Faux Determinism

Shalev Ben-David
Institute for Quantum Computing, Waterloo, Canada
University of Waterloo, Waterloo, Canada

M. H. Ebtehaj
Institute for Quantum Computing, Waterloo, Canada
University of Waterloo, Waterloo, Canada

Abstract

In the context of search problems, a pseudodeterministic algorithm is one which uses randomness but appears to be deterministic in its external behavior: it outputs the same answer when it is run on any one input multiple times. We introduce the relaxed notion of *computationally-secure pseudodeterminism*, which we call faux determinism. A faux-deterministic algorithm must appear to be deterministic to any external adversary with bounded resources. In other words, an algorithm is faux-deterministic if the task of "finding an input on which it fails to deterministically solve the search problem" is computationally intractable. In the black-box setting, we show that every verifiable search problem which has a randomized algorithm also has a faux-deterministic algorithm; this is not true of pseudodeterministic algorithms, which do not always exist. We highlight two reasons why this faux derandomization result is interesting: first, due to the computational indistinguishability, faux-deterministic algorithms may be used in place of pseudodeterministic ones in cryptographic settings. Second, from a conceptual point of view, our result helps explain why pseudodeterministic lower bounds are often hard to prove: such lower bounds must be inherently nonconstructive, since within $\mathsf{FZPP} \cap \mathsf{FNP}$, there is no efficient black-box algorithm for finding a hard input for a given faux-deterministic algorithm.

2012 ACM Subject Classification Theory of computation → Oracles and decision trees; Theory of computation → Pseudorandomness and derandomization; Theory of computation → Probabilistic computation

Keywords and phrases Decision Trees, Search Problems, Query Complexity, Pseudodeterminism, TFNP, Refutation

Digital Object Identifier 10.4230/LIPIcs.CCC.2026.39

Funding This research is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC), DGECR-2019-00027 and RGPIN-2019-04804.[1]

Acknowledgements We thank Dima Gavinsky for helpful discussions, as well as anonymous reviewers for helpful comments on a draft of this paper.

1 Introduction

In computational complexity, the power of randomness is usually modeled via complexity classes such as BPP, ZPP, or RP; these allow the randomness to affect the probability of error or the runtime of the algorithm.

In the context of search problems, the output of the algorithm may additionally depend on the randomization, even when the output is correct (there are typically multiple correct outputs for a given input to a search problem). This leads naturally to the notion of *pseudodeterminism*, introduced in [6]. A pseudodeterministic algorithm is defined to be an

[1] Cette recherche a été financée par le Conseil de recherches en sciences naturelles et en génie du Canada (CRSNG), DGECR-2019-00027 et RGPIN-2019-04804.

algorithm that may use randomness internally, but always outputs the same (correct) answer when run on the same input, even if run with different random seeds, except for a small probability of error.

Pseudodeterministic algorithms have been the subject of several lines of work in various computational models including query complexity [7, 11, 3], communication complexity [12], parallel and space-bounded computation [8, 10], and interactive proofs [9]. A major highlight is the recent result proving existence of a polynomial time pseudodeterministic algorithm that generates a canonical prime number for infinitely many input lengths [13, 5]. In the random oracle model, Yamakawa and Zhandry [14] also gave a search problem which can be solved efficiently using a quantum algorithm, but (assuming the Aaronson-Ambainis conjecture) cannot be solved efficiently by a pseudodeterministic quantum algorithm; they used this property for generating certified random bits, relying on the fact that any quantum algorithm which correctly solves the problem must give an output that has nontrivial entropy.

In this work, we introduce a variant of pseudodeterminism which we call faux determinism. A faux-deterministic algorithm need not succeed on all inputs; instead, it may fail to have good behavior on some inputs, so long as it is computationally difficult to find those inputs. In other words, while pseudodeterminism requires algorithms to be information-theoretically indistinguishable from deterministic algorithms in their output behavior, faux determinism merely requires algorithms to be computationally indistinguishable from deterministic algorithms relative to resource-bounded adversaries.

1.1 Definition and basic observations

We study faux determinism in the query (black-box) model.

Defining the model

We first clarify the definition. A faux-deterministic algorithm needs to have a random seed the adversary does not know about in advance, since otherwise finding an input on which the algorithm fails would be trivial (it could be hard-coded). For this reason, we define a faux-deterministic algorithm to be a probability distribution B over deterministic algorithms, with the property that for any computationally-bounded adversary A, if an algorithm B_r is sampled from this distribution, then the adversary A fails to find an input on which B_r fails when given oracle access to B_r.

In other words, our order of quantifiers is as follows: first, the faux-deterministic algorithm B is specified, except for a random seed r to be chosen later. Next, the adversary's algorithm A is chosen, in a way that may depend on the faux-deterministic algorithm B but not on its random seed. After this, the random seed r is chosen uniformly at random. Finally, A can make adaptive oracle calls to B_r, all with the same random seed r, which is now fixed. The goal of A is to find an input x to the search problem on which B_r fails, for this specific choice of random seed r. The algorithm B is called faux-deterministic if the task of A is impossible (except with superpolynomial resources).

In this work, we restrict our attention to the black-box query model; the algorithm B_r will be a decision tree, so B will be a probability distribution over decision trees, and the adversary A will be a (classical or quantum) query algorithm making queries to the function defined by B_r. In other words, the "truth table" of B_r will be considered the query input to the algorithm A, and strings x which are inputs to B_r are considered *indices* to the

exponentially long string B_r.[2] We often take the adversary A to be a randomized query algorithm, but we also consider the case where A is a quantum query algorithm, which makes queries to B_r in superposition. The formal definition is given in Definition 8.

To further clarify the definition, we make a few observations.

▶ **Lemma 1.** *A faux-deterministic algorithm is always a randomized algorithm.*

Proof. We show the contrapositive. If algorithm B fails as a randomized algorithm, there must exist a fixed input x such that $B_r(x)$ fails to give a valid output to the search problem with non-negligible probability over the randomness r. The adversary A can then deterministically output x without making any queries to B_r. With non-negligible probability over r, it will hold that B_r fails on the output x of A, which means that A refutes the proposed faux-deterministic algorithm B. ◀

While a faux-deterministic algorithm is always a randomized algorithm, the converse does not hold in general.

▶ **Lemma 2.** *A randomized algorithm, even with exponentially small error, need not satisfy the conditions of a faux-deterministic algorithm.*

Proof. Consider the FIND1 problem, the search problem in query complexity in which the input string x is promised to have Hamming weight at least $n/2$ and the goal is to output $i \in [n]$ such that $x_i = 1$. This has a randomized algorithm (output a random position i) whose error can be reduced by amplification (repeat k times to sample k random positions, query them to see if any of them are 1, and output the position of the first 1 found; this achieves error 2^{-k}).

This algorithm works on all inputs satisfying the Hamming weight promise. However, it is not faux-deterministic: when the randomness r is fixed, an adversary A can start by querying B_r on input x of high Hamming weight (such as $x = 1^n$), which gives a position i_1 such that $x_{i_1} = 1$; then it can construct an input y such that $y_{i_1} = 0$ and run $B_r(y)$, getting a different position i_2 such that $y_{i_2} = 1$; then find an input z with $z_{i_1} = z_{i_2} = 0$, querying it to find i_3; and so on. After k steps, the adversary will then find the entire set of k positions that the randomized algorithm B_r checks (which is a fixed set for any fixed choice of randomness r). If the adversary sets all those k positions to 0, the resulting string will be an input on which B_r fails. Finding this input requires only k queries to B_r, which is linear in the runtime of B_r, so B_r fails to be faux-deterministic (faux determinism requires superpolynomial security in the efficiency parameter). ◀

In fact, we can strengthen this result to show that some search problems which have a randomized algorithm lack *any* efficient faux-deterministic algorithm. However, we will later show in Theorem 5 that this does not happen for verifiable search problems such as FIND1.

▶ **Lemma 3.** *There is a black-box non-verifiable search problem which has an efficient randomized query algorithm but no efficient faux-deterministic algorithm.*

[2] One may object that the exponentially longer input string is not practical. For example, if the efficiency parameter is taken to be $\log n$, the adversary A must output strings of length n (and specify a string of length n whenever it makes a query to B_r). If one wants the adversary to specify those strings efficiently, one could switch to the "graybox" model, and represent each string x by a small circuit computing it. Our results should hold for this graybox model too. However, one must take care with graybox models, because depending on the allowed circuit sizes, some graybox models have FP = FBPP for verifiable search problems, causing a collapse of all the models we study in this work. We stick to the black-box model for simplicity.

Our proof of this lemma is very similar to the AOES construction in the concurrent work of [1].

Proof. Consider the following query problem on n bits: split the input into c blocks of n/c bits each, apply parity to each block, and output a string in $\{0,1\}^c$ purporting to solve the c parities; the output $z \in \{0,1\}^c$ is considered valid for an input x if it is *not all wrong*, that is, if for at least one $\ell \leq c$ it holds that z_ℓ is the value of the ℓ-th parity in the string x.

A randomized algorithm which makes no queries to x and guesses randomly will succeed except with error 2^{-c}. On the other hand, any faux-deterministic algorithm B can be refuted by a simple adversary A that employs the following strategy: first, A queries B_r on the input 0^n to get an output string $z \in \{0,1\}^c$. Next, A picks $i \in [n/c]$ at random, and flips the i-th bit of the ℓ-th parity block in the input 0^n for each $\ell \in [c]$ such that $z_\ell = 0$. In other words, A attempts to make all parities be wrong by changing only the i-th bit of each one. This results in a new input x (which has Hamming weight between 0 and c). Now, if B_r makes k queries to its input, then when i is chosen at random, with probability at least $1 - ck/n$, none of the i-th bits of any of the c parities in 0^n are queried by B_r. Therefore, with probability at least $1 - ck/n$, $B_r(x)$ still outputs the same output string z as before, which gets every parity wrong. Hence B_r fails on x, as desired.

Setting $c = \text{polylog}(n)$, the randomized algorithm has worst-case error $1/\text{ quasi-poly}(n)$ using 0 queries; on the other hand, the adversary A refutes B even if B uses $n^{1-o(1)}$ queries, with probability at least $1 - 1/\text{ poly}(n)$, using only a single query to B_r. Our definition of A was randomized, but it can also be made deterministic by choosing i to be the fixed bit in $[n/c]$ which works for as many random seeds r as possible. ◀

So far, we have seen that faux-deterministic algorithms are a proper subset of randomized algorithms. We next show that every pseudodeterministic algorithm, suitably amplified, is a faux-deterministic algorithm. It will also follow from Theorem 5 that not every faux-deterministic algorithm is pseudodeterministic; faux determinism is an intermediate condition between pseudodeterminism and randomized algorithms.

▶ **Lemma 4.** *A pseudodeterministic algorithm with negligible failure probability is always also a faux-deterministic algorithm.*

Proof. To see this, consider such an algorithm B, and let B_r denote the pseudodeterministic algorithm B with the randomness set to the fixed string r. Suppose there is an adversary strategy A such that (with non-negligible probability when r is chosen at random) A efficiently queries B_r to find an input x on which B_r fails. Now, since B is pseudodeterministic, it defines a deterministic function f from inputs to outputs. Moreover, since the failure rate of B is assumed to be negligible, the event that $B_r(z)$ outputs something different from $f(z)$ is negligible. Therefore, unless A makes exponentially many queries, the union bound dictates that with high probability over choice of r, the outputs A sees to its queries are identical to the outputs of f on those same queries.

Suppose then that the final output of A is a string x on which B_r fails. We can then produce x without any queries to B_r, simply by "querying" f instead of B_r, which is free (it has no dependence on r); this produces the same final output x on all but a negligible fraction of the random strings r. If A succeeds with non-negligible probability (over r), it means that the fixed string x is such that B fails on it with non-negligible probability (over r), which contradicts the assumption that B is a pseudodeterministic algorithm with negligible failure probability. ◀

To summarize, we have

$\mathsf{FP} \subseteq \mathsf{PseudoP} \subseteq \mathsf{FauxP} \subseteq \mathsf{FBPP}$.

