldots, i_t\}$ as the queries made by T in the all-zero branch. Thus, an string x with $|x| \geq \frac{n}{2}$ is an adverse input if and only if $x_i = 0$ for every $i \in I$. Let k be a natural number to be determined later. We consider a simple randomized adversary $\mathcal{A}$ that for $j \in [k]$, chooses uniformly random independent strings $\boldsymbol{X_j}$ with $|\boldsymbol{X_j}| = \frac{n}{2}$ and queries these strings to obtain $T[\boldsymbol{X_j}]$. Eventually, $\mathcal{A}$ outputs any string on which T fails to output a 1 index.

For a fixed $j \in [k]$, consider one of the strings $\boldsymbol{X_j}$. We define $\boldsymbol{S_j} \subset [n]$ to be the set of indices in $\boldsymbol{X_j}$ with value 0. Therefore, for $\boldsymbol{X_j}$ to be an adverse input we need to have $I \subset |\boldsymbol{S}_j|$. In total, we can choose any $\frac{n}{2} - t$ of the remaining $n - t$ indices for $\boldsymbol{X_j}$ to be an adverse input. On the other hand, we have $\binom{n}{\frac{n}{2}}$ possible choices for $\boldsymbol{X_j}$. Consequently, the probability that $I \subset |\boldsymbol{S}_j|$ can be lower bounded as

$$\begin{aligned}
\Pr_{\boldsymbol{X_j}}[T[\boldsymbol{X_j}] \notin \text{FIND1}(\boldsymbol{X_j})] &= \Pr_{\boldsymbol{X_j}}[I \subset \boldsymbol{S_j}] \\
&= \frac{\binom{n-t}{\frac{n}{2}-t}}{\binom{n}{\frac{n}{2}}} \\
&= \frac{\frac{n}{2} \ldots (\frac{n}{2} - t + 1)}{n \ldots (n - t + 1)} \\
&\geq \frac{(\frac{n}{2} - t)^t}{n^t} \\
&= \frac{1}{2^t}(1 - \frac{2t}{n})^t \\
&\geq \frac{1}{2^t}(1 - \frac{2t^2}{n}) \\
&\geq \frac{1}{2^{t+1}},
\end{aligned}$$

where in the sixth line we have used Bernoulli's inequality and in the last line we have used the assumption $t \leq \frac{\sqrt{n}}{2}$. Since $\boldsymbol{X_1}, \ldots, \boldsymbol{X_k}$ are independent, the probability that the

adversary succeeds in one of its queries is at least

$$\Pr_{\boldsymbol{X_1},\ldots,\boldsymbol{X_k}}\left[\bigvee_{j=1}^{k}\boldsymbol{T}[\boldsymbol{X_j}] \notin \mathsf{FIND1}(\boldsymbol{X_j})\right] \geq 1-(1-\frac{1}{2^{t+1}})^k \geq 1-e^{-\frac{k}{2^{t+1}}}.$$

Thus, for any tree T, after k iterations the algorithm succeeds with probability $1-e^{-\frac{k}{2^{t+1}}}$. Now, for a fixed distribution $\mathcal{T}$ and a random tree $\boldsymbol{T} \sim \mathcal{T}$ we can write

$$\begin{aligned}
&\Pr_{\boldsymbol{X}_1,\ldots,\boldsymbol{X}_k}\left[\Pr_{\boldsymbol{T}\sim\mathcal{T}_n}\left[\bigvee_{j=1}^{k}\boldsymbol{T}[\boldsymbol{X_j}] \notin \mathsf{FIND1}(\boldsymbol{X_j})\right]\right] \\
&= \Pr_{\boldsymbol{T}\sim\mathcal{T}_n}\left[\Pr_{\boldsymbol{X}_1,\ldots,\boldsymbol{X}_k}\left[\bigvee_{j=1}^{k}\boldsymbol{T}[\boldsymbol{X_j}] \notin \mathsf{FIND1}(\boldsymbol{X_j})\right]\right] \\
&\geq 1-e^{-\frac{k}{2^{t+1}}}.
\end{aligned}$$

Finally, maximizing the outer probability over the choices of $\boldsymbol{X}_1,\ldots,\boldsymbol{X}_k$, we obtain fixed strings $X_1,\ldots,X_k$ and a deterministic adversary that succeeds with probability at least $1-e^{-\frac{k}{2^{t+1}}}$ on the distribution $\mathcal{T}_n$. □

Remark. The adversarial upper bound given in proposition (3.19) is a non-adaptive algorithm that works on any distribution. However, in many cases - such as the examples discussed in the previous sections - for any fixed distribution $\mathcal{T}_n$, we can usually find much more efficient heuristic adversaries that learn the given tree in fewer queries. It is also important to note that this observation is not limited to distributions with a lot of structure, such as our construction. For example, a similar climbing method of finding earlier queries can be used to find an adverse input for the distribution of a uniformly random tree of depth t. As a result, the error probability in the above upper bound does not behave robustly to the number of queries. For instance, by querying only $2^{\frac{t}{2}}$ many inputs, the probability of error would be doubly exponential, $(\frac{1}{e})^{(\frac{1}{2})^{\frac{t}{2}}}$. ▷

3.6 Faux-determinism for search problems

As previously discussed, for input length $n \in \mathbb{N}$, the $\mathsf{FIND1}$ problem has a simple randomized algorithm $\mathcal{R} = \{T_1,\ldots,T_n\}$ where the tree T_i on input x queries x_i and if $x_i = 1$, it outputs i. Thus, each query in the tree construction of definition (3.4) can be viewed as

- querying a random tree $T_i \sim \mathcal{R}$;
- verifying whether $(x, T_i[x]) \in \mathsf{FIND1}$;
- and continuing to branch according to the above verification.

With a similar idea, we can generalize our results to obtain faux-deterministic algorithms for the class of all query search problems having efficient randomized and verification algorithms.

Theorem 3.20. *Let $\mathcal{S}$ be a search problem over alphabets Σ and Γ that has verifier $V = (V_n)$ of complexity $v(n)$ and a randomized algorithm $\mathcal{R} = (\mathcal{R}_n)$ with complexity $r(n)$. Then $\mathcal{S}$ has a $(2^{\frac{\sqrt{t}}{4}}, 2^{-\frac{\sqrt{t}}{4}})$-faux-deterministic algorithm of complexity $(r(n) + v(n))t$.*

Corollary 3.21. *If the search problem $\mathcal{S}$ has* $\mathrm{polylog}(n)$ *depth verifier and randomized algorithm, then $\mathcal{S}$ has a* $(\mathrm{quasipoly}(n), \mathrm{negl}(n))$*- faux deterministic algorithm of complexity* $\mathrm{polylog}(n)$.

Proof. We will directly use the reduction from [Gol+21] to reduce this problem to the $\mathsf{FIND1}$ problem using the following lemma.

Lemma 3.22. *Given randomized algorithm $\mathcal{R} = (\mathcal{R}_n)$ and verifier $V = (V_n)$ for $\mathcal{S}$, there exists a large enough constant c depending only on $|\Sigma|$ and a randomized algorithm $\mathcal{R}' = (\mathcal{R}'_n)$ such that,*

- $|\mathcal{R}'_n| = cn$,
- $\mathrm{C}^{\mathsf{dt}}(\mathcal{R}'_n) = \mathrm{C}^{\mathsf{dt}}(\mathcal{R}_n) = r(n)$,
- $\Pr_{\boldsymbol{R} \sim \mathcal{R}'_n}[T[x] \notin \mathcal{S}(x)] \leq \frac{1}{2}$.

Proof. This lemma can be obtained using union bound and Chernoff's bound. For the sake of conciseness, we postpone the proof to appendix (A). □

Let $R_1, \ldots, R_m$ be all of the decision trees in $\mathcal{R}'_n$ where $m = cn$. Given an input string x, the main idea is to implicitly create a new binary string y that assigns 0 or 1 to each decision tree R_i and run the faux-deterministic algorithm for $\mathsf{FIND1}$ on y. Formally, we define y as follows:

$$(y(x))_i := \begin{cases} 1 & V_n[x, R_i[x]] \in \mathcal{S}(x), \\ 0 & \text{otherwise.} \end{cases}$$

In other words, $y(x)_i = 1$ if and only if R_i outputs a correct answer on x. Note that by lemma (3.22), $\text{len}(y) = cn = O(n)$ and $|y|_1 \geq \frac{cn}{2}$.

Let the distribution $\mathcal{T} = (\mathcal{T}_n)$ be the $(2^{\sqrt{\frac{t}{4}}}, 2^{-\sqrt{\frac{t}{4}}})$-faux-deterministic algorithm in theorem (3.10), and let $\boldsymbol{T} \sim \mathcal{T}_m$ be a random decision tree from our construction. Consequently, our algorithm simulates $\boldsymbol{T}$ on the string y, and on each query y_i of $\boldsymbol{T}$, it runs $R_i[x]$ and outputs $V_n[x, R_i[x]] \in \{0, 1\}$. In the end, we output $R_i[x]$ where i is the index outputted by $\boldsymbol{T}$. Consequently, we define $\mathcal{F} = (\mathcal{F}_n)$ be the final distribution over these simulations. By definition, $\boldsymbol{T}$ makes at most t many queries, and in each query, we run a decision tree R_i and the verifier V of depths at most $r(n)$ and $v(n)$. Therefore, the total complexity of $\mathcal{F}_n$ is at most $(v(n) + r(n))t$.

It only remains to prove that $\mathcal{F}$ is a $(2^{\sqrt{\frac{t}{4}}}, 2^{-\sqrt{\frac{t}{4}}})$-faux deterministic algorithm for $\mathcal{S}$. Suppose for the sake of contradiction that there is an integer n and adversary $\mathcal{A}_n$ that finding an invalid input x for $\delta > 2^{-\sqrt{\frac{t}{4}}}$ fraction of trees in $\mathcal{F}_n$ after at most $q < 2^{-\sqrt{\frac{t}{4}}}$ many queries. Consequently, we can define a new adversary $\mathcal{A}'_m$ for $\mathcal{T}_m$ as follows. Given query access to a random tree $T \sim \mathcal{T}_m$, we simulate $\mathcal{A}_n$. In other words, whenever $\mathcal{A}_n$ queries x, we construct $y(x)$ by using $R_1, \ldots, R_m$ and V_n, query $y(x)$ to obtain $i = \boldsymbol{T}[y(x)]$, and output $R_i[x]$ to $\mathcal{A}_n$. Now, from the perspective of $\mathcal{A}_n$, a tree $\boldsymbol{F} \in \mathcal{F}$ corresponding to $\boldsymbol{T}$ is being queried. Therefore, for δ fraction of trees $\boldsymbol{T} \sim \mathcal{T}_n$ our simulation returns an input $\bar{x}$ for which $\boldsymbol{F}[\bar{x}] \notin \mathcal{S}(x)$. Hence, for the input $y(\bar{x})$ and the output $i = \boldsymbol{T}[y(\bar{x})]$, we must have $(y(\bar{x}))_i = V_n(\bar{x}, R_i[\bar{x}]) = 0$, which means, $y(\bar{x})$ is an adverse input for $\boldsymbol{T}$. But this contradicts the assumption about $\mathcal{T}$, and therefore, $\mathcal{F}$ is a $(2^{\sqrt{\frac{t}{4}}}, 2^{-\sqrt{\frac{t}{4}}})$-faux deterministic algorithm for $\mathcal{S}$.

□

3.7 Verifiers and sensitivity problem

Trivially, any faux-deterministic algorithm is a randomized algorithm itself, and therefore, having a randomized algorithm is a necessary condition in the theorem (3.20). On the contrary, the relation between the existence of faux-deterministic algorithms and verifiers is not as clear. Still, we can make interesting observations about the necessity of verifiers. We consider a modified version of the FIND1 problem with the same randomized and deterministic algorithm as FIND1 but does not have an efficient verifier.

3.7.1 FINDMEDIAN problem

In the FINDMEDIAN problem we are given any binary string $x \in \{0,1\}^n$ and,

- if $|x| \geq \frac{n}{2}$, we need to find an index i with $x_i = 1$,
- otherwise, $|x| < \frac{n}{2}$ and we can output any arbitrary index.

Definition 3.23 (FINDMEDIAN)**.** For $n \in \mathbb{N}$, let

$$\mathsf{FINDMEDIAN}_n = \{(x,i) | x_i = 1 \text{ or } |x| < \frac{n}{2}\}.$$

We define the search problem FINDMEDIAN over binary alphabets to be the sequence $\mathsf{FINDMEDIAN} := (\mathsf{FINDMEDIAN}_n)$. ▷

For inputs with $|x|_1 < \frac{n}{2}$, an algorithm solving FINDMEDIAN can output any index $i \in [n]$ and be correct. Therefore, any solution for FIND1 also solves the FINDMEDIAN problem.

On the other hand, an index i with $x_i = 0$ can still be a valid certificate if x does not have a large hamming weight. In other words, FINDMEDIAN has high certificate complexity. Therefore, unlike FIND1, FINDMEDIAN does not have an efficient verifier.

In the following, we prove that for any function $f : \{0,1\}^n \to [n]$ that can succesfully compute $\mathsf{FINDMEDIAN}_n$, we can efficiently find an input with high sensitivity for f.

