Randomized vs. Deterministic Separation in Time-Space Tradeoffs of Multi-Output Functions

Huacheng Yu
Princeton University, NJ, USA

Wei Zhan
University of Chicago, IL, USA

Abstract

We prove the first polynomial separation between randomized and deterministic time-space tradeoffs of multi-output functions. In particular, we present a total function that on the input of n elements in $[n]$, outputs $O(n)$ elements, such that:

- There exists a randomized oblivious algorithm with space $O(\log n)$, time $O(n \log n)$ and one-way access to randomness, that computes the function with probability $1 - O(1/n)$;
- Any deterministic oblivious branching program with space S and time T that computes the function must satisfy $T^2 S \geq \Omega(n^{2.5}/\log n)$.

This implies that logspace randomized algorithms for multi-output functions cannot be black-box derandomized without an $\widetilde{\Omega}(n^{1/4})$ overhead in time.

Since previously all the polynomial time-space tradeoffs of multi-output functions are proved via the Borodin-Cook method, which is a probabilistic method that inherently gives the same lower bound for randomized and deterministic branching programs, our lower bound proof is intrinsically different from previous works.

We also examine other natural candidates for proving such separations, and show that any polynomial separation for these problems would resolve the long-standing open problem of proving $n^{1+\Omega(1)}$ time lower bound for decision problems with polylog(n) space.

2012 ACM Subject Classification Theory of computation → Computational complexity and cryptography

Keywords and phrases Time-space tradeoffs, Randomness, Borodin-Cook method

Digital Object Identifier 10.4230/LIPIcs.ITCS.2024.99

Funding *Huacheng Yu*: Supported by Simons Junior Faculty Award – AWD1007164.
Wei Zhan: Supported by a Simons Investigator Award and by the National Science Foundation grant No. CCF-2007462. This work was done when WZ was in Princeton University.

1 Introduction

Time-space tradeoff is the phenomenon in computation where one could trade time for space by recomputing the intermediate results instead of storing them. Apart from being ubiquitous, time-space lower bounds also help us understand what can or cannot be efficiently computed with limited memory. However, lower bounds for decision problems turned out difficult to prove, and it remains a major open problem to prove lower bounds polynomially better than trivial in the general non-uniform model of sequential computation, i.e. branching programs.

By contrast, there is an abundance of lower bounds against functions that have multiple bits of output (polynomial in the length of the input). The study of time-space tradeoffs of multi-output functions started with Borodin and Cook [8] who showed that sorting n numbers within space S and time T requires $TS = \Omega(n^2)$. Their problem formulation and proof was later refined by Beame [5], who also showed a similar lower bound for listing unique elements. Tight time-space lower bounds was also proved for a variety of multi-output problems, including algebraic problems like matrix multiplication and inversion [2], frequency moments over sliding windows [6], and more recently, the memory game [9], printing and counting SAT assignments [19] and multiple collision finding [12], just to name a few.

Despite being problems from different backgrounds and of different nature, the proofs of their lower bounds are all direct applications of the original methods of Borodin and Cook [8]. A formal (but restrictive for certain applications) and detailed description of the Borodin-Cook method was given in [21, Section 10.11], and here we present a brief framework of the method:

1. Fix a distribution $\mathcal{D}$ over the inputs (often uniform), and find $a(S)$ such that given $a(S)$ bits in the input, only S bits of output are revealed on average.
2. Prove that for some large $c > 1$, given any decision tree of depth $a(S)$ and $c \cdot S$ bits of output assigned to each path in the tree, these outputs are correct with probability $2^{-\Omega(S)}$ under $\mathcal{D}$. This is usually the technical part of the proof.
3. Now split the branching program into stages of length $a(S)$, and by a union bound over the 2^S starting nodes of each stage, the above argument shows that most inputs under $\mathcal{D}$ cannot generate $c \cdot S$ bits output within a stage. This implies a lower bound of the form $ST/a(S) \geq m$ where $m = \text{poly}(n)$ is the number of output bits.

Note that here we state the method for proving lower bounds against deterministic branching programs. When the distribution $\mathcal{D}$ above is uniform, the method can be stated even more simply as a counting argument over the set of inputs, and the lower bounds hold for average-case complexity. But no matter how $\mathcal{D}$ is chosen, by Yao's Minimax Principle the above arguments actually provide lower bounds against distributions over deterministic branching programs, the most general model of randomized sequential computation. In other words, the Borodin-Cook method *inherently* give the same lower bound for deterministic and randomized computation. As all the previous lower bounds are proved using the Borodin-Cook method, the following question remains unanswered before this work:

Is there a polynomial separation between randomized and deterministic branching programs for time-space tradeoffs of multi-output functions?

Here we answer this question in the affirmative for oblivious branching programs, where the queries made to the input in the branching programs are independent of the input. We design a problem called (n, p)-Non-Occurring Elements, which can be solved efficiently by randomized oblivious algorithms, while any deterministic oblivious branching program solving the problem requires either polynomially larger space or polynomially longer time. Our problem has the additional advantage that it is a total function, which prevents the trivial separation where a randomized algorithm may have a sublinear running time (see our discussion in Section 1.2).

▶ **Definition 1.** *Let $n > 1$ and p be a prime factor of n. In the (n, p)-Non-Occurring Elements ((n, p)-NOE for short) problem, the input is an unordered list of n numbers $X = (x_1, \ldots, x_n) \in [n]^n$. The output is a set $Y \subseteq [n]$ such that:*

- *If for every $c \in [n]$, the number of times that c occurs in X is a multiple of p (0 included, so there are at most n/p distinct* occurring *elements), then Y consists of the (at least $n - n/p$) elements in $[n]$ that* do not occur *in X;*
- *Otherwise $Y = \varnothing$.*

▶ **Theorem 2.** *There is a randomized oblivious branching program with space $O(\log n)$ and time $\max\{1, n/p^2\} \cdot O(n \log n)$, that computes (n, p)-NOE with probability at least $1 - 2/n$. Moreover, the algorithm can be implemented with one-way access to random bits. On the other hand, any deterministic oblivious branching program with space S and time T that correctly computes (n, p)-NOE must satisfy $T^2(S + \log T) \geq \Omega(n^3/p)$.*

Theorem 2 will be proved in Section 3. Taking $n = p^2$, we get a polynomial separation with randomized upper bound $S = O(\log n), T = O(n \log n)$ and deterministic lower bound $T^2 S \geq \widetilde{\Omega}(n^{2.5})$.