In query complexity, for promise problems, all these inclusions are strict. The strictness of the first inclusion follows from the fact that randomized and deterministic algorithms can be separated for partial functions: a partial function can be interpreted as a search problem with one valid output for each input in the promise, and in that setting randomized algorithms are always pseudodeterministic.

However, the story changes for *verifiable* search problems: it actually holds that each randomized algorithm can be converted to a faux deterministic algorithm in a nontrivial way; this is Theorem 5. Therefore, the classes FauxP and FBPP (and also FZPP) collapse in the black-box model when restricted to verifiable search problems.

1.2 Faux derandomization

Our main contribution, other than introducing the notion of faux determinism, is the following "faux derandomization" theorem.

▶ **Theorem 5** (Faux derandomization). *In the query (black box) model, every search problem in* $\mathsf{FNP} \cap \mathsf{FBPP}$ *has a faux-deterministic algorithm, even in the setting of promise problems. In other words, if a search problem can be both verified efficiently and solved by a randomized algorithm, the randomized algorithm can be made faux-deterministic.*

More specifically, if the search problem has a verifier which uses v queries and a randomized algorithm which uses k queries, then for any t it has a faux-deterministic algorithm which uses $O((v+k)t)$ queries and cannot be refuted by an adversary making fewer than $2^{\Omega(\sqrt{t})}$ queries except with probability at most $2^{-\Omega(\sqrt{t})}$.

Moreover, this holds even with respect to a quantum adversary that makes quantum queries to the faux deterministic algorithm.

The statement of the theorem in its full generality can be found in Theorem 22. We highlight three ways to think about this theorem.

Computationally secure derandomization

Our theorem says that for verifiable search problems, any randomized algorithm can be "derandomized" in a computationally secure way: this is not true derandomization, but no polynomially-bounded adversary making black-box queries can distinguish it from true derandomization. More explicitly, starting with any randomized algorithm B, we can transform it to another randomized algorithm B' such that when the randomness r is fixed, B'_r solves the problem, even with *any subexponential number of (possibly adversarial) repeated uses*. The new algorithm has a "set it and forget it" random seed r; once it is fixed, even a computationally unbounded adversary which makes subexponentially many queries to the algorithm cannot find an instance on which the now-deterministic algorithm B'_r fails.

Since this "faux derandomized" algorithm is indistinguishable from a deterministic algorithm to computationally bounded adversaries, it can replace deterministic or pseudodeterministic protocols in cryptographic settings in which the latter are employed.[3]

[3] It should be noted, however, that there are no compelling cryptographic use cases for pseudodeterminism to date.

(To be sure, our algorithm still needs a random seed, so it may not look like derandomization; however, once the random seed is fixed, we get security against exponentially many adversarial inputs, giving us in some sense "exponentially long stretch" on the random seed length.)

Refutation paradigm

The refutation problems are studied in metacomplexity; see [4] and the references therein. Roughly speaking, if a task requires some amount of resources, the refutation task is to take as input an algorithm which uses fewer resources and demonstrate that it fails. Our result can be viewed as saying that in query complexity, the task of refuting pseudodeterministic algorithms for verifiable search problems is hard. Indeed, our faux-deterministic algorithms are just probability distributions over algorithms that pretend to be pseudodeterministic, but are not; yet when an algorithm is sampled from this distribution, it is hard to find an input on which it fails to be pseudodeterministic.

Nonconstructivity in pseudodeterministic lower bounds

Another interpretation of Theorem 5 is an explanation for the difficulty of proving pseudodeterministic lower bounds. The latter are notoriously difficult, even in the black-box setting: for instance, the pseudodeterministic complexity of the "find 1" problem from the proof of Lemma 2 is still open. Recall that in this task, the input is an n-bit string of Hamming weight at least $n/2$ and the goal is to output the position of a 1; this can be done in $O(1)$ randomized queries, requires $\Omega(n)$ deterministic queries, and its pseudodeterministic query complexity is only known to be between $\Omega(\sqrt{n})$ and $O(n)$, despite substantial attention from the community [11, 3]. What we show here is that the FIND1 problem has an $O(\text{polylog}\, n)$ faux-deterministic algorithm. This means that any lower bound must be nonconstructive: it must prove, for each algorithm B, the existence of an input string x on which it fails, but without outlining an efficient procedure for finding that string x (given oracle access to B). In other words, some natural lower bound techniques against pseudodeterminism also lower bound faux determinism, but since efficient faux-deterministic algorithms always exist (for $\mathsf{FBPP} \cap \mathsf{FNP}$), such techniques cannot rule out efficient pseudodeterministic algorithms.

One way to formalize the difficulty of proving a lower bound for FIND1 is via the following lemma.

▶ **Lemma 6.** *Consider the following search problem. The input is query access to a function f which solves the* FIND1 *problem: it maps strings of length n and Hamming weight at least $n/2$ to $[n]$, with the property that $x_{f(x)} = 1$ for every such x. The output is some input x of Hamming weight at least $n/2$ such that the block sensitivity of f at x is $\omega(\text{polylog}(n))$.*

Even though [11] show that every function solving FIND1 *has block sensitivity at least $\Omega(\sqrt{n})$ on some input, this search problem requires strictly more than $\text{poly}(n)$ queries to solve.*

Lemma 6 follows fairly straightforwardly from Theorem 5, as detailed in Section 5. But this consequence is perhaps surprising: combined with the existence of highly sensitive inputs ([11]), it gives a "meta" TFNP search problem which cannot be efficiently solved even with much weaker parameters.

Finally, we show that the parameters in Theorem 5 are quadratically tight (see Section 3.4).

▶ **Lemma 7.** *For any faux-deterministic algorithm for* FIND1 *which uses t queries, there exists an adversary strategy which uses 2^{t+1} queries and succeeds in refuting the faux-deterministic algorithm with probability at least $1/2$.*

1.3 Our techniques

We describe our approach for proving Theorem 5.

Reduction to FIND1

First, we use the existence of an FBPP algorithm B to reduce this task to the FIND1 problem. That is to say, on any input x, we can view the search problem as the task of finding a random seed r such that the output $B_r(x)$ passes verification by the verifier V; in other words, we want to find a random seed that causes the algorithm B to succeed at solving x, and we want to find this seed in a way that looks deterministic. We know that most seeds succeed (since B is a valid randomized algorithm), and we can check a seed (using the verifier for the search problem), so this amounts to a special case of the FIND1 problem in which we are given a black box which is known to accept most inputs r, and our goal is to "find a 1 in the string", that is, find a specific input r which is accepted. This reduction is somewhat standard for pseudodeterministic algorithms; see [11].

Attempts to faux-derandomize FIND1

Here is an example approach which does not work for designing a faux-deterministic algorithm for FIND1. We could try having B_r query a fixed (randomly chosen) subset S of $\log n$ bits of the input, and have it output the first index i for which $x_i = 1$. (This failed strategy also appeared earlier in Section 1.1; see Lemma 2.) For most x, there will be a 1 within those $\log n$ bits, so this algorithm will succeed. However, an adversary can proceed as follows: first it runs $B_r(x^1)$ on an arbitrary input x^1 to get an index $i_1 \in S$ with $x^1_{i_1} = 1$. Next, it picks an input x^2 close in Hamming distance to x^1 with $x^2_{i_1} = 0$, and runs $B_r(x^2)$ to get a different $i_2 \in S$. It repeats this, each time setting the previously chosen position of the input to 0. After around $|S|$ iterations, it will find an input on which B_r fails to find a 1.

The failure of the above non-adaptive approach to the design of B may suggest that adaptivity is important. We could try having B_r query adaptively (e.g. first query position 3, then based on its value, choose whether to query position 5 or 2, and then fork on the answer again, with paths growing exponentially and the positions being fixed in B_r but chosen randomly when r is random). The algorithm B_r can then pick the last 1 it saw down this path (alternatively, we could try the first 1 it saw). Unfortunately, this strategy also fails: the adversary can detect bits that are high up in the path (i.e. queried early), and set them to 0 successively, until the whole path is 0. This is easy for the adversary to do if $B_r(x)$ returns the first 1 it saw (it can just set that bit to 0 in the next input). If $B_r(x)$ returns the last 1 it saw, the adversary can detect indices i along the path taken by $B_r(x)$ by observing that flipping those bits changes the output of $B_r(x)$. It can binary search to find the bits along the path taken by $B_r(x)$, and just set them all to 0 in the next input. Repeating this process leads to inputs with more and more leading 0s in their path, until after around $\log^2 n$ queries the adversary can force B_r to fail.

A successful design

Our faux-deterministic algorithm works similarly to the fully-adaptive tree in the previous paragraph, but instead of forking on each bit we query, we fork on an OR of a batch of $\log n$ queried bits. In other words, the decision tree looks like a tree (of depth $\log n$) of ORs (of width $\log n$). We return the last 1 seen. The OR blocks are randomized when r is random.

This works because the OR function has very low sensitivity: on most inputs, the OR blocks will all evaluate to 1, and the path taken down the tree will simply be rightmost path. The adversary will be unable to find any input x on which the path of $B_r(x)$ is not the rightmost one until it has made some queries to B_r. Even then, it will only know bits in the rightmost block, and will be unable to find inputs x on which the path of $B_r(x)$ is not either the rightmost path, or the rightmost path with a left turn only at the very end. Indeed, after T queries to B_r, we show that the adversary can only find inputs x for which the path of $B_r(x)$ leads to the rightmost T leaves of the tree. To get the algorithm B_r to fail, however, requires all the OR blocks to evaluate to 0 (so that B_r never finds any 1 in the input at all); this means finding an input x for which the path of $B_r(x)$ is the leftmost path (consisting of all 0s for the OR blocks). Since there are 2^d leaves for depth d, the adversary requires exponentially many queries to B_r (where "exponential" is relative to the efficiency parameter $\log n$).

The main innovation here was the "tree of ORs" design: this lets the algorithm avoid being local (there are $\exp(d)$ many indices it might query for some input, where d is the number of queries it makes) while also letting it avoid having its path be sensitive to bit flips.

Quantum lower bound

Our faux-deterministic algorithm also works when the adversary is quantum, and makes superposition queries to B_r. Our proof of the quantum lower bound is based on a hybrid argument [2]; if Q is the adversary's quantum algorithm, we set $Q_0 = Q$ and make a sequence $Q_1, Q_2, \ldots$ where Q_1 is the quantum algorithm with the first query replaced by a "fake" quantum query, Q_2 is the quantum algorithm with the first two queries replaced by fakes, and so on, until all queries are replaced by fakes. We show that the final states (before measurement) are close between Q_t and Q_{t+1}, meaning that the fake initial queries are sufficiently accurate. This technique lets us show inductively that the first query of the real quantum algorithm must place extremely small amplitude on the paths of the tree that are not the rightmost path; the second query must place extremely small amplitude on the paths of the tree that are not the rightmost or second-rightmost paths; and so on.

2 Preliminaries

2.1 Notation

General notation

For $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, ..., n\}$. By $\mathrm{poly}(n)$ and $\mathrm{polylog}(n)$, we denote the family of all functions growing at most polynomially and poly-logarithmically, that is $f(n) = O(n^c)$ and $f(n) = O(\log^c n)$ for some constant c, respectively. We also define $\mathrm{negl}(n)$ to denote the family of negligible functions; that is, $f(n) = \mathrm{negl}(n)$ if for every constant $c > 0$, we have $f(n) = O(n^{-c})$.

For a binary string $x \in \{0,1\}^n$, we use $|x| = |\{i : x_i = 1\}|$ to denote its Hamming weight, and for a set $B \subset [n]$ we use x^B to denote the string obtained from x by flipping precisely the bits in B. For a single index $j \in [n]$, we write x^j to denote $x^{\{j\}}$.

For a distribution $\mathcal{D}$, we use the notation $x \sim \mathcal{D}$ to indicate that x is sampled according to $\mathcal{D}$, and use $x \in \mathcal{D}$ to denote that x lies in the support of $\mathcal{D}$. Let X be a random variable and let $\hat{X}$ be a fixed value in its range. With a slight abuse of notation, we may write $\hat{X}$ to denote the event $\{X = \hat{X}\}$ whenever X is clear from the context. The same convention applies to $\Pr[\cdot|\hat{X}]$ and $\mathbb{E}[\cdot|\hat{X}]$.

