Time-Space Tradeoffs and Query Complexity in Statistics, Coding Theory, and Quantum Computing

Widad Machmouchi

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2013

Reading Committee:

Paul Beame, Chair

Anna Karlin

Anup Rao

Program Authorized to Offer Degree:
Computer Science and Engineering

University of Washington

Abstract

Time-Space Tradeoffs and Query Complexity in
Statistics, Coding Theory, and Quantum Computing

Widad Machmouchi

Chair of the Supervisory Committee:
Professor Paul Beame
Computer Science and Engineering

Computational complexity is the field that studies the computational resources needed by algorithms to correctly solve computational problems. In this thesis, we derive lower bounds on the computational complexity of different problems for different classes of algorithms. The computational resources we consider are the running time, the memory usage and the input query complexity; i.e., the number of positions in the input that the algorithm reads. While deriving upper bounds involves constructing algorithms that run efficiently in their computational resources, lower bounds for a computational problem establish limits on how fast or space-efficient any algorithm solving that problem can be. We consider two main themes of lower bounds: time-space tradeoffs and query complexity lower bounds. Time-space tradeoffs define a relation between the running time and space of any algorithm solving a certain problems. They give a complete picture of the hardness of a problem by deriving a lower bound on one resource when the other is limited. Query complexity measures the depth of a decision tree computing the problem. We derive several lower bounds and upper bounds for different problems on different types of classical computational models and on quantum computers, where the queries to the input are represented by quantum operations operating on the qubits of the algorithm:

- We derive the largest time-space tradeoff known for a randomized algorithm solving an explicit problem. We consider the pointer jumping problem, also known as the outdegree-1

graph accessibility problem and derive a $T \in \Omega(n \log(n/S) \log\log(n/S))$ time-space trade-off for any randomized oblivious algorithm computing it. Oblivious algorithms have the property that the sequence of operations performed is independent of the value of the input. We derive a similar lower bound for randomized oblivious algorithms with boolean inputs.

- Frequency moments and order statistics are attributes computed on a sequence of numbers and are extremely relevant in various data analysis settings. We consider computing several copies of these functions on overlapping sequences of numbers by considering a window of a fixed length and sliding it over the input; these versions are called *sliding-window* versions of these functions. We give hardness separations between the sliding-window versions of these statistics. On one hand, we give a $TS \in \Omega(n^2)$ lower bounds for any algorithm computing sliding-window frequency moments. To complement our lower bound, we give a $TS \in \tilde{O}(n^2)$ comparison-based algorithm for sliding-window frequency moments. On the other hand we describe a $T^2S \in \tilde{O}(n^3)$ algorithm for sliding-window element distinctness. For order statistics, we derive a similar lower bound of $TS \in \Omega(n^2)$ for any algorithm computing a sliding-widow version of the median, while for the sliding window version of maximum (and minimum), we construct an $O(n \log n)$-time, logarithmic space algorithm.

- We study the increase in complexity on quantum computers when considering sliding-window frequency moments. We show that quantum computers do not require the same increase in complexity as their classical counterparts by constructing a quantum algorithm for sliding-window version of the zeroth frequency moments running in time $O(n^{3/2})$ and logarithmic space, thus beating the classical lower bound for this problem.

- In the area of error-correcting codes, we study the query complexity of error detection and correction algorithms. We prove that linear codes over large alphabets that are small in size and concentrated in terms of their codeword weights have testers and correctors that require only a constant number of queries to a word received across a noisy channel. Such codes are called *locally-testable* and *self-correctable*. As random codes of small size have the property

that their codeword weights are concentrated in a small range of values, they are also locally-testable and self-correctable with high probability.

- For linear codes with information-efficient encoding, we study the tradeoff between the time and space of their encoders, and their error-correction property. We derive properties on generator matrices of asymptotically good linear codes that guarantee an optimal time-space tradeoff of $TS \in \Omega(n^2)$ for any algorithm encoding them. Moreover, we prove a first step in an approach to establish these properties for all generator matrices of such codes.

- To compare classical computers to their quantum counterparts, we consider the query complexity of quantum algorithms computing frequency moments and the ONTO function that takes as input a function and determines whether it is surjective. We prove that any quantum computer computing the ONTO function or the parity of the zeroth frequency moment requires a linear number of queries to the input. Since the ONTO function with boolean inputs has a simple polynomial-size, constant-depth circuit, we obtain an almost linear lower bound on the query complexity of AC^0, a special class of circuits that have polynomial size, constant depth and unbounded fan-in for their gates.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

First and foremost, this work would not have been possible without my advisor Paul Beame. His mentoring, good advice, and encouragement throughout my years at UW were invaluable for my academic development. His extensive knowledge and creativity provided me with the right guidance through research paths that were too windy to lead anywhere or had the right idea to pursue. I am grateful for all the effort and care he showed me during the Princeton visit and advising me during the years where Venkat was away.

I am also thankful for Venkat Guruswami for advising me through my first few years at UW. Although we did not work together for long, I was lucky to experience his enthusiasm and brilliance and thankful for all the guidance he offered me. I would like to thank Anup Rao and Anna Karlin for their comments and for Rekha Thomas and Dan Suciu for being on my committee. I am in debt to Aram Harrow for all the discussions and explanations he offered when quantum effects stumped me and for being a great mentor during the last few years. My experience with research was that much richer after working with Raphaël Clifford during the Spring and Summer of 2012. Raphaël, thank you for being a great researcher to work with, thank you for all the advice and fun times we had together. You will remain a lifelong friend.

I am glad I got to share my PhD years with some of the best theory students who will remain lifelong friends of mine. From Theory night to the Theory retreat and recently to Theory lunch, it was great to hang out with such inspiring minds and fun personalities. Thank you for lending an ear when research got tough and helping me work through some hairy calculations. Everyone in the Theory group contributed to my academic and personal life and I will always be proud to have been a member. Thank you for every member, past and present.

My interest in Theory research was sparked by the wonderful Theory courses taught by my undergrad advisor Louay Bazzi at the American University of Beirut. I am thankful for his guidance

and mentoring with my final year project and applying to graduate schools. To Ghid, my partner in crime, I am glad we got to share a similar path in graduate school as we did in undergrad.

Being away from home and family was very difficult and I am thankful for all my Seattle friends that became like family to me, in chronological order: Rayyan, Elisa, Kira, Shayan, Hiba, Lukas, Layali Dabkeh group and ending with the Lebuget group.

Finally, to Dad, Mom, Nisrine, Nadine, Ayman and the latest two additions, Cecile and Ahmad, thank you for the support you provided and the love you sent my way while I was away from home. You were there when I needed sweet comforting words and when I needed motivation and encouragement. My extended family also played a big positive role in shaping who I am today. I hope I made all of you proud.

Widad Machmouchi

April 2013

DEDICATION

to my family, born into and acquired

Chapter 1

INTRODUCTION

Computational problems are ubiquitous in life, from finding shortest paths between two points, identifying patterns in data, to sorting sequences of numbers and analyzing their properties. Constructing efficient algorithms for such problems, both in terms of running time and storage, has always been a central goal in the design of algorithms. The computational efficiency of algorithms became more essential with the emergence of large data sets available to researchers with the advance of technology. A multitude of clever and computationally efficient algorithms have been created across different disciplines, from computer science, mathematics, physics, to biology, economics and finance. This huge success in designing algorithms and the quest for finding more efficient ones leads to the natural question: how computationally efficient can an algorithm be for a specific problem? Establishing limits and lower bounds on the resources for algorithms helps us answer that question and defines a stopping point for efforts invested towards finding more efficient algorithms. Such lower bounds give a deeper understanding of the difficulty of a problem and allow us to classify it with respect to other problems in terms of their computational hardness.

Unfortunately, the rich literature of efficient algorithms is unmatched in the area of lower bounds. While computing upper bounds for a problem involves analyzing the complexity of a single algorithm, lower bounds should hold for **all** the algorithms solving the problem. Deriving lower bounds on the computational resources needed to solve that problem requires thinking of an algorithm as an adversary. One should prove that any conceivable algorithm, regardless of how clever or unintuitive it is, requires a lot of resources to solve a certain problem, thus making finding lower bounds a difficult task. In this thesis, we consider two main aspects of lower bounds: *time-space tradeoffs* and *query complexity* lower bounds and study them in the areas of statistics, error-correcting codes, and quantum computing.

1.1 Time-Space Tradeoffs

The first lower bound theme we consider is time-space tradeoffs. Time-space tradeoffs give a relationship between the time and space usage of an algorithm: limiting one resource requires more of the other. Such a tradeoff gives a more complete picture of the complexity of a problem compared to separate lower bounds on the time or space alone, thus setting a landscape of all the algorithms solving the problem. Therefore, for every amount of storage allowed in a certain computational setting, the time-space tradeoff gives a lower bound on the running time of any algorithm solving the problem within that space restriction. Moreover, one of the important questions in complexity theory that is yet to be answered is whether polynomial time is equivalent to logarithmic space. Deriving space lower bounds for problems in P when a polynomial upper bound is enforced on the time can be seen as a first step towards that goal.

The computational model we use to derive time-space tradeoffs is the *branching program* model. Branching programs are a general sequential model of computation that are able to simulate Turing machines and random-access machines. Although those machines are the standard models used to construct algorithms, they are painful to use when deriving lower bounds since their computations involves many tedious details specific to the structure. In contrast, the underlying structure of a branching program is a simple directed rooted acyclic graph where each node represent the state of the machine at a given time in the computation. The time and space of a branching program are inherent in the graph; in fact, the time is the length of the longest path from the root to a sink (node with out-degree 0) and the space is the logarithm base 2 of the number of nodes in the graph. Moreover, since each state of the computation is encoded in a node, once that node is reached, no past information is needed to finish the computation. The next step is solely dependent in the input position read in that node. Thus, each node in the branching program can be viewed as the root of a smaller branching program performing some computation in the input. This makes branching programs a well-suited model to measure the progress of the computation in producing the output of functions.

Branching programs have been extensively used to derive time-space tradeoffs for a variety of problems: sorting (Borodin and Cook [43], Beame [34]), element distinctness (Ajtai [7], Beame, Saks, Sun and Vee [32]), matrix-vector product, matrix multiplication and convolution (Yesha [158],

Abrahamson [3, 5]), hashing (Mansour, Nisan and Tiwari [115]), and other explicit boolean functions (Ajtai [8], Beame, Saks, Sun and Vee [32]). The largest time-space tradeoff one can obtain on a branching program of binary input length n is $TS \in \Omega(n^2)$ since a full tree where each branch represents a possible input and each leaf represents the output of the function, has time n and space n.

Time-space tradeoff lower bounds for branching programs computing single-output functions are still very far from the $TS \in \Omega(n^2)$ time-space tradeoff possible on a branching program. Getting an $\Omega(n^{1+\epsilon})$ lower bound, for a positive ϵ, on the time-space tradeoff of an explicit bolean function has been an open problem for more than three decades. In fact, the largest time-space tradeoff on a general branching program is $T \in \Omega(n \log^2 n)$, when the space is in $O(n^\epsilon \log n)$ due to Beame and Vee [33] and $T \in \Omega(n \log(\frac{n \log n}{S}))$ for all values of T and S, due to Beame, Saks, Sun and Vee in [32]. The latter results also hold for randomized branching programs. Better time-space tradeoffs were obtained for a special class of branching programs called *oblivious branching programs*. *Oblivious* algorithms have the property that the sequence of operations performed is independent of the value of the input. Babai, Nisan and Szegedy [22] obtained a $TS = \Omega(n \log^2 n)$ time-space tradeoff for an explicit function computed on a deterministic oblivious branching program. However, this lower bound does not apply to randomized branching programs.

To improve these tradeoffs, we study time-space tradeoffs for randomized oblivious computations. The oblivious property makes these algorithms simpler to analyze and thus provides a first step towards time-space tradeoffs for randomized general algorithms. Moreover, these algorithms are useful in settings where the access to the input is predetermined, for example in *external sorts* [99]. In this case, the input to be sorted is stored in an (slow) external memory and the algorithm can transfer pre-determined small pieces at a time and perform operations on them. Oblivious computation is also essential in cryptographic settings where adversaries can learn information about the inputs by observing the behavior of the internal algorithms of the communicating parties. One can convert any algorithm into an oblivious one by simply querying all the input positions every time it needs to query the input at a specific position. If the original algorithm runs in time T, the oblivious one will run in time $O(nT)$. There are more efficient simulations that take a multi-tape Turing machine running in time T and convert it to an oblivious 2-tape Turing machine that runs

in time $O(T \log_2 T)$ [124]. Unfortunately, converting a 2-tape Turing machine to a single-tape one causes a quadratic increase in time, thus matching the running time of the naive algorithm described above. The problem we consider is the *out-degree-1 graph accessibility problem*, also known as the *pointer-jumping problem* and denoted $1GAP$. The graph accessibility problem comes up in a host of different areas from network and graph connectivity to parallel computation [64]. It has a very natural and efficient algorithm that is inherently non-oblivious: simply follow the edges from the start node until a node is revisited or the sink node is reached. The best oblivious algorithm known so far is the naive one that queries all input positions every time it needs to query a specific position, thus incurring a running time quadratic in the number of nodes and a logarithmic space. Moreover, $1GAP$ has a $T \in \Omega(n \log^2(n/S))$ time-space tradeoff on any deterministic oblivious branching program derived by Beame and Vee in [33] using the results of [22] . This leads to the following natural question:

Problem 1.1. *Are there more efficient randomized oblivious algorithms for* $1GAP$ *and what are the right time-space tradeoffs on oblivious branching programs?*

For time-space tradeoffs on general computations, we consider computing statistics on a sequence of numbers, namely the frequency moments, order statistics and the element distinctness problem. These quantities are an important aspect of data analysis and encapsulate significant information about the underlying data set. For example, the *Simpson index* [140] in ecology and the *Herfindahl-Hirschman index* [84, 83] in economics use the second frequency moment to measure the type diversity of the data set considered. The zeroth frequency moment, F_0, is the number of distinct elements in the sequence of numbers considered and the element distinctness problem decides whether all the numbers are distinct. Since these quantities can be easily computed by a sorting algorithm, it is natural to ask whether they satisfy the same time-space tradeoff as sorting: a tight $TS \in \Omega(n^2)$ [43, 34] lower bound matched by a comparison algorithm [123] for all $S \in O(n/\log n)$. The element distinctness problem was thought to be decision problem that can track the hardness of the sorting problem, and despite a $TS \in \Omega(n^{2-o(1)})$ time-space tradeoff for any comparison-based algorithm, the best lower bound known for the element distinctness problem, and hence frequency moments on general branching programs is $T \in \Omega(n\sqrt{\log(n/S)/\log\log(n/S)})$ derived in [32].

Since deriving lower bounds for single-output functions is a notoriously hard task, as a first step, we consider computing several copies of these functions on overlapping sequences of numbers by considering a window of a fixed length and sliding it over the input; these versions are called *sliding-window* functions. Sliding windows are prevalent in *data streaming* settings where the algorithm gets access to one symbol at a time from the stream and the purpose is to compute a function on subsets of the stream as the symbols arrive with the least amount of storage. Sliding-window statistics are relevant in analyzing time-series observations over short intervals of time, which arise in financial asset modeling for example.