▶ Remark 3. The (n, p)-NOE problem could be perceived as a "promised" version of the Non-Occurring Elements problem (in which the output at all times consists of the elements not occurring in X), and the latter problem has time-space tradeoff $TS = \Theta(n^2)$ for both deterministic and randomized branching programs [19]. The promise that every elements occurs a multiple of p times can be efficiently checked with randomness (Lemma 5), however there may as well be a deterministic algorithm that verifies the promise in almost-linear time and poly-log space (subject to Open Problem $\star$ below). The above facts imply that neither the Non-Occurring Elements problem nor the promise itself could demonstrate the desired separation.

1.1 Separations with Implications on Decision Problems

We shall stress that our proof of Theorem 2 is not technically hard. Indeed, to bypass the inherent disadvantage of the Borodin-Cook method, the key in our lower bound proof of Theorem 2 is to choose *adversarially* a distribution $\mathcal{D}$ that *depends on* the deterministic structure of the branching program instead of fixing a distribution $\mathcal{D}$ in advance. Arguably, the difficulty in proving a separation lies mostly in finding a proper total function where the adversarial method works. We demonstrate this difficulty by showing that, for several natural candidate problems whose best known deterministic algorithms are polynomially worse than randomized algorithms, proving a polynomial separation will lead to the resolution of the following open problem:

▶ **Open Problem $\star$.** *Find an explicit family of decision problems $F : \{0, 1\}^n \to \{0, 1\}$, such that any deterministic branching program with space $S \leq \operatorname{polylog}(n)$ that computes F requires time $T = n^{1+\Omega(1)}$.*

As we mentioned at the beginning of this paper, Open Problem $\star$ is a long-standing and notoriously hard problem. Even against oblivious branching programs, the best time-space lower bound is still $T = \Omega(n \log^2(n/S))$ proved by Babai, Nisan and Szegedy [3] three decades ago.

For a concrete example of our implication results, one of the candidate functions that we study can be very succinctly described as follows:

▶ **Definition 4.** *In the 2-StepPointerChasing (2-PC for short) problem, the input is a function $f : [n] \to [n]$, and the output consists of $(x, f(f(x)))$ for all $x \in [n]$.*

The 2-PC problem exhibits interesting phase transitions in time-space tradeoffs with different computation models. With non-oblivious queries, 2-PC can be solved deterministically in space $O(\log n)$ and time $O(n)$. On randomized oblivious branching programs, 2-PC obliges to the tradeoff $T^2 S = \widetilde{\Theta}(n^3)$. We conjecture that randomness is required for this tradeoff, and on deterministic oblivious branching programs the lower bound $TS \geq \widetilde{\Omega}(n^2)$ holds. However, it turns out that proving such a separation (in fact any lower bound polynomially better than $T^2 S = \Omega(n^3)$) is at least as hard as answering Open Problem $\star$. We will show this in Theorem 13, by relating 2-PC to a matching problem on explicit bipartite expanders.

Another example is the SetIntersection problem (given two sets A and B, output elements in $A \cap B$). The optimal randomized algorithm for SetIntersection is based on the small-space collision finding algorithm [6] on random (or pseudo-random [10, 18]) hash functions, with time-space tradeoff $T^2S = \widetilde{O}(n^3)$. The randomness seems essential in the original algorithm and its pseudo-random improvements; however, in Theorem 16 we give a black-box reduction from SetIntersection to the well-studied decision problem of ElementDistinctness, which shows that proving a polynomial separation for SetIntersection would answer Open Problem $\star$ on ElementDistinctness.

1.2 Related Works

Query Complexity

In essence, time-space tradeoff in non-uniform models is a study on query complexity with bounded memory. There are extensive results on separations for query complexity in different models, and readers can refer to [1] for a comprehensive lists of separations and relations between query complexities and other complexity measures on total functions.

Our result is in a parallel world to these separations: The separations in query complexity are based on *sub-linear* query algorithms, while we demonstrate our separation with *super-linear* lower bounds. In fact, for oblivious queries, there is no separation in query complexity: It was shown in [20] that no matter deterministic, randomized or even quantum, the oblivious (or non-adaptive) query complexity for *any* total boolean function that depends on n variables is always $\Omega(n)$. This is the major reason that we focus on total functions, so that our separation is substantial in the space-bounded setting.

Superfast Derandomization

With the common belief that $\mathsf{BPP} = \mathsf{P}$ and $\mathsf{BPL} = \mathsf{L}$ and the long line of literature under the "Hardness vs. Randomness" paradigm, several recent works take interest in the question of how fast derandomization could be. In particular, Chen and Tell [11] showed that under standard hardness assumption, linear overhead in the running time is possible, which is optimal assuming `#NSETH`.

In the space-bounded setting, Hoza [15] proved that any decision problem with space S, time $n \cdot \mathrm{poly}(S)$ and one-way access to randomness can be solved with the same space and time bound and only $O(S)$ random bits. It is not clear whether a similar result holds for full derandomization even for $S = O(\log n)$, while our Theorem 2 can be viewed as an impossibility result for multi-output functions, that a logspace randomized algorithm cannot be black-box derandomized (so that it keeps the oblivious query pattern) without an $\widetilde{\Omega}(n^{1/4})$ time overhead.

In order to remove the black-box assumption, a separation for general non-oblivious branching programs needs to be proved. Note that it is folklore that removing obliviousness on branching programs for multi-output functions requires polynomial overhead (see Section 4.1 for an example), and therefore our result does not translate directly into such a separation. However, we do conjecture that the lower bound in Theorem 2 still holds in the non-oblivious case, and could be tightened up to $TS \geq \widetilde{\Omega}(n^2)$, which would imply an *unconditional* $\widetilde{\Omega}(n)$ derandomization overhead.

Derandomizing Element Distinctness

In [6], Beame, Clifford and Machmouchi presented an randomized algorithm that solves ElementDistinctness (ED for short, that decides whether n input elements are all distinct) with tradeoff $T^2S = \widetilde{O}(n^3)$, which is strictly better than sorting. The algorithm requires access to a large random hash function that does not count towards the space usage. Recent works [10, 18] managed to bring down the seed length of the hash function to $O(\log^3 n)$ and thus acquiring an algorithm with the same tradeoff but also one-way access to random bits. They raised the question of how small the random seed length could be. At an extreme, one may wonder if the algorithm can be even fully derandomized, so that ED is solved deterministically with the same tradeoff.

This question cannot be answered in negative without answering Open Problem $\star$. However, there are numerous multi-output functions, such as SetIntersection, that uses the same algorithm to achieve the $T^2S = \widetilde{O}(n^3)$ tradeoff, and proving stronger deterministic lower bounds would also imply that the algorithm of [6] cannot be fully derandomized without polynomial overhead. Yet in Section 4.2, we show that for all such problems for which we have a tight randomized lower bound, this route is also impossible without answering Open Problem $\star$.