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 standard 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 entry x. We denote a *multi-register* system as $X = X_1 \dots X_k$, where $\mathcal{H}_X = \mathcal{H}_{X_1} \otimes \dots \otimes \mathcal{H}_{X_k}$. 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; that is, $\| U |\phi\rangle_X \|_2 = \| |\phi\rangle_X \|_2$ for all $|\phi\rangle_X$. With a slight 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 $|\alpha_x|^2$. 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 orthogonal projection matrices, meaning that $\Pi_i = \Pi_i^\dagger = \Pi_i^2$ and $\sum_{i=1}^k \Pi_i = I$. The outcome $i \in [k]$ of measurement $\mathcal{M}$ on state $|\phi\rangle$ occurs with probability $\| \Pi_i |\phi\rangle \|_2^2$, with the resulting state being $\frac{\Pi_i |\phi\rangle}{\| \Pi_i |\phi\rangle \|_2}$.

2.2 Search problems and query complexity measures

Let $\Sigma = (\Sigma_n)$ be a family of finite and non-empty sets. A *search problem* is a relation $S \subset \bigcup_{n\in\mathbb{N}} (\{0,1\}^n \times \Sigma_n)$. For $n \in \mathbb{N}$, we use $S_n := S \cap (\{0,1\}^n \times \Sigma_n)$ to denote the restriction of the problem to inputs of length n. For $x \in \{0,1\}^n$, the set of corresponding *certificates* (valid outputs) is $S_n(x) := \{y \in \Sigma_n : (x,y) \in S_n\}$; when n is clear from context we write $S(x)$ for $S_n(x)$. The *domain* and *range* of S are defined as $\mathrm{dom}(S) := \{x \in \{0,1\}^* : \exists y, (x,y) \in S\}$ and $\mathrm{ran}(S) := \{y : \exists x, (x,y) \in S\}$, respectively.

The problem S is called *total* if $\mathrm{dom}(S) = \{0,1\}^*$. Moreover, we say S is *Boolean* if $\mathrm{ran}(S) \subseteq \{0,1\}$, and *single-valued* if $|S(x)| = 1$ for every $x \in \mathrm{dom}(S)$. A Boolean single-valued problem is also called a *decision problem*.

Deterministic query complexity

A deterministic decision tree $T = (T_n)$ is a sequence of rooted binary trees T_n for input length $n \in \mathbb{N}$ where each internal node is labeled with an index $i \in [n]$, and has a right and a left child, each labeled as one of the possible values in $\{0,1\}$. Further, each leaf of T_n is labeled with a potential output in Σ_n.

For input $x \in \{0,1\}^n$, the value $T[x] := T_n[x]$ is computed as follows. Beginning with the root, if the current node is a leaf, output its label. Otherwise, the current node is labeled by some index $i \in [n]$. Query x_i and continue the computation from the corresponding child until a leaf is reached. The (deterministic) query complexity of T, denoted as $\mathsf{DEPTH}(T_n)$, is defined as its depth which is the maximum number of queries made on the worst-case input.

The *deterministic query complexity* of a search problem S, denoted by $D(S) = D(S_n)$, is the minimum depth of a decision tree T computing S, i.e., satisfying $T[x] \in S(x)$ for all $x \in \mathrm{dom}(S)$. We define the class FP to be the set of all search problems S such that $D(S_n) = \mathrm{polylog}(n)$.

Nondeterministic query complexity

A *verifier* V for a search problem S is a family of deterministic decision trees $V = (V_{n,y})$, where for each input length n and output $y \in \Sigma_n$, the tree $V_{n,y}$ outputs $V_{n,y}[x] = 1$ if $y \in S(x)$ and $V_{n,y}[x] = 0$ otherwise. We define the *nondeterministic query complexity* of V, denoted by $\mathsf{DEPTH}(V) = \mathsf{DEPTH}(V_n)$ as $\max_{y \in \Sigma_n} \mathsf{DEPTH}(V_{n,y})$.

The *nondeterministic query complexity* of a search problem, denoted by $N(S)$, is defined as the minimum depth of a verifier V for S. We define the class FNP to be the set of all search problems S such that $N(S_n) = \mathrm{polylog}(n)$.

Randomized and pseudodeterministic query complexity

A randomized decision tree $\mathcal{T} = (\mathcal{T}_n)$ is a sequence where each $\mathcal{T}_n$ is a probability distribution over deterministic decision trees of input length n. The query complexity of $\mathcal{T}$, denoted by $\mathsf{DEPTH}(\mathcal{T})$ is defined as the maximum depth over all decision trees T in the support of $\mathcal{T}_n$.

For an error parameter $\epsilon = \epsilon(n) \in (0, 1)$, the *randomized query complexity* of a search problem S, denoted by $R_\epsilon(S)$ is defined as the minimum depth of a randomized decision tree $\mathcal{T}$ such that $\Pr[\mathcal{T}_n[x] \notin S(x)] \leq \epsilon$ for all $x \in \mathrm{dom}(S_n)$. Whenever ϵ is omitted, we assume $\epsilon = 1/3$. We define the class FBPP to be the set of all search problems S such that $R(S_n) = \mathrm{polylog}(n)$.

Additionally, the *pseudodeterministic query complexity* of a search problem S, denoted by $D^{\mathrm{pseudo}}_\epsilon(S)$, is defined as the minimum depth of a randomized decision tree $\mathcal{T}$ such that for all $x \in \mathrm{dom}(S_n)$ there exists a canonical output $f(x) \in S(x)$ where $\Pr[\mathcal{T}_n[x] \neq f(x)] \leq \epsilon$. We define the class $\mathsf{PseudoP}$ to be the set of all search problems S such that $D^{\mathrm{pseudo}}_{1/3}(S_n) = \mathrm{polylog}(n)$.

Quantum query complexity

To define the quantum version of decision trees, we switch to the standard circuit model with query access. For a string $x \in \{0,1\}^n$, we define the *oracle unitary* O^x on *index* and *query* registers I and Q as the transformation $O^x |i\rangle_I |b\rangle_Q := |i\rangle_I |x_i \oplus b\rangle_Q$ for $i \in [n]$ and $b \in \{0,1\}$. A *quantum decision tree* $\mathcal{Q} = (\mathcal{Q}_n)$ is a sequence where each $\mathcal{Q}_n$ consists of unitaries $(U_0, \ldots, U_d)$ acting on index register I, query register Q, as well as an arbitrarily large work register W. We define the query complexity of $\mathcal{Q}$, denoted by $\mathsf{DEPTH}(\mathcal{Q}_n) := d(n)$. For an input $x \in \{0,1\}^n$, the execution of the algorithm $\mathcal{Q}[x]$ is defined as the measurement outcome of the work register (identifying a fixed set of computational-basis states with the elements of Σ_n) in the final state $|\psi\rangle = U_d O^x U_{d-1} O^x \ldots O^x U_0 |0\rangle_I |0\rangle_Q |0\rangle_W$.

Similar to the randomized case, the *quantum query complexity* of a search problem S, denoted by $Q_\epsilon(S)$ is defined as the minimum depth $d(n)$ of a quantum algorithm satisfying $\Pr[\mathcal{Q}[x] \notin S(x)] \leq \epsilon$ for all $x \in \mathrm{dom}(S)$. The class FBQP is defined as the set of search problems having query complexity $Q_{1/3}(S) = \mathrm{polylog}(n)$.

Block sensitivity

We recall the definition of block sensitivity for completeness. Let $f : \mathrm{dom}(f) \to \Sigma$ be a possibly partial function with $\mathrm{dom}(f) \subseteq \{0,1\}^*$, and let $x \in \mathrm{dom}(f) \cap \{0,1\}^n$. The *block sensitivity* of f on input x, denoted by $\mathrm{bs}(f,x)$, is defined as the maximum number b of disjoint blocks $B_1, \ldots, B_b \subset [n]$ of indices such that $x^{B_j} \in \mathrm{dom}(f)$ and $f(x) \neq f(x^{B_j})$ for all $j \in [b]$. The block sensitivity of f is defined as $\mathrm{bs}(f) = \max_{x \in \mathrm{dom}(f)} \mathrm{bs}(f,x)$.

2.3 Formal definition of faux determinism

In this section, we formalize the informal definition of faux determinism given in Section 1 using our query complexity notation.

For a search problem S and a decision tree T purporting to solve S on inputs of length n, we say $x \in \mathrm{dom}(S_n)$ is an *adversarial input* for T whenever $T[x] \notin S_n(x)$. We define the classical version of faux determinism as follows.

▶ **Definition 8** (faux determinism in query model)**.** *Let S be a search problem, $q = q(n) \in \mathbb{N}$ be a query upper bound, $\epsilon = \epsilon(n) \in [0,1)$ be an error parameter. We say that a distribution $\mathcal{T} = (\mathcal{T}_n)$ over decision trees is a* (q,ϵ)-faux-deterministic algorithm *for S if, for all $n \in \mathbb{N}$ and any deterministic*[4] *adversary A making fewer than $q(n)$ black-box queries of the form $x \in \mathrm{dom}(S_n)$ to a random decision tree $T \sim \mathcal{T}_n$, the probability that A finds an adversarial input is at most $\epsilon(n)$; i.e.,*

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

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:

$$\mathsf{DEPTH}(\mathcal{T}) := \max_{T \in \mathcal{T}} \mathsf{DEPTH}(T).$$

Next, we generalize the notion of faux-deterministic algorithms for quantum adversaries that are given query access to the oracle unitary

$$O^T |x\rangle_I |z\rangle_Z = \begin{cases} |x\rangle_I |T[x] \oplus z\rangle_Z & x \in \mathrm{dom}(S), \\ |x\rangle_I |z\rangle_Z & \text{otherwise}, \end{cases}$$

where we assume a fixed binary encoding of the output alphabet Σ_n, so that $T[x] \oplus z$ is well-defined.

▶ **Definition 9.** *We say that the distribution $\mathcal{T} = (\mathcal{T}_n)$ is a* quantum-secure (q,ϵ) faux-deterministic *algorithm for S whenever any quantum adversary $\mathcal{Q}$ making fewer than $q(n)$ many black-box queries to the oracle unitary O^T finds an adversarial input with probability at most $\epsilon(n)$; that is,*

$$\Pr[T \sim \mathcal{T}, x \leftarrow \mathcal{Q}[T]; x \in \mathrm{dom}(S_n) \text{ and } T[x] \notin S(x)] \leq \epsilon(n).$$

[4] In the above definition, the existence of deterministic and randomized adversaries are equivalent; i.e., if a randomized adversary succeeds with probability $\epsilon(n)$, we can choose the best deterministic adversary for each input length n to obtain a deterministic adversary that succeeds with probability at least $\epsilon(n)$ as well.

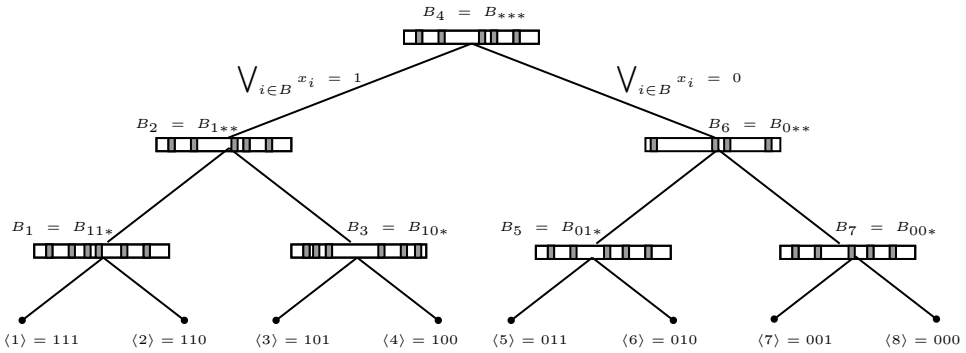

Figure 1 The Tree of ORs construction with depth $d = 3$. Extending the order $1 \prec * \prec 0$ induces a linear order on the leaves $\langle 1 \rangle, \ldots, \langle 8 \rangle$ and blocks $B_1, \ldots, B_7$..