Proposition 3.24. *Let $f : \{0,1\}^n \to [n]$ be a given function attempting to solve the total problem* FINDMEDIAN*. In at most $\frac{n^2}{2}$ many queries to f, we can*

- *either find an input x and indices $i_1, \ldots, i_{\frac{n}{2}} \in [n]$ with $f(x) \neq f(x)^{i_j}$ for $j \leq \frac{n}{2}$,*
- *or find an input x for which $f(x) \notin \mathsf{FINDMEDIAN}(x)$*

Therefore, we get the following corollary that gives us an efficient adversary for the FINDMEDIAN problem.

Corollary 3.25. *Suppose we are given any tree T of depth $t < \frac{n}{2}$ that claims to solve* FINDMEDIAN *on inputs of length n. Then, there exists a classical adversary that finds an adverse input for T in at most $\frac{n^2}{2}$ queries to T.*

Proof of corollary (3.25)*.* Given tree T, we consider the function defined by $f(x) := T[x]$ and run the adversary in proposition (3.24) on f. Since T has depth at most $\frac{n}{2}$, for any input x, at most $\frac{n}{2}$ many indices of x are queried in $T[x]$. Therefore, the adversary finds an input x for which $T[x] \notin \mathsf{FINDMEDIAN}(x)$ and this completes the proof. □

Proof of proposition (3.24). We start with the string $x(0) = 0^n$ where $x(0)_{f[x(0)]} = 0$. To obtain the next string $x(k+1)$ from the current string $x(k)$ we use the following algorithm for the adversary.

Obtaining $x(k+1)$ from $x(k)$

For each 0-index $j \in [n]$ with $x(k)_j = 0$,

1. we first flip the j-th bit to obtain the string $z = x(k)^j$, where
$$z_t = \begin{cases} 1 & t = j \\ x(k)_t & \text{otherwise;} \end{cases}$$
2. next, we make a query to f on z to obtain the index
$$i := f(z) = f(x(k)^j);$$
3. if $z_i = 0$, f still outputs a 0-index and we define
$$x(k+1) = (x(k))^j$$
so that we still have
$$(x(k+1))_{f(x(k+1))}.$$
4. If $z_i = 1$, f changes value from $x(k)$ to $x(k)^j$ and we continue with the next possible j with $x(k)_j = 0$.
5. If the string $x(k)$ is sensitive to all of its 0-indices, we define $x = x(k)$ and stop the algorithm.

In each step, the hamming weight of the string increases by 1 and therefore at the k-th step, we have $|x(k)|_1 = k$. Thus, if the algorithm takes more than $\frac{n}{2}$ steps, we reach a string x where $|x|_1 = \frac{n}{2}$, and $x_{f(x)} = 0$. In this case, we have found an input for which f does not compute $\mathsf{FINDMEDIAN}_n$.

Otherwise, the algorithm gives a string x with $|x|_1 < \frac{n}{2}$ such that,

i. $x_{f(x)} = 0$,

ii. For every 0-index $i \in [n]$ with $x_i = 0$ and we have $f(x^i) \neq f(x)$.

Since $|x|_1 < \frac{n}{2}$, we have found at least $\frac{n}{2}$ many sensitive indices for f on x. □

3.7.2 Pseudo-deterministic lower bounds for FIND1

In the proof of proposition (3.24), we introduced a new problem of finding a sensitive input for a function computing a hard problem, and in corollary (3.25) we reduced the problem of finding an adverse input to this problem. Subsequently, we can ask the same question for the FIND1 problem and any function $f : \mathcal{X}_n \longrightarrow [n]$ computing $f(x) \in \mathsf{FIND1}_n(x)$ where $\mathcal{X}_n = \{x \in \{0,1\}^n || x|_1 \leq \frac{n}{2}\}$ is the set of valid inputs for our problem.

Problem 3.26. Given $f : \mathcal{X}_n \longrightarrow [n]$ such that for all x, $f(x) \in \mathsf{FIND1}(x)$, find an input x and disjoint blocks of indices $B_1, \ldots, B_k$ such that $f(x) \neq f(x^{B_j})$ for $j \in [n]$.

Via polynomial-degree lower bounds and the Huang's resolution of sensitivity conjecture [Hua19], Goldwasser et al. [Gol+21] gave an existential proof that if f computes FIND1, then there exists an input that is sensitive on $\Omega(n^{\frac{1}{2}})$ disjoint blocks. On the other hand, even though such input exists, our faux-deterministic construction proves that solving problem (3.26) is not possible even for $k = \text{polylog}(n)$.

Proposition 3.27. *For f discussed in the above, no adversary can find such an input that has $\omega(\text{polylog}(n))$ sensitivity within $\text{poly}(n)$ many queries.*

Proof. We prove that if such an adversary exists, we can design another adversary that finds an adverse input for any given decision tree of depth $\text{polylog}(n)$. So, suppose we are given such a decision tree T. We define f to compute FIND1 with minimal distance to T. I.e.,

$$f(x) := \begin{cases} T[x] & T[x] \in \mathsf{FIND1}(x) \\ i & x_i = 1 \wedge T[x] \notin \mathsf{FIND1}(x) \end{cases}$$

Next, we simulate querying to f by querying T by

- outputting $T[x]$ if $T[x] \in \mathsf{FIND1}(x)$
- and returing an arbitrary index i with $x_i = 1$ otherwise.

Since f computes FIND1, it must have an input x and sensitive blocks $B_1, \ldots, B_k$ where $k = \Omega(\sqrt{n})$. Thus, the original adversary makes $\text{poly}(n)$ queries to find an input x and

disjoint blocks of indices $B_1, \ldots, B_k$ with $k = \omega(\text{polylog}(n))$ such that $f(x) \neq f(x^{B_j})$. If $T[x] \neq f(x)$, we have found an adverse input for T. So suppose $T[x] = f(x)$. Since T has depth polylog(n), it only queries polylog(n) many indices from x. Therefore, T queries from at most polylog(n) of the blocks. Then, for at least one of the blocks B_j, we must have,

$$f(x^{B_j}) \neq f(x) = T[x] = T[x^{B_j}]$$

As a result, we find an adverse input for T as $f(x^{B_j}) \neq T[x^{B_j}]$. □

Now, consider a pseudo-deterministic algorithm $\mathcal{T}$ computing FIND1, and let f be the function computing $f(x) := T[x]$. By above, even though f has high sensitivity, this sensitive input is not constructive in polynomial time. We view this observation as an evidence pointing towards the fact that any sensitivity lower bound for a pseudo-deterministic complexity of FIND1 should be non-constructive, whilst the usual non-constructive methods of lower bounding sensitivty such as polynomial method, induce at least a quadratic loss.

Chapter 4

Duality in TFNP

Previously, we studied the distribution of efficient algorithms $\mathcal{A}$ for a search problem $\mathcal{S}$ that can fool any efficient adversary attempting to find an adverse input for a randomly sampled $\boldsymbol{A} \sim \mathcal{A}$. Notice that in this setting, the problem of finding an adverse input for a faulty algorithm is itself a search problem. Therefore, shifting our viewpoint to the adversaries' side, we can ask whether there exists an adversary that can find an adverse input for any efficient algorithm purporting to solve $\mathcal{S}$. We call this new search problem the *dual* of the original problem and denote it by $\mathcal{S}^*$.

Roughly speaking, an algorithm A^* for a hard problem $\mathcal{S}^*$ takes another (efficient) algorithm A for $\mathcal{S}$ and has to output an input x such that $A[x] \notin \mathcal{S}(x)$. In this chapter, we will discuss interesting meta-complexity relations between the primal and dual problems, and we will provide several pieces of evidence for why the dual problem is analogous to the primal one.

We begin by recalling the observations mentioned in the introduction. Next, we briefly discuss why these observations should hold at an intuitive level and why proving these observations could be a challenging task in different computational models. Afterward, we will prove partial results for the Turing machine model and query model by formalizing some of the below observations.

Informal observation

Let $\mathcal{S}$ and $\mathcal{R}$ be two hard search problems. In different computational models, some or all of the following hold:

i. if $\mathcal{S}$ is verifiable, then $\mathcal{S}^*$ is also verifiable;

ii. if $\mathcal{S}$ is hard, then $\mathcal{S}^*$ is total;

iii. if $\mathcal{S}$ reduces to $\mathcal{R}$, then $\mathcal{R}^*$ reduces to $\mathcal{S}^*$;

iv. if $\mathcal{S}$ has a randomized algorithm then $\mathcal{S}^*$ is hard on average;

v. if $\mathcal{S}$ is hard on average, then $\mathcal{S}^*$ has a randomized algorithm.

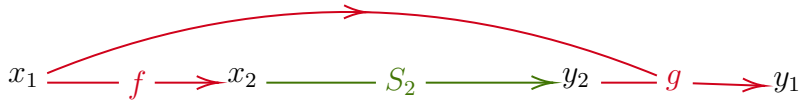
4.1 Intuitive overview

i. Suppose that for a hard search problem $\mathcal{S}$ we have a verifier V where $y \in \mathcal{S}(x)$ if and only if $V(x, y) = 1$. Also, let the input A of $\mathcal{S}^*$ be an efficient algorithm and let x be an arbitrary input for $\mathcal{S}$. Having the verifier V for $\mathcal{S}$, we can verify whether $(A, x) \in \mathcal{S}^*$ by checking if $V[x, A[x]] = 0$. Therefore, on an informal level the dual should also be verifiable;

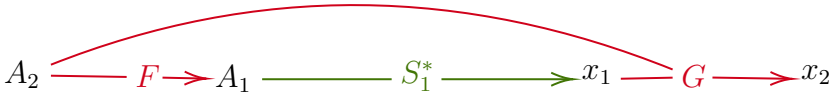
ii. Additionally, if the primal problem $\mathcal{S}$ is hard, then any efficient algorithm A should fail to solve $\mathcal{S}$. This means that for any input A of $\mathcal{S}^*$ there exists a certificate x such that $(A, x) \in \mathcal{S}^*$, and $\mathcal{S}^*$ is in this sense *total*.

As a result, for (hard) TFNP problems, we expect the dual problem to inherit characteristics of TFNP as well. Furthermore, as in observations (iii)-(v), we can relate the complexity of the primal and dual problems as follows.

iii. Suppose for two hard search problems $\mathcal{S}_1$ and $\mathcal{S}_2$ we have $\mathcal{S}_1 \preceq \mathcal{S}_2$. Then, intuitively, it should be more difficult to construct efficient algorithms that can fool algorithms of $\mathcal{S}_2^*$ compared to algorithms for $\mathcal{S}_1^*$. This is because, by the given reduction, from a faulty algorithm A_2 that is hard for $\mathcal{S}_2^*$, we can construct a faulty algorithm A_1 that is hard for $\mathcal{S}_1^*$. More specifically, suppose a general reduction (f, g) as in figure (4.1)



(a) Initial reduction from $\mathcal{S}_1$ to $\mathcal{S}_2$



(b) Constructed reduction from $\mathcal{S}_2$ to $\mathcal{S}_1$

Figure 4.1: Given a reduction (f, g) from $\mathcal{S}_1$ to $\mathcal{S}_2$, we can obtain another reduction (F, G) from $\mathcal{S}_2^*$ to $\mathcal{S}_1^*$.

is given, where we have

$$\left.\begin{array}{ll} \forall x_1 & x_2 = f(x_1) \\ \forall y_2 & \in \mathcal{S}_2(x_2) \end{array}\right\} \Longrightarrow g(x_1, y_2) \in \mathcal{S}_1(x_1). \tag{4.1}$$

Then, we can construct a new reduction (F, G) such that

$$\begin{aligned} A_1 &:= F(A_2), \\ x_2 = G(A_2, x_1) &:= f(x_1), \end{aligned}$$

where

$$\forall x_1, \quad A_1(x_1) = (F(A_2))(x_1) := g(x_1, A_2(f(x_1))). \tag{4.2}$$

Now, any $x_1 \in \mathcal{S}_1^*(A_1)$ is an adverse input for A_1. Therefore,

$$A_1(x_1) \notin \mathcal{S}_1(x_1). \tag{4.3}$$

Additionally, if for $y_2 := A_2(f(x_1))$ we have $y_2 \in \mathcal{S}(x_2)$, we will have

$$\begin{aligned} A_1(x_1) &= F(A_2)(x_1) && \text{(by definition (4.2))} \\ &= g(x_1, A_2(f(x_1))) && \text{(by } y_2 := A_2(f(x_1))\text{)} \\ &= g(x_1, y_2) \in \mathcal{S}_1(x_1) && \text{(by (4.1))} \end{aligned}$$

But this contraditcs eq. (4.3). Consequently,

$$\left.\begin{array}{ll} \forall A_1 & A_1 = F(A_2) \\ \forall x_1 & \in \mathcal{S}_1^*(A_1) \end{array}\right\} \Longrightarrow G(A_2, x_1) \in \mathcal{S}_2^*(A_2).$$

iv. As seen in the faux-determinism chapter, having a randomized algorithm for $\mathcal{S}$ is closely related to having a faux-deterministic algorithm which is equivalent to saying $\mathcal{S}^*$ is hard on average.

v. Additionally, if $\mathcal{S}$ is hard on average, then there exists a distribution of inputs for which every algorithm A for $\mathcal{S}$ fails with high probability. Therefore, a randomized algorithm for $\mathcal{S}^*$ can randomly sample from such a distribution for any given algorithm A to find an adverse input with high probability.