Quantum Time-Space Tradeoffs

Finally, we would like to mention that the situation of time-space tradeoffs of multi-output functions is entirely different in quantum computing. Klauck [16] presented a quantum algorithm for sorting n numbers with $T^2S = \widetilde{O}(n^3)$, which already provides a polynomial separation between quantum and randomized time-space tradeoff compared to the classical lower bound in [8, 5]. The separation is extra strong in the sense that the quantum algorithm uses only polylog(n) quantum memory.

On the other hand, lower bounds for quantum time-space tradeoffs are more scarce and the proofs for Step 2 in the quantum analogy of the Borodin-Cook methods are more ad-hoc. Currently, only two proof methods are known for quantum lower bounds: via direct-product theorems [17] and via the recording query technique [14].

2 Preliminaries

We use $[n]$ to denote the set $1, 2, \ldots, n$. We use asymptotic notations $\widetilde{O}$ and $\widetilde{\Omega}$ to hide poly-logarithmic factors in $O(\cdot)$ and $\Omega(\cdot)$. Since throughout this paper, the input size of interest is always polynomial in n, they always hide factors of polylog(n) regardless of the content in the parenthesis: For instance, $\widetilde{O}(1)$ in this paper always stands for $O(\text{polylog}(n))$.

Branching Programs

The computation models we consider in this paper are branching programs, which are general enough to model any computation with a proper notion of time and space. However, most of our upper bound results are uniform and can be implemented on more restrictive models such as RAMs.

A deterministic branching program is a layered DAG, with a unique initial vertex. When the inputs $(x_1, \ldots, x_n)$ of the problem are from the domain D, each vertex not in the last layer is labeled with $i \in [n]$, and has $|D|$ edges going out towards the next layer labeled with elements in D. The *computation path* on input $(x_1, \ldots, x_n)$ is the unique path that starts from the initial vertex, on each vertex labeled with i queries x_i, and then follows the outgoing edge labeled with the value of x_i, until reaching the last layer. We say that a branching program is with *space* S and *time* T if it consists of at most $T+1$ layers, and each contains at most 2^S vertices. The branching program is *oblivious*, if within each layer the labels on all the vertices are the same.

A randomized (oblivious) branching program with space S and time T is a distribution over deterministic (oblivious) branching programs with the same space and time bound. We say that a randomized branching program has *one-way access* to random bits, if in each layer the random labels on the vertices and outgoing edges are independent of the rest of the branching program.

Multi-Output Functions

For computing a decision problem, each node in the last layer of the branching program is labeled with 0 or 1, and the output is the label on the final node of the computation path. For computing a multi-output function $F : D^n \to R^m$ with the range set R, however, we allow outputting during the computation path: Each edge in the branching program is allowed to output arbitrarily many output statements $(j, y_j) \in [m] \times R$ that claims $F(x)_j = y_j$. The final output is the vector in R^m decided by the collection of output statements on the computation path, which must be complete and consistent. In many cases, the multi-output function computes a subset of R instead of a vector. In such cases, the output statements are simply elements $y \in R$, and the final output is the collection of these elements.

Finally, we note that as $|D|, |R|$ and m are all poly(n) in this paper, all the problems we consider can be converted into the binary domain and binary range, with only polylog(n) overhead in time and space for the branching programs computing them, which are ignored as we are focused on polynomial separations.

3 Polynomial Separation for Oblivious Computation

In this section we prove Theorem 2. The randomized upper bound is proved in Section 3.1 and the deterministic lower bound is proved in Section 3.2.

3.1 Randomized Oblivious Upper Bound

▶ **Lemma 5.** *There is a randomized algorithm using $O(\log n)$ space and $O(\log n)$ random bits that reads $X = (x_1, \ldots, x_n) \in [n]^n$ as a one-pass stream and satisfies that:*

- *If every $c \in [n]$ occurs in X a multiple of p times, the algorithm always accepts;*
- *Otherwise, the algorithm rejects with probability at least $1 - 2p^{-1/2} \log n$.*

Proof. The algorithm maintains a linear sketch of the frequencies of elements in $[n]$. Specifically, let $\alpha_1, \ldots, \alpha_n$ be uniformly and independently drawn from $\mathbb{F}_p$. The algorithm computes $\sum_i \alpha_{x_i}$ and accepts if the sum equals 0. If some $c \in [n]$ occurs not a multiple of p times, the factor before α_c in the sum is non-zero, and the sum equals 0 with probability $1/p$.

To reduce the random bit usage (the naive approach uses $n \log p$ random bits) we use Reed-Muller codes. Instead of drawing $\alpha_1, \ldots, \alpha_n$ independently, the algorithm draws $\beta_1, \ldots, \beta_m \in \mathbb{F}_p$ uniformly and independently, and let $\alpha_1, \ldots, \alpha_n$ be the values of monomials

$$\beta_1^{d_1} \beta_2^{d_2} \cdots \beta_m^{d_m}, \quad 0 \le d_1, \ldots, d_m < d.$$

By taking $d = p^{1/2}$ and $m = 2 \log n / \log p$, the number of such monomials is at least $d^m \ge n$. Since $m \log p = O(\log n)$, the algorithm can draw and store $\beta_1, \ldots, \beta_m$ directly. After reading $x_i = c \in [n]$, $(c-1)$ is decomposed in base d to obtain $d_1, \ldots, d_m$ in sequence, while the algorithm computes $\alpha_c = \beta_1^{d_1} \beta_2^{d_2} \cdots \beta_m^{d_m}$ and accumulates it to the sum $\sum_i \alpha_{x_i}$.