We define FauxP as the class of search problems S for which there exists a distribution $\mathcal{T}$ over decision trees of depth polylog(n) such that, for every polynomial p, $\mathcal{T}$ is $(p(n), \text{negl}(n))$-faux-deterministic. The class QFauxP is defined analogously, with security against quantum adversaries.

As mentioned in Section 1.3, our main technical step is designing a faux-deterministic algorithm for a FBPP $\cap$ FNP-complete problem FIND1 defined as follows.

▶ **Definition 10** (FIND1). *Let $n \in \mathbb{N}$. We define* FIND1_n *to be the following search problem: given an input $x \in \{0,1\}^n$ with Hamming weight $|x| \geq n/2$, find an index i such that $x_i = 1$; i.e.,* $\text{FIND1}_n := \{(x, i) \in \{0,1\}^n \times [n] |\ |x| \geq \frac{n}{2} \text{ and } x_i = 1\}$.

3 Faux-deterministic algorithm for FIND1

In this section, we present our faux-deterministic algorithm for the FIND1 problem. More specifically, in Section 3.1 we formally introduce the *Tree of ORs* algorithm and establish a few of its structural properties required for our analysis. We then analyze the algorithm against deterministic adversaries in Section 3.2 and against quantum algorithms in Section 3.3.

3.1 Tree of ORs construction

As previously discussed in Section 1.3, the construction of the Tree of ORs algorithm will be based on a complete binary tree T of depth d. For the rest of this section, we identify each vertex with a string $z \in \{0,1\}^r *^{d-r}$ for some $0 \leq r \leq d$: the root is $*^d$ $(r = 0)$, each internal vertex has the form $b_1 \cdots b_r *^{d-r}$ with $0 \leq r < d$, and the leaves are exactly $\{0,1\}^d$ $(r = d)$. For an internal vertex $b_1 \cdots b_r *^{d-r}$, its left and right children are $b_1 \cdots b_r 1 *^{d-r-1}$ and $b_1 \cdots b_r 0 *^{d-r-1}$, respectively.

▶ **Definition 11** (Tree of ORs). *Let $n, w = w(n), d = d(n) \in \mathbb{N}$, and let T be a binary tree of depth d. For each internal node $z = b_1 \ldots b_k *^{d-k}$ of T with $0 \leq k < d$, we assign an independent and uniformly random block of indices $B_z \subset [n]$ of size $|B_z| = w$.*

Then, on input $x \in \{0,1\}^n$, the algorithm begins at root node $z = *^d$ and proceeds recursively as follows: on an internal node $z = b_1 \ldots b_k *^{d-k}$ with $k < d$: if any of the indices $i \in B_z$ is a 1-index $(x_i = 1)$, the evaluation of the node is defined to be $B_z(x) := 1$, and we move to the left child $b_1 \ldots b_k 1 *^{d-k-1}$. Otherwise, the logical OR $(\vee)$ of indices in B_z is defined to be $B_z(x) := 0$ and we move to the right child $b_1 \ldots b_k 0 *^{d-k-1}$.

Eventually, the algorithm arrives at a leaf node $\ell \in \{0,1\}^d$. If this is the right-most leaf 0^d, no 1-indices are found and we err. Otherwise, there exists a unique $k \in \{0,\dots,d-1\}$ such that $\ell = b_1 \dots b_k 10^{d-k-1}$. As a result, the last node where a 1-index was found is an internal node $z^* = b_1 \dots b_k *^{d-k}$, and we output the smallest index $i \in B_{z^*}$ with $x_i = 1$.

For each input length n, our distribution $\mathcal{T}_n$ is a uniform distribution over all possible such trees with depth $d(n)$ and width $w(n)$. We may drop n whenever it is obvious from the context.

3.1.1 Linear ordering

To help analyze the algorithm, we extend the ordering $1 \preceq * \preceq 0$ to a linear order on all internal and leaf labels $z \in \{0,1\}^r *^{d-r}$ with $0 \le r \le d$ by lexicographic order; i.e., $1^d \prec 1^{d-1}* \prec 1^{d-1}0 \prec \cdots \prec 0^{d-1}1 \prec 0^{d-1}* \prec 0^d$. This ordering induces a natural left-to-right order on the leaves of the tree. For $\ell \in [2^d]$ let $\langle \ell \rangle$ denote the ℓ-th leaf in this order. Also, for $\ell \in [2^d - 1]$, we use B_ℓ to denote the ℓ-th block in this order (see Figure 1).

Intuitively, when reaching leaf $\langle \ell \rangle$ the output of the algorithm will be from the block B_ℓ. The following proposition formalizes this correspondence.

▶ **Proposition 12.** *Let $\ell \in [2^d - 1]$, and for label z, let z^+ and z^- denote the successor and predecessor in the linear order defined above, whenever it exists.*

- **i.** *between any two consecutive leaves $\langle \ell \rangle$ and $\langle \ell + 1 \rangle$ there exists a unique internal label $z = \langle \ell \rangle^+ = \langle \ell + 1 \rangle^-$. Conversely, each internal node $z \in \{0,1\}^r *^{d-r}$ is between two consecutive leaves $z^+ \succ z \succ z^-$.*
- **ii.** *the ℓ-th block in this order, denoted by B_ℓ, is exactly the block corresponding to label $\langle \ell \rangle^+$; i.e. $B_\ell = B_{\langle \ell \rangle^+}$.*
- **iii.** *a tree $T \in \mathcal{T}$ fails to output a 1-index if and only if it reaches the right-most leaf $\langle 2^d \rangle$. Moreover, for $\ell < 2^d$ the tree outputs a 1-index from the block B_ℓ if and only if it reaches the leaf $\langle \ell \rangle$.*

Proof. For (i), let $\langle \ell \rangle = b_1 \dots b_i 10^{d-i-1}$ for some $i \in \{0,\dots,d-1\}$. Then, by definition of the linear order, $\langle \ell + 1 \rangle = b_1 \dots b_i 01^{d-i-1}$. So, the only possible string in between is $z = b_1 \dots b_i *^{d-i}$, which is exactly equal to $z = \langle \ell + 1 \rangle^- = \langle \ell \rangle^+$. Similarly, each internal node $z = b_1 \dots b_i *^{d-i}$ is between two consecutive leaves $z^- = b_1 \dots b_i 10^{d-i-1}$ and $z^+ = b_1 \dots b_i 01^{d-i-1}$. Part (ii) is a direct consequence of (i). For (iii), recall that the algorithm branches left if and only if a 1-index is found. So, for a leaf $\langle \ell \rangle = b_1 \dots b_i 10^{d-i-1}$, the last block on which a 1-index is found is exactly $\langle \ell \rangle^+ = b_1 \dots b_i *^{d-i}$. ◀

3.1.2 Hybrid Lemma

We now prove the central property of the algorithm for security against adversaries. For $x \in \{0,1\}^n$ where $|x| \ge n/2$ and tree $T \in \mathcal{T}$, let $\ell_T(x)$ denote the leaf reached when executing T on x. Then, given a tree $T \sim \mathcal{T}$, the main goal of an adversary is to find such an input x that reaches the right-most leaf $\ell_T(x) = \langle 2^d \rangle$.

Assume that the adversary has already found the first $\ell - 1$ blocks $B_1 = \hat{B}_1, \dots, B_{\ell-1} = \hat{B}_{\ell-1}$. As a result, the adversary can choose a string x with $x_i = 0$ for every $i \in \bigcup_{j=1}^{\ell-1} \hat{B}_j$, which by Proposition 12 implies that $\ell_T(x) \succeq \langle \ell \rangle$. In the following lemma, we argue that if the adversary has no information about the remaining blocks $B_{\ell+1}, \dots, B_{2^d-1}$, then any such input x reaches exactly the next leaf $\ell_T(x) = \langle \ell \rangle$ with high probability over the choice of T; i.e., the adversary cannot "skip" any of the blocks.

▶ **Lemma 13.** *Let $\mathcal{T}$ be the distribution given in Definition 11 on inputs of length n with width w and depth d. Then, for any input $x \in \{0,1\}^n$ with $|x| \geq n/2$, for $\ell \in [2^d]$, and for any sequences of subsets $\hat{B}_1, \ldots, \hat{B}_{\ell-1} \subset [n]$ of size w, we have*

$$\Pr_{T\sim\mathcal{T}_n}\left[\ell_T(x) \preceq \langle\ell\rangle \middle| \hat{B}_1, \ldots, \hat{B}_{\ell-1}\right] \geq (1-\frac{1}{2^w})^d.$$

To prove the lemma, we first provide a necessary condition for $\ell_T(x) \preceq \langle\ell\rangle$, and then prove that this condition is satisfied.

For $\ell \in [2^d-1]$, and $0 \leq r < d$, we call $y \in \{0,1\}^r *^{d-r}$ a *1-prefix* of $\langle\ell\rangle$ whenever there exists $\hat{y}, \hat{z} \in \{0,1\}^*$ such that $\langle\ell\rangle = \hat{y}1\hat{z}$ and $y = \hat{y}*^{d-|\hat{y}|}$; that is, internal node y is in the path to leaf $\langle\ell\rangle$ with evaluation of B_y being 1.

▷ Claim 14. Assume that for every 1-prefix y of $\langle\ell\rangle$ we have $B_y(x) = 1$. Then $\ell_T(x) \preceq \langle\ell\rangle$.

Proof. For the sake of contradiction, suppose $\ell_T(x) \succ \langle\ell\rangle$, and let $\hat{y}$ be the largest common prefix of $\langle\ell\rangle$ and $\ell_T(x)$ with length m. Since $\ell_T(x) \neq \langle\ell\rangle$, we know that $\langle\ell\rangle$ and $\ell_T(x)$ differ in at least one index. Additionally, we have $\ell_T(x) \succ \langle\ell\rangle$. Therefore, we can write $\langle\ell\rangle$ and $\ell_T(x)$ as $\langle\ell\rangle = \hat{y}1\hat{z}$ and $\ell_T(x) = \hat{y}0\hat{z}'$. Now consider the 1-prefix $y = \hat{y}*^{d-m}$ of $\langle\ell\rangle$; since $\hat{y}$ is also a prefix of $\ell_T(x)$, the execution of tree T on x must have visited the internal block B_y. Also, since $\ell_T(x)$ extends $\hat{y}$ by a 0 symbol, the subset B_y must not have found a 1-index in the input x. Therefore, $B_y(x) = 0$, which is a contradiction to the assumption of the claim. ◁

Proof of Lemma 13. We prove that with high probability for any $\langle\ell\rangle$ the condition of the Claim 14 holds. Let y be an arbitrary 1-prefix of $\langle\ell\rangle$ where $\langle\ell\rangle = \hat{y}1\hat{z}$ and $y = \hat{y}*^{d-|\hat{y}|}$. By the defined linear order, we have $y \succ \langle\ell\rangle$ as $* \succ 1$. Therefore, by the natural ordering of the leaves and Proposition 12 we have $y \succ \langle\ell\rangle \succ \langle\ell\rangle^- = \langle\ell-1\rangle^+ \succ \langle\ell-2\rangle^+ \succ \cdots \succ \langle 1\rangle^+$. So, the block variable B_y is not among the variables $B_1, \ldots, B_{\ell-1}$, and the set B_y is sampled independent of the first $\ell-1$ sets.

Let $y_1, \ldots, y_r$ be all the 1-prefixes of $\langle\ell\rangle$. Note that since $\langle\ell\rangle \in \{0,1\}^d$ we have $r \leq d$. Consequently,

$$\begin{aligned}
\Pr_{T\sim\mathcal{T}_n}\left[\ell_T(x) \preceq \langle\ell\rangle \middle| \hat{B}_1, \ldots, \hat{B}_{\ell-1}\right] &\geq \Pr_{T\sim\mathcal{T}_n}\left[B_{y_1}(x)=1, \ldots, B_{y_r}(x)=1 \middle| \hat{B}_1, \ldots, \hat{B}_{\ell-1}\right] \\
&= \Pr_{T\sim\mathcal{T}_n}\left[B_{y_1}(x)=1, \ldots, B_{y_r}(x)=1\right] \\
&= \Pr_{T\sim\mathcal{T}_n}\left[B_{y_1}(x)=1\right] \ldots \Pr_{T\sim\mathcal{T}_n}\left[B_{y_r}(x)=1\right] \\
&\geq (1-\frac{1}{2^w})^r \\
&\geq (1-\frac{1}{2^w})^d,
\end{aligned}$$

where the first line is by Claim 14, second and third lines are by independence of blocks, and the fourth line is due to $|x| \geq n/2$. ◀

3.2 Security against classical adversaries

In this section, we will prove the security of the Tree of ORs construction against classical adversaries.