From a computational point of view, sliding windows also offer an alternative to *direct sums*, where one studies the amount of resources needed to compute several copies of the function on non-overlapping and independent inputs. Direct sum theorems state that to compute k copies of a function, one requires k times the resources needed to compute one copy of the function. One of the drawbacks of direct sums is the increase in input size when k different inputs to the functions are provided. The new input length grows linearly with k, thus making the increased factor in resources possibly smaller than k with respect to the new input length. In sliding windows, the input size stays of the same order since the inputs in the different windows have large overlap. Hence, we consider sliding-window versions of statistics related to sorting to separate them in terms of computational hardness, leading to the question:

Problem 1.2. *How does the computational complexity of frequency moments, element distinctness and order statistics compare over sliding-windows?*

In the light of the $TS \in O(n^{3/2} \log n)$ quantum algorithm that Klauck developed for sorting [96], we consider the complexity of sliding windows frequency moments on quantum computers and compare it to their classical complexity:

Problem 1.3. *What is the quantum complexity of computing sliding-window frequency moments?*

Our final time-space tradeoff applies to communication settings with errors. Error-correcting codes are combinatorial objects used in communication to provide error correction and detection across noisy channels. A message to be communicated across the channel is encoded using that code to produce a word that is longer in length, called *codeword*. Other than their use in commu-

nication, error-correcting codes are essential tools in different areas of computer science such as pseudorandomness and error-resilient database systems. The two main important aspects in designing these codes are their information-efficiency and error-resilience. One way to formalize these notions is with the two parameters, the *rate* and *minimum distance*. The rate is the ratio of the length of the message encoded to that of a codeword and the minimum distance is the minimum distance between any two codewords in the code. These are opposing parameters in the sense that to improve the error resilience of the code, one needs to add more redundancy to the encoded word; however this added redundancy will decrease the information content in the encoded message with respect to its length. Therefore, it is necessary to construct codes that have the right balance between those two parameters: a positive rate and a minimum distance that is linear in the length of the codeword. Such codes are called *asymptotically good*. However, in some constructions, the minimum distance of a code is hard to estimate directly from its description. The complexity of the encoding (the process of taking a message and mapping it to an encoded word) has been used to estimate their minimum distance. In fact, in a surprising result, Bazzi and Mitter [26] proved that a code with positive rate necessarily has a sublinear minimum distance if the encoding runs in linear time and sublinear space. Hence, a relationship between the rate and minimum distance, and the computational efficiency of encoding would helps us identify asymptotically good codes by simply examining the resources used by their encoders. Such a relationship would answer the following question:

Problem 1.4. *How computationally efficient can the encoding be for a code with positive rate and linear minimum distance?*

1.2 Query Complexity Lower Bounds

The second theme in this thesis is query complexity. In this setting, we restrict the algorithm's access to the input and measure how many input positions it queries. The query complexity of a function is the depth of a decision tree computing it. This can be used as a proxy to the running time when the number of positions read from the input is smaller than the input length. To model the restricted access of the algorithm to the input, the input is provided as an oracle that the algorithm can query with a position $i \in [n]$ and the oracle returns the input symbol at that position.

We study the query complexity of error detection and correction algorithms for error-correcting codes defined over large alphabets. In this setting, an algorithm queries symbols from a word received across the channel and the purpose is to test whether it is a valid codeword, or compute an individual symbol from the codeword closest to it. Ideally, we would like the algorithm to only query a constant number of positions in the received word. Codes having algorithms that test membership and correct individual errors with a constant number of queries are called *locally-testable* and *self-correctable*. The locality of the algorithms make these codes very useful in various areas of theoretical computer science, e.g. *Probabilistically Checkable Proofs* [20] where one wishes to verify an alleged proof for a claim by only reading a constant number of positions in the proof, and *Private Information Retrieval Schemes* [56], where one wishes to learn the information at a specific position in a database without revealing that position. one of the main unanswered questions in local-testability and self-correctability is whether there exist codes with positive rate that have these properties. All the known constructions of locally-testable and self-correctable codes have rates going to 0, as their block length tends to infinity. To characterize the role of the rate in the local-testability and self-correctability in general codes, one can ask:

Problem 1.5. *What are the necessary and sufficient conditions for codes over large alphabets to be locally-testable and self-correctable?*

We also study query complexity in the context of quantum computing. Quantum computers are theoretical models of computation that simulate quantum physics and promise large speed-ups in running time, compared to their classical counterparts. With the multitude of attempts at creating such a quantum computer, the need arises to study the computational power of these machines and their limits. The computational resource often used to measure the efficiency of a quantum computer is its input query complexity. Query complexity turns out to be the main aspect to optimize in the design of many quantum algorithms, as reflected in the currently known quantum algorithms: Simon's algorithm for finding the period of a function [139], Shor's algorithm for factoring and discrete log [138], and Grover's search algorithm [77]. These algorithms offer significant speed-ups over their classical counterpart and it is natural to ask which functions do not gain any significant speed-ups when computed by a quantum algorithms. Several lower bounds have been derived on the quantum query complexity for different problems. Bennett, Bernstein, Brassard and Vazirani [38]

derived an $\Omega(\sqrt{n})$ lower bound on computing OR, thus proving that Grover's search algorithm is optimal. Aaronson and Shi derived an $\Omega(n^{2/3})$ lower bound for the quantum query complexity of element distinctness [1] and Brassard and Høyer and Tapp derived an $\Omega(n^{1/3})$ lower bound for the collision problem [47]. However, these (tight) lower bounds are significantly smaller than the linear number of queries required for a classic algorithm. Moreover, these problems can be computed by AC^0 circuits. These are polynomial-size constant-depth unbounded fan-in circuits that have limited computational power. On the other hand, Beals, Buhrman, Cleve, Mosca and de Wolf [27] proved that parity, a problem that cannot be computed by an AC^0 circuit, requires $n/2$ queries to its inputs to be computed by a quantum algorithm. Hence, it is natural to ask:

Problem 1.6. *What functions computed by* AC^0 *circuits require a linear number of queries on a quantum algorithm?*

1.3 Overview of the Results

We present the contributions in this thesis by area, starting first with time-space tradeoffs for randomized oblivious branching programs and for sliding-window statistics, then lower bounds and constructions in coding theory and finally query complexity lower bounds in quantum computing.

1.3.1 Time-Space tradeoffs for randomized oblivious branching programs

We partially answer Problem 1.1 and prove a time-space tradeoff lower bound of $T \in \Omega\left(n \log(\frac{n}{S}) \log\log(\frac{n}{S})\right)$ for randomized oblivious branching programs to compute $1GAP$. Along with the simple deterministic time n and space $O(\log n)$ RAM (random access machine) algorithm for $1GAP$, this time-space tradeoff gives a separation between randomized oblivious and general branching programs.

We also give a similar time-space tradeoff of $T \in \Omega\left(n \log(\frac{n}{S}) \log\log(\frac{n}{S})\right)$ for Boolean randomized oblivious branching programs computing $GIP\text{-}MAP$, a variation of the generalized inner product problem that can be computed in time n and space $O(\log^2 n)$ by a deterministic Boolean branching program.

These are the first lower bounds for randomized oblivious branching programs computing explicit functions that apply for $T \in \omega(n \log n)$. They also show that any simulation of general

branching programs by randomized oblivious ones requires either a superlogarithmic increase in time or an exponential increase in space.

These results appeared previously in the joint work with Paul Beame [30], published in the *Proceedings of the IEEE Conference on Computational Complexity, 2011*.

1.3.2 Time-Space tradeoffs for sliding-window frequency moments

We derive new time-space tradeoff lower bounds and algorithms for exactly computing statistics of input data, including frequency moments, element distinctness, and order statistics, that are simple to calculate for sorted data. In particular, we develop a randomized algorithm for the element distinctness problem whose time T and space S satisfy $T \in \tilde{O}(n^{3/2}/S^{1/2})$, smaller than previous lower bounds for comparison-based algorithms, showing that element distinctness is strictly easier than sorting for randomized branching programs.

We further show that our element distinctness algorithm can be extended at only a polylogarithmic factor cost to solve the element distinctness problem over sliding windows [59], where the task is to take an input of length $2n - 1$ and produce an output for each window of length n, giving n outputs in total.

In contrast, we show a time-space tradeoff lower bound of $T \in \Omega(n^2/S)$ for randomized multiway branching programs, and hence standard RAM and word-RAM models, to compute the number of distinct elements, F_0, over sliding windows. The same lower bound holds for computing the low-order bit of F_0 and computing any frequency moment F_k for $k \neq 1$.

We complement this lower bound with a $T \in \tilde{O}(n^2/S)$ comparison-based deterministic RAM algorithm for exactly computing F_k over sliding windows, nearly matching both our general lower bound for the sliding-window version and the comparison-based lower bounds for a single instance of the problem. We also consider the computations of order statistics over sliding windows.

The above lower and upper bounds answer the question posed in Problem 1.2 by proving a separation between frequency moments and their parities, and the element distinctness problem in sliding windows. These results were joint work with Paul Beame and Raphaël Clifford [28] and under review for publication in the 54^{th} *Annual IEEE Symposium on Foundations of Computer Science*.

Moreover, we show that quantum algorithms have an advantage over their classical counterparts. We do so by construction an $O(n^{3/2})$ time, logarithmic space quantum algorithm that computes sliding-window frequency moments, thus beating the $TS \in \Omega(n^2)$ classical time-space tradeoff.

1.3.3 Local Testability of sparse linear codes over large alphabets

We prove that linear codes over a finite field $\mathbb{F}_q$ of polynomial size (and hence, vanishing rate) and codeword weights concentrated around $\left(1 - \frac{1}{q}\right) n$ are locally testable and self-correctable. This result partially resolves Problem 1.5 by deriving two sufficient conditions for the local-testability and self-correctability of codes over large alphabets. It also implies that sparse random linear codes also are locally testable and self-correctable, since they satisfy these two conditions with high probability. Moreover, as an attempt to prove the necessity of the vanishing rate for these properties to hold for general codes, we show that random codes that have superpolynomial size and linear minimum distance cannot be tested or corrected with a constant number of queries, with high probability.

These results were published previously [112] in the *Proceedings of the 2010 IEEE International Symposium on Information Theory*.

1.3.4 Time-space tradeoffs for encoding asymptotically good codes

We describe an approach to deriving an optimal $TS \in \Omega(n^2)$ time-space tradeoff for branching programs encoding asymptotically good linear codes. We derive properties of the generator matrix that are sufficient to for this tradeoff to hold. As a first step to establish these properties in the generator matrix of an asymptotically good code, we prove that any such code contains an asymptotically good code that has no low-dimension projections. This result was obtained jointly with Shachar Lovett during a visit to the Institute of Advanced Studies, Princeton during Winter 2011. We formulate a conjecture regarding the generator matrix, which, if proven, would answer the question of Problem 1.4.

1.3.5 Quantum query complexity lower bounds

We construct a function computed by a simple AC^0 circuit that requires a linear number of queries for a quantum algorithm to compute it correctly with high probability. This lower bound implies an almost linear $\Omega(n/\log n)$ lower bound on the quantum query complexity of AC^0, and thus answering Problem 1.6. These results were obtained jointly with Paul Beame [31] and published in the journal of *Quantum Information & Computation*, volume 12, number 7-8, pages 670-676.

Our lower bound implies also an $\Omega(\frac{n^2}{\log^2 n})$ lower bound on the formula size of the same AC^0 function that improves on the Krapchencko's method for formula size lower bound. Moreover, as the parity of F_0 is a special case of our AC^0 function, we get a separation between the parity of F_0 and the element distinctness problem in the quantum query model.

1.4 Organization of the Thesis

This thesis is organized in three parts. The first part studies time-space tradeoffs derived in the branching program model. In Chapter 2, we introduce and motivate branching programs, along with a high-level view of the techniques for deriving time-space tradeoffs. Our time-space tradeoff for randomized branching programs is presented in Chapter 3. Time-space tradeoffs for sliding-window statistics along with their algorithms are explained in Chapter 4.

The second part focuses on error-correcting codes. We present the needed background from coding theory in Chapter 5 and derive conditions for local-testability and self-correctability of sparse linear codes in Chapter 6. In Chapter 7, we present our approach to get optimal time-space tradeoffs for encoders of asymptotically good codes.

Finally, in the third part, we derive our lower bounds on the quantum query complexity of AC^0 and our algorithm for sliding-window frequency moments.

Part I

TIME-SPACE TRADEOFFS

Chapter 2

COMPUTATIONAL MODELS

In 1936, Alan Turing [146] introduced the Turing machine as a universal model of computation that is able to simulate any computational procedure. Turing machines formalized the notion of an algorithm. The resources it uses in its computation were formalized by Hartmanis and Stearns in [82]. Such resources are the running time, i.e the number of basic operations (these are operations that are done in one time step with respect to the computational method used) performed by the algorithm and the space, i.e. the number of memory units used (read and written) by the algorithm.

Random-access machines (RAMs) are an abstract model of computation that was introduced by Cook and Reckhow [57] in 1972. RAMs are machines consisting of a collection of registers with two special registers, the *control unit* where the description of the machine is stored and the *program counter*, which determines which instruction to be executed next. They have since been used as the main model of computation to develop algorithms and to study their time and space complexity. They are able to simulate any Turing machine with roughly the same resources. Moreover, they offer random-access to the input, as opposed to the sequential access of the head of a Turing machine. The computer science literature is very rich with fast and inventive algorithms, with a wide range of applications. This success in designing algorithms lead to the natural questions: How efficient can a RAM be in its time and space usage? Are there problems that RAMs cannot solve efficiently?

The resources used by an algorithm are measured in terms of its input length. Thus the time and space complexity are represented as functions $T : \mathbb{N} \to \mathbb{R}$ and $S : \mathbb{N} \to \mathbb{R}$ respectively, such that $T(n)$ and $S(n)$ are the number of time steps and units of memory, respectively, that the algorithm uses in its computation. When considering space complexity, we divide the memory usage of an algorithm into 3 main types. The *input* space is the space used to store the input to the algorithm and is considered read-only; i.e, the algorithm can access any memory unit in that space and read it but not write to it. The *working* space is the space used by the algorithm to perform its mathematical operations. The *output* space is the space where the algorithm produces its outputs and is considered

write-only; i.e., the algorithm can access any memory unit to write the value of an output but not read from it. As the input space and output space are determined by the problem to solve, we derive bounds on the working space of the algorithm, thus allowing for space usage to be sublinear in the input length.

The task of deriving complexity lower bounds takes a different approach compared to designing algorithms. To get a desired lower bound, one should prove that *every* algorithm solving the problem should use at least the amount of resources dictated by the lower bound. That seems like a hard task as many algorithms can be clever and counter-intuitive. Evidently, the field of computational complexity has had far fewer successes in finding lower bounds on problems compared with finding efficient algorithms for them.

Although one might want to derive lower bounds on a single resource, this approach might not give a complete picture about the difficulty of a problem. Consider for example the problem of sorting n integers: Insertion Sort runs in $\Theta(n^2)$ time in the worst case, but use a logarithmic amount of working space memory; Merge Sort, despite running in $O(n \log n)$ time in the worst case, requires $\Theta(n \log n)$ of working space. Therefore, time-space tradeoffs that gives a relation between the space and time of an algorithm are able to better capture the complexity of a problem.