Now the sum $\sum_i \alpha_{x_i}$ is a total degree md polynomial in $\mathbb{F}_p$ on variables $\beta_1, \ldots, \beta_m$, where the the coefficients are the frequencies of elements in $[n]$ occurring in X. If every $c \in [n]$ occurs in X a multiple of p times, the polynomial is always zero; Otherwise, the polynomial is non-zero, and by the Schwartz-Zippel Lemma, the probability that the polynomial evaluates to zero is at most $md/p \leq 2p^{-1/2} \log n$. ◀

▶ **Lemma 6.** *Suppose $X = (x_1, \ldots, x_n) \in [n]^n$ satisfies that every $c \in [n]$ occurs in X either 0 or at least p times. Then there is a randomized oblivious algorithm using $O(\log n)$ space and $O(n^2 p^{-2} \log n)$ time, with one-way access to random bits, that solves* Non-Occurring Elements *on X with probability at least $1 - 1/n$.*

Proof. Let $R \subseteq [n]$ be a random multi-set of size $r = 3np^{-1} \ln n$. As every occurring element occurs at least p times, the probability that $\{x_i \mid i \in R\}$ does not contain all occurring elements in X is at most

$$n \cdot (1 - p/n)^r \leq n \cdot e^{-3 \ln n} = n^{-2}. \tag{1}$$

The algorithm goes for n/p rounds, in each round independently samples such a multi-set R of size r, and queries x_i for $i \in R$. The algorithm also stores an number j, which is initialized as 0, and in each round j is updated to the next smallest occurring number

$$j' = \min \{x_i > j \mid i \in R\} \cup \{n+1\}.$$

At the end of each round, the algorithm outputs every number strictly between the pre-updated j and j'. By (1) and a union bound, with probability at least $1 - 1/n$, in every round $\{x_i \mid i \in R\}$ contains all occurring elements (where there are at most n/p of them). In this case j goes through all occurring elements in order, and thus the outputs are exactly the non-occurring ones.

The overall time complexity is $rn/p = O(n^2 p^{-2} \log n)$, and since elements in R can be sampled sequentially to compute j' and no need to be stored, the only space usage is for storing j and j' which is $O(\log n)$. ◀

Note that Lemma 5 solves a decision problem and thus can be repeated for $O(\log n)$ times to amplify the success probability to $1 - 1/n$. Then combined with Lemma 6, we obtain the desired randomized oblivious upper bound of space $O(\log n)$ and time $O((1 + n/p^2) \cdot n \log n)$.

3.2 Deterministic Oblivious Lower Bound

▶ **Lemma 7.** *Any deterministic oblivious branching program with space S and time T that correctly computes (n,p)-NOE must satisfy $T^2(S + \log T) \geq \Omega(n^3/p)$.*

Proof. Divide the branching program into $\ell = 2T/n$ stages, each of which contains $T/\ell = n/2$ queries. We first construct a partition $\mathcal{P}$ on $[n]$ that consists of n/p parts of size p as follows:

1. Initially, let $\mathcal{P} = \varnothing$.
2. For each stage k of the branching program, let Q_k be the set of indices queried in the stage. Arbitrarily pick $r = n^2/(4Tp)$ disjoint sets of size p outside $\bigcup\{P \in \mathcal{P}\} \cup Q_k$, and add them into $\mathcal{P}$.
3. Finally after going through all the stages, arbitrarily partition the remaining elements in $[n]$ into sets of size p.

Notice that during Step 2, the total number of elements in $\bigcup\{P \in \mathcal{P}\}$ never exceeds

$$r\ell p = \frac{n^2}{4Tp} \cdot \frac{2T}{n} \cdot p = n/2.$$

As $|Q_k| \le n/2$, this implies that Step 2 is always feasible.

We define a distribution $\mathcal{D}$ of $X \in [n]^n$ as follows: For every part $P \in \mathcal{P}$, uniformly and independently pick $c \in [n]$ and let $x_i = c$ for all $i \in P$. Notice that the (n,p)-NOE problem is identical to NON-OCCURRING ELEMENTS on $\mathrm{supp}(\mathcal{D})$, i.e., every element occurs a multiple of p times. Now consider the probability

$$\Pr_{X\sim\mathcal{D}}[\text{On input } X\text{, at least } m \text{ distinct elements are outputted in stage } k]. \tag{2}$$

Since for each input $X \in \mathrm{supp}(\mathcal{D})$ there are at least $n - n/p \ge n/2$ non-occurring elements, there must exist a stage k such that the above probability is at least $1/\ell$ for $m = n/(2\ell)$.

On the other hand, Step 2 in the construction of $\mathcal{P}$ implies that, given the query answers in stage k (i.e. x_i for all $i \in Q_k$), there are at least r parts in $\mathcal{P}$ whose values in X are still uniformly random. When the query answers are given, there are at most 2^S different collections of outputs in stage k (dictated by the starting state of the stage), and if m distinct elements are outputted and thus non-occurring, each one of the r parts is consistent with these outputs with probability $1 - m/n$, as the elements in these parts should be not in the output. Therefore the probability in (2) is upper bounded by

$$2^S \cdot \left(1 - \frac{m}{n}\right)^r = 2^S \cdot \left(1 - \frac{n}{4T}\right)^{\frac{n^2}{4Tp}} \le 2^S \cdot e^{-\frac{n^3}{16T^2p}}.$$

As the probability is at least $1/\ell \ge 1/T$, we have

$$S - \frac{\log e \cdot n^3}{16T^2p} \ge -\log T \quad \Rightarrow \quad T^2(S + \log T) \ge \Omega(n^3/p). \qquad \blacktriangleleft$$

4 Separations that Imply Decision Lower Bounds

In this section we present several natural candidates of multi-output function for randomized vs. deterministic separations, and show that actually proving such separations will lead to answering Open Problem $\star$. These results can be perceived in two ways: On one hand, these are currently barrier results implying that proving separations for these natural problems is difficult, which is where the (n,p)-NOE problem in our main result stands out; On the other hand, one may hope that future developments in proving lower bounds for multi-output functions will help towards the final resolution of Open Problem $\star$.

Before getting into the concrete examples, we note that every multi-output function $F : \{0,1\}^n \to \{0,1\}^m$ can be converted to a decision problem $F' : \{0,1\}^n \times [m] \to \{0,1\}$ defined as $F'(x,i) = F(x)_i$. Therefore, if F' can be computed in space $O(\log n)$ and time $\widetilde{O}(n)$, then F can trivially be computed in space $O(\log n)$ and time $\widetilde{O}(mn)$. Our results in this section holds non-trivially with better time complexity than $\widetilde{O}(mn)$. However, this implication is still useful as it makes decision problems and single-output functions (whose outputs are in $[n]$, or generally have length $m = \mathrm{polylog}(n)$) morally equivalent with respect to Open Problem $\star$: Any lower bound for a single-output function that is polynomially better than trivial implies a corresponding lower bound for a decision problem.

4.1 Pointer Chasing and Expanders

Recall the definition of the 2-StepPointerChasing problem:

▶ **Definition 8.** *In the 2-StepPointerChasing (2-PC for short) problem, the input is a function $f : [n] \to [n]$, and the output consists of $(x, f(f(x)))$ for all $x \in [n]$.*

For non-oblivious algorithms, 2-PC can be easily solved in deterministic space $O(\log n)$ and time $O(n)$, by querying f on each x and adaptively on $f(x)$. On the other hand we have the following lower bound on oblivious algorithms for 2-PC. The proof is a direct application of Borodin-Cook method on the uniform distribution, and thus omitted.