▶ **Theorem 15.** *Let $\mathcal{T} = (\mathcal{T}_n)$ be the distribution in Definition 11 with width and depth parameters $w = w(n)$ and $d = d(n)$. Then, $\mathcal{T}_n$ is a $(2^d, d \cdot 2^{d-w})$-faux-deterministic algorithm of complexity $w \cdot d$ for the* FIND1 *problem.*

Proof. Clearly, the tree has query complexity $w \cdot d$. So, let A be an arbitrary deterministic adversary that has query access to a uniformly randomly chosen tree $T \sim \mathcal{T}_n$, 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 the oracle provides additional information to the adversary as well. More specifically, when querying $x_{(k)}$ with $\ell_T(x_{(k)}) = \langle j \rangle$ the oracle returns

- all of the blocks $B_1, \ldots, B_j$,
- and all of the first k blocks $B_1, \ldots, B_k$.

We can now lower bound the probability p that the algorithm does not err on $x_{(q)}$, i.e., $\ell_T(x_{(q)}) \neq 0^d$, as follows.

$$\begin{aligned}
p &:= \Pr_{T \sim \mathcal{T}_n} \left[\ell_T(x_{(q)}) \neq 0^d\right] \\
&\geq \Pr_{T \sim \mathcal{T}_n} \left[\ell_T(x_{(1)}) \preceq \langle 1 \rangle, \ldots, \ell_T(x_{(q)}) \preceq \langle q \rangle\right] \\
&= \prod_{\ell=1}^{q} \Pr_{T \sim \mathcal{T}_n} \left[\ell_T(x_{(\ell)}) \preceq \langle \ell \rangle \middle| \ell_T(x_{(1)}) \preceq \langle 1 \rangle, \ldots, \ell_T(x_{(\ell-1)}) \preceq \langle \ell - 1 \rangle\right] \\
&= \prod_{\ell=1}^{q} p_\ell,
\end{aligned}$$

where p_ℓ is the probability that the ℓ-th query reaches at most the ℓ-th leaf conditioned on the assumption that all of the previous $\ell - 1$ queries have resulted in a leaf before $\langle \ell \rangle$. Therefore, each of these queries has revealed at most the first $\ell - 1$ blocks $B_1 = \hat{B}_1, \ldots, B_{\ell-1} = \hat{B}_{\ell-1}$. Additionally, we assumed that after $\ell - 1$ queries, the oracle returns all of the first $\ell - 1$ blocks. Hence, $x_{(\ell)}$ depends only on the first ℓ blocks and we can write

$$p_\ell = \Pr_{T \sim \mathcal{T}_n} \left[\ell_T(x_{(\ell)}) \preceq \langle \ell \rangle \middle| \hat{B}_1, \ldots, \hat{B}_{\ell-1}\right] \geq (1 - \frac{1}{2^w})^d,$$

where we have used the Lemma 13 in the last inequality for $\ell \in [q]$. Therefore,

$$p = \prod_{\ell=1}^{q} p_\ell \geq \prod_{\ell=1}^{q} (1 - \frac{1}{2^w})^d \geq (1 - \frac{1}{2^w})^{dq} \geq 1 - \frac{dq}{2^w} \geq 1 - \frac{d2^d}{2^w}.$$ ◀

▶ **Corollary 16.** *Let $t = t(n) \in \mathbb{N}$ be an increasing function. 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 $\epsilon > 0$ and $t = \log^{2+\epsilon}(n)$, we have a $(\text{quasi-poly}(n), \text{negl}(n))$-faux-deterministic algorithm of depth $\text{polylog}(n)$.*

Proof. Letting $w = \frac{3}{4}\sqrt{t}$ and $d = \frac{1}{4}\sqrt{t}$ we obtain a tree with total depth $\frac{3}{16}t < t$. By Theorem 15, 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 $2^{\frac{\sqrt{\log^{2+\epsilon}(n)}}{4}} = \text{quasi-poly}(n)$ and $2^{-\frac{\sqrt{\log^{2+\epsilon}(n)}}{4}} = \text{negl}(n)$. ◀

3.3 Security against quantum adversaries

In this section, we will prove the security of the Tree of ORs construction against quantum adversaries.

▶ **Theorem 17.** *Let $\mathcal{T} = (\mathcal{T}_n)$ be the distribution in Definition 11 with width and depth parameters $w = w(n)$ and $d = d(n)$. Then, $\mathcal{T}$ is a quantum-secure $(2^d - 1, \sqrt{d} \cdot 2^{d+2-w/2})$-faux-deterministic algorithm of depth $w \cdot d$ for* FIND1.

Proof. Let $\mathcal{Q} = (U_0, U_1, \ldots, U_q)$ be a quantum algorithm that makes q queries to a randomly sampled decision tree $T \sim \mathcal{T}_n$ to obtain $|\psi\rangle = U_q O^T \ldots U_1 O^T |\psi_0\rangle$, where $|\psi_0\rangle = U_0 |0\rangle$ is the initial state, and O^T is defined as

$$O^T |x\rangle_I |z\rangle_Z = \begin{cases} |x\rangle_I |T[x] \oplus z\rangle_Z & |x| \geq n/2, \\ |x\rangle_I |z\rangle_Z & \text{otherwise.} \end{cases}$$

Then, the algorithm performs a measurement on the final state $|\psi\rangle$ to obtain the output string $x = \mathcal{Q}[T] \in \{0,1\}^n$.

Let $B_1, \ldots, B_{2^d-1}$ denote the corresponding query blocks in the tree T according to Proposition 12. Similar to the Proof of Theorem 15, for the sake of simplicity, we assume that after the k-th query, the adversary has access to all of the first k blocks $B_1, \ldots, B_k \subset [n]$.

Hybrid argument. We are going to use a hybrid argument by constructing a series of quantum algorithms $\mathcal{Q}^0 = \mathcal{Q}$, $\mathcal{Q}^1$, …, and $\mathcal{Q}^q = \mathcal{Q}'$ as follows: for $k \in [q-1]$, the algorithm $\mathcal{Q}^k$ is defined by replacing the first k queries to O^T with the sequence of simulated queries $O^{T^1}, \ldots, O^{T^k}$ while maintaining the remaining $q - k$ queries as the original oracle O^T. Formally, we have the construction in Algorithm 1.

Algorithm 1 Construction of the hybrid $\mathcal{Q}^k$.

Defining the partial simulation T^k (see Figure 2). Assume that the first k blocks $\hat{B}_1, \ldots, \hat{B}_k$ and an input $x \in \{0,1\}^n$ are given. Then, $T^k[x]$ simulates the structure of T, beginning at the root node where at each step a block B_ℓ for some $\ell \in [2^d - 1]$ is queried.

- If the queried block is one of the given blocks $\ell \leq k$, we execute $\hat{B}_\ell[x]$ and traverse left ($\hat{B}_\ell[x] = 1$) or right ($\hat{B}_\ell[x] = 0$) accordingly. Otherwise, $\ell > k$, and we traverse left as if $B_\ell[x] = 1$.
- Finally, the algorithm reaches a leaf node. Let $L := \{\ell \in [k] : \hat{B}_\ell[x] = 1\}$. If $L = \emptyset$, we output an arbitrarily fixed index $i \in [n]$. Otherwise, L has a largest element ℓ^*, and we output an index $j \in \hat{B}_{\ell^*}$ satisfying $x_j = 1$.

The simulated unitary O^{T^k}. Naturally, we define the unitary O^{T^k} as

$$\forall x \in \{0,1\}^n \qquad O^{T^k} |x\rangle_I |z\rangle_Z = \begin{cases} |x\rangle_I |T^k[x] \oplus z\rangle_Z & |x| \geq \frac{n}{2}, \\ |x\rangle_I |z\rangle_Z & \text{otherwise.} \end{cases}$$

The hybrid $\mathcal{Q}^k$. Given blocks $\hat{B}_1, \ldots, \hat{B}_k$, the algorithm $\mathcal{Q}^k$ evolves the initial state $|\psi_0\rangle$ as follows.

- **Simulation phase:** For each $t \in [k]$, the algorithm replaces the t-th query with a simulated query to O^{T^t}. That is, we let $|\psi_0^k\rangle = |\psi_0\rangle$, and recursively define $|\psi_t^k\rangle := U_t O^{T^t} |\psi_{t-1}^k\rangle$ for $t = 1, \ldots, k$.
- **Faithful phase:** For $t = k+1, \ldots, q$, the algorithm proceeds with the remaining queries to the real oracle O^T: $|\psi_t^k\rangle := U_t O^T |\psi_{t-1}^k\rangle$.
- **Measurement:** finally, we perform the measurement in the computational basis on $|\psi_q^k\rangle$ to obtain the output string $x = \mathcal{Q}^k[T] \in \{0,1\}^n$.

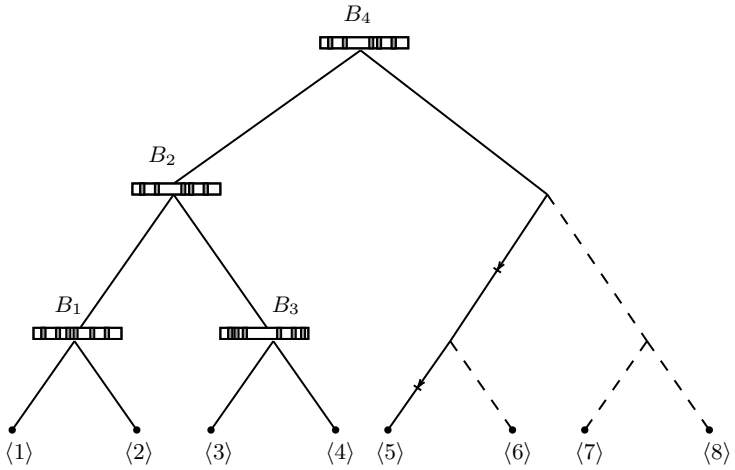

Figure 2 The simulated tree T^4 from T with the knowledge of B_1, B_2, B_3, and B_4. The quantum algorithm simulating them only computes the nodes of the tree for which the blocks are known. For instance, for any input x with $\ell_T(x) \preceq \langle 4\rangle$ the computed values of T and T^4 are equal; $T[x] = T^4[x]$.

Proof overview. Before proceeding into the details of the proof, we give an overview of the proof strategy. Let $|\psi\rangle := |\psi_q^0\rangle$ and $|\psi'\rangle := |\psi_q^q\rangle$ be the final states before measurement of the initial and final algorithms $\mathcal{Q}$ and $\mathcal{Q}'$. For a fixed tree $T \in \mathcal{T}$, we also define the projection $\Pi_1(T)$ to be the projection to states having input registers $|x\rangle_I$, where x is an *adversarial input* for T. I.e., valid inputs $|x| \geq n/2$ 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. As a result, $\Pi_0(T) + \Pi_1(T) = I$. We can then obtain the probability that $\mathcal{Q}$ and $\mathcal{Q}'$ find an adversarial input (conditioned on tree T) as

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

We begin by bounding the "expected success probability" $p = \mathbb{E}_{T\sim\mathcal{T}}[\sqrt{p(T)}]$ through the following steps.

1. **Bounding the Euclidean distance.** First, we prove that, the expected Euclidean distance between the pre-measurement final states of any consecutive pairs $|\psi_q^{k+1}\rangle$ and $|\psi_q^k\rangle$ is small. As a result, we obtain an upper bound on $\mathbb{E}_T[\|\,|\psi'\rangle - |\psi\rangle\,\|]$.
2. **Bounding p'.** Next, we prove that $\mathbb{E}_T[p'(T)]$ is small with high probability as it does not make any queries to the real tree T.
3. **Putting the above together.** We then obtain an upper bound for p using the upper bounds on $\|\,|\psi\rangle - |\psi'\rangle\,\|$ and our bound on $\mathbb{E}_T[p'(T)]$.