4.2 Formalization challenges

Up to this point, we have not yet talked about any exact formulation of the mentioned properties. Even though these rough observations seem natural, making them accurate in any computational model can quickly become a complicated and confusing task. This is mainly due to the fact that any choice in the definition of the dual problem faces compatibility issues with the primal problem.

Specifically, the notions of verifiability, hardness, totality, randomized algorithms, and reductions are not necessarily robust enough to be extendable to the dual problem. Therefore, we may need to adjust or restrict the usual definitions to account for such issues. For example, consider the notion of hardness for a search problem $\mathcal{S}$. If $\mathcal{S}$ is partially hard only on a few input sizes, then $\mathcal{S}^*$ will not be total since we may have efficient algorithms that solve $\mathcal{S}$ for infinitely many input lengths. On the other hand, problems of interest are often *continuously* hard or easy on different input lengths, and such non-continuous problems are usually artificial.

To make this worse, the usual definitions of the above notions can be very fragile when subjected to small changes in computational models, especially when considering a meta-complexity problem. As a result, we have to be very careful about the implications of any change or restrictions to the regular definitions. This ultimately makes the task of finding the proper definitions much harder. For instance, adapting to some non-uniform models, such as circuits, might completely collapse the distinctions between randomized and deterministic algorithms. Additionally, in uniform computational models, the description size of an input A for $\mathcal{S}^*$ does not encapsulate its complexity.

As a result, it is currently unclear to us how to prove all of the above in the usual computational models, and we do not know if a computational model satisfies any formalization of these observations simultaneously. However, we can still obtain partial results in

the usual non-uniform query and uniform Turing machine models that we present in the following sections.

4.3 Turing-machine model

In the white-box model, we consider the dual problems for a hard search problem $\mathcal{S} \in \mathsf{TFNP}$, where TFNP is the class of search problems that are both total and verifiable. Roughly, in the dual problem $\mathcal{S}^*$, we are given a polynomial time Turing machine M purporting to solve $\mathcal{S}$ along with an integer m. Then, our goal is to find an input x with a length of at least m such that $M[x] \notin \mathcal{S}[x]$. For proper interpretations of the previously mentioned observations in the Turing machine model, we will be able to prove the following.

Informal Theorem. *Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be two "hard" search problems in TFNP. Then, we have*

i. $\mathcal{S}_1^*$ is also in TFNP, and

ii. If $\mathcal{S}_1$ reduces to $\mathcal{S}_2$, then $\mathcal{S}_2^*$ also reduces to $\mathcal{S}_1^*$.

In the following, we first provide and adapt preliminary definitions such as a *hard search problem* for the Turing machine and formally define the dual problem. Then, we will prove each item in the above separately.

4.3.1 Preliminaries for dual problem in Turing machine model

Search problems in Turing machine model and binary encodings When working with Turing machines, it is more convenient to work with *binary inputs and outputs*. Therefore, as previously discussed in chapter 2, we can view search problems $\mathcal{S}$ as binary relations

$$\mathcal{S} \subset \{0,1\}^* \times \{0,1\}^*,$$

where we interchangeably denote input $x \in \mathsf{dom}(\mathcal{S})$ and output $y \in \mathsf{range}(\mathcal{S})$ with their *binary encodings* $\langle x \rangle$ and $\langle y \rangle$.[1]

[1]For more information on the chosen binary encoding, please refer to chapter 2.

For complexity theory purposes, paddable encodings simplify a lot of unnecessary complex details of dealing with Turing machines and their uniform nature. Therefore, we choose a *paddable* encoding for which strings $\langle x \rangle$ and $\langle x \rangle 0^r$ denote the same input for our search problem. One can consider such padding as a comment that does not affect the actual meaning of the string. Therefore, for our search problem and integers r and s, we will have

$$(\langle x \rangle, \langle y \rangle) \in \mathcal{S} \Leftrightarrow (\langle x \rangle 0^r, \langle y \rangle 0^s) \in \mathcal{S}.$$

Further, without loss of generality, we assume that *correctly formatted Turing machines* respect such an encoding. In other words, for a padded string $\langle x \rangle 0^r$, a correctly formatted Turing machine ignores the padding and just runs on input $\langle x \rangle$.[2]

Complexity classes We use the standard complexity classes FP, FBPP, and FNP as the class of search problems having polynomial time deterministic, randomized, and verification algorithms. Consequently, TFNP will be the class of total search problems in FNP.

We note that in complexity theory, hard problems are generally considered to be problems that do not have polynomial time deterministic algorithms ($\mathcal{S} \notin$ FP). For the purpose of dual problems, however, a mere super-polynomial lower bound makes the job of the dual algorithm extremely hard. Therefore, for the purpose of our results, we consider quasipoly(n) lower bounds for the primal problems.

Therefore, we use quasiFP to denote the class of search problems having Turing machines computing them in time $2^{\text{polylog}(n)}$.

Continuity If a Turing machine M cannot solve $\mathcal{S}$, it will fail to solve $\mathcal{S}$ on infinitely many output lengths. Therefore, for any integer m, there exists an input with $|x| \geq m$ such that $M[x] \notin \mathcal{S}(x)$. However, the gap between the input lengths on which M fails might be too large. In such scenarios, an algorithm for $\mathcal{S}^*$ might not even be able to write such an input in polynomial time.

This issue mainly arises when our initial search problem is not *continuous* in the sense that its complexity varies with input lengths. Such problems are mostly artificial, however, they still satisfy the definition of search problem. For instance, consider a variant of finding a satisfying assignment problem where on even input lengths, any possible output is correct. To discard such manufactured problems, we introduce a notion of continuity for search problems.

[2]Note that this assumption does not change the complexity of a search problem as the algorithm still needs to solve the problem for non-padded strings over all input lengths as well.

Definition 4.1 (Continuity in TM model)**.** Let $\mathcal{S}$ be a search problem and $\mathcal{C}$ be a complexity class of search problems. Then, a search problem $\mathcal{S} \notin \mathcal{C}$ is *$\mathcal{C}$-continuous* if every algorithm in $\mathcal{C}$ fails to solve $\mathcal{S}$ on infinitely many input lengths. More specifically, $\mathcal{S} \notin \mathcal{C}$ is $\mathcal{C}$-continuous if for all infinite subsets $I \subseteq \mathbb{N}$,

$$\mathcal{S} \cap (\{x \in \{0,1\}^* |\ |x| \in I\} \times \{0,1\}^*) \notin \mathcal{C}.$$

▷

Reductions For the sake of reductions in the Turing machine model, we consider a generalized notion of Karp reductions as follows.

Definition 4.2. Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be two search problems. We say $\mathcal{S}_1$ is *poly-time Karp reducible* to $\mathcal{S}_2$ and denote $\mathcal{S}_1 \preceq_{poly} \mathcal{S}_2$, whenever there exists Boolean functions $f, g : \{0,1\}^* \to \{0,1\}^*$ such that,

i. f and g are computable in polynomial time, i.e.,

$$f, g \in \mathsf{FP};$$

ii. For all $x_1 \in \mathsf{dom}(\mathcal{S}_1)$, the string $x_2 = f(x_1)$ is an input for $\mathcal{S}_2$,

$$x_2 \in \mathsf{dom}(\mathcal{S}_2);$$

iii. For any corresponding output of x_2 such as $y_2 \in \mathcal{S}_2(x_2)$, g computes a correct certificate for x_1 based on y_2,

$$g(x_1, y_2) \in \mathcal{S}_1(x_1).$$ ▷

4.3.2 Dual problem in Turing machine model

Having rigorously formalized the definitions of continuity, hard problems, and reductions, we begin defining the dual problem by first considering correct *input and output formatting*.

Definition 4.3 (Input formatting)**.** Suppose $\mathcal{S} \in \mathsf{TFNP}$ is a search problem in the Turing machine model. The dual problem $\mathcal{S}^*$ takes inputs of the format,

$$X = (\langle M \rangle, 1^k, 1^m, 1^{km^k}),$$

where

i. $\langle M \rangle$ is the canonical binary encoding of a correctly formatted Turing machine M;

ii. k is a natural number denoting the polynomial $p_k(n) = kn^k$;

iii. m is a natural number such that $m \geq 2^{k|M|}$.[3]

We can interpret the above input as M claiming to solve $\mathcal{S}$ in time p_k and the dual algorithm has to find a counterexample input of length at least m. Therefore, given an input

$$X = (\langle M \rangle, 1^k, 1^m, 1^{km^k}),$$

the goal of $\mathcal{S}^*$ is to either,

- prove that M does not run in time p_k;
- or find an input of length m on which M fails to output a correct answer.

▷

Remark. To have any chance of finding such an input, the dual algorithm would need to simulate M for at least $p_k(m)$ many steps. Therefore, we give the dual algorithm a string of length km^k as part of the input.

Also, note that if m is not sufficiently large compared to $|M|$, a Turing machine can be constructed to solve $\mathcal{S}$ using brute force on inputs of length m. Adding the last condition $m \geq 2^{k|M|}$ guarantees such inputs are not valid and eases the task of the dual algorithm to find a counterexample input of length at least m. ▷

With respect to the above input, we can now define the correct output formatting as well.

Definition 4.4 (Output formatting)**.** For an input

$$X = (\langle M \rangle, 1^k, 1^m, 1^{km^k})$$

formatted as above, a correctly formatted output Y is a tuple

$$Y = (y, 1^{k|y|^k}),$$

such that $|y| \geq m$. ▷

[3]For simplicity, we denote the binary length $|\langle M \rangle|$ as simply $|M|$.

Remark. The reason why we additionally add the string $1^{k|y|^k}$ is that any possible verifier for $\mathcal{S}^*$ should be able to run M on y and check if it takes more than $p_k(y)$ many steps. ▷

Finally, we define the dual problem as follows.

Definition 4.5 (Dual problem in Turing machine model)**.** Let $\mathcal{S}$ be a search problem. If a binary string X is not correctly formatted as in definition (4.3), we define $\mathcal{S}^*(X)$ to include everything:

$$\mathcal{S}^*(X) = \{0,1\}^*.$$

On the other hand, for any correctly formatted input string $X = (\langle M\rangle, 1^k, 1^m, 1^{km^k})$ and correctly formatted output string $Y = (y, 1^{k|y|^k})$, we have $Y \in \mathcal{S}(X)$ if and only if either

- y is a counterexample to M running in time p_k: $\mathsf{TIME}(M, y) > k|y|^k$,
- or y is a counterexample to M solving $\mathcal{S}$: $M[y] \notin \mathcal{S}(y)$.

▷

4.3.3 Totality and verifiability of $\mathcal{S}^*$

We begin by proving that the dual of a hard TFNP problem remains in TFNP as well.

Proposition 4.6. *Let $\mathcal{S} \in \mathsf{TFNP} \setminus \mathsf{quasiFP}$ be a $\mathsf{quasiFP}$-continuous search problem. Then, $\mathcal{S}^*$ is also a problem in TFNP.*

Proof. We first prove that $\mathcal{S}^*$ also has a verifier and then prove it is total.

$\mathcal{S}^*$ is verifiable: Let V be a polynomial time verifier for $\mathcal{S}$ running in time $q(n)$ such that

$$V(x, y) = 1 \Leftrightarrow y \in \mathcal{S}(x).$$

We will construct a verifier V^* for $\mathcal{S}^*$ such that we get

$$V^*(X, Y) = 1 \Leftrightarrow Y \in \mathcal{S}(X).$$

To construct V^*, suppose the pair of input-output (X, Y) for $\mathcal{S}^*$ is given. Next, we will give a verifier algorithm as follows.

Verifier for $\mathcal{S}^*$

1. If X is not formatted correctly, every output Y is a correct certificate, and we accept. Thus, we have $V^*(X, Y) := 1$.

2. Suppose X is formatted correctly, and for some M, k, and m we have

$$X = (\langle M \rangle, k, 1^m, 1^{km^k}),$$

   where $m \geq 2^{k|M|}$. Now, if Y is not formatted correctly, it cannot be a correct certificate, and we reject it: $V^*(X, Y) := 0$.

3. Otherwise, Y is also formatted correctly and for some y we have

$$Y = (y, 1^{k|y|^k}),$$

   where $|y| \geq m$.

   Now, if the computation of $M[y]$ does not halt after $k|y|^k$ many steps or if $V(y, M[y]) = 0$, we accept, and otherwise, we reject.

$$V^*(X, Y) = (M[y] \text{ does not halt in } k|y|^k) \text{ or } (V(y, M[y]) = 0)$$

Clearly, this algorithm is a verifier for $\mathcal{S}^*$. For the run-time, the first two steps take the most linear time. Additionally, the third step takes time at $k|y|^k = O(|Y|)$ for running $M[y]$ and $q(|(y, M[y])|) = O(q(|Y|))$ for computing $V(y, M[y])$. Therefore,

$$\mathsf{TIME}(V^*, (X, Y)) = O(|X| + |Y|) + O(|Y|) + q(|Y|) = O(|(X, Y)|).$$

Thus, we have $\mathcal{S}^* \in \mathsf{FNP}$.