▶ **Proposition 9.** *Any randomized oblivious branching program with space S and time T that solves 2-PC must satisfy $T^2S \geq \widetilde{\Omega}(n^3)$.*

Notice that Proposition 9 provides an example of polynomial separation between oblivious and non-oblivious time-space tradeoffs of total functions (in contrast, for decision problems the best separation between oblivious and non-oblivious branching programs is barely super-logarithmic [7]). The bound is also tight and can be achieved via the following simple algorithm: In each round pick two random subsets $X, Y \subseteq [n]$ with $|X| = |Y| = \sqrt{nS}$. We store at most $\widetilde{O}(S)$ pairs of $(x, f(x)) \in X \times Y$ by querying f on X, and output the corresponding $(x, f(f(x)))$ by querying f on Y. Each pair in a round is found with probability close to S/n, and thus $\widetilde{O}(n/S)$ rounds suffices.

The above algorithm heavily relies on the fact that Y is decided entirely by randomness and hardwired into the branching programs. A natural question is whether the same time-space tradeoff holds for oblivious computation with weaker notions of randomness, or even without randomness at all. In Theorem 13 below, we show that proving impossibility results to this question will give answers to Open Problem $\star$. We first need to introduce the single-output function, ExpanderMatching based on the explicit unbalanced bipartite expanders by Guruswami, Umans and Vadhan [13].

▶ **Definition 10.** *A bipartite graph $\Gamma \subseteq [n] \times [m]$ is a (k, a)-expander if for every subset $L \subseteq [n]$ with $|L| \leq k$, the number of the neighbors of L is at least $a \cdot |L|$.*

▶ **Theorem 11** ([13]). *For every constant $\alpha > 0$, given $n \in \mathbb{N}$ and $k \leq n$, there is an explicitly constructed bipartite graph $\Gamma_{\alpha,n,k} \subseteq [n] \times [m]$ which is a $(k, 1)$-expander, with $|\Gamma_{\alpha,n,k}| = \widetilde{O}(n)$ and $m \leq \widetilde{O}(k^{1+\alpha})$.*

The original result in [13] is stronger than stated in Theorem 11, with the expansion factor a arbitrarily close to the degree $|\Gamma_{\alpha,n,k}|/n = \mathrm{polylog}(n)$. For our application, we only need expansion to be no less than 1. We use the graph to construct an explicit single-output function as follows:

▶ **Definition 12.** *The (α, n, k)-ExpanderMatching problem is a function $[n]^k \times [m] \to [n] \cup \{\perp\}$, with m decided by Theorem 11. Given the input $L \in [n]^k$ and $y \in [m]$, we think of L as a subset of $[n]$ with $|L| \leq k$. There exists a matching for L in $\Gamma_{\alpha,n,k}$ because of the $(k, 1)$-expander property, and we consider the lexicographically smallest matching $\mathcal{M} : L \to [m]$ in $\Gamma_{\alpha,n,k}$. The output of the problem is $\mathcal{M}^{-1}(y)$ if it exists, or $\perp$ if not.*

▶ **Theorem 13.** *For every constant $\alpha > 0$, if (α, n, k)-ExpanderMatching can be solved by deterministic oblivious branching programs with space $\widetilde{O}(1)$ and time $\widetilde{O}(k)$, then for every $S \leq n$, there is a deterministic oblivious branching program solving 2-PC with space $\widetilde{O}(S)$ and time $\widetilde{O}(\sqrt{n^{3+\alpha}/S})$.*

Proof. We partition $[n]$ into blocks $B_1 \sqcup \cdots \sqcup B_{n/k}$ of size k, with k to be optimally chosen later. The deterministic oblivious algorithm for 2-PC consists of $n/(kS)$ stages, where in each stage we output $(x, f(f(x)))$ all x in S consecutive blocks. In order to do so we need to query f on $f(B_i)$, but as the queries are oblivious, we instead query f on the neighbors of y in $\Gamma_{\alpha,n,k}$ for each $y \in [m]$. Since $|f(B_i)| \leq k$, the matching for $f(B_i)$ provides all the answers for $x \in B_i$. More concretely, the algorithm is described as Algorithm 1.

Algorithm 1 Deterministic Oblivious Algorithm for 2-PC.

```
for $\ell \leftarrow 0, \ldots, n/(kS) - 1$ do
    for $y \in [m]$ do
        for $i \in [S]$ do
            Apply $(\alpha, n, k)$-ExpanderMatching on $f(B_{i+\ell S}) \in [n]^k$ and $y$;
            Store the answer $u_i \in [n] \cup \{\perp\}$.
        foreach $v \in [n]$ such that $(v, y) \in \Gamma_{\alpha,n,k}$ do
            Query $f(v)$;
            if $v = u_i$ for some $i \in [S]$ then attach $f(u_i)$ to $u_i$.
        for $x \in B_{\ell S+1} \sqcup \cdots \sqcup B_{(\ell+1)S}$ do
            Query $f(x)$;
            if $f(x) = u_i$ for some $i \in [S]$ then output $(x, f(u_i))$.
```

To prove the correctness, it suffices to show that every $x \in [n]$ is outputted. This is guaranteed in every block $B_{i+\ell S}$, as when y goes through $[m]$, every element in $f(B_{i+\ell S})$ is matched and appears as u_i at some point.

The space complexity is clearly $\widetilde{O}(S)$ as the bottleneck is storing u_i and $f(u_i)$ for $i \in [S]$. To identify the time complexity, notice that f is queried in all three inner loops. For each ℓ and y, the (α, n, k)-Expander Matching algorithm makes $\widetilde{O}(kS)$ queries in total, while querying $f(x)$ for $x \in B_{\ell S+1} \sqcup \cdots \sqcup B_{(\ell+1)S}$ also takes $O(kS)$ time. Besides, for each ℓ, querying $f(v)$ for every edge $(v, y) \in \Gamma_{\alpha,n,k}$ takes $|\Gamma_{\alpha,n,k}| = \widetilde{O}(n)$ time. Therefore the total number of oblivious queries is

$$\frac{n}{kS}\left(m \cdot \widetilde{O}(kS) + \widetilde{O}(n)\right) = \widetilde{O}\left(k^{1+\alpha} n + \frac{n^2}{kS}\right).$$

Taking $k = \sqrt{n/S}$, the above expression is upper bounded by $\widetilde{O}(\sqrt{n^{3+\alpha}/S})$. ◀

As a direct corollary of Theorem 13, if we managed to prove a polynomial separation between randomized and deterministic oblivious time-space tradeoffs of 2-StepPointerChasing, it would imply a strong lower bound for (α, n, k)-ExpanderMatching for some $\alpha > 0$ and thus would answer Open Problem $\star$.