Finally, we convert the bound on $\mathbb{E}_T[\sqrt{p(T)}]$ to the total success probability.

Bounding the Euclidean distance. Let $\langle\ell\rangle$ be the ℓ-th leaf in the tree as in Proposition 12 and $\ell_T(x)$ be the reached leaf when executing T on x. The strategy is as follows: for any valid input x if $\ell_T(x) \preceq \langle k\rangle$, the tree only queries among the blocks $B_1 = \hat{B}_1, \ldots, B_k = \hat{B}_k$ and outputs an index from one of these blocks. Thus, for any such input x, $T[x] = T^{k+1}[x]$ unless $\ell_T(x) \succ \langle k+1\rangle$. Furthermore, $\mathcal{Q}^k$ does not have any information about $B_{k+1}, \ldots, B_{2^d}$

before its first real query to O^T. Consequently, applying Lemma 13, we can prove that $\ell_T(x) \preceq \langle k+1 \rangle$ 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. Formally, we have the following.

▷ Claim 18. The following holds.

i. For $k \in \{0, \dots, q-1\}$, $\mathbb{E}_T \left[\left\| |\psi_q^{k+1}\rangle - |\psi_q^k\rangle \right\| \right] \leq \sqrt{\frac{4d}{2^w}}$;

ii. $\mathbb{E}_T \left[\left\| |\psi'\rangle - |\psi\rangle \right\| \right] \leq q\sqrt{\frac{4d}{2^w}}$.

Proof. By the unitary invariance of the Euclidean norm, we have

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

where we also used the fact that $|\psi_k^{k+1}\rangle = |\psi_k^k\rangle$. For $x \in \{0,1\}^n$, let Π_x be the projection to the states having input register $|x\rangle_I$, where $\sum_{x\in\{0,1\}^n} \Pi_x = I$. Observe that for $x \in \{0,1\}^n$, if $T^{k+1}[x] = T[x]$ we get $\left(O^{T^{k+1}} - O^T \right) \Pi_x |\psi_k^k\rangle = 0$. So, we can write

$$\left\| |\psi_q^{k+1}\rangle - |\psi_q^k\rangle \right\| = \left\| \sum_{x\in\{0,1\}^n} \left(O^{T^{k+1}} - O^T \right) \Pi_x |\psi_k^k\rangle \right\| \tag{1}$$

$$\begin{aligned} &= \left\| O^{T^{k+1}} \sum_{\substack{x\in\{0,1\}^n \\ T^{k+1}[x]\neq T[x]}} \Pi_x |\psi_k^k\rangle - O^T \sum_{\substack{x\in\{0,1\}^n \\ T^{k+1}[x]\neq T[x]}} \Pi_x |\psi_k^k\rangle \right\| \\ &\leq \left\| O^{T^{k+1}} \sum_{\substack{x\in\{0,1\}^n \\ T^{k+1}[x]\neq T[x]}} \Pi_x |\psi_k^k\rangle \right\| + \left\| O^T \sum_{\substack{x\in\{0,1\}^n \\ T^{k+1}[x]\neq T[x]}} \Pi_x |\psi_k^k\rangle \right\| \\ &= \left\| \sum_{\substack{x\in\{0,1\}^n \\ T^{k+1}[x]\neq T[x]}} \Pi_x |\psi_k^k\rangle \right\| + \left\| \sum_{\substack{x\in\{0,1\}^n \\ T^{k+1}[x]\neq T[x]}} \Pi_x |\psi_k^k\rangle \right\| \\ &= 2 \left\| \sum_{\substack{x\in\{0,1\}^n \\ T^{k+1}[x]\neq T[x]}} \Pi_x |\psi_k^k\rangle \right\|. \end{aligned} \tag{2}$$

Now, consider the state $|\psi_k^k\rangle$; by definition, this state has been obtained by applying unitaries $T^1, \dots, T^k$, and therefore all the coefficients in $|\psi_k^k\rangle$ only depend on sets $\hat{B}_1, \dots, \hat{B}_k$. Consequently, conditioned on $B_1 = \hat{B}_1, \dots, B_k = \hat{B}_k$, we can write

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

where all coefficients α_x are fixed complex numbers. As a result, we have

$$\mu_{\hat{B}_1,\dots,\hat{B}_k} := \mathop{\mathbb{E}}_T \left[\left\| |\psi_q^{k+1}\rangle - |\psi_q^k\rangle \right\|^2 \middle| \hat{B}_1, \dots, \hat{B}_k \right] \leq \mathop{\mathbb{E}}_T \left[4 \sum_{\substack{x\in\{0,1\}^n \\ T^{k+1}[x]\neq T[x]}} |\alpha_x|^2 \middle| \hat{B}_1, \dots, \hat{B}_k \right]$$

$$\leq 4 \sum_{x\in\{0,1\}^n} |\alpha_x|^2 \Pr_T\left[T[x] \neq T^{k+1}[x] \middle| \hat{B}_1, \ldots, \hat{B}_k\right].$$

Recall that if $\ell_T(x) \preceq \langle k+1\rangle$ then $T^{k+1}[x] = T[x]$. Applying Lemma 13, we obtain

$$\begin{aligned}\Pr_T\left[T[x] \neq T^{k+1}[x] \middle| \hat{B}_1, \ldots, \hat{B}_k\right] &\leq 1 - \Pr_T\left[\ell_T(x) \preceq \langle k+1\rangle \middle| \hat{B}_1, \ldots, \hat{B}_k\right] \\ &\leq 1 - (1 - \frac{1}{2^w})^d \\ &\leq \frac{d}{2^w}.\end{aligned}$$

As a result, we get the upper bound $\mu_{\hat{B}_1,\ldots,\hat{B}_k} \leq \frac{4d}{2^w} \sum_{x\in\{0,1\}^n} |\alpha_x|^2 = \frac{4d}{2^w}$. Conditioning on $\hat{B}_1, \ldots, \hat{B}_k$, we compute the expectation as

$$\mathop{\mathbb{E}}_T\left[\left\||\psi_q^{k+1}\rangle - |\psi_q^k\rangle\right\|^2\right] = \sum_{\hat{B}_1,\ldots,\hat{B}_k} \mu_{\hat{B}_1,\ldots,\hat{B}_k} \cdot \Pr_T\left[\hat{B}_1 \ \ldots, \hat{B}_k\right] \leq \frac{4d}{2^w}.$$

Finally, we apply Jensen's inequality to get

$$\mathop{\mathbb{E}}_T\left[\left\||\psi_q^{k+1}\rangle - |\psi_q^k\rangle\right\|\right] \leq \sqrt{\mathop{\mathbb{E}}_T\left[\left\||\psi_q^{k+1}\rangle - |\psi_q^k\rangle\right\|^2\right]} \leq \sqrt{\frac{4d}{2^w}}.$$

For part (ii), we can bound the distance between these two states as

$$\mathop{\mathbb{E}}_T\left[\left\||\psi'\rangle - |\psi\rangle\right\|\right] = \mathop{\mathbb{E}}_T\left[\left\|\sum_{k=1}^{q} |\psi_q^k\rangle - |\psi_q^{k-1}\rangle\right\|\right] \leq \sum_{k=1}^{q} \mathop{\mathbb{E}}_T\left[\left\||\psi_q^k\rangle - |\psi_q^{k-1}\rangle\right\|\right] \leq q\sqrt{\frac{4d}{2^w}}. \qquad \triangleleft$$

Bounding $\mathbb{E}_T[p'(T)]$. Note that $\mathcal{Q}'$ depends only on the first q blocks, with $q < 2^d - 1$. As a result, block B_{2^d-1} is independent of the output of $\mathcal{Q}'$, and we can show the following.

▷ Claim 19. $\mathbb{E}_T[p'(T)] \leq \frac{1}{2^w}$.

Proof. Conditioning on values $\hat{B}_1, \ldots, \hat{B}_q$, we have

$$\mathop{\mathbb{E}}_T[p'(T)] = \sum_{\hat{B}_1,\ldots,\hat{B}_q} \mathop{\mathbb{E}}_T\left[p'(T) \mid \hat{B}_1, \ldots, \hat{B}_q\right] \Pr_T\left[\hat{B}_1, \ldots, \hat{B}_q\right]. \tag{3}$$

Conditioning further on output of $\mathcal{Q}'$, we have

$$\begin{aligned}&\mathop{\mathbb{E}}_T\left[p'(T)|\hat{B}_1, \ldots, \hat{B}_q\right] \\ =& \sum_{y\in\{0,1\}^*, |y|\geq\frac{n}{2}} \mathop{\mathbb{E}}_T\left[p'(T) \mid y \leftarrow \mathcal{Q}'[T], \hat{B}_1, \ldots, \hat{B}_q\right] \Pr\left[y \leftarrow \mathcal{Q}'[T] \mid \hat{B}_1, \ldots, \hat{B}_q\right] \\ =& \sum_{y\in\{0,1\}^*, |y|\geq\frac{n}{2}} \mathop{\mathbb{E}}_T\left[T[y] \notin \mathsf{FIND1}(y) | y \leftarrow \mathcal{Q}'[T], \hat{B}_1, \ldots, \hat{B}_q\right] \Pr[y \leftarrow \mathcal{Q}'[T] \mid \hat{B}_1, \ldots, \hat{B}_q] \\ \leq& \sum_{y\in\{0,1\}^*, |y|\geq\frac{n}{2}} \mathop{\mathbb{E}}_T\left[B_{2^d-1}(y) = 0 | y \leftarrow \mathcal{Q}'[T], \hat{B}_1, \ldots, \hat{B}_q\right] \Pr[y \leftarrow \mathcal{Q}'[T] \mid \hat{B}_1, \ldots, \hat{B}_q] \\ =& \sum_{y\in\{0,1\}^*, |y|\geq\frac{n}{2}} \mathop{\mathbb{E}}_T[B_{2^d-1}(y) = 0] \Pr[y \leftarrow \mathcal{Q}'[T] \mid \hat{B}_1, \ldots, \hat{B}_q] \\ \leq& \sum_{y\in\{0,1\}^*, |y|\geq\frac{n}{2}} \frac{1}{2^w} \Pr[y \leftarrow \mathcal{Q}'[T] \mid \hat{B}_1, \ldots, \hat{B}_q] \\ =& \frac{1}{2^w},\end{aligned}$$

where the fourth line is because $T[y] \notin \mathsf{FIND1}(y)$ implies $B_{2^d-1}(y) = 0$, the fifth line is because B_{2^d-1} is independent of the first q blocks, and the sixth line is because the probability that B_{2^d-1} misses all 1-indices of y is at most 2^{-w}, and the last line is because we assumed the algorithm always outputs a valid string $|y| \geq n/2$. Plugging 2^{-w} in (3) we get the desired result. ◁

Putting the above together. We are now ready to upper bound $\mathbb{E}_T[p(T)]$. Note that, for any tree $T \in \mathcal{T}_n$,

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

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

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

By Claim 18 part (ii), the first expectation in the above is upper bounded by $q\sqrt{4d/2^w}$. Moreover, by Claim 19, we can bound the second part $\mathbb{E}_T[\sqrt{p'(T)}] \leq \sqrt{\mathbb{E}_T[p'(T)]} \leq \sqrt{1/2^w}$, where we once again used Jensen's inequality. Putting these together, we get $\mathbb{E}_T[\sqrt{p(T)}] \leq q\sqrt{4d/2^w} + \sqrt{1/2^w} = (2q\sqrt{d}+1)2^{-\frac{w}{2}}$. Moreover, since $0 \leq p(T) \leq 1$, we have $p(T) \leq \sqrt{p(T)}$, and therefore

$$\Pr_{T,\mathcal{Q}}\left[x \leftarrow \mathcal{Q}[T]; T[x] \notin \mathsf{FIND1}\right] = \mathop{\mathbb{E}}_{T}[p(T)] \leq \mathop{\mathbb{E}}_{T}\left[\sqrt{p(T)}\right] \leq (2q\sqrt{d}+1)2^{-\frac{w}{2}}.$$