$\mathcal{S}^*$ is total: It only remains to prove that $\mathcal{S}$ is total. By definition, for an incorrectly formatted input X, any string is a certificate in $\mathcal{S}^*(X)$. Thus, without loss of generality, we only consider correctly formatted inputs. We will utilize and prove the following lemma.

Lemma 4.7. *There exists a finite set of correctly formatted inputs $T = \{X_1, \ldots, X_t\}$ such that for any other correctly formatted input $X \notin T$ we have a linear size certificate $Y = (y, km^k)$, where $|Y| = O(|X|)$ and*

$$(X, Y) \in \mathcal{S}^*.$$

Additionally, for the finitely many remaining inputs $X_1, \ldots, X_t$, the algorithm can find certificates $Y_1, \ldots, Y_t$ by brute force. Therefore, we prove that for every input X, there exists a Y of size at most $O(|X|)$ such that $(X, Y) \in \mathcal{S}^*$. This completes the proof of totality for $\mathcal{S}^*$. □

Proof of lemma (4.7). Suppose this is not the case. Then, there are infinitely many correctly formatted strings

$$X_i = (\langle M_i \rangle, k_i, 1^{m_i}, 1^{k_i m_i^{k_i}}),$$

for $i \in \mathbb{N}$ such that,

- $m_i \geq 2^{k_i |M_i|}$;
- X does not have any certificate of type $Y = (y, k_i m_i^{k_i})$.

By the definition (4.5), this means that for any input of length $|y| = m_i$,

i. $M_i[y] \in \mathcal{S}(y)$);

ii. $\mathsf{TIME}(M_i, y) \leq k_i m_i^{k_i}$.

Now, consider the Turing machine $\mathcal{M}$ defined below where on any input y,

1. enumerates over every pair (M, k) with $k|M| \leq \log |y|$;
2. simulates $M[y]$ for $k|y|^k$ many steps;
3. Checks whether $M[y]$ outputs a correct answer in $\mathcal{S}(y)$ by calling the verifier V.

By (i) and (ii), this algorithm solves $\mathcal{S}$ on infinitely many input lengths m_i for $i \in \mathbb{N}$. Furthermore, on each input y, there are at most $|y|^2$ many possible pairs $|M|, k \leq \log |y|$, and on each pair, the algorithm takes time $\text{poly}(k|y|^k)$. Thus, the algorithm runs in total quasi-polynomial time

$$\text{poly}(k|y|^k).|y|^2 = \text{poly}((\log y)|y|^{\log y}).|y|^2 = \text{quasipoly}(|y|).$$

But this contradicts the $\mathsf{quasiFP}$-continuous assumption for $\mathcal{S}$. This completes the proof of the lemma (4.7). □

4.3.4 Comparing complexities of $\mathcal{S}_1$ and $\mathcal{S}_2$

Proposition 4.8. *Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be two search problems in* TFNP\quasiFP *that are* quasiFP*-continuous. Then, if $\mathcal{S}_1 \preceq_{\text{poly}} \mathcal{S}_2$ we have $\mathcal{S}_2^* \preceq_{\text{poly}} \mathcal{S}_1^*$.*

Proof. Suppose there exists polynomial time functions $f : \{0,1\}^n \to \{0,1\}^{l(n)}$ and g such that,

i. f maps an input x_1 of $\mathcal{S}_1$ to an input $x_2 = f(x_1)$ of $\mathcal{S}_2$;

ii. g finds a certificate for x_1 based on a certificate $y_2 \in \mathcal{S}_2(x_2)$:

$$g(x_1, y_2) \in \mathcal{S}_1(x); \tag{4.4}$$

iii. $f, g \in \mathsf{FP}$.

Without loss of generality, we can pad $f(x)$ with enough zeros to always have length $l(n) = dn^d$ for a constant $d \geq 1$. Additionally, we assume that $f(x)$ and $g(x, y)$ run in time $q(n) = rn^r$, where n is the input size and $r \geq d \geq 3$.

We construct functions F and G such that,

I. F maps an input X_2 of $\mathcal{S}_2^*$ to an input $X_1 = F(X_2)$ of $\mathcal{S}_1^*$;

II. G finds a certificate for X_2 based on a certificate $Y_1 \in \mathcal{S}_1^*(X_1)$:

$$G(X_2, Y_1) \in \mathcal{S}_2^*(X_2); \tag{4.5}$$

III. $F, G \in \mathsf{FP}$.

Constructing F: We start by defining F and stating some of its important properties below.

$F(X_2)$

For an incorrectly formatted input X_2, we define $F(X_2) = \perp$ to output an empty string. For a correctly formatted input X_2 of $\mathcal{S}_2^*$, where

$$X_2 = (\langle M_2 \rangle, k_2, 1^{m_2}, 1^{k_2 m_2^{k_2}}),$$

we define $F(X_2)$ to be such that,

$$X_1 := F(X_2) = (\langle M_1 \rangle, k_1, 1^{m_1}, 1^{k_1 m_1^{k_1}}).$$

Moreover, M_1, k_1, and m_1 are obtained as follows.

- The new Turing machine M_1 on input x_1 outputs $M_1[x_1] = g(x_1, M_1[f(x_1)])$;
- $k_1 = r^3 k_2$;
- $m_1 = \max\{2^{k_1|M_1|}, m_2\}$.

Clearly, the above definition satisfies (I).

Properties of F: The following lemma summarizes all of the other properties of F that is needed to make our reduction work. For conciseness, we postpone the proof of the lemma to the end of this section.

Lemma 4.9. *There exists a large enough integer $N \in \mathbb{N}$, such that for a correctly formatted input*

$$X_2 = (\langle M_2 \rangle, k_2, 1^{m_2}, 1^{k_2 m_2^{k_2}}),$$

with $m_2 \geq N$, and

$$X_1 := F(X_2) = (\langle M_1 \rangle, k_1, 1^{m_1}, 1^{k_1 m_1^{k_1}})$$

as above we have

a. $|M_1| \leq c\frac{\log m_2}{k_2}$ *for some fixed c independent of X_2;*

b. $k_1 m_1^{k_1} \leq \text{poly}(k_2 m_2^{k_2})$*;*

c. For $n \geq N$ we have $k_1 n^{k_1} \geq n^{k_2 r^2} + k_2 (dn^d)^{k_2}$;

d. For an input x_2 with $|x_2| \geq m_2 \geq N$, either

- *$M_2[x_2]$ does not halt after $k_2|x_2|^{k_2}$ many steps,*
- *or* $\mathsf{TIME}(M_1, x_1) \leq \mathsf{TIME}(M_2, f(x_1)) + |x_1|^{k_2 r^2}$.

Now, we can proceed to define G as follows and prove that for F and G (II) holds.

Constructing G: Suppose we are given an input X_2 of $\mathcal{S}_2^*$ and have computed the new input $X_1 = F(X_2)$. Additionally, let $Y_1 \in \mathcal{S}_1^*(X_1)$ be a certificate for X_1.

$G(X_2, Y_1)$

1. If X_2 is not correctly formatted, any arbitrary string Y_2 is a correct certificate for X_2. Therefore, we output an arbitrary string.
2. Otherwise, we have
$$X_2 = (\langle M_2 \rangle, k_2, 1^{m_2}, 1^{k_2 m_2^{k_2}}),$$
$$X_1 = (\langle M_1 \rangle, k_1, 1^{m_1}, 1^{k_1 m_1^{k_1}}).$$
3. If $m_2 < N$ in lemma (4.9), we can find a counterexample input Y_2 for X_2 with brute force. Thus, we can additionally assume $m_2 > N$.
4. Now, let $Y_1 = (x_1, k_1|x_1|^{k_1})$ be the certificate of X_1 in the above, where we have $|x_1| \geq m_1$ and either
   - $\mathsf{TIME}(M_1, x_1) > k_1|x_1|^{k_1}$,
   - or $M_1[x_1] \notin \mathcal{S}(x_1)$

   Now, consider the string $x_2 = f(x_1)$. We claim that
$$Y_2 := (x_2, 1^{k_2|x_2|^{k_2}})$$
   is in $\mathcal{S}_2^*(X_2)$. Consequently, (II) will hold for all of the above cases.

Proof of claim. If $M_2[x_2]$ does not halt after $k_2|x_2|^{k_2}$, we are done as x_2 is a counterexample to M_2 running in time $k_2 n^{k_2}$. Therefore, suppose $M_2[x_2]$ halts after $k_2|x_2|^{k_2}$ many steps.

- If $\mathsf{TIME}(M_1, x_1) > k_1|x_1|^{k_1}$, we have

$$\begin{aligned}
\mathsf{TIME}(M_2, x_2) &= \mathsf{TIME}(M_2, f(x_1)) && \text{(By } x_2 = f(x_1)\text{)} \\
&\geq \mathsf{TIME}(M_1, x_1) - |x_1|^{k_2 r^2} && \text{(By (d))} \\
&> k_1|x_1|^{k_1} - |x_1|^{k_2 r^2} && \text{(By } \mathsf{TIME}(M_1, x_1) > k_1|x_1|^{k_1}\text{)} \\
&\geq |x_1|^{k_2 r^2} + k_2(d|x_1|^d)^{k_2} - |x_1|^{k_2 r^2} && \text{(By (c))} \\
&= k_2(d|x_1|^d)^{k_2} \\
&= k_2|x_2|^{k_2}. && \text{(by } |x_2| = d|x_1|^d\text{)}
\end{aligned}$$

  This means that M does not halt in time $k_2|x_2|^{k_2}$, which contradicts our assumption.

- Therefore, we must have $M_1[x_1] \notin \mathcal{S}(x_1)$. But then $M_2[x_2]$ cannot be in $\mathcal{S}(x_2)$ as otherwise we should have $g(x_1, M_2[x_2]) \in \mathcal{S}_1(x_1)$ by the condition (ii). However,

$$g(x_1, M_2[x_2]) = M_1[x_1] \notin \mathcal{S}(x_1),$$

  which is a contradiction.

□

F and G are in FP: It only remains to prove (III). First, we have finitely many possible correctly formatted inputs X_2 with $m_2 < N$ as we always have $k_2|M_2| \leq \log m_2$. Therefore, for such inputs, the brute force method takes at most constant time. Otherwise, we have $m_2 \geq N$. Consequently, for $X_1 = F(X_2)$ we have the followings hold.

1. by (a), M_1 can be constructed in time $\log m_2 = \mathrm{poly}(|X_2|)$;
2. Since r is a fixed integer, $k_1 = r^3 k_2$ is also easily obtainable and we can write $1^{r^3 k_2}$ in time $\mathrm{poly}(k_2) = \mathrm{poly}(|X_2|)$;
3. by (b), $k_1 m_1^{k_1} = \mathrm{poly}(k_2 m_2^{k_2})$. Therefore, we can write 1^{m_1} and $1^{k_1 m_1^{k_1}}$ in polynomial time as well.

Therefore, $X_1 = F(X_2)$ is computable in polynomial time. Now, let

$$Y = (x_1, 1^{k_1|x_1|^{k_1}})$$

be a correct certificate for X_1. We can obtain $x_2 = f(x_1)$ in polynomial time as well. Additionally, by (c), $k_1|x_1|^{k_1} \geq k_2|x_2|^{k_2}$. Therefore, we can write $(x_2, 1^{k_2|x_2|^{k_2}})$ in polynomial time as well. This proves that G is also computable is polynomial time, and (III) is proven. □

Proof of lemma (4.9). We find integers N_0, N_1, N_2 for each element and take the maximum value $N = \max\{N_0, N_1, N_2\}$ to obtain the results simultaneously.

a. Note that M_1 in the above can be described with length

$$|M_1| = |M_2| + |f| + |g| + O(1).$$

Since f and g are fixed Turing machines, we obtain

$$|M_1| = |M_2| + O(1).$$

Additionally, since X_2 is correctly formatted, we have $m_2 \geq 2^{k_2|M_2|}$. Therefore,

$$|M_2| \leq \frac{\log m_2}{k_2}.$$

Now, there exists a large enough N_0 and constant c such that for $m_2 > N_0$, we have,

$$|M_1| \leq \frac{\log m_2}{k_2} + O(1) \leq c\frac{\log m_2}{k_2}.$$

b. We know that $m_1 = \max\{2^{k_1|M|_1}, m_2\}$. Therefore, we either have $m_1 = 2^{k_1|M|_1}$ or $m_1 = m_2$.