4.2 Element Distinctness and Collision Finding

We recall the definition of the ElementDistinctness problem.

▶ **Definition 14.** *In the ElementDistinctness (ED for short) problem, the input is a list of n elements from a fixed domain D, with $|D| = \mathrm{poly}(n)$. The output is 1 if all elements are distinct, and 0 otherwise.*

A randomized algorithm for ED with $T^2 S = \widetilde{O}(n^3)$ was given in [6], and it was later improved to use only one-way access to randomness in [10, 18]. Based on the same algorithm, they also showed that the SetIntersection problem (given two sets A and B of size n, output $A \cap B$) can be solved in $T^2 S = \widetilde{O}(n^3)$, and the tradeoff is known to be optimal [12]. Different variants of this problem was also studied, such as memory games [9] and n-collision finding [12], which share the same tight tradeoff for randomized algorithms.

Here we present a general form of SetIntersection, that covers all the variants when two sets that each contains no duplicates are given, and show its black-box relationship with ED:

▶ **Definition 15.** *In the* SetCollision *problem, the input contains two sets $A, B \subseteq D$ given as unordered lists $(a_1, \dots, a_n)$ and $(b_1, \dots, b_n)$ that contain no duplicated elements in each list itself. The output consists of all collisions, that are triples (i, j, x) such that $a_i = b_j = x$.*

▶ **Theorem 16.** *If ED can be solved deterministically with space $\widetilde{O}(1)$ and time $\widetilde{O}(n)$, then* SetCollision *can be solved deterministically with space $\widetilde{O}(1)$ and time $\widetilde{O}(n^{3/2})$. Furthermore, if the algorithm for ED is oblivious, then for every $S \le n$, there is a deterministic (non-oblivious) algorithm that solves* SetCollision *with space $\widetilde{O}(S)$ and time $\widetilde{O}(\sqrt{n^3/S})$.*

Proof. We first present a simple divide-and-conquer algorithm $\mathcal{A}$ for solving SetCollision. The algorithm $\mathcal{A}(\ell, s, s')$, which find the collisions between two intervals of length l, is described recursively as Algorithm 2. For the sake of simplicity we assume that ℓ is a power of 2.

Algorithm 2 Divide-and-Conquer Algorithm $\mathcal{A}(\ell, s, s')$.

```
if $\ell = 1$ then
    if $a_s = b_{s'}$ then Output $(s, s', a_s)$;
    return.
if ED$(a_s, \dots, a_{s+\ell-1}, b_{s'}, \dots, b_{s'+\ell-1}) = 1$ then return.
let $\ell' \leftarrow \ell/2$;
Sequentially execute $\mathcal{A}(\ell', s, s')$, $\mathcal{A}(\ell', s+\ell', s')$, $\mathcal{A}(\ell', s, s'+\ell')$ and $\mathcal{A}(\ell', s+\ell', s+\ell')$.
```

It is easy to see that $\mathcal{A}(\ell, s, s')$ outputs all the collisions between the two intervals $a[s, \dots, s+\ell-1]$ and $b[s', \dots, s'+\ell-1]$, since whenever $\ell > 1$ and there exists at least one collision (which is checked by the ED call), the algorithm splits each interval into two halves, and solve all four pairs of sub-intervals with the four recursive calls. Hence $\mathcal{A}(n, 1, 1)$ solves SetCollision.

The space usage of $\mathcal{A}(n, 1, 1)$ is $\widetilde{O}(1)$, since there are $O(\log n)$ levels of recursion and each recursive call locally uses $\widetilde{O}(1)$ space. To bound the time usage, the key observation is that there are at most n collisions. Therefore, although there could be as much as $(n/\ell)^2$ possible recursive calls to $\mathcal{A}$ at the level of recursion with interval length ℓ, there are in fact at most $O(n)$ actual calls within each level, while the rest are prematurely stopped because of the ED check. Taking the summation over $\ell = 2^t$ for $t = 0, \dots, \log n$, the total time usage of $\mathcal{A}(n, 1, 1)$ bounded by

$$\sum_{t \le \frac{1}{2}\log n} O(n) \cdot \widetilde{O}(2^t) + \sum_{t > \frac{1}{2}\log n} \left(\frac{n}{2^t}\right)^2 \cdot \widetilde{O}(2^t) = \widetilde{O}(n^{3/2}).$$

When the space S is larger, in order to leverage the space advantage and reduce the time usage we need to *parallelize* the algorithm $\mathcal{A}$. However, the core of algorithm $\mathcal{A}$ is the black-box ED algorithm, whose instances cannot be parallelized if they are highly adaptive. Therefore from now on, we assume that the space-$\widetilde{O}(1)$ and time-$\widetilde{O}(n)$ ED algorithm is oblivious.

To understand how oblivious ED algorithm helps parallelization, consider the recursion level with $\ell = \sqrt{n}$. At this level, we need to answer $\mathrm{ED}(a_s, \ldots, a_{s+\ell-1}, b_{s'}, \ldots, b_{s'+\ell-1})$ for all n pairs of $s, s' \in \{1, \sqrt{n}+1, \ldots, n-\sqrt{n}+1\}$. We can call the oblivious ED algorithm to solve the instance with $s = s' = 1$, and call it again to solve another instance with $s = \sqrt{n}+1, s' = 1$. Because the algorithm is oblivious, whenever a_i (resp. b_i) is queried in the first instance, $a_{i+\sqrt{n}}$ (resp. b_i) is queried in the second instance at the exact same time step. That means the two algorithm instance can be interleaved, using double the space while the queries to B do not need to be repeated. Take a step further, we can interleave the 4 instances of ED with $s, s' \in \{1, \sqrt{n}+1\}$, using 4 times the space but only *double* the time.

In our actual algorithm, we partition $\{1, \sqrt{n}+1, \ldots, n-\sqrt{n}+1\}$ into $\sqrt{n/S}$ groups, each of size $\sqrt{S}$. With the idea stated above, for each pair of groups of s and s', we can solve all the ED instances within this pair (there are S instances) with space $\widetilde{O}(S)$ and time $\widetilde{O}(\sqrt{nS})$. As there are n/S pairs of groups, the overall time usage all the ED instances at level $\ell = \sqrt{n}$ is $\widetilde{O}(\sqrt{n^3/S})$.