Moreover, when deriving lower bounds on time and space, one should use a computational model that is sufficiently general so that the lower bounds reflect an intrinsic property of the problem, and not restrictions on the power of the model itself. Such a model should have a composition that reflects the resources it uses, mainly time and space. In fact, lower bounds were derived for the sorting problems on Turing machines with one input head and *straight-line programs* [118, 145]. Straight-line programs are a model of computation such that the sequence of operations is independent of the value of the input; it is "oblivious" to that order. The sequential access to the input in a Turing machine and the obliviousness of straight-line programs represent restrictions of the computational model, rather than a hardness of the problem. Although RAMs offer random-access to the input and have unstructured computation, their computational complexity is hard to analyze by just inspecting their description. Therefore, the *branching program*, a general unstructured sequential model of computation with random-access to the input was defined and formalized in a sequence of works [97, 144, 45, 43]. Branching programs are labeled graphs, where each node in the program

is identified with one of the possible internal states of the sequential machine and the label denotes the input queried in that state and hence they capture the notion of both time and space in their description. In their pioneering work [43], Borodin and Cook devised the first technique to derive time-space tradeoff on general branching programs and applied it to obtain a $TS \in \Omega(n^2/\log n)$ for sorting n integers, each in the range $[1, n^2]$.

In the remainder of this chapter, we introduce and define branching programs and their different types. We also give an overview of the two main techniques used in deriving time-space tradeoffs that we also use to produce our results. Moreover, we introduce some of the tools that we will be using throughout the thesis: multi-party communication complexity, Yao's minimax principle, and martingales.

2.1 Branching Programs

Before we start with preliminaries, we explain some of the standard notations we use:

- $[n]$ to denote the set $\{1, 2, \ldots n\}$.
- $\tilde{O}(f(n))$ to suppress the poly-logarithmic factors in $O(f(n)\log^c n)$, for any $c > 0$.
- $\tilde{\Omega}(f(n))$ to suppress the poly-logarithmic factors in $\Omega(f(n)\log^c n)$, for any $c > 0$.
- $\log x$ to denote the logarithm base 2 of x.
- $n^{(p)}$ to denote the product $n(n-1)\ldots(n-(p-1))$.

Let D and R be two finite sets and n and m two positive integers.

DEFINITION 2.1. *A deterministic* single-output D-way branching program *is a rooted connected directed acyclic graph with nodes of out-degree 0, called* sink nodes*. Each sink node is labeled with a symbol from R, each non-sink node is labeled with an input index from $[n]$ and every edge is labeled with a symbol from D. The out-degree of a non-sink node is $|D|$ and the labels on the outgoing edges are all distinct.*

In the case where more than one output is desired, the number of sink nodes will increase as $|R|^m$ where m is the number of outputs produced. This would lead to an exponential blow-up in the number of sink nodes of the branching program. Instead, we use a variation of the branching program model where an output value is produced along the edges of the underlying graph.

DEFINITION 2.2. *A deterministic* multi-output D-way branching program *is a rooted connected directed acyclic graph with one node of out-degree 0, called the* sink node*. Each non-sink node is labeled with an input index from $[n]$ and every edge is labeled with a symbol from D. The out-degree of a non-sink node is $|D|$ and the labels on the outgoing edges are all distinct. Moreover, each edge is labeled with a possibly empty subset of the output indices and their corresponding values from R.*

When $|D| = 2$, we call the branching program *boolean*. We associate each path in the branching program with its node and edge labels.

DEFINITION 2.3. *Let π be a directed path in a branching program. The* queries *of π, denoted by Q_π, are the set of indices of symbols queried by π. The* answers *to those queries are represented by the function $A_\pi : Q_\pi \to D$ and for multi-output branching programs, the* outputs *produced along π by a* partial *function $Z_\pi : [m] \to R$.*

Note that we are abusing the term "function" in the above definition since a path in the branching program might have conflicting answers to the same query. We assume that the last correct answer is what is retained in the definition.

When provided with an input $x \in D^n$, the branching program computes by starting at the root and proceeding along the nodes of the graph by querying the input position associated with the label of each node and then, following the edge with the label from D that is equal to the symbol in x at that position. For single-output branching programs, the output of the branching program is the label of the sink node reached. For multi-output branching programs, the outputs are produced along the edges crossed. As a deterministic branching program produces one set of output values on a given input at the end of the computation, we can associate it with the function it computes.

DEFINITION 2.4. *A* computation *(in a branching program B) on x is a directed path π from the source to a sink in B whose queries to the input are consistent with x, i.e $A_\pi(i) = x_i$, for all*

$i \in Q_\pi$. *Moreover, the* output *of B on x, denoted $B(x)$, is defined as $B(x) = Z_\pi([m])$. A branching program B is said to* compute *a function f if for every $x \in D^n$, $B(x) = f(x)$.*

Sometimes, the input to the branching program is chosen according to some probability distribution.

DEFINITION 2.5. *Let μ be a probability distribution over D^n. A branching program B* computes *f* under μ with error at most η *if and only if $B(x) = f(x)$ for all but an η-fraction of $x \in D^n$ under distribution μ.*

The complexity measures associated with a branching program are the *time* and *space*.

DEFINITION 2.6. *The* time *T of a branching program is the length of the longest path from the root to a sink. The* size *is the number of nodes in the branching program and the* space *S is the logarithm base 2 of the size.*

Therefore, $S \geq \log T$. Note that the space here represents the working space of the computational model. The input to the branching program is assumed to be stored in a read-only memory and the output is written to a write-only memory. The space measures the number of different states of the computation, represented by the the nodes of the graph.

When considering a probability distribution over the input, the complexity measures of the branching program can be taken to be the averages of time and space.

DEFINITION 2.7. *Let μ be a probability distribution over D^n and let B be a branching program. The* average time *of B denoted $\overline{T}$ over distribution μ is the expectation of the length of the computation of B on x, when x is chosen according to μ.*

To define the notion of *average space*, we need a closer look at the space of a branching program, as described by Abrahamson in [5]. Let $size_B$ denote the size B and consider a numbering of the nodes of the branching program from $\{1, \ldots, size_B\}$. Then relative to a specific numbering, the space of the branching program on an input x can be viewed as the logarithm of the largest number of a node in the computation of B on x.

DEFINITION 2.8. *Let μ be a probability distribution over D^n and let B be a branching program.*

The average space *of B denoted $\overline{S}$ over distribution μ is the minimum over all numberings, of the expectation of the space of B on x when x is chosen according to μ.*

Branching programs are also used to model randomized computation.

DEFINITION 2.9. *A* randomized branching program *$\mathcal{B}$ is a probability distribution over deterministic branching programs with the same input set. $\mathcal{B}$ computes a function f with error at most η if for every input $x \in D^n$, $\Pr_{B\sim\mathcal{B}}[B(x) = f(x)] \geq 1 - \eta$. The* time *(respectively* space*) of a randomized branching program is the maximum time (respectively space) of a deterministic branching program in the support of the distribution.*

It is also useful to consider non-deterministic computations on branching programs.

DEFINITION 2.10. *A* non-deterministic branching program *B is a single-output branching program where we allow some of the nodes to be unlabeled and of to have an arbitrary out-degree, called* guessing nodes*. A computation can choose any of the unlabeled edges out of such nodes. For an input x, $B(x) = a$ if and only if there exists a path from the root to a sink with label $a \in R$ and the queries at the labeled nodes of the path are consistent with x. The guessing nodes are not counted in the time or size of the branching program.*

2.1.1 Branching programs versus general sequential models

Branching programs describe general unstructured models of computation where each node in the program is identified with one of the possible internal states of the sequential machine. We assume that the branching program has random access to the input, in the sense that it can read any input position in one unit of time. Moreover, we assume that the program will know an entire input variable if it accesses any bit of that input; thus we are ignoring the $\log |D|$ factor it costs to query a given variable, where D is the domain of the underlying function. The input is assumed to be stored in a read-only memory and the output written on a write-only memory. These memories are not counted towards the space used by the branching program. Each output symbol is produced as a unit, in the sense that all the bits of a given output symbols are produced in the same time unit.

Branching programs simulate other general sequential models like multi-tape Turing machines and RAMs. In the propositions below, we only show simulations for machines with a single output.

The arguments are easily generalized to multi-output machines.

Proposition 2.1. *Let M be a Turing machine with 1 read-only input head and 1 read-write head that runs in time T and use working space S. Then there exists a branching program B that simulates M in time T and space $O(\log n + S)$, where n is the input length.*

Proof. We will simulate the machine on the branching program by representing each configuration of the machine by a node. Since the input is stored in a read-only tape and the machine uses only S units of working space, hence it has $n|Q|\,|\Sigma|^S$ configurations, where Q is the set of states of the machine and Σ is the tape alphabet. Therefore the branching program B needs $n|Q|\,|\Sigma|^S$ nodes to represent these configurations. The label of each node is given by the position of the read-only input head of the corresponding configuration. The edges between the nodes corresponding to different configurations are determined by the transition function of the machine. The space of the branching program is therefore $\log(n|Q|\,|\Sigma|^S) = S\log|\Sigma| + \log|Q| + \log n$ and the time is T . The proposition follows as $|Q|, k$ and $|\Sigma|$ are constants. □

We get similar bounds for branching programs simulating RAMs. We assume that RAMs can read all the bits in a register in one unit of time, but the space is counted in term of the total bits in the registers used by the machine.

Proposition 2.2. *Let M be RAM with n read-only input registers, running in time T and use S registers of read-write memory, each of size b bits. Then there exists a 2^b-way branching program B that simulates M in time T and space bS.*

Proof. We represent the possible 2^{bS} states of the read-write memory used by the machine by the nodes of the branching program. Hence, we need 2^{bS} nodes in B. The input position to be read at the next time step is stored in a read-write register that is part of the working memory, hence the label of each node corresponds to the read-only input register read at each time step by the machine. An edge connect two nodes u and v if and only if v represents the state of the machine after reading the input at the label of u and performing the instruction from the instruction register. Therefore, B runs in time T and uses space bS. □

We make the following assumptions on the complexities of Turing machines and RAMs:

1. the time is at least n, where n is the input length; this assumption implies that the function computed depends on all its input positions, and

2. the space of a Turing machine or a RAM is at least $\log n$; this allows for storage of the head position for a Turing machine or the input position register of a RAM.

Therefore lower bounds on branching programs imply the same lower bounds on Turing machines and RAMs.

2.1.2 Some special classes of branching programs

When analyzing a branching program on an input, it is useful to have a node reached from the source by paths that are all of the same length.

DEFINITION 2.11. *A branching program is* leveled *if the nodes are divided into an ordered collection of sets each called a* level *where edges are between consecutive levels only. For a leveled branching program, the* width *is the maximum number of nodes on any of its levels and thus the space of a width W leveled branching program is at least* $\log W$.

Any branching program B of time T and space S can be made leveled by running Algorithm 2.1 on it. At then end of the algorithm , the nodes in B' belong to T different levels and the edges are

Algorithm 2.1 LEVEL-BP

input: B
output: B'

1: Create T copies of B and order the copies from 1 to T. Each copy will correspond a level in the final branching program B'.
2: Remove all edges from the copies.
3: Let S be a subset of the nodes that contains the root from copy 1.
4: **while** S is not empty **do**
5: Let i be the copy number of the nodes in S.
6: Connect the nodes in S to the nodes in copy $i + 1$ that correspond to their neighbor in B.
7: Replace the nodes in S by their neighbors in copy $i + 1$.
8: **end while**
9: Remove all nodes with no edges.

only between consecutive levels. Therefore there is no increase in time. The number of nodes has

increased by a factor of T, hence the space increases by an additive factor of $\log T$. Since $S \geq \log T$, the final branching program has time T and space $O(S)$. Therefore, for the remainder of the thesis, we assume that our branching programs are leveled.

Sometimes, we restrict the number of times an input position is queried along any path (not necessarily a valid computation) from source to sink.

DEFINITION 2.12. *A branching program is called* read-k *if every input position is queried at most k times on any path from source to sink, whether or not the answers on that path are consistent. A* read-once branching program *is a read-k branching program with $k = 1$.*

In comparison-based computations, the only operations allowed are comparisons between symbols at different input positions. Note that we do not allow comparison of an input position with a constant.

DEFINITION 2.13. *A* comparison branching program *B is a 3-way branching program where each non-sink node is labeled with a pair of indices $i, j \in [n]$ with $i < j$ and has precisely three out-edges with labels from $\{<, =, >\}$, corresponding to the relative position of inputs x_i and x_j in the total order of the inputs. A* computation *(in B) on x is a directed path π from the source to a sink in B whose edge labels are consistent with the relative order of the symbols at the input positions of the label of the originating node.*

To simulate straight-line programs, we define the notion of obliviousness for a branching program.

DEFINITION 2.14. *An* oblivious *branching program is a leveled branching program in which the same input symbol is queried by all nodes at the same level. Hence, the sequence of input positions queried by the program is the same across all the paths of the branching program.*

We also define a generalized version of oblivious branching programs where the obliviousness is enforced only on a subset of the input variables.

DEFINITION 2.15. *Let I be a subset of $[n]$. A branching program is I-*oblivious *if it is leveled and any level at which a variable with an index in I is queried, the same variable is queried at each node in that level.*

For $I \subseteq [n]$, we define the *I-time*, *I-size*, and *I-width* of a branching program to denote these measures where we only count nodes at which variables with indices in I are queried.

Similarly to randomized branching programs, we define *randomized oblivious* branching programs as a probability distribution over deterministic oblivious branching programs. The time and space are defined in the same fashion.

2.2 Yao's minimax Principle

Yao's minimax principle provides a tool that allows us to characterize the η-error randomized complexity of f in terms of its complexity when its input is chosen according to some probability distribution. We call the latter complexity the *distributional complexity* of f.

DEFINITION 2.16. *Let $\eta > 0$ and μ be a probability distribution over the inputs to a function f. Then the* η-error distributional complexity of f with respect to μ *is the minimum complexity of a deterministic algorithm A that computes f with probability at least $1 - \eta$, where the probability is over the random choice of the inputs according the distribution μ. The* η-error distributional complexity of f *is the maximum η-error distributional complexity of f with respect to μ, where the maximum is over the set of probability distributions μ over the inputs to f.*

Yao's mimimax principle states that the randomized complexity of a function is equal to its distributional complexity. In this thesis, we only use one side of this equality that allows us to derive lower bounds on the randomized complexity of f by deriving lower bounds on its distributional complexity. We formally state this easy half of Yao's mimimax principle.

Proposition 2.3. *[153] Let C be a class of algorithms and $\mathcal{C}$ be a distribution over C such that the probability that an algorithm $A \in C$ chosen according to $\mathcal{C}$ computes a function f is at least $1 - \eta$, for some $\eta > 0$. Then, for every distribution μ on the inputs to algorithms in C, there exists an algorithm A^* in the support of $\mathcal{C}$ that computes f under μ with error at most η, i.e $\Pr_{x\sim\mu}[A^*(x) = f(x)] \geq 1 - \eta$.*

Proof. For each input x, by the definition of $\mathcal{C}$, $\Pr_{A\sim\mathcal{C}}[A(x) = f(x)] \geq 1 - \eta$. Consider any probability distribution μ over the inputs. Then $\Pr_{x\sim\mu,\ A\sim\mathcal{C}}[A(x) = f(x)] \geq 1 - \eta$. Therefore, there exists an algorithm A^* from the support of $\mathcal{C}$ such that $\Pr_{x\sim\mu}[A^*(x) = f(x)] \geq 1 - \eta$. □

Moreover, if T and S are the maximum time and space used by any algorithm in the support of $\mathcal{C}$, then the algorithm A^* uses time at most T and space at most S. The proposition also holds when we consider T and S to be the average time and space of algorithms over the distribution $\mathcal{C}$.