- If $m_1 = 2^{k_1|M_1|}$, then for $n \geq N_0$ in (a) we have,

$$\begin{aligned}
k_1 m_1^{k_1} &= k_1\big(2^{k_1|M_1|}\big)^{k_1} && (\text{by } m_1 = 2^{k_1|M_1|}) \\
&= r^3 k_2 (2^{k_1|M_1|})^{r^3 k_2} && (\text{by } k_1 = r^3 k_2) \\
&\leq r^3 k_2\big(2^{(k_1 c\frac{\log m_2}{k_2})}\big)^{r^3 k_2} && (\text{by (a)}) \\
&= r^3 k_2\big(2^{(r^3 k_2 c\frac{\log m_2}{k_2})}\big)^{r^3 k_2} && (\text{by } k_1 = r^3 k_2) \\
&\leq r^3 k_2\big(2^{(r^3 c\log m_2)}\big)^{r^3 k_2} \\
&\leq r^3 k_2\big(m_2^{cr^3}\big)^{r^3 k_2} \\
&\leq r^3 k_2\big(m_2^{k_2}\big)^{cr^6} \\
&\leq r^3\big(k_2 m_2^{k_2}\big)^{cr^6} \\
&= \text{poly}(k_2 m_2^{k_2}).
\end{aligned}$$

- If $m_1 = m_2$, then,

$$k_1 m_1^{k_1} = k_1 m_2^{k_1} = (r^3 k_2) m_2^{r^3 k_2} = \text{poly}(k_2 m_2^{k_2}).$$

Therefore, for $n \geq N_0$ we have,

$$k_1 (m_1)^{k_1} \leq \text{poly}(k_2 m_2^{k_2}).$$

c. Let $p_1(n)$ and $p_2(n)$ be the polynomials,

$$\begin{aligned} p_1(n) &= k_1 n^{k_1} = r^3 k_2 n^{r^3 k_2}, \\ p_2(n) &= n^{k_2 r^2} + k_2 (d n^d)^{k_2} \leq n^{k_2 r^2} + k_2 (r n^r)^{k_2}. \end{aligned}$$

We know that $r \geq 3$. Additionally, for $n \geq r \geq 3$ we have,

$$\begin{aligned} p_1(n) - p_2(n) &= r^3 k_2 n^{r^3 k_2} - (n^{k_2 r^2} + k_2 (r n^r)^{k_2}) \\ &\geq r^3 k_2 n^{r^3 k_2} - n^{k_2 r^2} - k_2 (n.n^r)^{k_2}) \\ &= r^3 k_2 n^{r^3 k_2} - n^{k_2 r^2} - k_2 n^{(r+1) k_2} \\ &\geq r^3 k_2 n^{r^3 k_2} - (k_2 + 1) n^{k_2 r^3} \\ &\geq 0. \end{aligned}$$

Therefore, there exists a large enough integer $N_1 \in \mathbb{N}$ such that for $n \geq m_2 \geq N_1$, we have $p_1(n) \geq p_2(n)$.

d. If M_2 takes more than $k_2 |x_2|^{k_2}$ many steps on x_2 we are done. Otherwise, we assume

$$\mathsf{TIME}(M_2, x_2) \leq k_2 |x_2|^{k_2}.$$

Note that f outputs string of length $l(|x_1|)$. Thus, we have

$$\begin{aligned} |x_2| = |f(x_1)| = l(|x_1|) = d|x_1|^d \leq r|x_1|^r, \\ |M_2[x_2]| \leq k_2 |x_2|^{k_2} \leq k_2 (r|x_1|^r)^{k_2}, \end{aligned}$$

where we have used the fact that $d < r$ in the first line. Additionally, since g and f run in time rn^r, we have,

$$\begin{aligned} \mathsf{TIME}(M_1, x_1) &= \mathsf{TIME}(f, x_1) &+ \mathsf{TIME}(M_2, x_2) &+ \mathsf{TIME}(g, (x_1, |M_2[x_2]|)) \\ &\leq r|x_1|^r &+ \mathsf{TIME}(M_2, f(x_1)) &+ r(|x_1| + |M_2[x_2]|)^r \\ &= \mathsf{TIME}(M_2, f(x_1)) &+ o(|x_1|^{k_2 r^2}). \end{aligned}$$

Therefore, there exists an integer N_2 such that for any x_2 and

$$|x_2| = |f(x_1)| \geq |x_1| \geq m_1 \geq m_2 \geq N_2,$$

we have,

$$\mathsf{TIME}(M_1, x_1) \leq \mathsf{TIME}(M_2, f(x_1)) + |x_1|^{k_2 r^2}.$$

By taking $N = \max\{N_0, N_1, N_2\}$, we can satisfy all the requirements of the lemma. □

4.4 Query model

Similar to the faux-deterministic setting, we define the dual problem in the query model to also be yet another query problem. As a result, an algorithm for the dual problem is a decision tree that can query the outputs of an another given tree T. To formally define the dual problem as a query search problem, we first need to generalize notions such as search problems and reductions in the query model.

4.4.1 Preliminaries for dual problem in query model

Search problems in query model

One initial difficulty for the rigorous formalization of dual in the query setting is that the problem $\mathcal{S}_n^*$ has inputs and alphabets that can depend on n. Precisely, the inputs for $\mathcal{S}_n^*$ are strings of length $2^{O(n)}$ corresponding to outputs of decision trees where each index corresponds to an input x for the tree. Thus, we generalize the definition of search problems to allow Σ, Γ, and input lengths to grow as n increases.

Definition 4.10 (Generalized search problems)**.** A *(search) problem* is a tuple $\mathcal{S} = (S, \Sigma, \Gamma, l)$ where

- $\Sigma = \{\Sigma_n\}_{n=1}^{\infty}$ and $\Gamma = \{\Gamma_n\}_{n=1}^{\infty}$ denote the finite inputs and output alphabets corresponding to each $n \in \mathbb{N}$,
- $l : \mathbb{N} \to \mathbb{N}$ is a length function determining the input length $l(n)$ for each $n \in \mathbb{N}$,
- $S = \{S_n\}_{n=1}^{\infty}$ is a family of relations where $S_n \subseteq \Sigma_n^{l(n)} \times \Gamma_n$ denotes the input-output pairs for the n-th problem instance.

For technical reasons, we also require that for all $n \in \mathbb{N}$ we have

1. $l(n+1) \geq l(n)$,
2. $\log |\Sigma_n| = O(l(n))$,
3. $|\Gamma_n| \leq |\Sigma_n|^{l(n)}$

The first constraint requires the length l not to decrease as n grows, the second condition is there so that alphabet size does not increase faster than the input length, and the third property means that number of possible outputs is at most as much as the number of possible inputs.

Whenever Σ, Γ, and l are obvious from the context, we denote a search problem simply with $\mathcal{S}$. For each natural number $n \in \mathbb{N}$, with abuse of notation, we denote the corresponding n-th instance relation as $\mathcal{S}_n := S_n$. ▷

Remark. As before, for each $x \in \Sigma_n^{l(n)}$ we define the set of corresponding certificates as $\mathcal{S}(x) := \mathcal{S}_n(x) = \{y \in \Gamma_n | (x, y) \in \mathcal{S}_n\}$, the domain as $\mathsf{dom}(\mathcal{S}) := \bigcup_n \mathsf{dom}(\mathcal{S}_n)$, and the range as $\mathsf{range}(\mathcal{S}) := \bigcup_n \mathsf{range}(\mathcal{S}_n)$. Also, we call a problem *total* if $\mathsf{dom}(\mathcal{S}) = \bigcup_n \Sigma_n^{l(n)}$. The definitions of decision trees and algorithms such as $T = (T_n)$ remain as before, where T_n solving $\mathcal{S}_n$ has input-output alphabets Σ_n and Γ_n with input length $l(n)$. ▷

We also need to modify the definition of efficient algorithms as we cannot expect to have polylog(n) complexity while the input length for $\mathcal{S}_n$ is exponential in n. Therefore, we call a deterministic, randomized, or verification algorithm efficient whenever it has polylog depth in the size of its input length $l(n)$. Subsequently, we can define complexity classes in the query model as follows.

Definition 4.11. For $k \in \mathbb{N}$, we define the classes $\mathsf{FP}^{\mathsf{dt}}(k)$, $\mathsf{FBPP}^{\mathsf{dt}}(k)$, and $\mathsf{FNP}^{\mathsf{dt}}(k)$ as the class of search problems $\mathcal{S} = (S, \Sigma, \Gamma, l)$ having deterministic, randomized, and verifier decision trees of complexities $log^k(l(n))$ for large enough n. Moreover, we define the class $\mathsf{TFNP}^{\mathsf{dt}}(k)$ to be the class of total search problems in $\mathsf{FNP}^{\mathsf{dt}}(k)$. Accordingly, we define the polylog(n) variant of of the above classes as,

$$\mathsf{FP}^{\mathsf{dt}} = \bigcup_{k \in \mathbb{N}} \mathsf{FP}^{\mathsf{dt}}(k) \qquad \mathsf{FBPP}^{\mathsf{dt}} = \bigcup_{k \in \mathbb{N}} \mathsf{FBPP}^{\mathsf{dt}}(k)$$

$$\mathsf{FNP}^{\mathsf{dt}} = \bigcup_{k \in \mathbb{N}} \mathsf{FNP}^{\mathsf{dt}}(k) \qquad \mathsf{TFNP}^{\mathsf{dt}} = \bigcup_{k \in \mathbb{N}} \mathsf{TFNP}^{\mathsf{dt}}(k)$$

▷

Reductions

We also will formalize the definition of reductions in the query model. For the sake of simplicity, we only consider reductions where we efficiently transform inputs of a search problem $x \in \mathcal{S}_n$ to inputs of another problem $x \in \mathcal{S}'_n$ and then use the output of $y' \in \mathcal{S}'_n(x')$ to obtain an output $f_n(y') \in \mathcal{S}(x)$. We call such reductions in query model Karp reductions and formalize them as follows.

Definition 4.12 (Decision forest)**.** We call a list of decision trees $\mathcal{L} = (T^1, \ldots, T^M)$ on input-output alphabets Σ, Σ' with input lengths N a *decision forest*. For any input $x \in \Sigma^N$ we define $\mathcal{L}[x]$ to be a string in Σ^M defined as,

$$\mathcal{L}[x] := T^1[x] \ldots T^M[x] \in \Sigma'^M,$$

and define the complexity of $\mathcal{L}$ as the maximum depth of all trees in the list,

$$\mathrm{C}^{\mathsf{dt}}(\mathcal{L}) := \max_{i \in [M]} \mathrm{C}^{\mathsf{dt}}(T^i).$$

▷

Definition 4.13 ((Karp) reduction in query model)**.** Now, let $\mathcal{S} = (S, \Sigma, \Gamma, l)$ and $\mathcal{S}' = (S', \Sigma', \Gamma', l')$ be two search problems. We say $\mathcal{S}$ reduces to $\mathcal{S}'$ and denote $\mathcal{S} \preceq_{\text{polylog}} \mathcal{S}'$ if there exists

- Natural numbers $N, d \in \mathbb{N}$
- a family of decision forests $\mathcal{L} = (\mathcal{L}_n)$ over input-output alphabets Σ_n and Σ'_n
- a family of functions $f = (f_n)$ where $f_n : \Gamma'_n \to \Gamma_n$,

such that for all $n \geq N$

1. $\mathrm{C}^{\mathsf{dt}}(\mathcal{L}_n) \leq \log^d l(n)$
2. $l'(n) = \theta(l(n))$
3. For all $x \in \mathsf{dom}(\mathcal{S}_n)$, $f_n(\mathcal{S}'_n(\mathcal{L}[x])) \subset \mathcal{S}_n(x)$

More accurately, we can write $\mathcal{S} \preceq_d \mathcal{S}'$ to denote the complexity of the decision forest in the above definition as well. ▷

Intuitively, we transform each input x of $\mathcal{S}$ to an input $x' = \mathcal{L}[x]$ of $\mathcal{S}'$ where x'_i is determined by running the i-th decision tree in $\mathcal{L}$ on x. (i) ensures that transformation of x to $x' = \mathcal{L}_n[x]$ is efficient, (ii) means the length of x' is not far from length of x, and (iii) means that for any output $y \in \mathcal{S}'_n(\mathcal{L}[x])$, the obtained value for $\mathcal{S}'$, $f_n(y)$, is a correct certificate for $\mathcal{S}_n$. Therefore, for $n \geq N$, having an algorithm $\mathcal{T}$ for $\mathcal{S}'$, we can answer the queries of $\mathcal{T}$ to x' by running T_i's on x, and use the transformation f to obtain a result for $\mathcal{S}(x)$ from $\mathcal{S}'(x')$.

Lemma 4.14. *Suppose $\mathcal{S} \preceq_d \mathcal{S}'$. If $\mathcal{S}'$ is in $\mathsf{FP}^{\mathsf{dt}}(k)$ ($\mathsf{FBPP}^{\mathsf{dt}}(k)$), then $\mathcal{S} \in \mathsf{FP}^{\mathsf{dt}}(k+d)$ ($\mathcal{S} \in \mathsf{FBPP}^{\mathsf{dt}}(k+d)$).*

Continuity of search problems

For the dual problem to be definable, we also need a notion of continuity for a search problem $\mathcal{S}$ that states $\mathcal{S}_n$ and $\mathcal{S}_m$ are fundamentally equally hard with respect to their input sizes for different values of $m, n \in \mathbb{N}$. We proceed with a similar definition to the Turing machine section.

Definition 4.15 (Continuity)**.** Let $\mathcal{C}$ be a complexity class. A search problem $\mathcal{S} \notin \mathcal{C}$ is a $\mathcal{C}$-continuous search problem if any algorithm in $\mathcal{C}$ fails to solve $\mathcal{S}$ on infinitely many input lengths. More specifically, $\mathcal{S} \notin \mathcal{C}$ is $\mathcal{C}$-continuous if for all infinite subsets $I \subseteq \mathbb{N}$,

$$\mathcal{S} \cap (\bigcup_{n \in I} \mathsf{dom}(\mathcal{S}_n) \times \mathsf{range}(\mathcal{S}_n)) \notin \mathcal{C}.$$

for every infinite subset $I \subset \mathbb{N}$, $\mathcal{S}$ is polylog($l(n)$) solvable on inputs $n \in I$ if and only if it is polylog($l(n)$) solvable on all the inputs $n \in \mathbb{N}$. ▷

This definition ensures that we are not dealing with a problem that, for instance, is hard on even instances $n = 2k$, yet is easy on the odd instances $n = 2k + 1$. Therefore, almost every natural problem in query complexity satisfies this property as obtained lower bounds usually correspond to every input length $n \in \mathbb{N}$.