Finally, since $q \leq 2^d - 1$, $p \leq 2 \cdot (2 \cdot 2^d\sqrt{d}+1) \cdot 2^{-w/2} \leq \sqrt{d} \cdot 2^{d+2-w/2}$. ◀

▶ **Corollary 20.** *Let $t = t(n) \in \mathbb{N}$ be an increasing function. Then, the* FIND1 *problem has a quantum-secure $(2^{\frac{\sqrt{t}}{8}}, 2^{-\frac{\sqrt{t}}{8}})$-faux-deterministic algorithm of complexity t. Specifically, for any $\epsilon > 0$ and $t = \log^{2+\epsilon}(n)$, we have a quantum-secure $(\text{quasi-poly}(n), \text{negl}(n))$-faux-deterministic algorithm of depth $\text{polylog}(n)$.*

Proof. For large enough t, letting $w = \frac{7}{8}\sqrt{t}$ and $d = \frac{1}{8}\sqrt{t}+1$ in Theorem 17, we get a faux-deterministic algorithm of depth $dw \leq t$ such that for any attacker with less than $2^{\frac{1}{8}\sqrt{t}} \leq 2^{\frac{1}{8}\sqrt{t}+1} - 1$ queries will succeed at finding an invalid output with probability at most $\sqrt{d} \cdot 2^{d+2-w/2} \leq 2^{\frac{2}{8}\sqrt{t}-\frac{7}{16}\sqrt{t}} \leq 2^{\frac{-\sqrt{t}}{8}}$. ◀

3.4 Partial converse

In this section, we present a partial converse to our faux-deterministic construction: while our construction of depth t is secure against adversaries making $2^{O(\sqrt{t})}$ many queries, we show that for any distribution over decision trees of depth t, a non-adaptive adversary can find an adversarial input in $2^{O(t)}$ many queries.

▶ **Proposition 21.** *Let $\mathcal{T}_n$ be any distribution over decision trees with input length n and total depth $t = t(n) \leq \sqrt{n}/2$. Then, for $k \in \mathbb{N}$, $\mathcal{T}_n$ cannot be a $(k, 1 - e^{\frac{k}{2^{t+1}}})$-faux-deterministic algorithm for the* FIND1 *problem. Specifically, there exists a classical adversary that makes $k = 2^{t+1}$ many queries and succeeds with probability at least $1/2$ over the choice of the tree from the distribution.*

Proof. Let $T \in \mathcal{T}$ be a fixed decision tree of depth $t \leq \sqrt{n}/2$, and let $I := \{i_1, \ldots, i_s\}$ with $s \leq t$ denote the set of indices queried by T on the input 0^n; that is, the branch where all query outcomes are 0. Thus, a string x with $|x| \geq n/2$ is an adversarial input ($T[x] \notin \mathsf{FIND1}(x)$) 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 X^j with $|X^j| = n/2$ and queries these strings to obtain $T[X^j]$. Eventually, $\mathcal{A}$ outputs any X^j such that $T[X^j] \notin \mathsf{FIND1}(X^j)$, whenever one exists.

For a fixed $j \in [k]$, define $Z^j \subset [n]$ to be the set of indices in X^j with value 0. Therefore, X^j is an adversarial input if and only if $I \subset Z^j$. Consequently, we have

$$\Pr_{X^j}[T[X^j] \notin \mathsf{FIND1}(X^j)] = \Pr_{X^j}[I \subset Z^j] \geq \frac{\binom{n-t}{n/2-t}}{\binom{n}{n/2}} \geq \frac{(n/2-t)^t}{n^t} = \frac{1}{2^t}(1-\frac{2t}{n})^t$$
$$\geq \frac{1}{2^t}(1-\frac{2t^2}{n}) \geq \frac{1}{2^{t+1}},$$

where we have used $s \leq t$, Bernoulli's inequality and the assumption $t \leq \frac{\sqrt{n}}{2}$. Since $X^1, \ldots, X^k$ are independent, for a fixed tree T, the probability that the adversary succeeds in one of its queries is at least

$$\Pr_{X^1,\ldots,X^k}\left[\exists j : T[X^j] \notin \mathsf{FIND1}(X^j)\right] \geq 1 - (1-\frac{1}{2^{t+1}})^k \geq 1 - e^{-\frac{k}{2^{t+1}}}.$$

Note that the above probability is independent of the fixed tree $T \in \mathcal{T}$. As a result, we have $\Pr_{T\sim\mathcal{T},X^1,\ldots,X^k}\left[\exists j : T[X^j] \notin \mathsf{FIND1}(X^j)\right] \geq 1 - e^{-k/2^{t+1}}$. Maximizing over the choice of $X^1, \ldots, X^k$, we obtain a deterministic adversary which proves the desired result. ◀

4 Faux-deterministic algorithm for verifiable search problems

As previously discussed, for input length $n \in \mathbb{N}$, the $\mathsf{FIND1}$ problem has a simple randomized algorithm $\mathcal{R} = (\mathcal{R}_n)$, where $\mathcal{R}_n = \{T_1, \ldots, T_n\}$ and each tree T_i on input x queries x_i and, if $x_i = 1$, outputs i. Thus, each query in the tree construction of Definition 11 can be viewed as

- querying a random tree $T_i \sim \mathcal{R}_n$;
- 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 all search problems that have efficient randomized algorithms and verifiers.

▶ **Theorem 22.** *Let $\Sigma = (\Sigma_n)$ and let $S \subset \bigcup_{n\in\mathbb{N}}(\{0,1\}^n \times \Sigma_n)$ be a search problem with verifier $V = (V_{n,y})_{n\in\mathbb{N},y\in\Sigma_n}$ of complexity $v(n)$ and a randomized algorithm $\mathcal{R} = (\mathcal{R}_n)$ with complexity $r(n)$ with error at most $1/3$. Let $t = t(n) \in \mathbb{N}$ be an increasing function. Then, there exists a faux-deterministic algorithm $\mathcal{F}$ of complexity $(r(n) + v(n)) \cdot t$ such that*

- *S is $(2^{\frac{\sqrt{t}}{4}}, 2^{-\frac{\sqrt{t}}{4}})$-secure against classical adversaries;*
- *S is $(2^{\frac{\sqrt{t}}{8}-1}, 2^{-\frac{\sqrt{t}}{8}})$-secure against quantum adversaries.*

The proof follows the strategy of [11], adapting their reduction to our setting. We start with the following folklore amplification lemma that intuitively reduces the number of decision trees in the randomized algorithm to a linear number.

▶ **Lemma 23.** *Let $S = (S_n)$ where $S_n \subset \{0,1\}^n \times \Sigma_n$ for some output alphabet $\Sigma = (\Sigma_n)$. There exists a large enough constant c such that the following holds. Given a randomized*

algorithm $\mathcal{R} = (\mathcal{R}_n)$ for S with $\Pr_{R\sim\mathcal{R}_n}[R[x] \notin S(x)] \leq 1/3$ for all $x \in \mathrm{dom}(S_n)$, there exists a randomized algorithm $\mathcal{R}' = (\mathcal{R}'_n)$ where $\mathcal{R}'_n$ consists of cn many decision trees, has the same complexity as $\mathcal{R}$, and satisfies $\Pr_{R\sim\mathcal{R}'_n}[R[x] \notin S(x)] \leq 1/2$.

For the sake of completeness, we provide the proof as follows.

Proof. Let c be a constant to be determined later, and let $R_1, \ldots, R_m \sim \mathcal{R}_n$ be uniformly random and independent decision trees sampled from $\mathcal{R}_n$ where $m = cn$. For each input $x \in \mathrm{dom}(S_n)$ and index $i \in [m]$, we define the random variable $X_i^x := V_{n,R_i(x)}[x]$ to denote whether R_i computes a correct output for input x or not. By definition, for each such x, the random variable $X^x := \sum_{i=1}^m X_i^x$ denotes the number of decision trees outputting a correct answer for x. Our goal is to show that with non-zero probability over the choice of R_i, for all inputs $x \in \mathrm{dom}(S_n)$, we have $X^x \geq m/2$.

Let $x \in \mathrm{dom}(S_n)$ be a fixed input. By linearity of expectation, we have $\mathbb{E}_{R_1,\ldots,R_m}[X^x] = \sum_{i=1}^m \mathbb{E}_{R_i}[X_i^x] \geq 2m/3$, where we used the fact that the randomized algorithm $\mathcal{R}_n$ succeeds with probability at least 2/3. Moreover, since $X_1^x, \ldots, X_m^x$ are independent, we can use the multiplicative form of Chernoff's bound to obtain

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

Thus, by union bound we can get $\Pr_{R_1,\ldots,R_m}\left[\exists x : X^x \leq \frac{m}{2}\right] \leq 2^n e^{-m/48}$. For $c > 48\ln 2$, the above probability is less than 1. Consequently, there exists fixed $R_1, \ldots, R_m \in \mathcal{R}_n$ such that for every $x \in \mathrm{dom}(S_n)$ at least $m/2$ of the R_i output a correct answer. So, a uniform distribution over $\mathcal{R}' := \{R_1, \ldots, R_m\}$ yields the desired result. ◀

Proof of Theorem 22. Let c be the constant in Lemma 23, and let $R_1, \ldots, R_m$ be the corresponding decision trees in $\mathcal{R}'_n$ where $m = cn$. Given an input string x, the main idea is to define a binary string $y(x) \in \{0,1\}^m$ that assigns 0 or 1 to each decision tree R_i and run the faux-deterministic algorithm for FIND1 on $y(x)$. Formally, we set

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

In other words, $y(x)_i = 1$ if and only if R_i outputs a correct answer on x. Note that by Lemma 23, $y(x)$ is a binary string of length $m = cn$ and has Hamming weight $|y(x)| \geq \frac{m}{2}$.

Let $\mathcal{T} = (\mathcal{T}_m)$ be the faux-deterministic algorithm from Theorem 15 with depth parameter t, applied to inputs of length m, and let $T \sim \mathcal{T}_m$. For $T \in \mathcal{T}_m$ we construct an algorithm F_T that on input x simulates $T[y(x)]$ as follows: whenever T queries an index $i \in [m]$, we run $R_i[x]$ and output $y(x)_i \in \{0,1\}$ (namely 1 iff $V_{n,R_i[x]}[x]$ accepts). At the end, the tree T returns an index $i \in [m]$, and we output $R_i[x]$. Let $\mathcal{F} = (\mathcal{F}_n)$ be the distribution of F_T where $T \sim \mathcal{T}_m$. It is easy to verify that $\mathsf{DEPTH}(\mathcal{F}_n) \leq (v(n) + r(n)) \cdot t$.

It only remains to prove that $\mathcal{F}$ is a faux-deterministic algorithm for S. We do this by reducing an adversary for $\mathcal{F}$ to an adversary for $\mathcal{T}_m$ (on FIND1 inputs $y(x)$) as follows. For $n \in \mathbb{N}$, let A be an adversary (classical or quantum) that makes at most q queries and finds an adversarial input for $F_T \sim \mathcal{F}_n$, where the probability is over $T \sim \mathcal{T}_m$. We construct a corresponding adversary A' for $\mathcal{T}_m$ by running the algorithm A with a simulation of $\mathcal{F}_n$ as follows.

- **Classical setting.** In this case, for an input $x \in \{0,1\}^n$, we can directly simulate a query to $F_T[x]$ with a single query to $T[y(x)]$: using $R_1, \ldots, R_m$ and the verifier V, we construct the string $y(x)$; then, we query T on input $y(x)$ to obtain an index $i = T[y(x)]$, and return $R_i[x]$ to A.