More generally, using the same idea, we design a parallelized version of $\mathcal{A}$, which is the algorithm $\mathcal{A}^*(\ell, (s_i, s'_i)_{i\in I})$ described as Algorithm 3 below, that takes as an argument a list of $|I| \le S$ pairs of s and s'.

Algorithm 3 Parallelized Divide-and-Conquer Algorithm $\mathcal{A}^*(\ell, (s_i, s'_i)_{i\in I})$.

```
if $\ell = 1$ then
    for $i \in I$ do
        if $a_{s_i} = b_{s'_i}$ then Output $(s_i, s'_i, a_{s_i})$;
    return.
Solve $e_i \leftarrow \mathrm{ED}(a_{s_i}, \ldots, a_{s_i+\ell-1}, b_{s'_i}, \ldots, b_{s'_i+\ell-1})$ for all $i \in I$ in parallel;
let $\ell' \leftarrow \ell/2$, $Q \leftarrow \varnothing$;
foreach $i \in I$ such that $e_i = 0$ do
    for $(\Delta s, \Delta s') \leftarrow (0,0), (\ell',0), (\ell',0), (\ell',\ell')$ do
        Add $(s_i + \Delta s, s'_i + \Delta s')$ to the queue $Q$;
        if $|Q| = S$ or reaching the end of the algorithm then
            Execute $\mathcal{A}^*(\ell', Q)$;
            $Q \leftarrow \varnothing$.
```

It is clear from the description that $\mathcal{A}^*(\ell, (s_i, s'_i)_{i\in I})$ functions the same as the sequential execution of $\mathcal{A}(\ell, s_i, s'_i)$ for all $i \in I$. Our final algorithm for SetCollision is to run sequentially $\mathcal{A}^*(\sqrt{n}, G \times G')$, for all G and G' chosen from the $\sqrt{n/S}$ groups of size $\sqrt{S}$ that partitions $\{1, \sqrt{n}+1, \ldots, n-\sqrt{n}+1\}$, and thus it correctly outputs all collisions between set A and B. Each recursive call of $\mathcal{A}^*$ uses $O(S \log n)$ space locally, plus the $\widetilde{O}(S)$ space to compute at most S instances of ED in parallel. As there are $O(\log n)$ levels of recursion, the overall space usage is $\widetilde{O}(S)$.

To bound the time usage, we first examine how much time is used to solve S instances of ED in parallel. Fix the initial argument G and G' at the start of the recursion and focus on one level of recursion with interval length ℓ. At this level, one instance of the ED algorithm takes $\widetilde{O}(\ell)$ time. Since the input intervals for these ED instances are either the same or disjoint, each query is repeated for at most $|G| \cdot \sqrt{n}/\ell$ times at its parallel places after the interleaving parallelization. Thus the time usage for solving ED is $\widetilde{O}(|G| \cdot \sqrt{n}) = \widetilde{O}(\sqrt{nS})$. As the rest of steps take $O(S) \le O(\sqrt{nS})$ time, altogether each recursive call of $\mathcal{A}^*$ locally takes $\widetilde{O}(\sqrt{nS})$ time, regardless of the level of recursion.

On the other hand, let us call a recursive call $\mathcal{A}^*(\ell, (s_i, s'_i)_{i\in I})$ *complete* if $|I| = S$, and *incomplete* if $|I| < S$. Since there are at most n collisions, at each level of the recursion there are at most $O(n/S)$ complete calls, while each call produces at most one incomplete call in the next level. Initially there are n/S calls, and therefore the total number of calls to $\mathcal{A}^*$ in our final algorithm is $\widetilde{O}(n/S)$. So the total running time is $\widetilde{O}(\sqrt{n^3/S})$. ◀

Since SetCollision has the randomized lower bound $T^2S = \widetilde{\Omega}(n^3)$, Theorem 16 implies that any polynomial separation between randomized and deterministic time-space tradeoffs of SetCollision (or its variants such as SetIntersection) would answer Open Problem $\star$ on ElementDistinctness.

Notice that in the reduction of Theorem 16, the input guarantee that both lists A and B are sets is only used so that ED decides the distinctness between the two lists. Without the guarantee, we can instead resort to the ListDistinctness problem studied in [4].

▶ **Definition 17.** *In the ListDistinctness problem (LD for short), the input contains two unordered lists $(a_1, \dots, a_n)$ and $(b_1, \dots, b_n)$ from a fixed domain D, with $|D| = \mathrm{poly}(n)$. The output is 1 if there exist $i, j \in [n]$ such that $a_i = b_j$, and 0 otherwise.*

LD is at least as harder as ED, and while ED can be solved in $\widetilde{O}(1)$ space and $\widetilde{O}(n^{3/2})$ time, no algorithm even with $n^{o(1)}$ space and $n^{2-\Omega(1)}$ time was known for LD. The proof of Theorem 16 can be altered to show that the problem of n-Collision reduces deterministically to LD:

▶ **Definition 18.** *In the n-Collision problem, the input is an unordered lists $(a_1, \dots, a_n)$ of elements in D. The output consists of n distinct collisions, that are triples (i, j, x) such that $i \neq j$ and $a_i = a_j = x$, or all of the collisions if there are less than n of them.*

Strictly speaking, the n-Collision problem is not a function, but rather a relational problem, as the collection of outputted collisions is not uniquely determined. However, a time-space lower bound of $T^2S = \widetilde{\Omega}(n^3)$ is still know for n-Collision [12].

▶ **Theorem 19.** *If LD can be solved deterministically with space $\widetilde{O}(1)$ and time $\widetilde{O}(n)$, then n-Collision can be solved deterministically with space $\widetilde{O}(1)$ and time $\widetilde{O}(n^{3/2})$. Furthermore, if the algorithm for LD is oblivious, then for every $S \leq n$, there is a deterministic (non-oblivious) algorithm that solves n-Collision with space $\widetilde{O}(S)$ and time $\widetilde{O}(\sqrt{n^3/S})$.*

Proof. Notice that the collisions found in the algorithms in Theorem 16 are all distinct. By setting a global counter for the number of collisions already found and outputted, the algorithms and proofs in Theorem 16 can be copied verbatim to show a reduction to LD from the problem k-ListCollision, where the input consists of two unordered lists of size n that may contain duplicates, and the output contains k collisions (if exist) between the two lists. If LD can be solved deterministically with space $\widetilde{O}(1)$ and time $\widetilde{O}(n)$, then the deterministic algorithm for k-ListCollision works in space $\widetilde{O}(S)$ and time $\max\{m, n\} \cdot \widetilde{O}(\sqrt{n/S})$, where m is the actual number of collisions outputted (S can be arbitrary when the algorithm for LD is oblivious, and $S = O(1)$ in the general case).