Therefore, if one wishes to derive a lower bound on the complexity of a randomized branching program computing a function f with error η, it is enough to find a "hard" distribution μ over the inputs of f and prove that any deterministic branching program computing f under μ with error at most η satisfies the desired lower bound. We will use Yao's minimax lemma to drive our time-space tradeoffs in Chapters 3 and 4.

2.3 Multiparty Communication Complexity

Communication complexity measures the amount of information that must be exchanged between different parties in order to compute a function f when the input to that function is divided among the different parties. The notion of communication complexity was formalized by Yao [154] but was first introduced by Abelson [2] in the context of distributed computing of real functions: a function $f(x, y)$ is to be computed by two parties such that the first party only knows x, the second only knows y and the messages exchanged are bits.

In the above scenario, there is no overlap in the information to which the parties have access. Each party has access to a piece of the input that is private to them. In certain scenarios, it is desired that a party has access to all but a portion of the input that is hidden from them. In that case, the information they have access to is not private anymore since other parties have access to it too. Hence, we can view the portion of the input hidden from a party as being written on their forehead, thus allowing all the other parties to see it. This is particularly relevant in a broadcast setting when one party communicates information to all but a subset of the other parties. The private and public settings of accessing information are equivalent when only two parties are communicating and are different when there are three or more parties. The public information communication model is known as the *number-on-forehead model* and was first introduced in [53].

DEFINITION 2.17. *Let $f : D^n \to \{0, 1\}$. Assume that p parties, each having access to a part of the input x, wish to communicate in order to compute $f(x)$. The set $[n]$ is partitioned into p pieces, $\mathcal{P} = \{P_1, P_2, \ldots, P_p\}$ such that each party i has access to every input symbol whose index is in P_j*

for $j \neq i$. The parties communicate by broadcasting bits to the other parties, which can be viewed as writing the bits on a common board, switching turns based on the content of the board. The function is computed when its value is written on the board by one of the parties. At that point, the communication halts. The number-on-forehead (NOF) multiparty communication complexity *of f with respect to the partition $\mathcal{P}$, denoted $C_{\mathcal{P}}(f)$, is the minimum total number of bits written. When there is a standard partition of the input into p-parties associated with a given function f, we will simply write $C_p(f)$ instead of $C_{\mathcal{P}}(f)$.*

When the input to the function is chosen according to a probability distribution μ, we are interested in the communication complexity of a protocol that is correct with a high probability over the choice of the input.

DEFINITION 2.18. *Let μ be a probability distribution over D^n. The* (μ, ϵ)-distributional communication complexity *of f with respect to $\mathcal{P}$, denoted $D^{\mu}_{\epsilon,\mathcal{P}}(f)$, is the minimum number of bits exchanged in a NOF communication protocol with input partition $\mathcal{P}$ that computes f correctly on a $1-\epsilon$ fraction of the inputs weighted according to μ.*

Again, we replace the $\mathcal{P}$ by p when $\mathcal{P}$ is a p-partition that is understood from the context.

Many computational models can be expressed in terms of a communication protocol and hence lower bounds on the communication complexity of a function can translate to lower bounds in these models. One such model are ACC^0 circuits. These are polynomial-size boolean circuits having OR, AND, and NOT gates as is standard, in addition to a special type of gates called MOD_p, for some fixed prime p, that compute the sum of its boolean inputs modulo that prime. In a series of work starting with Allender [10], then Yao [156] and ending with Beigel and Tarui [35], it was proven that ACC^0 circuits can be simulated by depth-2 *threshold circuits*. This type of circuits has gates that evaluate to 1 only when the number of 1's in their input exceeds a certain threshold. Goldmann and Håstad [72] established a relationship between multiparty communication complexity and threshold circuits complexity and proved that any function computed by a depth-2 threshold circuit of size R where the gates connected to the inputs have fan-in at most s, have a $1 + s \log R$-bits multiparty communication protocol with s parties. Combining this result with the simulation of ACC^0 circuits with depth-2 threshold circuits implies that any function computed by an ACC^0 circuit has a poly-logarithmic multiparty protocol with a logarithmic number of parties. Hence lower bounding the

multiparty communication complexity of functions would give lower bounds on the size of boolean circuits with MOD_p gates computing them.

Moreover, multiparty communication complexity is an important tool in deriving lower bounds for branching programs. It was first used in in [53] to derive a time-space tradeoff on general branching programs deciding whether k integers sum to n, of the form: any constant-width branching program computing that function requires superlinear (in n) time. In [22], Babai, Nisan and Szegedy established a connection (implicit in [11] for two-party communication complexity) between multi-party communication complexity and time-space tradeoffs for oblivious branching programs. Using that connection, they define an explicit boolean function related to the generalized inner product and derive a $T = \Omega(n \log^2(n/S))$ time-space tradeoff for any oblivious branching program computing it. Subsequently, Beame and Vee [33] used the same connection to derive the same time-space tradeoff for a more natural function, the out-degree-1 graph accessibility problem, on any deterministic oblivious branching program. to derive time-space tradeoffs for oblivious branching programs. We explain the connection between NOF multiparty communication complexity and oblivious branching programs in Section 2.5.2 and we show our time-space tradeoffs using multiparty communication complexity in Chapter 3.

2.4 Doob Martingales and the Azuma-Hoeffding Inequality

Martingales are a sequence of random variables that are defined over a sequence of samples from the same probability space. Martingales have the property that at any point in the sequence, the expected value of the next random variable conditioned on the previous observations is equal to value of the present random variable. Formally,

DEFINITION 2.19. *Let $Y_{i=1}^n$ and $X_{i=0}^n$ be two sequences of random variables defined on a common probability space such that $\mathbb{E}[X_i|Y_1, Y_2, \ldots, Y_{i-1}] = X_{i-1}$, for all $i \in [n]$. Then the sequence $X_{i=1}^n$ is a* martingale *with respect to $Y_{i=1}^n$.*

In certain settings, it is useful to study the concentration of the values of the X_i's. The Azuma-Hoeffding inequality gives a concentration bound on the random variables of a martingale.

Theorem 2.4 (**Azuma-Hoeffding Inequality**). *[85, 21] Let $X_{i=0}^n$ be a martingale such that $X_i -$*

$X_{i-1} \leq c_i$ *for all* $i \in [n]$. *Then, for every* $\lambda > 0$, $\Pr[|X_n - X_0| \geq \lambda] \leq e^{-\frac{\lambda^2}{2\sum_{i=1}^{n} c_i^2}}$.

We also use the Chernoff bound, which is a special case of the Azuma-Hoeffding inequality, applied to the the sum of the Y_i's only when they are independent.

Theorem 2.5 (**Chernoff Bound**). *[55] Let* $Y_{i=1}^{n}$ *be* n *independent random variables taking values in* $\{0, 1\}$, *such that for all* $i \in [n]$, $\mathbb{E}[Y_i] = p_i$, *for some* $p_i > 0$. *Let* $X = \sum_{i=1}^{n} Y_i$ *be a random variable and let* $\mu = \sum_{i=1}^{n} p_i$ *be its expectation. Then, for any* $\delta \in (0, 1]$, $\Pr[X \leq (1-\delta)\mu] \leq e^{-\mu\delta^2/2}$.

In this thesis, we use *Doob martingales*, a type of martingale defined as the expectation of a random variable defined in terms of the sequence of observations sampled.

DEFINITION 2.20 (Doob Martingale). *Let* $Y_{i=1}^{n}$ *be a sequence of random variables and let* A *be a random variable defined over the same probability space. Then the sequence of random variables* $X_i = \mathbb{E}[A|Y_1, Y_2, \ldots, Y_i]$, $i = 0, \ldots, n$ *defines a martingale. Moreover* $X_0 = \mathbb{E}[A]$ *and* $X_n = A$.

For a more complete introduction to martingales, the reader is referred to [152].

2.5 Lower Bounds for Branching Programs

The general approach to derive lower bounds for branching programs involves dividing the program into several small-depth layers and analyzing the computation within each layer, since all we know about the past computation is contained in the limited memory available at the beginning of each layer.

We are interested in lower bounds on explicit functions. For lower bounds on non-explicit function, using counting arguments similar to Shannon's [136] and Muller's [117], one can prove that almost any boolean function on n inputs requires a branching program of size $\Omega(2^n/n)$ [89].

The first lower bound for an explicit function on a general branching program was derived by Borodin and Cook in [43], where they gave a $TS \in \Omega(n^2/\log n)$ time-space tradeoff for sorting n elements, each encoded in $2\log n$ bits. The same technique was later used by Yesha in [158] to derive time-space tradeoffs for matrix-multiplication ($TS \in \Omega(n^3)$) and discrete Fourier transform ($TS \in \Omega(n^2)$ over a prime field). In [3, 5], Abrahamson gave a general outline for deriving lower

bounds for multi-output (randomized) branching programs and applied it to many algebraic problems including matrix-vector product, matrix multiplication and convolution. In [34], Beame gave an optimal $TS \in \Omega(n^2)$ time-space tradeoff for finding unique elements in a list of n integers and thus improved the sorting lower bound to $TS \in \Omega(n^2)$. This technique was also applied to universal hashing by Mansour, Nisan and Tiwari [115], where they showed that a hashing funtion from n bits to m bits require a time-space tradeoff $TS \in \Omega(nm)$ on any general branching program.

All the above functions produce multiple outputs and the technique applied to obtain those large time-space tradeoffs requires having a large number of inputs. For single-output functions, this technique yields trivial lower bounds, as we show below in Section 2.5.1. These functions proved to be hard to derive time-space tradeoffs for on general branching programs. The first time-space tradeoffs for single-output functions were derived on constant-width branching programs due to their relation to multiparty communication complexity [53]. For larger values of width and space, the first (and largest) time-space tradeoffs for explicit functions were derived on oblivious branching programs by exploiting their connection to communication complexity as we explain in Section 2.5.2.

Using techniques similar to two-party communication complexity, Alon and Maass [11] gave a $T \in \Omega(n \log(n/S))$ time-space tradeoff for the boolean set equality problem on oblivious branching programs. By increasing the number of parties to $k > 2$, Babai, Nisan and Szegedy [22] derived a $T \in \Omega(n \log^2(n/S))$ tradeoff for computing a boolean function related to the generalized inner product problem. Beame and Vee [33] reduced the boolean generalized inner product problem to the out-degree-1 graph accessibility problem and derived the same time-space tradeoff for oblivious branching programs over larger domain, thus separating oblivious branching programs from general ones.

Read-k branching programs were also studied in the context of time-space tradeoffs for single-output functions. Other than being an intermediary class between general and oblivious branching programs, the motivation behind studying these restricted branching programs comes from the interest in studying branching programs that takes linear time, which means that every input is read a constant number of time on average. Proving lower bound on read-k programs helps shed light on linear-time general branching programs. For read-once branching programs, several lower bounds on the size were derived. Masek [116] and Wegener [151] derived an $\Omega(n^2)$ lower bound on the size

of a read-once branching program computing the majority function. Since the majority can be computed by read-twice branching programs in size $O(n^2/\log n)$ [111], this gives a separation between read-once and read-twice branching programs. This separation was made exponential by Žák [149] and Wegener [150], who proved independently that determining whether a graph on n is exactly a clique on $n/2$ nodes requires a read-once branching program of size in $\Omega(e^n)$, where n is the number of nodes in the graph and the graph is in the adjacency-matrix representation (the same lower bound hold when the graph simply contains such a clique). Combining that lower bound with the read-twice branching program of size in $n^{O(1)}$ computing that function [150], we get the exponential separation between read-once and read-twice branching programs. In [46], Borodin, Razborov and Smolensky extended that separation to non-determistic branching program by showing that the read-once lower bound holds for the same function on non-deterministic branching programs. Moreover, they exhibited an explicit boolean function f such that any read-k branching program computing f requires space $\Omega(\frac{n}{4^k k^3})$. Okol'nishnikova [121] gave separations between non-deterministic read-k branching programs, for different values of k. She exhibited for each $k \geq 2$, a sequence of boolean functions on n inputs whose size is exponentially (in n) larger on read-k non-deterministic branching programs, compared to the size of a read-$(k \ln k/\ln 2 + C)$ non-deterministic branching programs computing the same function. In [143], T.S. Jayram strengthened that separation by exhibiting functions that have linear-size read-$(k+1)$ branching programs while any read-k branching programs computing those function requires sizes exponential in $n^{1/k+1}$.

The first lower bound for an explicit single-output function on a general branching program of superpolynomial size was derived by Beame, Saks and Thathachar in [29]. They devised a novel lower bound technique and exhibited an explicit family of boolean functions for which any sub-exponential size branching program requires length at least $1.0178n$. For larger domains D, the authors obtained a much stronger lower bound compared to the boolean case. They constructed a family of functions such that for every constant $k > 0$, any multi-way branching program computing the function with length kn requires space $S \in \Omega(n \log^{1-\epsilon} |D|)$, for any $\epsilon > 0$, where $|D| \geq 2^{2(k+2)}$. Hence, if $T \in o(n \log\log n)$, then $S \in \Omega(n \log^{1-\epsilon} n)$ for any $\epsilon > 0$. These functions are based on quadratic forms $x^T M x$, where the matrix M is a General Fourier Transform matrix. Note that simply writing the input to a multi-way branching program in binary would increase the

input size by a factor of $\log n$, thus rendering the time lower bounds trivial in terms of the new input length. Thus, deriving lower bounds for boolean branching programs turned out to be a harder task.

Shortly after, Ajtai [7] improved the lower bound for multi-way branching programs and showed that a decision version of the Hamming distance problem (given n binary vectors, each of length $c \log n$, is there a pair that have distance at most $\frac{1}{4}c \log n$?) requires more than $\epsilon n \log n$ space when the time is linear, for some constant $\epsilon > 0$. Moreover, he proved a weaker ϵn lower bound on the space for linear time for multi-way branching programs computing the *element distinctness problem*: given n integers, decide whether or not they are all distinct.

Following that, in a breakthrough work, Ajtai [8] gave the first superlinear lower bound on time for boolean branching programs. The author built on his techniques from [7] and constructed a boolean function based on quadratic forms such that no boolean branching program with space at most ϵn computes the function in time $T = kn$, for all positive integers k and $\epsilon > 0$. Ajtai's results were eventually improved by Beame, Saks, Sun and Vee in [32]. They showed that any randomized branching program computing the boolean quadratic form considered by Ajtai requires time $T \in \Omega\left(n\sqrt{\frac{\log(n/S)}{\log\log(n/S)}}\right)$. As for functions with larger domains, they constructed functions over large fields related to those quadratic forms and show a time lower bound of $T \in \Omega(n \log(\frac{n \log n}{S}))$ for any randomized multi-way branching program computing it. Moreover, they derived a time-space tradeoff of $T \in \Omega(n\sqrt{\log(n/S)/\log\log(n/S)})$ for the element distinctness problem (thus imroving Ajtai's result [7] when the space is in $n^{o(1)}$).