- **Quantum setting.** In the quantum setting, we simulate each call to the oracle unitary $O^{F_T} : |x\rangle_I |z\rangle_Z \to |x\rangle_I |z \oplus F_T[x]\rangle_Z$ (where we use the fixed binary encoding of Σ_n in the output register Z) with *two* calls to the unitary O^T, defined on registers Y and Q (with I and Z being treated as workspace and left unchanged) by

$$O^T |x\rangle_I |y\rangle_Y |i\rangle_Q |z\rangle_Z = \begin{cases} |x\rangle_I |y\rangle_Y |i \oplus T[y]\rangle_Q |z\rangle_Z & |y| \geq m/2, \\ |x\rangle_I |y\rangle_Y |i\rangle_Q |z\rangle_Z & \text{otherwise.} \end{cases}$$

  First, we prepare two extra registers Y and Q consisting of m and $\log m$ qubits, respectively. Additionally, using the classical algorithms $R_1, \ldots, R_m$ and V, we construct two unitary maps U_y and U_R defined by their evolution $U_y |x\rangle_I |0\rangle_Y = |x\rangle_I |y(x)\rangle_Y$ and $U_R |x\rangle_I |i\rangle_Q |z\rangle_Z = |x\rangle_I |i\rangle |z \oplus R_i[x]\rangle_Z$. Then, we simulate O^{F_T} by applying the unitaries $U_y^\dagger O^T U_R O^T U_y$. More specifically, we have the following evolution.

$$\begin{aligned} |x\rangle_I |0\rangle_Y |0\rangle_Q |z\rangle_Z &\xrightarrow{U_y} |x\rangle_I |y(x)\rangle_Y |0\rangle_Q |z\rangle_Z \\ &\xrightarrow{O^T} |x\rangle_I |y(x)\rangle_Y |T[y(x)]\rangle_Q |z\rangle_Z \\ &\xrightarrow{U_R} |x\rangle_I |y(x)\rangle_Y |T[y(x)]\rangle_Q |z \oplus R_{T[y(x)]}\rangle_Z \\ &\xrightarrow{O^T} |x\rangle_I |y(x)\rangle_Y |0\rangle_Q |z \oplus R_{T[y(x)]}\rangle_Z \\ &\xrightarrow{U_y^\dagger} |x\rangle_I |0\rangle_Y |0\rangle_Q |z \oplus R_{T[y(x)]}\rangle_Z , \end{aligned}$$

  which is equal to $|x\rangle_I |0\rangle_Y |0\rangle_Q |z \oplus F_T[x]\rangle_Z$.

In either case, from the perspective of A', an algorithm $F_T \in \mathcal{F}$ corresponding to the tree T is being queried. So, if A finds an adversarial input for F_T, our simulation returns an input $\bar{x}$ for which $F_T[\bar{x}] \notin S(\bar{x})$. Hence, for the input $y(\bar{x})$ and the output $i = T[y(\bar{x})]$, we must have $(y(\bar{x}))_i = 0$ (equivalently, $V_{n,R_i[\bar{x}]}[\bar{x}] = 0$), which means $y(\bar{x})$ is an adversarial input for T. As a result, if A succeeds with probability δ (over $T \sim \mathcal{T}_m$), then A' succeeds with probability δ.

On the other hand $\mathcal{T}_m$ is $(2^{\sqrt{t}/4}, 2^{-\sqrt{t}/4})$-secure against classical adversaries (Corollary 16) and $(2^{\sqrt{t}/8}, 2^{-\sqrt{t}/8})$-secure against quantum adversaries (Corollary 20). Since the classical reduction uses exactly q queries to the FIND1 oracle, while the quantum reduction uses $2q$ queries, the theorem follows for classical adversaries making fewer than $2^{\sqrt{t}/4}$ queries and quantum adversaries making fewer than $2^{\sqrt{t}/8-1}$ queries. ◀

Similar to Corollary 16 and Corollary 20, from the above we immediately get the following corollary.

▶ **Corollary 24.** *If the search problem S has a verifier and a randomized algorithm of depth* $\mathrm{polylog}(n)$ *with error at most* $1/3$*, then S has a* $(\text{quasi-poly}(n), \mathrm{negl}(n))$*-faux-deterministic algorithm of depth* $\mathrm{polylog}(n)$.

Proof. For any $\epsilon > 0$, take $t = \log^{2+\epsilon}(n)$ and apply Theorem 22; see Corollary 16 and Corollary 20 for the security bounds. ◀

5 Sensitivity problem

In this section, we consider the meta-complexity problem of finding an input with high block sensitivity.

▶ **Definition 25.** *Let $s = s(n) \in \mathbb{N}$. We define the sensitivity problem* $\text{findBS}^s = (\text{findBS}^s)_n$ *as follows. The input is oracle access to a function $f : \{0,1\}^n \to [n]$ such that for every $x \in \{0,1\}^n$ with $|x| \geq n/2$, we have $f(x) \in \mathsf{FIND1}(x)$ (equivalently, $x_{f(x)} = 1$). The goal is to find an input x with $|x| \geq n/2$ such that f has block sensitivity at least $\text{bs}(f,x) \geq s(n)$ on x, whenever one exists.*

Goldwasser et al. [11] showed that every function f solving $\mathsf{FIND1}$ ($f(x) \in \mathsf{FIND1}(x)$ for all x with $|x| \geq n/2$) has block sensitivity at least $\text{bs}(f) = \Omega(\sqrt{n})$. We use this fact to prove the following theorem.

▶ **Theorem 26.** *Let $s = s(n)$ be an integer valued function satisfying $\omega(1) \leq s(n) \leq o(\sqrt{n})$. Then, any algorithm solving* findBS^s *requires $2^{\sqrt{s}/5}$ many queries to its oracle to solve this problem with success probability at least $2^{-\sqrt{s}/5}$. In other words, any solver using fewer than $2^{\sqrt{s}/5}$ oracle queries cannot succeed with probability more than $2^{-\sqrt{s}/5}$. Specifically, for $\omega(\text{polylog}(n)) \leq s \leq o(\sqrt{n})$, the required query bound $2^{\sqrt{s}/5}$ is super-polynomial, so any such algorithm for* findBS^s *requires more than* $\text{poly}(n)$ *queries.*

Proof. For the sake of contradiction, suppose there exists a randomized algorithm $\mathcal{A}$ that solves findBS^s with success probability at least $2^{-\sqrt{s}/5}$ while making fewer than $2^{\sqrt{s}/5}$ many queries to a given function f in the promise of findBS^s. We show how to refute any tree T of depth $s-2$ purporting to solve $\mathsf{FIND1}$ with success probability greater than $2^{-\sqrt{s}/5} > 2^{-\sqrt{s-2}/4}$ while making fewer than $2^{\sqrt{s}/5} < 2^{\sqrt{s-2}/4}$ queries to T. This yields a contradiction to $(2^{\sqrt{s-2}/4}, 2^{-\sqrt{s-2}/4})$-security in Corollary 16.

So suppose we are given access to a decision tree T. We implicitly define $f : \{0,1\}^n \to [n]$ as follows:

$$f(x) := \begin{cases} T[x] & T[x] \in \mathsf{FIND1}(x), \\ \min\{i \in [n] : x_i = 1\} & \text{otherwise.} \end{cases}$$

Note that $f(x) \in \mathsf{FIND1}(x)$ whenever $|x| \geq n/2$, and therefore f is a valid function in the promise of findBS^s. Moreover, each query to $f(x)$ can be simulated with exactly one query to $T[x]$ and checking whether $T[x] \in \mathsf{FIND1}(x)$.

To refute T, we simulate $\mathcal{A}$ on f by making $2^{\sqrt{s}/5} < 2^{\sqrt{s-2}/4}$ many queries to T. As a result, with probability at least $2^{-\sqrt{s}/5}$, we obtain an input x with $|x| \geq n/2$ that has block sensitivity at least $b \geq s$.

So, let $i := T[x]$. We show that $x_i \neq 1$, which means we found an adversarial input $T[x] \notin \mathsf{FIND1}(x)$. So suppose $x_i = 1$, and therefore $f(x) = T[x]$. Let $Q \subset [n]$ with $|Q| = s-2$ be the set of queries made by T when executed on x. By disjointness of the sensitive blocks and since $b > s \geq |Q| + 2$, we have $b - |Q| \geq 2$. Therefore, there exists at least two disjoint sensitive block $B_1, B_2 \subset [n]$ satisfying $f(x^{B_a}) \neq f(x)$ and $B_a \cap Q = \emptyset$ for $a \in \{1,2\}$. Consequently, T generates the same outputs index $T[x^{B_a}] = T[x] = f(x) \neq f(x^{B_a})$ when executed on these two new inputs. By definition of f, it must be the case that $(x^{B_b})_i \neq 1 = x_i$, and hence, $i \in B_b$. But we argued $B_1 \cap B_2 = \emptyset$ which is a contradiction, completing the proof. ◀

References

1 Hugo Aaronson, Tom Gur, and Jiawei Li. Pseudo-deterministic Quantum Algorithms, 2026. arXiv:2602.17647. doi:10.48550/arXiv.2602.17647.

2 Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and Weaknesses of Quantum Computing. *SIAM Journal on Computing*, 26(5):1510–1523, 1997. doi:10.1137/S0097539796300933.

3 Arkadev Chattopadhyay, Yogesh Dahiya, and Meena Mahajan. Pseudo-Deterministic Query Complexity of Search Problems. *Computational Complexity*, 34(2):20, 2025. doi:10.1007/s00037-025-00266-7.

4 Lijie Chen, Ce Jin, Rahul Santhanam, and R. Ryan Williams. Constructive Separations and Their Consequences. In *Proceedings of the 62nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 646–657, 2022. doi:10.1109/FOCS52979.2021.00069.

5 Lijie Chen, Zhenjian Lu, Igor Oliveira, Hanlin Ren, and Rahul Santhanam. Polynomial-Time Pseudodeterministic Construction of Primes. *Journal of the ACM*, 73(2):14:1–14:43, 2026. doi:10.1145/3803408.

6 Eran Gat and Shafi Goldwasser. Probabilistic Search Algorithms with Unique Answers and Their Cryptographic Applications. Technical Report TR11-136, Electronic Colloquium on Computational Complexity, 2011. URL: https://eccc.weizmann.ac.il/report/2011/136/.

7 Oded Goldreich, Shafi Goldwasser, and Dana Ron. On the possibilities and limitations of pseudodeterministic algorithms. In *Proceedings of the 4th Innovations in Theoretical Computer Science Conference (ITCS)*, ITCS '13, pages 127–138. Association for Computing Machinery, 2013. doi:10.1145/2422436.2422453.

8 Shafi Goldwasser and Ofer Grossman. Perfect Bipartite Matching in Pseudo-Deterministic RNC. Technical Report TR15-208, Electronic Colloquium on Computational Complexity, 2015. URL: https://eccc.weizmann.ac.il/report/2015/208/.

9 Shafi Goldwasser, Ofer Grossman, and Dhiraj Holden. Pseudo-Deterministic Proofs. In Anna R. Karlin, editor, *Proceedings of the 9th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 94 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ITCS.2018.17.

10 Shafi Goldwasser, Ofer Grossman, Sidhanth Mohanty, and David P. Woodruff. Pseudo-Deterministic Streaming. In Thomas Vidick, editor, *Proceedings of the 11th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 151 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 79:1–79:25. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ITCS.2020.79.

11 Shafi Goldwasser, Russell Impagliazzo, Toniann Pitassi, and Rahul Santhanam. On the Pseudo-Deterministic Query Complexity of NP Search Problems. In Valentine Kabanets, editor, *Proceedings of the 36th Conference on Computational Complexity (CCC)*, volume 200 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 36:1–36:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CCC.2021.36.

12 Mika Göös, Nathaniel Harms, Artur Riazanov, Anastasia Sofronova, Dmitry Sokolov, and Weiqiang Yuan. Pseudodeterministic Communication Complexity, 2025. arXiv:2511.04794. doi:10.48550/arXiv.2511.04794.

13 Igor Carboni Oliveira and Rahul Santhanam. Pseudodeterministic Constructions in Subexponential Time. Technical Report TR16-196, Electronic Colloquium on Computational Complexity, 2016. URL: https://eccc.weizmann.ac.il/report/2016/196/.

14 Takashi Yamakawa and Mark Zhandry. Verifiable Quantum Advantage without Structure. *Journal of the ACM*, 71(3):20:1–20:50, 2024. doi:10.1145/3658665.