Now notice that the complete graph over n vertices can be partitioned into a set of complete bipartite graphs, 2^{t-1} of which being of size $(n/2^t, n/2^t)$ for $t = 1, \dots, \log n$. We apply n-ListCollision on each pairs of lists of size $n/2^t$ defined by these bipartite graphs, until n collisions are found. This clearly solves the n-Collision problem with space $\widetilde{O}(S)$. Suppose that the number of collisions actually outputted on each bipartite graph is $m_1, m_2, \dots$ respectively, then the total time usage is

$$\begin{aligned}&\max\left\{m_1,\frac{n}{2}\right\}\cdot\widetilde{O}\left(\sqrt{\frac{n}{2S}}\right)+\max\left\{m_2,\frac{n}{4}\right\}\cdot\widetilde{O}\left(\sqrt{\frac{n}{4S}}\right)+\max\left\{m_3,\frac{n}{4}\right\}\cdot\widetilde{O}\left(\sqrt{\frac{n}{4S}}\right)+\cdots\\ &\leq\ (m_1+m_2+\cdots)\cdot\widetilde{O}\left(\sqrt{\frac{n}{2S}}\right)+\sum_{t=1}^{\log n}2^{t-1}\cdot\frac{n}{2^t}\cdot\widetilde{O}\left(\sqrt{\frac{n}{2^tS}}\right)\quad=\quad\widetilde{O}\left(\sqrt{n^3/S}\right).\quad\blacktriangleleft\end{aligned}$$

Similarly, we have the corollary of Theorem 19 that any polynomial separation between randomized and deterministic time-space tradeoffs of n-Collision (or its variants such as MemoryGame [9]) would answer Open Problem $\star$ on ListDistinctness.

References

1 Scott Aaronson, Shalev Ben-David, Robin Kothari, Shravas Rao, and Avishay Tal. Degree vs. approximate degree and quantum implications of huang's sensitivity theorem. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1330–1342. ACM, 2021. doi:10.1145/3406325.3451047.

2 Karl R. Abrahamson. Time-space tradeoffs for algebraic problems on general sequential machines. *J. Comput. Syst. Sci.*, 43(2):269–289, 1991. doi:10.1016/0022-0000(91)90014-V.

3 László Babai, Noam Nisan, and Mario Szegedy. Multiparty protocols, pseudorandom generators for logspace, and time-space trade-offs. *J. Comput. Syst. Sci.*, 45(2):204–232, 1992. doi:10.1016/0022-0000(92)90047-M.

4 Nikhil Bansal, Shashwat Garg, Jesper Nederlof, and Nikhil Vyas. Faster space-efficient algorithms for subset sum, k-sum, and related problems. *SIAM J. Comput.*, 47(5):1755–1777, 2018. doi:10.1137/17M1158203.

5 Paul Beame. A general sequential time-space tradeoff for finding unique elements. *SIAM J. Comput.*, 20(2):270–277, 1991. doi:10.1137/0220017.

6 Paul Beame, Raphaël Clifford, and Widad Machmouchi. Element distinctness, frequency moments, and sliding windows. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 290–299. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.39.

7 Paul Beame and Widad Machmouchi. Making branching programs oblivious requires superlogarithmic overhead. In *Proceedings of the 26th Annual IEEE Conference on Computational Complexity, CCC 2011*, pages 12–22. IEEE Computer Society, 2011. doi:10.1109/CCC.2011.35.

8 Allan Borodin and Stephen A. Cook. A time-space tradeoff for sorting on a general sequential model of computation. *SIAM J. Comput.*, 11(2):287–297, 1982. doi:10.1137/0211022.

9 Amit Chakrabarti and Yining Chen. Time-space tradeoffs for the memory game. *arXiv preprint*, 2017. arXiv:1712.01330.

10 Lijie Chen, Ce Jin, R. Ryan Williams, and Hongxun Wu. Truly low-space element distinctness and subset sum via pseudorandom hash functions. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 – 12, 2022*, pages 1661–1678. SIAM, 2022. doi:10.1137/1.9781611977073.67.

11 Lijie Chen and Roei Tell. Simple and fast derandomization from very hard functions: eliminating randomness at almost no cost. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 283–291. ACM, 2021. doi:10.1145/3406325.3451059.

12 Itai Dinur. Tight time-space lower bounds for finding multiple collision pairs and their applications. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020 – 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 405–434. Springer, 2020. doi:10.1007/978-3-030-45721-1_15.

13 Venkatesan Guruswami, Christopher Umans, and Salil P. Vadhan. Unbalanced expanders and randomness extractors from parvaresh-vardy codes. *J. ACM*, 56(4):20:1–20:34, 2009. doi:10.1145/1538902.1538904.

14 Yassine Hamoudi and Frédéric Magniez. Quantum time-space tradeoff for finding multiple collision pairs. In Min-Hsiu Hsieh, editor, *16th Conference on the Theory of Quantum Computation, Communication and Cryptography, TQC 2021, July 5-8, 2021, Virtual Conference*, volume 197 of *LIPIcs*, pages 1:1–1:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.TQC.2021.1.

15 William M. Hoza. Typically-correct derandomization for small time and space. In Amir Shpilka, editor, *34th Computational Complexity Conference, CCC 2019, July 18-20, 2019, New Brunswick, NJ, USA*, volume 137 of *LIPIcs*, pages 9:1–9:39. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CCC.2019.9.

16 Hartmut Klauck. Quantum time-space tradeoffs for sorting. In Lawrence L. Larmore and Michel X. Goemans, editors, *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 69–76. ACM, 2003. doi:10.1145/780542.780553.

17 Hartmut Klauck, Robert Špalek, and Ronald de Wolf. Quantum and classical strong direct product theorems and optimal time-space tradeoffs. *SIAM J. Comput.*, 36(5):1472–1493, 2007. doi:10.1137/05063235X.

18 Xin Lyu and Weihao Zhu. Time-space tradeoffs for element distinctness and set intersection via pseudorandomness. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 5243–5281. SIAM, 2023. doi:10.1137/1.9781611977554.ch190.

19 Dylan M. McKay and R. Ryan Williams. Quadratic Time-Space Lower Bounds for Computing Natural Functions with a Random Oracle. In Avrim Blum, editor, *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*, volume 124 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 56:1–56:20, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ITCS.2019.56.

20 Ashley Montanaro. Nonadaptive quantum query complexity. *Inf. Process. Lett.*, 110(24):1110–1113, 2010. doi:10.1016/j.ipl.2010.09.009.

21 John E. Savage. *Models of computation – exploring the power of computing*. Addison-Wesley, 1998.