The above results remain the largest lower bounds for randomized multi-way single-output (general) branching programs. For their deterministic counterparts, the best lower bound known for an explicit funtion is due to Beame and Vee [33], where they construct a boolean function such that any branching program computing it in space $S \leq n^{\epsilon} \log |D|$ requires time $T \in \Omega(n \log^2 n)$, for any $\epsilon > 0$, though the function is not known to be computable in NP.

For a complete exposition of branching programs and their time-space tradeoffs, the reader is referred to [151]. Now, we present the lower bound techniques, divided according to the number of outputs of the functions computed.

2.5.1 Time-space tradeoff technique for multi-output functions

Assume that a branching program correctly computes a function f on a large fraction of the inputs. The argument goes as follows: after dividing the branching program into layers, the layers are broken into several small-depth branching programs rooted at the top nodes of the layer. For each input, we should have at least one small-depth branching program producing a large number of the outputs for that input. Then, we prove that this small branching program correctly computes that number of outputs only for a small fraction of the inputs. Since the branching program is correct for a large fraction of inputs, we should have enough such small-depth branching programs to compensate for the small fraction of the inputs for which the small-depth branching programs correctly compute the outputs.

We illustrate the argument as presented by Abrahamson in [5].

Let $f : D^n \rightarrow R^m$ and μ be a probability distribution over D^n. Let B be a leveled η-error deterministic branching program computing f under μ in time T and space S, and $x \in D^n$ be an input to B.

DEFINITION 2.21. *Let P be a branching program and let π be a directed path in P. π k-solves input x w.r.t. to f on some subset $L \subseteq [m]$ if and only if:*

1. *$x_i = A_\pi(i)$, for all $i \in Q_\pi$,*
2. *the domain of the partial function Z_π has size at least k, and*
3. *$Z_\pi(j) = f(x)_j$ for all elements $j \in L$ in the domain of Z_π, where $f(x)_j$ is the j^{th} output of f.*

P k-solves x on L if and only if the computation of P on x k-solves x on L.

Using the notion of k-solvability, we define the probability that a branching program k-solves random inputs of a function.

DEFINITION 2.22. *Let $f : D^n \rightarrow R^m$ be a function and let μ be a probability distribution over D^n. Let $L \subseteq [m]$ be a subset of the outputs of f and h and $k \leq |L|$ be positive. For a branching program P, define $pr^L_{f,\mu}(P, k)$ as the probability that P k-solves an input x with respect to $f(x)$ on*

L, when x is chosen according to μ. Define $pr^L_{f,\mu}(h,k)$ as the maximum, over all depth h branching programs P, of $pr^L_{f,\mu}(P,k)$.

To derive lower bounds on the complexities of multi-output functions, it suffices to upper bound $pr^L_{f,\mu}(h,k)$, for an appropriate setting of h and k. The following theorem is generalization of the argument in [5].

Theorem 2.6. *Let B be an η-error branching program computing a function $f : D^n \to R^m$ under an input distribution μ in time T and space S, such $\eta \leq 1 - 2^{-\delta S}$. Let $L \subseteq [m]$ be a subset of the outputs of f, $h \leq n$ and $k \leq \frac{|L|h}{T}$ be positive integers. If $pr^L_{f,\mu}(h,k) \leq c^k$ for some $c < 1$, then $TS \geq \frac{|L|h}{1+\delta}\log(1/c)$. In particular, if $L \in \Theta(m)$, $h \in \Theta(n)$ and $\delta \in \Theta(1)$, then $TS \in \Omega(nm\log(1/c))$.*

Proof. Let $h \leq n$ and $k \leq \frac{|L|h}{T}$ be positive integers. We divide B into $\frac{T}{h}$ stages (we ignore floors and ceilings for the purpose of this proof).

DEFINITION 2.23. *Let v be a node in the top level of a stage. $B_{v,h}$ is the D-way branching program formed by the subgraph whose root is v and non-sink nodes are the nodes of B reachable from v within that stage. The sink node of $B_{v,h}$ is connected to all the nodes of the bottom level of that subgraph. Moreover, $B_{v,h}$ has at most height h.*

Since there are at most $\frac{T}{h}$ stages, for every input x, there should be a stage producing at least $\frac{|L|h}{T} \geq k$. By the correctness of B, for x chosen according to μ and with probability at least $1-\eta$, there exists some branching program $B_{v,h}$ in some stage that k-solves x. On the other hand, the probability that a branching program $B_{v,h}$ in some stage k-solves the random input x is at most $pr^L_{f,\mu}(h,k)$ as defined above. B can have at most 2^S such branching programs. Applying the union bound, we get the following inequality: $1 - \eta \leq 2^S pr^L_{f,\mu}(h,k)$ and hence $S \geq \log(1-\eta) - \log pr^L_{f,\mu}(h,k)$. Replacing η and $pr^L_{f,\mu}(h,k)$ by their bounds, the theorem follows. □

Since any branching program $B_{v,h}$ can simply produce an arbitrary fixed value for an output independent of its input and this value would be correct for an input x with probability $1/|R|$, $c \geq \frac{1}{|R|}$and the best lower bound one can derive using that technique is $TS \in \Omega(nm\log|R|)$.

We use this technique to derive lower bounds for branching programs computing frequency moments in Chapter 4 and propose an approach based on this technique to derive time-space tradeoffs for branching programs encoding error correcting codes in Chapter 7.

2.5.2 Time-space tradeoff technique for single-output functions

The approach to derive lower bounds for multi-output functions measure the progress made by the branching program in terms of the correct outputs that are produced. If we apply it to single-output functions, the best time-space tradeoff we obtain is $TS \in \Omega(n \log n)$. This bound is trivial since $T \geq n$ and $S \geq \log n$, for any function. Therefore, a new technique is required. Indeed, the approach adopted for single-output functions makes use of the fact that if a function $f : D^n \rightarrow \{0,1\}$ is computed by a branching program with small time and space, then the set $f^{-1}(1)$ of inputs should have a special structure. The lower bounds follow by proving that $f^{-1}(1)$ does not have such a structure.

As we only consider oblivious branching programs when deriving lower bounds for single-output functions in this thesis, we present an overview of the technique as it applies to that specific class. The lower bounds for the oblivious case rely on exploiting the relationship between the computation of the function on the oblivious program and its communication complexity.

Let $f : D^n \rightarrow \{0,1\}$ and $I \subseteq [n]$. Let B be a deterministic I-oblivious branching program of width W computing f in I-time T. Since B is I-oblivious, it yields a sequence s of queries to the input symbols indexed by I of length T. We divide s into r segments. We assign each segment to one set in $A_1, A_2, \ldots, A_p$ corresponding to each of the p parties in a NOF communication setting. Since the A_i's contain the input positions that each party can see, we need to find a subset $C_i = I \backslash A_i$ that can be placed on the forehead of party i, for each party i. Moreover, the C_i's should be disjoint. Hence, the C_i's form a partition $\mathcal{P}'$ of a subset $I' \subseteq I$ and for all s_j assigned to a set A_i, there is a C_i such that the variables of C_i do not appear in any of these s_j's. For any fixing of the values of input variables in $[n] \setminus I'$, we get a p-party communication setting with partition $\mathcal{P}'$ to compute the function f' resulting from f by fixing the values of input variables in $[n] \setminus I'$.

The following proposition relates the complexity of the branching program to that of the communication complexity of f', when the input to f is chosen according to some distribution μ on D^n.

It is adapted from [22, 33] for I-oblivious branching programs approximating a function.

Proposition 2.7. *Let $f : D^n \to \{0,1\}$. Let B be a deterministic I-oblivious branching program of I-width W that approximates f under a probability distribution μ with error at most ϵ. Let $\mathcal{P}'$ be a partition of a subset $I' \subseteq I$ of $[n]$ such that the distribution μ on D^n has fixed values for all input variables indexed by $[n] \setminus I'$. Suppose that the query sequence of B can be written as $s_1 \dots s_r$ such that for each s_i, there is some class $P_{j_i} \in \mathcal{P}'$ whose variables do not appear in s_i. Then for any partition $\mathcal{P}$ that extends $\mathcal{P}'$ to all of $[n]$, $(r-1)\log(W) + 1 \geq D^{\mu}_{\epsilon,\mathcal{P}}(f)$.*

Proof. Associate party j with each class P_j of $\mathcal{P}$ and place all input variables in class P_j on the forehead of party j. For segment s_1, party j_1 simulate B by starting at the root and querying the input variables in s_1 and following the computation according to B. When she reaches the bottom level of segment s_1, she broadcasts the name of the node in B she reaches. Since this node is in I, $\log W$ bits are enough to specify that node. For $i = 2, \dots, r$, party j_i simulates B on segment s_i and broadcasts the name of the node of B reached at the end of the segment. The output of the protocol is the label of the sink node that party j_r reaches. □

Therefore, to derive the time-space tradeoff, one should divide the sequence s into $s_1 \dots s_r$ such that the resulting partition $\mathcal{P}'$ yields a large distributional communication complexity for f, $D^{\mu}_{\epsilon,\mathcal{P}}(f)$. In Chapter 3, we use this technique to derive time-space tradeoffs for an explicit function on randomized oblivious branching programs.

Chapter 3

RANDOMIZED OBLIVIOUS BRANCHING PROGRAMS

An algorithm is *oblivious* (sometimes also called *input-oblivious*) if and only if its every operation, operand, as well as the order of those operations is determined independent of its input. Certain models of computation, such as circuits or straight-line programs are inherently oblivious. However, many computing models such as Turing machines and random access machines (RAMs), which use non-oblivious operations such as indirect addressing, are not, though fairly efficient simulations of these general models by their more restricted oblivious variants have been shown [124, 9].

Deterministic oblivious branching programs have been studied in many contexts. Indeed the much-studied *ordered binary decision diagrams* (or OBDDs) [50] correspond to the special case of deterministic oblivious branching programs that are also read-once. Our lower bound approach follows a line of work based on another reason to consider oblivious algorithms: their behavior is restricted and thus simpler to analyze than that of general algorithms. Alon and Maass [11] showed $T \in \Omega(n \log(n/S))$ lower bounds for certain natural Boolean functions on deterministic oblivious branching programs and this lower bound tradeoff was increased by Babai, Nisan, and Szegedy [22] to $T \in \Omega(n \log^2(n/S))$ for a different Boolean function based on the generalized inner product. Beame and Vee [33] also used a variant of [22] using generalized inner product to give an $\Omega(\log^2 n)$ factor separation between general branching programs and deterministic oblivious branching programs by proving a $T \in \Omega(n \log^2(n/S))$ lower bound for the $1GAP_n$ problem. However, that separation does not apply to the randomized simulations nor to the Boolean branching programs that we consider.

Though a number of time-space tradeoff lower bounds have been proven for natural problems in NP for general deterministic and nondeterministic [29, 7, 8] and randomized [32] computation, all of the lower bounds are sub-logarithmic and, naturally, none can yield a separation between general and randomized oblivious branching programs. Indeed the largest previous lower bounds for solving decision problems on randomized branching programs are of the form $T \in \Omega(n \log(n/S))$ which

is at most a logarithmic factor larger than the trivial time bound of n. These bounds also apply to randomized *read-k* branching programs (which roughly generalize oblivious branching programs for related problems) for $k \in O(\log n)$ [132]. No prior separations of randomized oblivious computations from general computation have been known.

Our main result implies that a superlogarithmic increase in time or an exponential increase in space is necessary to convert a general algorithm to a randomized oblivious one. We derive this separation by considering a very simple problem for deterministic RAM algorithms, the pointer jumping problem, also known as the out-degree-1 graph accessibility, $1GAP_n$.

3.1 Pointer Jumping and Generalized Inner Product

We consider the out-degree-1 directed graph reachability problem, $1GAP$, which is also known as the pointer jumping problem. Define $1GAP_n : [n+1]^n \to \{0,1\}$ such that $1GAP_n(x) = 1$ iff there is a sequence of indices $i_1, 1_2, \ldots, i_\ell$ such that $i_1 = 1$, $i_\ell = n+1$ and $x_{i_j} = i_{j+1}$ for all $j = 2, \ldots \ell - 1$. The $1GAP_n$ problem is s-t connectivity in $(n+1)$-vertex directed graphs of out-degree 1, where $x_1, \ldots, x_n$ represent the out-edges of nodes 1 through n, s is 1, and t is $n+1$ and vertex $n+1$ has a self-loop.

We will relate the complexity of $1GAP_n$ for randomized oblivious branching programs to that of the *generalized inner product* problem $GIP_{p,n}$ for a suitable value of p. $GIP_{p,n} : (\{0,1\}^n)^p \to \{0,1\}$ is given by $GIP_{p,n}(z_1, \ldots, z_p) = \oplus_{j=1}^{n} \bigwedge_{i=1}^{p} z_{ij}$. The standard input partition for $GIP_{p,n}$ places each z_i in a separate class. Babai, Nisan, and Szegedy [22] proved that under the uniform distribution the p-party NOF communication complexity of $GIP_{p,n}$ is large. We also will use the fact that $GIP_{p,n}$ can be computed easily by a leveled width-4, oblivious read-once branching program.

3.2 Contributions

Our lower bounds apply not only to randomized oblivious RAM algorithms but also to more powerful randomized oblivious branching programs. Our precise results are the following.

Theorem 3.1. *Let $\epsilon < 1/2$. Randomized oblivious branching programs or random access machines computing $1GAP_n$ using time T, space S and with error at most ϵ require $T \in \Omega\left(n \log(\frac{n}{S}) \log\log(\frac{n}{S})\right)$.*

Since $1GAP_n$ can be computed by a RAM algorithm in time n and space $O(\log n)$, which follows the path from vertex 1 maintaining the current vertex and a step counter, we immediately obtain the following corollary.

Corollary 3.2. *Any method for converting deterministic random access machine algorithms to randomized oblivious algorithms requires either an $n^{1-o(1)}$ factor increase in space or an $\Omega(\log n \log\log n)$ factor increase in time.*

The $1GAP_n$ problem has input variables from a linear-sized domain that the RAM can read in one step. Because of this, the lower bound for computing $1GAP_n$ is at most $\Omega(\log\log(n/S))$ larger than the number of its input bits and so is sub-logarithmic for Boolean branching programs. However, we also obtain analogues of the above results for Boolean branching programs computing a variant of the generalized inner product problem that we denote by $GIP\text{-}MAP$.

Theorem 3.3. *Let $\epsilon < 1/2$. Any randomized oblivious Boolean branching program computing $GIP\text{-}MAP_N$ with time T, space S and error at most ϵ requires then $T \in \Omega\left(N \log(\frac{N}{S}) \log\log(\frac{N}{S})\right)$.*

3.3 Techniques

Our argument uses the connection between oblivious branching programs and (best-partition) communication complexity that we described in Section 2.5.2. We also make use of the fact that inputs to the $1GAP_n$ problem conveniently encode any function computable by a small branching program. More precisely, we show that, since the generalized inner product GIP can be computed by a constant-width read-once branching program, we can convert any oblivious branching programs for $1GAP_n$ to a (partially) oblivious branching program that computes $Permuted\text{-}GIP$ which takes as input both a GIP input z and a permutation π to determine how the GIP function is applied to z. The permutation allows us to convert the lower bound for GIP in fixed-partition multiparty communication complexity to a best-partition lower bound, reminiscent of a similar conversion in the context of 2-party communication complexity [105]. Though that idea would have been sufficient for the deterministic non-Boolean separations in [33], we need much more here. The key to our argument is a way of extending this idea to a more involved analysis that yields a reduction that works for distributional complexity. Our Boolean lower bounds follow for $GIP\text{-}MAP$, a version of $Permuted\text{-}GIP$ based on pseudo-random permutations.