4.4.2 Dual problems in query model

Since query complexity is a non-uniform model, restrictions such as $\mathrm{C}^{\mathsf{dt}}(T) = \text{polylog}(n)$ do not make sense as a given decision tree T for the dual problem has a fixed depth.

Additionally, having only a black-box query access to the tree T, the algorithm does not know $\mathrm{C}^{\mathsf{dt}}(T)$ to be able to refute it if $\mathrm{C}^{\mathsf{dt}}(T) > \mathbb{D}^{\mathsf{dt}}(\mathcal{S})$. Thus, we need a promise that only trees that cannot solve $\mathcal{S}$ are valid inputs $\mathcal{S}^*$. Because of these reasons, we consider fixed depths of type $\log^k(n)$ where k is a constant.

Definition 4.16. We define $\mathsf{Tree}_{n,k}(\Sigma, \Gamma, l)$ to be the set of all decision trees with input and output alphabets and length Σ, Γ, and $l(n)$, and has depth at most $\log^k(l(n))$. Additionally, for $T \in \mathsf{Tree}_{n,k}(\Sigma, \Gamma, l)$ with abuse of notation, we use T to denote a string in $\Gamma^{|\Sigma|^{l(n)}}$ as well. ▷

Definition 4.17 (Dual problem in query model)**.** Let $\mathcal{S} = (S, \Sigma, \Gamma, l)$ be a search problem and $k \in \mathbb{N}$ be an integer. We define the relations S^k_n as

$$S^k_n := \{(T, x) | T \in \mathsf{Tree}_{n,k}(\Sigma, \Gamma, l), \mathrm{C}^{\mathsf{dt}}(T) < \mathbb{D}^{dt}(\mathcal{S}_n), x \in \mathsf{dom}(\mathcal{S}_n), T[x] \notin \mathcal{S}_n(x)\}$$

For any $k \in \mathbb{N}$, the dual problem has input alphabet $\Sigma^*_n := \Gamma_n$, output alphabet $\Gamma^*_n = \Sigma^{l(n)}_n$, and length $l^*(n) = |\Sigma_n|^{l(n)}$. Therefore, for $n \in \mathbb{N}$,

i $\log |\Sigma^*_n| = \log |\Gamma_n| = O(\log |\Sigma_n|^{l(n)}) = O(\log l^*(n)) = O(l^*(n))$,

ii $|\Gamma^*_n| = |\Sigma^{l(n)}_n)| = |\Sigma_n|^{l(n)} = l^*(n) \leq |\Sigma_n|^{l(n)}$,

iii $l(n) \leq l(n+1)$.

Thus, $\mathcal{S}^k$ satisfies the conditions of a generalized search problem. ▷

4.4.3 Duality results in query model

We can now state our results in the query model as follows.

Theorem 4.18. *Let $\mathcal{S}$ and $\mathcal{S}'$ be two search problems in $\mathsf{TFNP}^{\mathsf{dt}}(v) \setminus \mathsf{FP}^{\mathsf{dt}}$ that are $\mathsf{FP}^{\mathsf{dt}}$-continuous. In other words, $\mathcal{S}$ and $\mathcal{T}$ have verifiers of depth v, but do not have any deterministic algorithm of depth $\log^k(n)$ for any fixed k. Then,*

i For $k \in \mathbb{N}$, the dual problem $\mathcal{S}^k$ is in $\mathsf{FNP}^{\mathsf{dt}}(0)$,

ii If $\mathcal{S} \in \mathsf{FBPP}^{\mathsf{dt}}(r)$, then for any $k > r + v + 3$, we have $\mathcal{S}^k \notin \mathsf{FBPP}^{\mathsf{dt}}$ ($\mathcal{S}^k$ is hard on average).

iii If $\mathcal{S} \notin \mathsf{FBPP}^{\mathsf{dt}}$ ($\mathcal{S}$ is hard on average), then $\mathcal{S}^k \in \mathsf{FBPP}^{\mathsf{dt}}(1)$ for any k.

iv If $\mathcal{S} \preceq_d \mathcal{S}'$ *then for every fixed* k, $\mathcal{S}'^k \preceq_{\text{polylog}} \mathcal{S}^{k+d+1}$

Proof. Given query access to outputs of decision tree T and an input x, we can simply query x from T and check whether $T[x] \in \mathcal{S}(x)$ or not; therefore, we have a verifier of depth one to check $(T, x) \in \mathcal{S}^k$ and (i) holds. We prove (ii) using Yao's principal[Yao77] in query complexity and our construction of faux-deterministic algorithms.

Suppose $\mathcal{S}$ has a randomized algorithm and verifier of depths $\log^r(l(n))$ and $\log^v(l(n))$. Now, by proposition (3.20), we have a family of decision trees $\mathcal{T} = (\mathcal{T}_n)$ that have depths $\log^{r+v+3}(l(n))$ where any deterministic adversary of depth $\text{poly}(l(n))$ fails to find an adverse input for a randomly chosen tree $T \sim \mathcal{T}$. Additionally, we have,

$$\text{polylog}(l^*(n)) = \text{polylog}(|\Sigma_n|^{l(n)}) = \text{poly}(l(n) \log |\Sigma(n)|) = \text{poly}(l(n))$$

Therefore, there exists a distribution of inputs for $\mathcal{S}^{r+v+3}$ such that no deterministic algorithm of depth $\text{polylog}(l^*(n))$ can solve it with non-negligible probability. Consequently, by Yao's minimax theorem, $\mathcal{S}^{r+v+3}$ does not have a $\text{polylog}(l^*(n))$ randomized algorithm, and $\mathcal{S}^{r+v+3} \notin \mathsf{FBPP}^{\mathsf{dt}}$. Conversely, if $\mathcal{S} \notin \mathsf{FBPP}^{\mathsf{dt}}$, then for any k and a large enough $n \geq N_0$ we have we have $\log^k l(n) \leq \mathbb{R}^{\mathsf{dt}}(\mathcal{S}_n)$. Now, again by Yao's principal[Yao77], for $n \geq N_0$, there exists a distribution of inputs $\mathcal{D}_n$ such that for any deterministic decision tree T_n with $\mathrm{C}^{\mathsf{dt}}(T_n) < \log^k l(n) \leq \mathbb{R}^{\mathsf{dt}}(\mathcal{S}_n)$ we have,

$$\Pr_{\boldsymbol{x} \sim \mathcal{D}_n}[T_n[x] \notin \mathcal{S}(x)] \geq \frac{2}{3}.$$

Therefore, the adversary sampling a random query from $\mathcal{D}_n$ can solve $\mathcal{S}_n^k$ for $n \geq N_0$ with one query, and for $n < N_0$, we can solve $\mathcal{S}_n^k$ by a simple randomized algorithm of depth $O(1)$. Therefore, $\mathcal{S}^k \in \mathsf{FBPP}^{\mathsf{dt}}(1)$.

For (iii), suppose $\mathcal{S} \preceq_d \mathcal{S}'$, and let $\mathcal{L}$, N_0, f be given as in the reduction definition. Then, for $n \geq N$

- $\mathrm{C}^{\mathsf{dt}}(\mathcal{L}_n) \leq \log^d l(n)$,
- $l'(n) = \theta(l(n))$,
- For all $x \in \mathsf{dom}(\mathcal{S}_n)$, $f_n(\mathcal{S}'(\mathcal{L}[x])) \subset \mathcal{S}(x)$.

Now, given tree T'_n for $\mathcal{S}'^*_n$ where T' is a decision tree from $\mathsf{Trees}_{n,k}(\Sigma', \Gamma', l')$ we define a new tree T_n for $\mathcal{S}$ as,

$$T_n[x] := f(T'[\mathcal{L}_n[x]])$$

Note that, for large enough $n > \max\{N_0, N_1\}$,

$$\mathrm{C}^{\mathsf{dt}}(T) \leq \mathrm{C}^{\mathsf{dt}}(T').\,\mathrm{C}^{\mathsf{dt}}(\mathcal{L}_n) \leq \log^k(l'(n)).\log^d(l'(n)) \leq \log^{k+d}(\theta(l(n))) \leq \log^{k+d+1}(l(n))$$

Thus, for at most finitely many n, the tree T_n obtained from T'_n solves $\mathcal{S}_n$. Otherwise, $\mathcal{S}$ has a polylog depth algorithm by the continuity condition. Therefore, there exists an integer N_2 such that for $n \geq \max\{N_0, N_1, N_2\}$, T_n is in the promise of $\mathcal{S}_n^{k+d+1}$ and $\mathcal{S}_n^{k+d+1}(T_n)$ is non-empty. Now for any $x \in \mathcal{S}_n^{k+d+1}(T_n)$, where $T_n[x] \notin \mathcal{S}(x)$, we must have $\mathcal{L}_n[x] \notin \mathcal{S}_n'^k(T'_n)$ as otherwise

$$T_n[x] = f(\mathcal{L}_n[x]) \in f(\mathcal{S}'(\mathcal{L}_n[x])) \subset \mathcal{S}(x)$$

which is a contradiction. Therefore, we have our reduction. □

Chapter 5

Conclusion

We conclude this thesis by summarizing our results and outlining possible future directions for research on faux-deterministic algorithms and duality within TFNP.

5.1 Faux-determinism

In chapter 3, we proved existence of $(2^{O(t^{\frac{1}{2}})}, 2^{O(t^{\frac{1}{2}})})$-faux deterministic algorithms for the FIND1 problem in the query model. In our construction, we utilized $\sqrt{t}$ of the total queries in the non-adaptive blocks to hide sensitive indices from the adversary. However, our simple randomized attack on faux-deterministic algorithms of depth t gave an upper bound of $2^{O(t)}$ for the adversary's complexity. Therefore, a natural question would be whether a better construction matches the trivial upper bound in proposition (3.19).

Problem 5.1. Is there a $(O(2^t), O(2^{-t}))$-faux-deterministic algorithm for FIND1 problem?

Additionally, we generalized our construction in query complexity for a family of search problems in $\mathsf{TFNP}^{dt} \cap \mathsf{FBPP}^{\mathsf{dt}}$. We observed that the condition of the randomized algorithm is necessary. We also suggested that construction is heavily dependent on the verification algorithm by providing a natural extension of the FIND1 problem to the FINDMEDIAN problem that did not have a verifier and a faux-deterministic algorithm. However, we did not prove any general relation between faux-deterministic complexity and verification complexity.

Problem 5.2. What are the relations between faux-deterministic complexity and verification, sensitivity, and certificate complexities of problems in $\mathsf{TFNP}^{\mathsf{dt}} \setminus \mathsf{FP}^{\mathsf{dt}}$?

Besides query complexity, we can extend definition of faux-deterministic algorithms to other computational models such as communication complexity. An interesting direction would be to improve or lift the results in query complexity to communication complexity.

Problem 5.3. Can we extend the results for faux-determinism in query complexity to communication complexity as well?

5.2 Duality

In Chapter 4, we gave an intuitive overview of why our observations are natural for the dual problem, and formalized several of those observations for the dual problem in the purely white-box Turing machine model and the purely black-box query model. However, several questions remained unanswered.

- In the Turing machine model, we did not prove any relation between average-case hardness and having a randomized algorithm for the primal and dual problems. For appropriate definitions of average-case hardness, we ask the following problem.

  Problem 5.4. Do the following observations hold for the dual problem in Turing machine model?

  iv if $\mathcal{S}$ has a randomized algorithm then $\mathcal{S}^*$ is hard on average;

  v if $\mathcal{S}$ is hard on average, then $\mathcal{S}^*$ has a randomized algorithm.

- Additionally, our results in Turing machine model require a lower-bound of a $\mathcal{S} \in \mathsf{TFNP} \setminus \mathsf{quasiFP}$. An interesting next step would be to see if we can relax the requirement to $\mathcal{S} \in \mathsf{TFNP} \setminus \mathsf{FP}$.

  Problem 5.5. Can we prove the same duality results in Turing machine model for search problems in $\mathsf{TFNP} \setminus \mathsf{FP}$?

- In the query model, for an integer k, we defined the corresponding dual problem of $\mathcal{S}$ as $\mathcal{S}^k$ where only decision trees of depth $\log^k n$ are inside the promise. However, a more general definition for dual problem can accept any decision tree T of depth $d < \mathrm{C}^{\mathsf{dt}}(\mathcal{S})$. As a result, we ask the following question.

  Problem 5.6. Can the duality properties be proven for the more general definition of the dual problem in query complexity?

- Finally, another direction for studying duality within TFNP is considering different computational models. Specifically, we can have any combination of uniform or non-uniform, black-box, gray-box, or white-box computational models.

  Problem 5.7. Is there a natural computational model for which all of the duality properties are satisfied simultaneously?

References

[Aar21] Scott Aaronson. "Open Problems Related to Quantum Query Complexity". In: *ACM Transactions on Quantum Computing* 2.4 (Dec. 2021). ISSN: 2643-6809. DOI: 10.1145/3488559. URL: https://doi.org/10.1145/3488559.