3.4 Related Work

The questions we consider here were inspired by results of Ajtai [9] (and similar results of Damgård, Meldgaard, and Nielsen [58]) who, eliminating cryptographic assumptions from [73, 122, 74], showed efficient simulations of general RAM algorithms by randomized algorithms that are oblivious and succeed with high probability, with only a polylogarithmic factor overhead in both time and space. However, our separations do not apply in the context of their simulations for two reasons. First, their simulations assume that the original RAM algorithm only has sequential access to its input, in which case the small space upper bound for $1GAP_n$ does not apply (or alternatively the original RAM algorithm has linear space which a deterministic oblivious branching program can use to compute any function in linear time). Second, our lower bounds apply only to *randomized oblivious* simulations, in which the sequence of locations accessed must be independent of the input but may depend on the random choices, whereas Ajtai's simulations are more general *oblivious randomized* simulations in that the probability distribution of the sequence of locations accessed is input independent[1].

3.5 Reducing Permuted Problems to $1GAP$

In order to derive time-space tradeoffs for the $1GAP$ problem, we will use a reduction from a permuted version of GIP. Since the idea is more general we state the reduction more generally.

Let $f : \{0,1\}^N \to \{0,1\}$ be a Boolean function. Define the promise problem $Permuted\text{-}f : \{0,1\}^N \times [N]^N \to \{0,1\}$ by $Permuted\text{-}f(z,\pi) = f(z_{\pi(1)}, z_{\pi(2)}, \ldots z_{\pi(N)})$ where π is guaranteed to be a permutation.[2]

Lemma 3.4. *Let $n = Nw + 1$. If $f : \{0,1\}^N \to \{0,1\}$ is a Boolean function computable by a width w oblivious read-once branching program then there is a reduction g from $Permuted\text{-}f$ to $1GAP_n$ mapping (z,π) to x such that the value of each x_i depends on π and at most one bit of z,*

[1] A simple algorithm that flips a random bit r and then chooses which order to read bits x_2 and x_3 based on the value of $x_1 \oplus r$ is oblivious randomized but not randomized oblivious: Each order of reading x_2, x_3 has probability $1/2$ independent of the input, but for each fixed value of r, the order of reading x_2, x_3 depends on the value of x_1

[2] The function is well-defined even when π is not a permutation. Determining whether or not π is a permutation is precisely the $ELEMENT\text{-}DISTINCTNESS$ problem [155, 34, 32] with range $[N]$ whose time-space tradeoff complexity is open, as we discuss in Chapter 4.

whose index is independent of π.

Proof. Let B_f be a width-w oblivious read-once branching program computing f. We assume without loss of generality that B_f queries the bits of z in the order $z_1, z_2, \ldots, z_N$; if not, we can modify the construction below by applying the fixed ordering given by the queries of B_f. Given π, the function $Permuted\text{-}f$ is computed by a modification of B_f^π that replaces each query to z_j in B_f by a query to $z_{\pi(j)}$.

Vertex $n+1$ for the $1GAP_n$ problem will correspond to the 1-sink of B_f^π. Vertex 1 will point to the start node of B_f^π and will also be identified with the 0-sink node of B_f^π. More precisely, x_1 has value $w * (\pi(1) - 1) + 2$, assuming that the first node in that level is the start node.

Vertices 2 though n will correspond to the nodes of B_f^π. For $j \in [N]$ and $k \in [w]$, vertex $i = (j-1) * w + k + 1$ will correspond to the k-th node of the level in B_f^π that queries z_j. Note that, given i, the values j and k are uniquely determined. More precisely, given $i \in [2, n]$, the value of x_i is determined as follows: Determine j and k. Query z_j [3]. Also query[4] π to determine $\ell = \pi^{-1}(j)$, the level in B_f^π at which z_j is queried. Unless $\ell = N$, the next variable that B_f^π will query is $z_{\pi(\ell+1)}$. Suppose that the 0- and 1-outedges from the k-th node at level ℓ in B_f are to the k_0-th and k_1-th nodes at level $\ell + 1$ in B_f respectively; then the value of x_i will be $(\pi(\ell+1) - 1) * w + k_{z_j} + 1$. Otherwise, set the value of x_i depending on z_j to either 1 or $n+1$ depending on whether the corresponding edge goes to the 0-sink or 1-sink. Correctness is immediate. □

Overloading notation we identify the subset $[N]$ of the $2N$ indices of $Permuted\text{-}f$ with z.

Corollary 3.5. *Let $n = Nw + 1$ and $f : \{0,1\}^N \to \{0,1\}$ be a Boolean function computable by a width-w oblivious read-once branching program. If there is a (randomized) oblivious branching program for $1GAP_n$ of time T, space S, and width W, then there is a (randomized) z-oblivious branching program for $Permuted\text{-}f$ with z-time T, z-space S, and z-width W.*

Proof. Use the reduction g given by Lemma 3.4 to replace each query to an input x_i of $1GAP_n$ by the single query of z_j depending on i and then query π as given by g. There is one z-query for each

[3]This is where we need the branching program to be oblivious: each level is associated with a specific variable from z.

[4]This is where we need that π is a permutation.

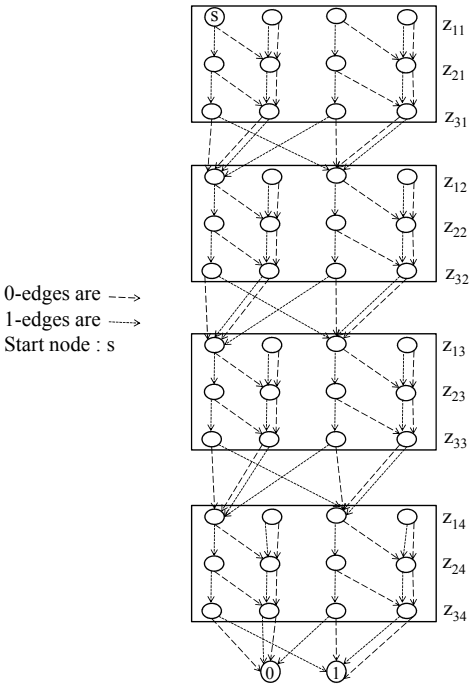

Figure 3.1: A width 4 branching program computing $GIP_{3,4}$.

x-query. (We do not care that queries to π are efficient.) □

To apply Corollary 3.5 to $GIP_{p,m}$, we show that $GIP_{p,m}$ has a read-once oblivious branching program

Proposition 3.6. *Let $p, m \geq 0$ and $N = mp$, There is a width-4 read-once oblivious branching program for $GIP_{p,m}$.*

Proof. We need to keep track of the value of the AND of each block of size p and of the value of the XOR. The construction shown in Figure 3.1 is easily generalized to a width-4, $pm + 1$-time branching program computing $GIP_{p,m}$, for any p and m.

□

3.6 Time-Space Tradeoff Lower Bound for $Permuted\text{-}GIP$ *and* $1GAP$

Following the last section, to derive lower bounds for $1GAP_n$ we consider randomized z-oblivious branching programs for $Permuted\text{-}GIP_{p,m}$. Since such programs are distributions over deterministic programs, as usual we invoke the easy half of Yao's Lemma [153] described in Chapter 2 to show that it is enough to find a distribution μ such that every ϵ-error deterministic z-oblivious branching program approximating $Permuted\text{-}GIP_{p,m}$ under μ will have a large time-space tradeoff.

The distribution μ we consider is the uniform distribution on the inputs of $Permuted\text{-}GIP_{p,m}$, i.e. the uniform distribution on all inputs $(z_1, z_2, \ldots, z_p, \pi) \in (\{0,1\}^m)^p \times S_{[p]\times[m]}$ where $S_{[p]\times[m]} \cong S_N$ is the set of all permutations of the $N = pm$ inputs to $GIP_{p,m}$. We will write μ as $\mathcal{Z} \times \mathcal{U}$, where $\mathcal{Z}$ is the uniform distribution on $\{0,1\}^{pm}$ and $\mathcal{U}$ is the uniform distribution on $S_{[p]\times[m]}$. The random permutation π induces a random partition of the bits of z into blocks of size p: $U_1, U_2, \ldots, U_m$ where each U_i contains bits $z_{\pi(1,i)}, z_{\pi(2,i)}, \ldots, z_{\pi(p,i)}$.

We will apply the reduction from I-oblivious branching programs to multiparty communication complexity given by Proposition 2.7 in Chapter 2 to obtain our lower bound. To do this we will break the branching program for $Permuted\text{-}GIP_{p,M}$ into r layers consisting of many time steps and randomly assign each layer to the party that will simulate the layer. The following technical lemma will be useful in arguing that in the course of this assignment, it is likely that blocks of p tuples in the $Permuted\text{-}GIP_{p,m}$ input distribution can be placed on the foreheads of p different players.

Lemma 3.7. *For every $\delta > 0$, there is a $p_\delta > 0$ such that for every integer $p \geq p_\delta$ and $d \leq \frac{1}{8}\, p \log p$, the following holds: Let $G = (L, R)$ be a bipartite graph, where $|L| = |R| = p$. For each $i \in L$, repeat the following process $d_i \leq d$ times: independently at random choose a node j from R with probability $1/p$ and add edge (i, j) if it is not already present. Let $H = K_{p,p} - G$ be the graph resulting from subtracting G from $K_{p,p}$. Then H has a perfect matching with probability at least $1 - \delta$. In particular, if $p \geq 69$, this probability is at least $15/16$.*

Note that Lemma 3.7 is asymptotically tight with respect to d since, by the standard coupon collector analysis, for any $c > 1/\ln 2$ and $p \geq cp \log p$, the probability that even a single left vertex

has a neighbor in the graph H goes to 0 as p goes to infinity. Indeed its proof follows from the fact that below the coupon-collector bound the complement graph is a random bipartite graph of relatively large left degree and hence likely contains a matching. Moreover, we use Hall's Theorem that relates the presence of a matching in a bipartite graph to the presence of a subset of the left nodes with a shrinking neighborhood.

Theorem 3.8 (Hall's Theorem). *[80] Let $H = (L, R)$ be a bipartite graph. For a subset $S \subseteq L$, let $N_H(S)$ denote the neighborhood of S in R in the graph H. Then H has a matching if and only if for every subset $S \subseteq L$, $|N_H(S)| \geq |S|$.*

Proof of Lemma 3.7 . By Hall's theorem we can upper bound the probability that there is no perfect matching in $H = K_{p,p} - G$ by the probability that there is some $S \subseteq L$ with $1 \leq |S| \leq p-1$ and $|N_H(S)| \leq |S|$. (Any witnessing set S' for Hall's Theorem must be non-empty and the case that $|S'| = p$ is included in the probability that there is a set S with $|N_H(S)| \leq |S| = p-1$.) Fix $S \subseteq L$, let $|S| = s$, and fix $T \subseteq R$ such that $|T| = s$. Now $N_H(S) \subseteq T$ if and only if every $i \in S$ has an edge to every $j \in R \setminus T$ in the original graph G. (i, j) is not an edge in G if j is not one of the d_i choices for i; thus, we have $\Pr[(i,j) \text{ is an edge in } G] = 1 - \left(1 - \frac{1}{p}\right)^{d_i} \leq 1 - \left(1 - \frac{1}{p}\right)^{d} \leq 1 - 4^{-d/p} \leq 1 - \frac{1}{p^{1/4}}$, since $d_i \leq d \leq \frac{1}{8}\, p \log p$ and $\left(1 - \frac{1}{p}\right)^{d} \geq 4^{-d/p}$ for $p \geq 2$. For each $j \in R \setminus T$, these events are negatively correlated, hence $\Pr[\forall j \in R \setminus T, (i,j) \text{ is an edge in } G] \leq \left(1 - \frac{1}{p^{1/4}}\right)^{p-s}$. Since the choices for each $i \in S$ are independent, it follows that:

$$\Pr[\forall i \in S, \forall j \in R \setminus T, (i,j) \text{ is an edge in } G] \leq \left(1 - \frac{1}{p^{1/4}}\right)^{s(p-s)}.$$

By a union bound, we have

$$\begin{aligned}
&\Pr[\exists S \subseteq L,\ T \subseteq R,\ \text{s. t. } |T| = |S| \text{ and } N_H(S) \subseteq T] \\
&\quad \leq \sum_{s=1}^{p-1} \binom{p}{s}^2 (1 - \frac{1}{p^{1/4}})^{s(p-s)} \\
&\quad \leq \sum_{1 \leq s \leq p/2} \binom{p}{s}^2 (1 - \frac{1}{p^{1/4}})^{sp/2} + \sum_{p-1 \geq s \geq p/2} \binom{p}{s}^2 (1 - \frac{1}{p^{1/4}})^{(p-s)p/2}
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{1\le s\le p/2} \binom{p}{s}^2 (1-\frac{1}{p^{1/4}})^{sp/2} + \sum_{1\le p-s\le p/2} \binom{p}{p-s}^2 (1-\frac{1}{p^{1/4}})^{(p-s)p/2} \\
&\le 2 \sum_{1\le s\le p/2} \left[p^2(1-\frac{1}{p^{1/4}})^{p/2}\right]^s \\
&\le 2\sum_{s\ge 1} \left[p^2 e^{-p^{3/4}/2}\right]^s \\
&\le \frac{2p^2 e^{-p^{3/4}/2}}{1-p^2e^{-p^{3/4}/2}} \le \delta
\end{aligned}
$$

provided that $p \ge p_\delta$, where p_δ is a constant such that $\frac{2p_\delta^2 e^{-p_\delta^{3/4}/2}}{1-p_\delta^2 e^{-p_\delta^{3/4}/2}} \le \delta$. For $\delta = \frac{1}{16}$, $p_\delta \le 69$. Therefore, $H = K_{p,p} - G$ has a perfect matching with probability at least $15/16$, for $p \ge 69$. □

The following is our main lemma showing that we can convert z-oblivious branching programs approximating $Permuted\text{-}GIP$ under the hard distribution μ to an efficient communication protocol for $GIP_{p,m'}$ under the uniform distribution.

Lemma 3.9. *Let $p, m > 0$ be positive integers and let $N = mp$. Assume that $69 \le p \le \sqrt{N}/2$ and let $k \le \frac{1}{16} p \log p$. Let B be a z-oblivious branching program of z-length $T \le kN$ and z-width W that approximates $Permuted\text{-}GIP_{p,m}$ with error at most ϵ under the probability distribution μ. Then, for $m' = \frac{m}{2^{p+4}}$ and $\epsilon' = \epsilon + e^{-\frac{m}{2^{2p+3}}} + \frac{1}{p} + \frac{1}{8}$, under the uniform distribution on inputs there is a deterministic ϵ'-error NOF p-party protocol for $GIP_{p,m'}$ of complexity at most $k^2 p^3 2^{p+3} \log W$.*

Proof. Let $I = [N]$ be the set of input positions in z. Since B is z-oblivious we can let s be the sequence of at most kN elements from I that B queries in order. Divide s into r equal segments $s_1, s_2, \ldots, s_r$, each of length $\frac{kN}{r}$, where r will be specified later. Independently assign each segment s_i to a set in $A_1, A_2, \ldots, A_p$ with probability $1/p$. Denote this probability distribution by $\mathcal{A}$. Each A_j represents the elements of I that party j will have access to and hence we will place a subset of $I \setminus A_j$ on the forehead of party j. Since the sets $I \setminus A_j$, $j = 1, \ldots, p$, might overlap, they might not form a partition of I into p sets.

The distribution $\mu = (\mathcal{Z}, \mathcal{U})$ on $Permuted\text{-}GIP_{p,m}$ inputs randomly divides the elements of I into m blocks $U_1, U_2, \ldots, U_m$, each of size p. We calculate the probability, under the random assignment of segments to parties and z-bits to blocks, that we obtain a relatively large number of

blocks such that, for every party j, each block contains exactly one z-bit not belonging to A_j. We bound the probability that a block has z-bits such that:

(1) they do not occur too frequently in the sequence,

(2) their assignments to parties are independent, and

(3) each z-bit can be placed on the forehead of a different party.

Claim 3.10. *Except with probability at most $e^{-m/2^{2p+3}}$ over $\mathcal{U}$, all bits in at least $m/2^{p+2}$ blocks are read at most $2k$ times in s.*

By Markov's inequality, at least half the z-bits appear at most $2k$ times in s; call this set I_{2k}. Hence, $|I_{2k}| \geq N/2$. For a fixed $\ell \in [m]$, the probability that U_ℓ contains z-bits that appear at least $2k$ times in s is equal to $\binom{|I_{2k}|}{p}/\binom{N}{p}$. Hence,

$$\begin{aligned}
&\Pr_{\mathcal{U}}[\text{ all } z\text{-bits in } U_\ell \text{ appear at most } 2k \text{ times in } s] \\
&\geq \binom{N/2}{p}/\binom{N}{p} \\
&= \frac{N/2}{N} \cdots \frac{(N/2-(p-1))}{(N-(p-1))} \\
&= 2^{-p} \cdot \prod_{i=0}^{p-1} \left(1 - \frac{i}{N-i}\right) > \frac{1}{2^p}\left(1 - \frac{2p^2}{N}\right) \geq \frac{1}{2^{p+1}}
\end{aligned}$$

since $p \leq \sqrt{N}/2$. Hence, the expected number of such blocks is at least $\frac{m}{2^{p+1}}$. Let $\mathcal{E}_1$ be the event that the number of blocks for which all z-bits appear at most $2k$ times in s is at least $\frac{m}{2^{p+2}}$, which is at least half the expected number.

For simplicity here we use a single index i to select the bits of z. Let B_i be the block that z-bit i falls into according to the distribution $\mathcal{U}$. Let Y be the number of blocks for which all z-bits appear at most $2k$ times in s. We use Doob martingales and the Azuma-Hoeffding inequality as explained in Chapter 2.4 to bound the concentration of Y around its mean. The random variables $Y_t = \mathbb{E}[Y|B_1, B_2, \ldots, B_t]$ form a Doob martingale, with $Y_0 = \mathbb{E}[Y] \geq \frac{m}{2^{p+1}}$ and $Y_m = Y$. Let $\mathcal{E}_1$

be the event that Y is at least $m/2^{p+2}$. Then, by the Azuma-Hoeffding inequality, we have

$$\begin{aligned}\Pr_{\mathcal{U}}[\overline{\mathcal{E}_1}] &= \Pr\left[|Y_m - Y_0| \geq \frac{m}{2^{p+2}}\right] \\ &\leq e^{-2\frac{\left(\frac{m}{2^{p+2}}\right)^2}{m}} = e^{-m/2^{2p+3}}.\end{aligned}$$

Claim 3.11. *Except with probability at most $1/p$ over $\mathcal{U}$, there are at most $m/2^{p+3}$ blocks in which any two z-bits are queried in the same segment.*

Let $t(i)$ be the number of segments in which z-bit i appears, and write $i \sim i'$ if z-bits i and i' appear in the same segment at least once. Then $\sum_i t(i) \leq kN$ and the number of i' such that $i \sim i'$ is at most $t(i)kN/r$. A z-bit i is in a given block U_ℓ with probability $1/m$ and the events that i and i' are mapped there are negatively correlated. Hence for every $\ell \in [m]$, we have

$$\begin{aligned}&\Pr_{\mathcal{U}}[\exists i, i' \in U_\ell \text{ such that } i \sim i'] \\ &\leq \sum_i \sum_{i' \sim i} \frac{1}{m^2} \leq \sum_i \frac{t(i)kN}{r}\,\frac{1}{m^2} = \frac{k^2N^2}{rm^2} = \frac{p^2k^2}{r}.\end{aligned}$$

Setting $r = 8k^2p^32^p$, the expected number of such bad blocks is at most $m/(p2^{p+3})$. Hence, by Markov's inequality,

$$\Pr_{\mathcal{U}}\left[\text{ \# of blocks with } \sim z\text{-bits} \geq m/2^{p+3}\right] \leq 1/p.$$

Let $\mathcal{E}_2$ be the event that there are at least $m'' = m/2^{p+3}$ blocks such that each z-bit in these blocks is read at most $2k$ times and no two z-bits in any given block are read in the same segment. Call these blocks *good*. Then the above claims imply that $\Pr_{\mathcal{U}}[\overline{\mathcal{E}_2}] \leq e^{-m/2^{2p+3}} + 1/p$. For the remainder we condition on event $\mathcal{E}_2$.

Given that a block has z-bits occurring at most $2k$ times in s and no two z-bits appear in the same segment, we calculate the probability over $\mathcal{A}$ that the assignment of segments to parties ensures that each of the z-bits in the block can be placed on the forehead a different party. For such a block U_ℓ, we construct a bipartite graph where the left vertices represent the p z-bits in the block and the right vertices represent the p parties. We add an edge (i, j) if z-bit i **cannot** be assigned to party

j, because it is read in a segment in A_j. Observe that each z-bit in block U_ℓ can be placed on the forehead of a different party if and only if this graph contains a perfect matching. Since the segments are assigned to the various A_j independently, each with probability $1/p$, the resulting distribution on the graph is equivalent to that of the graph $K_{p,p} - G$ in Lemma 3.7 with $d = 2k \leq \frac{1}{8}\, p \log p$.

Since $p \geq 69$, we can apply Lemma 3.7 to say that the probability that the graph associated with block U_ℓ does not contain a perfect matching is at most $1/16$. By Markov's inequality, the probability, conditioned on $\mathcal{E}_2$, that fewer than $m''/2$ such blocks contain a perfect matching is at most $1/8$.

Let $\mathcal{E}_3 \subseteq supp(\mathcal{U}) \times supp(\mathcal{A})$ be the event that there are at least $m' = m/2^{p+4}$ blocks whose z-bits can be placed on the foreheads of different parties. Combining all the above, $\mathcal{E}_3$ occurs except with probability at most $\epsilon_1 = e^{-\frac{m}{2^{2p+3}}} + \frac{1}{p} + \frac{1}{8}$ over the distributions $\mathcal{U}$ and $\mathcal{A}$. There must be some choice of $A = (A_1, \ldots, A_p)$ for which the probability, over the distribution $\mathcal{U}$ on the grouping of blocks, that $\mathcal{E}_3$ does not occur is at most the average ϵ_1. We fix such an A.

Since the branching program B correctly computes $Permuted\text{-}GIP_{p,m}$ under distribution μ with probability at least $1 - \epsilon$, there must be some choice π of the permutation on the bits of $z_1, z_2, \ldots, z_p$ with $(\pi, A) \in \mathcal{E}_3$ such that B correctly computes $Permuted\text{-}GIP_{p,m}$ with probability at least $1-\epsilon-\epsilon_1$ under the distribution μ conditioned on the choice of π. This conditional distribution is now determined entirely by the uniform distribution $\mathcal{Z}$. Let I', with $|I'| = m'$, be the set of blocks witnessing event $\mathcal{E}_3$. By simple averaging there must be some assignment ζ to the blocks not in I' so that B correctly computes $Permuted\text{-}GIP_{p,m}$ with probability at least $1 - \epsilon - \epsilon_1$ under distribution μ conditioned on the choice of π and assignment ζ.

By construction, we can create a p-partition $\mathcal{P}'$ of the set of pm' bits in the blocks in I' so that each class contains precisely one z-bit from each block. We extend $\mathcal{P}'$ to a partition $\mathcal{P}$ of all of $[N]$ by arbitrarily assigning to each class one z-bit of each block not in I' and dividing the N values representing π equally among the parties. Applying Proposition 2.7 with $r = k^2p^3 2^{p+3}$, we obtain a deterministic NOF p-party protocol of complexity $k^2p^3 2^{p+3} \log W$ for $Permuted\text{-}GIP_{p,m}$ with error $\epsilon' = \epsilon + \epsilon_1$ under distribution μ'.

We reduce the communication problem for $GIP_{p,m'}$ under the uniform distribution to that for $1GAP_n$ under μ' by observing that the inputs to the $GIP_{p,m'}$ problem on a uniformly random

input can be embedded into the unfixed blocks of the $Permuted\text{-}GIP_{p,m}$ instances induced by the permutation π given by the distribution μ'. □

We now apply the following lower bound derived Babai, Nisan and Szegedy for $GIP_{p,m}$ under the uniform distribution.

Proposition 3.12. *[22]* $D_{\epsilon,p}^{uniform}(GIP_{p,m})$ *is* $\Omega(m/4^p + \log(1-2\epsilon))$.

Theorem 3.13. *Let* $\epsilon < 1/2$. *There is a* $p \leq \log(m/S)$ *such that if a randomized* z*-oblivious branching program computes* $Permuted\text{-}GIP_{p,m}$ *with time* T*, space* S *and error at most* ϵ*, then* $T \in \Omega\left(N \log(\frac{N}{S}) \log\log(\frac{N}{S})\right)$ *where* $N = pm$.

Proof. Let $\tilde{B}$ be a z-oblivious randomized branching program computing $Permuted\text{-}GIP_{p,m}$ of time T and width W. If $\epsilon > 1/5$, we amplify the success probability to $4/5$ by running $\tilde{B}$ a constant number C of times, as is standard with probability amplification. The new z-oblivious branching program computes $Permuted\text{-}GIP_{p,m}$ in time CT, width CW and error at most $1/5$: it has C stages, each representing one run of $\tilde{B}$ on the input and each stage $i = 0, \ldots, C-1$ contains i copies of $\tilde{B}$ corresponding to the number of 1's in the outputs of the previous $i-1$ runs of $\tilde{B}$. Therefore, we can assume without loss of generality that the error ϵ of $\tilde{B}$ is $< 1/5$.

Apply Yao's Lemma to $\tilde{B}$ using distribution μ to obtain a deterministic z-oblivious branching program B with the same time and space bounds that computes $Permuted\text{-}GIP_{p,m}$ with error at most ϵ under μ.

Let $T = kN$ and let p be the smallest integer ≥ 69 such that $k \leq \frac{1}{16} p \log p$. If $p \geq \log(N/S)/4$ then the result is immediate so assume without loss of generality that $69 \leq p < \log(N/S)/4$. Let $\epsilon_1 = e^{-\frac{m}{2^{2p+3}}} + \frac{1}{p} + \frac{1}{8}$ which is $< 1/5$ for these values of p.

Since $69 \leq p \leq \sqrt{N}/2$ we can combine Lemma 3.9 and Proposition 3.12 to say that there is a constant C independent of N and p such that

$$k^2 p^3 2^{p+3} \log W \geq C(\frac{m'}{4^p} + \log(1-2\epsilon')),$$

where $m' = m/2^{p+4} = \frac{N}{p2^{p+4}}$ and $\epsilon' = \epsilon + \epsilon_1 \leq 2/5$. Rewriting m' and k in terms of N and p and using $S \geq \log W$, we obtain $Sp^7 \log^2 p \geq C_1 n 4^{-2p}$, for some constant C_1. Simplifying and taking

logarithms, we have $p \geq C_2 \log(\frac{N}{S})$, for some constant C_2. Since p is the smallest integer ≥ 69 such that $k \leq \frac{1}{16} p \log p$, we have $k \geq C_3 \log(\frac{N}{S}) \log\log(\frac{N}{S})$ for some constant C_3 and the theorem follows. □

Using Theorem 3.13 together with Proposition 3.6 and Corollary 3.5, we immediately get our claimed time-space tradeoff for randomized oblivious branching programs computing $1GAP_n$.

Corollary 3.14. *Let $\epsilon < 1/2$. If a randomized oblivious branching program computes $1GAP_n$ with time T, space S and error at most ϵ, then $T \in \Omega\left(n \log(\frac{n}{S}) \log\log(\frac{n}{S})\right)$.*

3.7 Time-Space Tradeoff Lower Bound for Randomized Oblivious Boolean Branching Programs

$1GAP_n$ requires $\Omega(n \log n)$ bits of input and hence is unsuitable for separations involving Boolean branching programs. As with $1GAP_n$, specifying the permutation in the input to $Permuted\text{-}GIP_{p,m}$ also requires $\Theta(N \log N)$ bits where $N = pm$. Instead, we consider $GIP\text{-}MAP$, an extension of $GIP_{p,n/p}$ where the input bits are shuffled by an almost $2p$-wise independent permutation and arranges these bits into the p vectors $z_1, z_2, \ldots, z_p$ that are the input to $GIP_{p,n/p}$. The key difference is that specifying the permutation requires only $O(p \log n)$ bits.

DEFINITION 3.1. *A set of permutations $\mathcal{F}$ of A with $|A| = n$ is a* δ-almost t-wise independent *family of permutations iff for every set of t distinct points $a_1, \ldots, a_t \in A$ and for π chosen uniformly from $\mathcal{F}$, the distribution of $(\pi(a_1), \ldots, \pi(a_t))$ is δ-close to the uniform distribution on sequences of t distinct points from A. It is simply* t-wise independent *if $\delta = 0$.*

For any prime power q, the set of permutations on $\mathbb{F}_q$ given by $\mathcal{F}_{2,q} = \{f_{a,b} \mid a \neq 0, b \in \mathbb{F}_q\}$ where $f_{a,b}(x) = ax + b$ over $\mathbb{F}_q$ is a pairwise independent family of permutations. For $t > 2$, the family $\mathcal{F}_{t,q}$ consisting of all polynomials of degree $t-1$ over $\mathbb{F}_q$ is a t-wise independent family of functions but there is no analogous subset of this family that yields a t-wise independent family of permutations. After publishing our results, Kuperberg, Lovett and Peled [103] gave a non-constructive proof that t-wise independent family of permutations exist. Since their proof is only existential, we still need to use almost t-wise independent permutations. While there are now a number of constructions of almost $2p$-wise independent random permutations in the literature, for

simplicity we fix a construction of Naor and Reingold [119] based on analyzing variants of Luby and Rackoff's pseudorandom permutation construction that uses Feistel operations [110]. They showed that a simple combination of two copies of each of these two kinds of pseudorandom families yields a family of permutations that is δ-almost t-wise independent and pairwise independent provided t is not too large and δ is not too small.