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009. DOI: 10.1017/CBO9780511804090.

[Adl78] Leonard Adleman. "Two theorems on random polynomial time". In: *19th Annual Symposium on Foundations of Computer Science (sfcs 1978)*. 1978, pp. 75–83. DOI: 10.1109/SFCS.1978.37.

[Amb17] Andris Ambainis. *Understanding Quantum Algorithms via Query Complexity*. 2017. arXiv: 1712.06349 [quant-ph].

[AS16] Noga Alon and Joel H. Spencer. *The Probabilistic Method*. 4th. Wiley Publishing, 2016. ISBN: 1119061954.

[Ats06] Albert Atserias. "Distinguishing SAT from Polynomial-Size Circuits, through Black-Box Queries". In: *Proceedings of the 21st Annual IEEE Conference on Computational Complexity*. CCC '06. USA: IEEE Computer Society, 2006, pp. 88–95. ISBN: 0769525962. DOI: 10.1109/CCC.2006.17. URL: https://doi.org/10.1109/CCC.2006.17.

[Bd02] Harry Buhrman and Ronald de Wolf. "Complexity measures and decision tree complexity: a survey". In: *Theoretical Computer Science* 288.1 (2002). Complexity and Logic, pp. 21–43. ISSN: 0304-3975. DOI: https://doi.org/10.1016/S0304-3975(01)00144-X. URL: https://www.sciencedirect.com/science/article/pii/S030439750100144X.

[Bea+01] Robert Beals et al. “Quantum Lower Bounds by Polynomials”. In: *J. ACM* 48.4 (July 2001), pp. 778–797. ISSN: 0004-5411. DOI: 10.1145/502090.502097. URL: https://doi.org/10.1145/502090.502097.

[Ben+97] Charles H. Bennett et al. “Strengths and Weaknesses of Quantum Computing”. In: *SIAM Journal on Computing* 26.5 (Oct. 1997), pp. 1510–1523. DOI: 10.1137/s0097539796300933.

[Bra+23] Vladimir Braverman et al. *Lower Bounds for Pseudo-Deterministic Counting in a Stream*. 2023. arXiv: 2303.16287 [cs.DS].

[BT21] Andrej Bogdanov and Luca Trevisan. *Average-Case Complexity*. 2021. arXiv: cs/0606037 [cs.CC].

[CDM23] Arkadev Chattopadhyay, Yogesh Dahiya, and Meena Mahajan. “Query Complexity of Search Problems”. In: *Electron. Colloquium Comput. Complex.* TR23-039 (2023). ECCC: TR23-039. URL: https://eccc.weizmann.ac.il/report/2023/039.

[Che+22] Lijie Chen et al. “Constructive Separations and Their Consequences”. In: *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, July 2022. DOI: 10.1109/focs52979.2021.00069. URL: https://doi.org/10.1109%2Ffocs52979.2021.00069.

[Che+23] Lijie Chen et al. *Polynomial-Time Pseudodeterministic Construction of Primes*. 2023. arXiv: 2305.15140 [cs.CC].

[GG11] Erann Gat and Shafi Goldwasser. “Probabilistic Search Algorithms with Unique Answers and Their Cryptographic Applications”. In: *Electron. Colloquium Comput. Complex.* TR11 (2011).

[GGR13] Oded Goldreich, Shafi Goldwasser, and Dana Ron. “On the Possibilities and Limitations of Pseudodeterministic Algorithms”. In: *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*. ITCS ’13. Berkeley, California, USA: Association for Computing Machinery, 2013, pp. 127–138. ISBN: 9781450318594. DOI: 10.1145/2422436.2422453. URL: https://doi.org/10.1145/2422436.2422453.

[Gol+20] Shafi Goldwasser et al. “Pseudo-Deterministic Streaming”. In: *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*. Ed. by Thomas Vidick. Vol. 151. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2020, 79:1–79:25. ISBN: 978-3-95977-134-4. DOI:

10.4230/LIPIcs.ITCS.2020.79. URL: https://drops.dagstuhl.de/opus/volltexte/2020/11764.

[Gol+21] Shafi Goldwasser et al. "On the Pseudo-Deterministic Query Complexity of NP Search Problems". In: *36th Computational Complexity Conference (CCC 2021)*. Ed. by Valentine Kabanets. Vol. 200. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 36:1–36:22. ISBN: 978-3-95977-193-1. DOI: 10.4230/LIPIcs.CCC.2021.36. URL: https://drops.dagstuhl.de/opus/volltexte/2021/14310.

[Gol08] Oded Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008. DOI: 10.1017/CBO9780511804106.

[Gro96] Lov K. Grover. "A Fast Quantum Mechanical Algorithm for Database Search". In: *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*. STOC '96. Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 1996, pp. 212–219. ISBN: 0897917855. DOI: 10.1145/237814.237866. URL: https://doi.org/10.1145/237814.237866.

[GST07] Dan Gutfreund, Ronen Shaltiel, and Amnon Ta-Shma. "If NP Languages Are Hard on the Worst-Case, Then It is Easy to Find Their Hard Instances". In: *Comput. Complex.* 16.4 (Dec. 2007), pp. 412–441. ISSN: 1016-3328. DOI: 10.1007/s00037-007-0235-8. URL: https://doi.org/10.1007/s00037-007-0235-8.

[HÅs+99] Johan HÅstad et al. "A Pseudorandom Generator from any One-way Function". In: *SIAM Journal on Computing* 28.4 (1999), pp. 1364–1396. DOI: 10.1137/S0097539793244708. eprint: https://doi.org/10.1137/S0097539793244708. URL: https://doi.org/10.1137/S0097539793244708.

[Hua19] Hao Huang. *Induced subgraphs of hypercubes and a proof of the Sensitivity Conjecture*. 2019. arXiv: 1907.00847 [math.CO].

[IW97] Russell Impagliazzo and Avi Wigderson. "P = BPP If E Requires Exponential Circuits: Derandomizing the XOR Lemma". In: *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*. STOC '97. El Paso, Texas, USA: Association for Computing Machinery, 1997, pp. 220–229. ISBN: 0897918886. DOI: 10.1145/258533.258590. URL: https://doi.org/10.1145/258533.258590.

[Kab00] V. Kabanets. “Easiness assumptions and hardness tests: trading time for zero error”. In: *Proceedings 15th Annual IEEE Conference on Computational Complexity*. 2000, pp. 150–157. DOI: 10.1109/CCC.2000.856746.

[KN96] Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, 1996. DOI: 10.1017/CBO9780511574948.

[Lás+91] Lovász László et al. “Search problems in the decision tree model”. In: vol. 8. Nov. 1991, pp. 576–585. ISBN: 0-8186-2445-0. DOI: 10.1109/SFCS.1991.185422.

[LOS21] Zhenjian Lu, Igor C. Oliveira, and Rahul Santhanam. “Pseudodeterministic Algorithms and the Structure of Probabilistic Time”. In: *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2021. Virtual, Italy: Association for Computing Machinery, 2021, pp. 303–316. ISBN: 9781450380539. DOI: 10.1145/3406325.3451085. URL: https://doi.org/10.1145/3406325.3451085.

[NC10] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010. DOI: 10.1017/CBO9780511976667.

[Nis89] Noam Nisan. “CREW PRAMS and decision trees”. In: *SIAM J. Comput.* 20 (1989), pp. 999–1007.

[NW94] Noam Nisan and Avi Wigderson. “Hardness vs randomness”. In: *Journal of Computer and System Sciences* 49.2 (1994), pp. 149–167. ISSN: 0022-0000. DOI: https://doi.org/10.1016/S0022-0000(05)80043-1. URL: https://www.sciencedirect.com/science/article/pii/S0022000005800431.

[OS17] Igor Oliveira and Rahul Santhanam. “Pseudodeterministic constructions in subexponential time”. In: June 2017, pp. 665–677. DOI: 10.1145/3055399.3055500.

[Ros98] Sheldon M. Ross. *A First Course in Probability*. Fifth. Upper Saddle River, N.J.: Prentice Hall, 1998.

[Yao77] Andrew Chi-Chin Yao. “Probabilistic Computations: Toward a Unified Measure of Complexity”. In: *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*. SFCS ’77. USA: IEEE Computer Society, 1977, pp. 222–227. DOI: 10.1109/SFCS.1977.24. URL: https://doi.org/10.1109/SFCS.1977.24.

[Yao82] Andrew C. Yao. “Theory and application of trapdoor functions”. In: *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*. 1982, pp. 80–91. DOI: 10.1109/SFCS.1982.45.

APPENDICES

Appendix A

Remaining Proofs

In this appendix, we will provide the postponed proofs in the main body.

Proof of lemma (3.17). First, we can remove the equal unitaries in both $\left|\psi_q^{k+1}\right\rangle$ and $\left|\psi_q^k\right\rangle$ to obtain

$$\begin{aligned}\left\| \left|\psi_q^{k+1}\right\rangle - \left|\psi_q^k\right\rangle \right\| &= \left\| U_q U^{\boldsymbol{T}} \dots U_{k+1} U^{\boldsymbol{T^{k+1}}} \left|\psi_k^k\right\rangle - U_q U^{\boldsymbol{T}} \dots U_{k+1} U^{\boldsymbol{T}} \left|\psi_k^k\right\rangle \right\| \\ &= \left\| U^{\boldsymbol{T^{k+1}}} \left|\psi_k^k\right\rangle - U^{\boldsymbol{T}} \left|\psi_k^k\right\rangle \right\| \\ &= \left\| \left(U^{\boldsymbol{T^{k+1}}} - U^{\boldsymbol{T}}\right) \left|\psi_k^k\right\rangle \right\|.\end{aligned}$$

Now, for $x \in \{0,1\}^n$, we define Π_x to be the projection to the states having input register $|x\rangle_I$. Consequently, we have $\sum_{x\in\{0,1\}^n} \Pi_x = I$, and we can write

$$\begin{aligned}\left\| \left|\psi_q^{k+1}\right\rangle - \left|\psi_q^k\right\rangle \right\| &= \left\| \left(U^{\boldsymbol{T^{k+1}}} - U^{\boldsymbol{T}}\right) \left|\psi_k^k\right\rangle \right\| \\ &= \left\| \left(U^{\boldsymbol{T^{k+1}}} - U^{\boldsymbol{T}}\right)\Big(\sum_{x\in\{0,1\}^n} \Pi_x \left|\psi_k^k\right\rangle \Big) \right\| \\ &= \left\| \sum_{x\in\{0,1\}^n} \left(U^{\boldsymbol{T^{k+1}}} - U^{\boldsymbol{T}}\right)\Pi_x \left|\psi_k^k\right\rangle \right\|. \qquad \text{(A.1)}\end{aligned}$$

Note that if $\boldsymbol{T^{k+1}}[x] = \boldsymbol{T}[x]$ we have $(U^{\boldsymbol{T^{k+1}}} - U^{\boldsymbol{T}})\Pi_x \left|\psi_k^k\right\rangle = 0$, as the unitaries $U^{\boldsymbol{T^{k+1}}}$ and $U^{\boldsymbol{T^{k+1}}}$ are equivalent when applied to $|x\rangle$. Therefore, we can continue (A.1) as

$$\begin{aligned}
\left\| \left|\psi_q^{k+1}\right\rangle - \left|\psi_q^k\right\rangle \right\| &= \left\| \sum_{\substack{x\in\{0,1\}^n \\ \boldsymbol{T^{k+1}}[x]\neq\boldsymbol{T}[x]}} \left(U^{\boldsymbol{T^{k+1}}} - U^{\boldsymbol{T}}\right)\Pi_x \left|\psi_k^k\right\rangle \right\| \\
&= \left\| U^{\boldsymbol{T^{k+1}}} \sum_{\substack{x\in\{0,1\}^n \\ \boldsymbol{T^{k+1}}[x]\neq\boldsymbol{T}[x]}} \Pi_x \left|\psi_k^k\right\rangle - U^{\boldsymbol{T}} \sum_{\substack{x\in\{0,1\}^n \\ \boldsymbol{T^{k+1}}[x]\neq\boldsymbol{T}[x]}} \Pi_x \left|\psi_k^k\right\rangle \right\| \\
&\leq \left\| U^{\boldsymbol{T^{k+1}}} \sum_{\substack{x\in\{0,1\}^n \\ \boldsymbol{T^{k+1}}[x]\neq\boldsymbol{T}[x]}} \Pi_x \left|\psi_k^k\right\rangle \right\| + \left\| U^{\boldsymbol{T}} \sum_{\substack{x\in\{0,1\}^n \\ \boldsymbol{T^{k+1}}[x]\neq\boldsymbol{T}[x]}} \Pi_x \left|\psi_k^k\right\rangle \right\| \\
&= \left\| \sum_{\substack{x\in\{0,1\}^n \\ \boldsymbol{T^{k+1}}[x]\neq\boldsymbol{T}[x]}} \Pi_x \left|\psi_k^k\right\rangle \right\| + \left\| \sum_{\substack{x\in\{0,1\}^n \\ \boldsymbol{T^{k+1}}[x]\neq\boldsymbol{T}[x]}} \Pi_x \left|\psi_k^k\right\rangle \right\| \\
&= 2\left\| \sum_{\substack{x\in\{0,1\}^n \\ \boldsymbol{T^{k+1}}[x]\neq\boldsymbol{T}[x]}} \Pi_x \left|\psi_k^k\right\rangle \right\|. \qquad \text{(A.2)}
\end{aligned}$$

Now, consider the state $\left|\psi_k^k\right\rangle$; by definition, this state has been obtained by applying unitaries $\boldsymbol{T^1}, \ldots, \boldsymbol{T^k}$, and therefore all the coefficients in $\left|\psi_k^k\right\rangle$ only depend on sets $\boldsymbol{B}^1, \ldots, \boldsymbol{B}^k$. Consequently, conditioned on $\boldsymbol{B}^1 = B^1, \ldots, \boldsymbol{B}^k = B^k$, we can write

$$\left|\psi_k^k\right\rangle = \sum_{x\in\{0,1\}^n} \alpha_x \left|x\right\rangle_I \left|\phi_x\right\rangle_{Q,W,O},$$

where all coefficients α_x are fixed complex numbers. Therefore, conditioned on $\boldsymbol{B}^1 = B^1, \ldots, \boldsymbol{B}^k = B^k$ from (A.2) we get,

$$\left\| \left|\psi_q^{k+1}\right\rangle - \left|\psi_q^k\right\rangle \right\|^2 \leq 4 \sum_{\substack{x\in\{0,1\}^n \\ \boldsymbol{T^{k+1}}[x]\neq\boldsymbol{T}[x]}} \alpha_x^2.$$

Thus, we have

$$\begin{aligned}
\mu_{B^1,\ldots,B^k} &:= \mathop{\mathbb{E}}_{\boldsymbol{T}}\left[\left\| \left|\psi_q^{k+1}\right\rangle - \left|\psi_q^{k}\right\rangle \right\|^2 \middle| \boldsymbol{B}^1 = B^1, \ldots, \boldsymbol{B}^k = B^k\right] \\
&\leq 4 \mathop{\mathbb{E}}_{\boldsymbol{T}}\left[\sum_{\substack{x\in\{0,1\}^n \\ \boldsymbol{T}^{\boldsymbol{k+1}}[x]\neq \boldsymbol{T}[x]}} \alpha_x^2 \middle| \boldsymbol{B}^1 = B^1, \ldots, \boldsymbol{B}^k = B^k\right] \\
&\leq 4 \sum_{x\in\{0,1\}^n} \alpha_x^2 \Pr_{\boldsymbol{T}}\left[\boldsymbol{T}[x] \neq \boldsymbol{T}^{k+1}[x] \middle| \boldsymbol{B}^1 = B^1, \ldots, \boldsymbol{B}^k = B^k\right].
\end{aligned}$$

Now, note that for each x, if $l_{\boldsymbol{T}}(x) \geq l_k$, then the simulation of $\boldsymbol{T^{k+1}}$ is faithful to $\boldsymbol{T}$ and we obtain the same value $\boldsymbol{T^{k+1}}[x] = \boldsymbol{T}[x]$. Therefore, we can now use lemma (3.8) to obtain the following:

$$\begin{aligned}
&\Pr_{\boldsymbol{T}}\left[\boldsymbol{T}[x] \neq \boldsymbol{T}^{k+1}[x] \middle| \boldsymbol{B}^1 = B^1, \ldots, \boldsymbol{B}^k = B^k\right] \\
\leq &\Pr_{\boldsymbol{T}}\left[l_T(x) \prec l_{k+1} \middle| \boldsymbol{B}^1 = B^1, \ldots, \boldsymbol{B}^k = B^k\right] \\
= &1 - \Pr_{\boldsymbol{T}}\left[l_T(x) \succeq l_{k+1} \middle| \boldsymbol{B}^1 = B^1, \ldots, \boldsymbol{B}^k = B^k\right] \\
\leq &1 - (1 - \frac{1}{2^w})^h \\
\leq &\frac{h}{2^w}.
\end{aligned}$$

As a result, we get the upper bound

$$\begin{aligned}
\mu_{B^1,\ldots,B^k} &\leq 4 \sum_{x\in\{0,1\}^n} \alpha_x^2 \Pr_{\boldsymbol{T}}\left[\boldsymbol{T}[x] \neq \boldsymbol{T}^{k+1}[x] \middle| \boldsymbol{B}^1 = B^1, \ldots, \boldsymbol{B}^k = B^k\right] \\
&\leq \frac{4h}{2^w} \sum_{x\in\{0,1\}^n} \alpha_x^2 \\
&= \frac{4h}{2^w}.
\end{aligned}$$

Hence, we can compute the total expectation using the conditional expectations to get

$$\begin{aligned}
&\mathop{\mathbb{E}}_{\boldsymbol{T}}\left[\left\| \left|\psi_q^{k+1}\right\rangle - \left|\psi_q^{k}\right\rangle \right\|^2\right] \\
&= \sum_{B^1,\ldots,B^k} \mathop{\mathbb{E}}_{\boldsymbol{T}}\left[\left\| \left|\psi_q^{k+1}\right\rangle - \left|\psi_q^{k}\right\rangle \right\|^2 \middle| \boldsymbol{B}^1 = B^1, \ldots, \boldsymbol{B}^k = B^k\right] \Pr_{\boldsymbol{T}}\left[\boldsymbol{B}^1 = B^1, \ldots, \boldsymbol{B}^k = B^k\right] \\
&= \sum_{B^1,\ldots,B^k} \mu_{B^1,\ldots,B^k} \Pr_{\boldsymbol{T}}\left[\boldsymbol{B}^1 = B^1, \ldots, \boldsymbol{B}^k = B^k\right] \\
&\leq \frac{4h}{2^w}.
\end{aligned}$$

Finally, we finish the proof by using Jensen's inequality to obtain,

$$\begin{aligned}
\mathop{\mathbb{E}}_{\boldsymbol{T}}\left[\left\| \left|\psi_q^{k+1}\right\rangle - \left|\psi_q^{k}\right\rangle \right\|\right] &\leq \sqrt{\mathop{\mathbb{E}}_{\boldsymbol{T}}\left[\left\| \left|\psi_q^{k+1}\right\rangle - \left|\psi_q^{k}\right\rangle \right\|^2\right]} \\
&\leq \sqrt{\frac{4h}{2^w}}.
\end{aligned}$$

□

Proof of lemma (3.18). First, we can condition the possible values for the blocks to obtain

$$\mathop{\mathbb{E}}_{\boldsymbol{T}}\left[p'(T)\right] = \sum_{B^1,\ldots,B^q} \mathop{\mathbb{E}}_{\boldsymbol{T}}\left[p'(T)|\boldsymbol{B}^1 = B^1,\ldots,\boldsymbol{B}^q = B^q\right] \Pr_{\boldsymbol{T}}\left[\boldsymbol{B}^1 = B^1,\ldots,\boldsymbol{B}^q = B^q\right].$$

Note that the output of $\mathcal{Q}'$ is independent of $\boldsymbol{B}^{2^d}$ as the algorithm $\mathcal{Q}'$ has only access to the first $q < 2^d$ blocks and does not make any real queries to the tree. Therefore, we can further condition on output of $\mathcal{Q}'$ to obtain

$$\begin{aligned}
&\mathop{\mathbb{E}}_{\boldsymbol{T}}\left[p'(\boldsymbol{T})|\boldsymbol{B}^1,\ldots,\boldsymbol{B}^q\right] \\
&= \sum_{y\in\{0,1\}^*,|y|_1\geq\frac{n}{2}} \mathop{\mathbb{E}}_{\boldsymbol{T}}\left[p'(\boldsymbol{T})|y \leftarrow \mathcal{Q}'[\boldsymbol{T}],\boldsymbol{B}^1,\ldots,\boldsymbol{B}^q\right] \Pr\left[y \leftarrow \mathcal{Q}'[\boldsymbol{T}]|\boldsymbol{B}^1,\ldots,\boldsymbol{B}^q\right] \\
&= \sum_{y\in\{0,1\}^*,|y|_1\geq\frac{n}{2}} \mathop{\mathbb{E}}_{\boldsymbol{T}}\left[\boldsymbol{T}[y] \notin \mathsf{FIND1}(y)|y \leftarrow \mathcal{Q}'[\boldsymbol{T}],\boldsymbol{B}^1,\ldots,\boldsymbol{B}^q\right] \Pr\left[y \leftarrow \mathcal{Q}'[\boldsymbol{T}]|\boldsymbol{B}^1,\ldots,\boldsymbol{B}^q\right].
\end{aligned} \tag{A.3}$$

Additionally, if we have $\boldsymbol{B}^{2^d}(x) = 1$, then we get $\boldsymbol{T}[x] \in \mathsf{FIND1}(x)$. Moreover, since $\boldsymbol{B}^{2^d}(x) = 1$ is independent of the first q blocks, at most $\frac{1}{2^w}$ all the random indices in $\boldsymbol{B}^{2^d}$ miss 1 values in x. Thus,

$$\begin{aligned}
\mathop{\mathbb{E}}_{\boldsymbol{T}}\left[p'(\boldsymbol{T})|\boldsymbol{B}^1,\ldots,\boldsymbol{B}^q\right] &\leq \sum_{y\in\{0,1\}^*,|y|_1\geq\frac{n}{2}} \frac{1}{2^w} \Pr\left[y \leftarrow \mathcal{Q}'[T]|\boldsymbol{B}^1,\ldots,\boldsymbol{B}^q\right] \\
&= \frac{1}{2^w}.
\end{aligned}$$

Finally, from (A.3) we get

$$\mathop{\mathbb{E}}_{\boldsymbol{T}}\left[p'(T)\right] \leq \frac{1}{2^w},$$

and by Jensen's inequality

$$\mathop{\mathbb{E}}_{\boldsymbol{T}}\left[\sqrt{p'(T)}\right] \leq \sqrt{\frac{1}{2^w}}.$$

□

Proof of lemma (3.22). For $\mathcal{S}$, let $\mathcal{R}_n$ and V_n be the randomized distribution and verifier of complexity polylog(n) on inputs of length n. Let $\boldsymbol{R_1}, \ldots, \boldsymbol{R_m} \sim \mathcal{R}_n$ be m uniformly random independent trees chosen from $\mathcal{R}_n$. For each input $x \in \Sigma^n$ and index $i \in [m]$, we define the random variable $\boldsymbol{X}_i^x := V_n[x, \boldsymbol{R}_i(x)]$ to denote whether $\boldsymbol{R}_i$ computes a correct output for x or not. By the definition, for each input $x \in \Sigma^n$, the random variable $\boldsymbol{X^x} := \sum_i \boldsymbol{X_i^x}$ denotes number of decision trees outputting a correct answer for x. Our goal is to show that with non-zero probability over the choice of $\boldsymbol{R}_i$'s, for all inputs $x \in \Sigma^n$, we have $\boldsymbol{X^x} \geq \frac{m}{2}$.

Let $x \in \Sigma^n$ be a fixed input. Using linearity of expectation for all $x \in \Sigma^n$, we have

$$\mathop{\mathbb{E}}_{\boldsymbol{R_1},\ldots,\boldsymbol{R_m}}\left[\boldsymbol{X^x}\right] = \sum_{i=1}^{m} \mathop{\mathbb{E}}_{\boldsymbol{R_i}}\left[\boldsymbol{X_i^x}\right] = \sum_{i=1}^{m} \Pr_{\boldsymbol{R_i}}\left[\boldsymbol{X_i^x} = 1\right] \geq \frac{2m}{3},$$

where in the last step we used the fact that the randomized algorithm $\mathcal{R}_n$ succeeds with probability at least $\frac{2}{3}$. Moreover, since $\boldsymbol{R_1}, \ldots, \boldsymbol{R_m}$ are independent, the $\boldsymbol{X^{x_1}}, \ldots, \boldsymbol{X^{x_m}}$ are independent as well, and by the multiplicative form of Chernoff's bound, we have

$$\Pr_{\boldsymbol{R_1},\ldots,\boldsymbol{R_m}}\left[\boldsymbol{X^x} \leq \frac{m}{2}\right] = \Pr_{\boldsymbol{R_1},\ldots,\boldsymbol{R_m}}\left[\boldsymbol{X^x} \leq (1-\frac{1}{4})\frac{2m}{3}\right] \leq e^{\frac{-\frac{2m}{3}(\frac{1}{4})^2}{2}} = e^{-\frac{m}{48}}.$$

Thus, by union bound we can get

$$\Pr_{\boldsymbol{R_1},\ldots,\boldsymbol{R_m}}\left[\exists x, \boldsymbol{X^x} \leq \frac{m}{2}\right] \leq \sum_{x\in\Sigma^n} \Pr_{\boldsymbol{R_1},\ldots,\boldsymbol{R_m}}\left[\boldsymbol{X^x} \leq \frac{m}{2}\right] \leq |\Sigma|^n e^{-\frac{m}{32}}.$$

For $m > 32n.\ln|\Sigma|$, the above probability is non-zero, and there exists $R_1, \ldots, R_m \in \mathcal{R}_n$ such that for every $x \in \mathsf{dom}(\mathcal{S}_n)$ at least $\frac{m}{2}$ of R_i's output a correct answer. Consequently, a uniform distribution over $\mathcal{R}' := \{R_1, \ldots, R_m\}$ will complete the proof of lemma. □