Let w be an integer and for $\ell = w, 2w$, identify the elements of $\mathbb{F}_{2^\ell}$ with $\{0,1\}^\ell$. The construction uses the *Feistel operator* F_f on $2w$ bits which maps (x, y) for $x, y \in \{0,1\}^w$ to $(y, f(x) \oplus y)$ where $f : \{0,1\}^w \to \{0,1\}^w$. Define a family $\mathcal{F}_t^w$ of permutations on $\mathbb{F}_{2^{2w}}$ as the set of all functions constructed as follows: For each independent choice of h_1, h_2 from $\mathcal{F}_{2,2^{2w}}$ and f_1, f_2 from $\mathcal{F}_{t,2^w}$ define the permutation

$$\pi_{h_1,h_2,f_1,f_2} = h_2^{-1} \circ F_{f_2} \circ F_{f_1} \circ h_1.$$

Observe that $8w + 2tw$ bits suffice to specify an element of $\mathcal{F}_t^w$.

Proposition 3.15. *([119] Corollary 8.1) Let w be an integer and t be an integer. Then $\mathcal{F}_t^w$ is δ-almost t-wise independent family of permutations on $\mathbb{F}_{2^{2_w}}$ for $\delta = t^2/2^w + t^2/2^{2w}$ that also forms a pairwise independent family of permutations.*

DEFINITION 3.2. *Let N be a positive integer and $n = 2^{2w}$ be the largest even power of 2 such that $n + \log^2 n \leq N$. Let p be a power of 2 such that $2 \leq p \leq \frac{1}{8}\log n$ (of which the are fewer than $\log\log n$ possibilities).*

Define $GIP\text{-}MAP_N : \{0,1\}^N \to \{0,1\}$ as follows: We interpret input bits $n+1, \ldots, n + \log\log\log n$ as encoding the value $p \leq \frac{1}{8}\log n$ and the next $8w + 4pw \leq \frac{3}{4}\log^2 n$ bits as encoding a permutation π from $\mathcal{F}_{2p}^w$ which we identify with permutation on $[n]$.

$$GIP\text{-}MAP_N(x_1 x_2 \ldots x_n, p, \pi) = GIP_{p,n}(z_1, z_2, \ldots, z_p),$$

where $z_i = x_{\pi((i-1)n/p+1)} \ldots x_{\pi(in/p)}$, $i = 1, \ldots, p$.

Proposition 3.16. *$GIP\text{-}MAP_N$ is computable by a deterministic Boolean branching program using time N and $O(\log^2 N)$ space.*

Proof. The program begins with a full decision tree that first reads the bits of the encoding of p and then the bits encoding π. At each leaf of the tree, the program contains a copy of the width 4 branching program computing $GIP_{p,n/p}$ where variable z_{ij} is replaced by $x_{\pi((i-1)n+j)}$. □

We obtain the following time-space tradeoff lower bound for $GIP\text{-}MAP_N$.

Theorem 3.3. *Let $\epsilon < 1/2$. Any randomized oblivious Boolean branching program computing $GIP\text{-}MAP_N$ with time T, space S and error at most ϵ requires then $T \in \Omega\left(N \log(\frac{N}{S}) \log\log(\frac{N}{S})\right)$.*

Proof. The proof follows the basic structure of the argument for $Permuted\text{-}GIP_{p,m}$, except that we now fix the p that is part of the input. Let n be given as in the definition of $GIP\text{-}MAP_N$. The δ-almost $2p$-wise independence of the permutation ensures that the probability that a block of the permuted $GIP_{p,n/p}$ problem has all its variables accessed at most $2k$ times is roughly 2^{-p} and that these events are roughly pairwise independent. The pairwise independence of the permutation ensures that two variables in a block are unlikely to be assigned to the same segment.

Let $\tilde{B}$ be a randomized oblvious branching program computing $GIP\text{-}MAP_N$ and assume without loss of generality that the error ϵ of $\tilde{B}$ is $< 1/5$. We can assume without loss of generality that $\log(n/S) \geq 2^{10}$ or the result follows immediately. Otherwise let p be the largest power of 2 such that $p \leq \frac{1}{8}\log(n/S)$. Then $p \geq 69$. Let μ_p be the uniform distribution over the input bits to $GIP\text{-}MAP_N$ conditioned on the fixed value of p. Apply Yao's Lemma to $\tilde{B}$ using distribution μ_p to obtain a deterministic oblivious branching program B with the same time and space that computes $GIP\text{-}MAP_N$ with error at most ϵ under μ_p.

Suppose that the time of B, $T \leq kn$ where $k = \frac{1}{16}p\log p$. Since B is oblivious we can let s be the sequence of at most kn elements from the set of input positions $I = [n]$ that B queries, in order. (We do not include the input positions queried outside of $[n]$ since their values will eventually be fixed.) Divide s into r equal segments $s_1, s_2, \ldots, s_r$, each of length at most $\frac{kn}{r}$, where r will be specified later. Independently assign each segment s_i to a set in $A_1, A_2, \ldots, A_p$ with probability $1/p$. Denote this probability distribution $\mathcal{A}$. Each A_j represents the elements of L that party j will have access to and hence we will place a subset of $I \setminus A_j$ on the forehead of party j. Since the sets $L \setminus A_j$, $j = 1, \ldots, p$, might overlap, they might not form a partition of I into p sets.

The permutation π randomly divides $[n]$ into $m = n/p$ blocks $U_1, U_2, \ldots, U_m$, each of size

p where block U_j contains the j^{th} bits of the vectors $z_1, z_2, \ldots, z_p$ as given in the definition of $GIP\text{-}MAP_N$. By construction, the distribution of π is δ-almost $2p$-wise independent for $\delta = 8p^2/\sqrt{n}$.

We now follow many of the lines of the remainder of the proof of Lemma 3.9 and give full details where the proofs differ. The key difference in the calculations is that we no longer have a truly random permutation. The parameter n here corresponds to N in the proof of Lemma 3.9.

As before, we calculate the probability, under the random assignment of segments to parties and elements of $[n]$ to blocks, that we obtain a relatively large number of blocks such that, for every party j, each block contains exactly one element of $[n]$ not belonging to A_j. To do this, we bound the probability that a block has elements of $[n]$ such that:

(1) they do not occur too frequently in the sequence,

(2) their assignments to parties are independent, and

(3) each element can be placed on the forehead of a different party.

In the proof of Lemma 3.9, conditioned on (1) and (2), the argument for (3) is independent of the choice of π and depends only on the randomness of the assignment of segments to parties. The proof of (2) depends only on the pairwise independence of π which is guaranteed here by Proposition 3.15. Only the proof of part (1) needs to be modified substantially.

As before, we first remove all input indices that appear more than $2k$ times in the sequence s. By Markov's inequality, at least half the input indices appear at most $2k$ times in s. Let the first $n/2$ elements of this set be G.

Therefore, for $\ell \in [m]$, let Y_ℓ be the indicator function for the event that $U_\ell \subset G$. Then since π is δ-almost $2p$-wise independent

$$\begin{aligned}
\Pr_{\mu_p}[Y_\ell = 1] &= \Pr_{\mu_p}[U_\ell \subset G] \\
&\geq (n/2)^{(p)}/n^{(p)} - \delta \\
&> 2^{-p} - 2^{-p-1}p^2/n - \delta \\
&= 2^{-p} - \delta'
\end{aligned}$$

where $\delta' = \delta + 2^{-p-1}p^2/n < 9p^2/\sqrt{n}$. Similarly and more simply, $\Pr_{\mu_p}[Y_\ell = 1] \le 2^{-p} + \delta \le 2^{-p} + \delta'$. Let $\mathcal{E}_1$ be the event that the number of blocks for which all elements appear at most $2k$ times in s is at least $m'' = \frac{n}{p2^{p+1}}$.

We use the second moment method to upper bound $\Pr_{\mu_p}[\overline{\mathcal{E}_1}]$. Let Y be the number of blocks for which all elements appear in G. Then $Y = \sum_{\ell \in [n/p]} Y_\ell$ and $|\mathbb{E}(Y) - \frac{n}{p2^p}| \le \frac{n}{p}\delta'$. Since the Y_i are indicator variables $Var(Y) = \mathbb{E}(Y) + \sum_{i \ne j} Cov(Y_i, Y_j)$ where $Cov(Y_i, Y_j) = \Pr[Y_i Y_j = 1] - \Pr[Y_i = 1]\Pr[Y_j = 1]$. Since the outputs of π are δ-almost $2p$-wise independent, we have: $\Pr[Y_i Y_j = 1] = \Pr[U_i \cup U_j \subseteq G] \le 2^{-2p} + \delta \le 2^{-2p} + \delta'$. Therefore

$$\begin{aligned}
Var(Y) \le & \frac{n}{p}2^{-p} + \frac{n}{p}\delta' \\
& + \frac{n}{p}(\frac{n}{p} - 1)[2^{-2p} + \delta' + (2^{-p} - \delta')^2] \\
= & \frac{n}{p}2^{-p} + \frac{n}{p}\delta' + \frac{n}{p}(\frac{n}{p} - 1)\delta'[1 + 2^{1-p} - \delta'] \\
\le & \frac{n}{p}2^{-p} + \frac{2n^2}{p^2}\delta'.
\end{aligned}$$

Now $\mathcal{E}_1$ holds if $Y \ge m'' = \frac{n}{p2^{p+1}} \ge \mathbb{E}(Y) - \frac{n}{p}(2^{-p-1} + \delta')$. So by Chebyshev's inequality, we have

$$\begin{aligned}
\Pr_{\mu_p}[\overline{\mathcal{E}_1}] &\le \Pr\left[|Y - \mathbb{E}(Y)| \ge \frac{n}{p}(2^{-p-1} + \delta')\right] \\
&\le \frac{Var(Y)}{(\frac{n}{p}(2^{-p-1} + \delta'))^2} \\
&\le \frac{(n/p)2^{-p} + 2(n/p)^2\delta'}{(n/p)^2(2^{-p-1} + \delta')^2} \\
&\le \frac{p2^{p+2}}{n} + 2\delta' \\
&= \frac{p2^{p+2}}{n} + \frac{18p^2}{\sqrt{n}}
\end{aligned}$$

Since $69 \le p \le \frac{1}{8}\log n$, we obtain $\Pr_{\mu_p}[\overline{\mathcal{E}_1}] \le 2^{-3p}$.

As in the proof of Lemma 3.9 let $t(i)$ be the number of segments in which i appears, and write $i \sim i'$ if elements i and i' appear in the same segment at least once. Then the number of i' such that $i \sim i'$ is at most $t(i)kn/r$ and $\sum_i t(i) = kn$. By construction the random permutation π is pairwise

independent and hence it maps any two input bits $i \neq i' \in [n]$ to two randomly chosen distinct points in $[n]$. Therefore the probability that they are both chosen for some block U_j is precisely $p(p-1)/n(n-1) \leq p^2/n^2$. Hence for every $\ell \in [n/p]$, we have

$$\begin{aligned}&\Pr_{\mu_p}[\exists i, i' \in U_\ell \text{ such that } i \sim i'] \\ &\leq \sum_i \sum_{i' \sim i} \frac{p^2}{n^2} = \sum_i \frac{t(i)kn}{r} \frac{p^2}{n^2} = \frac{k^2p^2n^2}{rn^2} = \frac{k^2p^2}{r}.\end{aligned}$$

Setting $r = k^2p^3 2^{p+2}$, the expected number of such blocks is at most $\frac{n}{p^2 2^{p+2}} = m''/(2p)$. Hence, by Markov's inequality, $\Pr_{\mu_p}[$ the number of blocks with $\sim$ tuples $\geq m''/2] \leq \frac{1}{p}$. Let $\mathcal{E}_2$ be the event that there are at least $\frac{m''}{2} = \frac{n}{p2^{p+2}}$ blocks such that each bit in these blocks is read at most $2k$ times and no two bits in any block are read in the same segment. Then $\Pr_{\mu_p}[\overline{\mathcal{E}_2}] \leq 2^{-3p} + 1/p$.

Conditioned on the event $\mathcal{E}_2$, the probability that the number of blocks among the $m' = m''/2$ blocks guaranteed by $\mathcal{E}_2$ for which elements that can be placed on the forehead of the p different parties is independent of the choice of π and depends only on the assignment $\mathcal{A}$. By the same calculation as that of Lemma 3.9 with the value of m'' here, except with a probability of $1/8$, conditioned on $\mathcal{E}_2$, there are at least $m' = \frac{n}{8p2^p}$ blocks whose elements can be placed on the foreheads of different parties. Let $\mathcal{E}_3$ be the probability over the joint distribution of μ_p and $\mathcal{A}$ that there are at least m' such blocks. The $\Pr[\overline{\mathcal{E}}_3]$ is at most $\epsilon_1 = 2^{-3p} + 1/p + 1/8$. There must be some choice of $A = (A_1, \ldots, A_p)$ for which the probability, over the distribution μ_p on the grouping of blocks, that $\mathcal{E}_3$ does not occur is at most the average ϵ_1. We fix such an A.

Since the branching program B correctly computes $GIP\text{-}MAP_N$ under distribution μ_p with probability at least $1 - \epsilon$, there must some choice $\boldsymbol{\pi}$ of the permutation that groups the elements of $[n]$ into blocks with $(\boldsymbol{\pi}, A) \in \mathcal{E}_3$ such that B correctly computes $GIP\text{-}MAP_N$ with probability at least $1 - \epsilon - \epsilon_1$ under the distribution μ_p conditioned on the choice of $\boldsymbol{\pi}$. (This conditional distribution is now determined entirely by the uniform distribution over $\{0,1\}^n$.)

Let I', with $|I'| = m'$ be the set of blocks witnessing event $\mathcal{E}_3$. By averaging there must be some assignment ζ to the blocks not in I' so that B correctly computes $GIP\text{-}MAP_N$ with probability at least $1 - \epsilon - \epsilon_1$ under distribution μ conditioned on the choice of the permutation $\boldsymbol{\pi}$ and assignment ζ. Let μ' be this conditional distribution which is uniform on the inputs appearing in the blocks of

I'. As in the proof of Lemma 3.9 we can use the branching program B to obtain a deterministic p-party communication protocol of complexity at most $rS = k^2p^3 2^{p+2}S$ that computes $GIP_{p,m'}$ with the standard input partition for a uniformly random input in $\{0,1\}^{pm'}$ with error at most $\epsilon' = \epsilon + \epsilon_1 < 2/5$.

Hence, by Proposition 3.12, there is an absolute constant C such that $k^2p^3 2^{p+2}S \geq \frac{Cm'}{4^p} = \frac{Cn}{8p2^{3p}}$. Since $k = \frac{1}{16}p\log p$, we obtain $2^{4p}p^6\log^2 p \geq 8Cn/S$ which contradicts the assumption that p is the largest power of 2 smaller than $\frac{1}{8}\log(n/S)$ for n/S sufficiently large.

Our only hypothesis was that $T \leq kn$ so we must have $T > kn = \frac{1}{16}np\log p$ which is at least $cn\log(n/S)\log\log(n/S))$ for some constant $c > 0$. Since n is $\Theta(N)$, the theorem follows. □

3.8 Discussion and Open Questions

Our results apply to randomized oblivious algorithms and are the largest explicit time-space tradeoff lower bounds known for randomized non-uniform computational models. However, it would be interesting to extend these bounds to more powerful classes of randomized branching programs, in particular oblivious randomized ones where the probability distribution on the input sequence is independent of the input. We conjecture that $1GAP_n$ is also hard for this stronger oblivious randomized model. It is important to note that if we applied Yao's Lemma directly on this model then we would lose the requirement of obliviousness when the randomness is fixed, hence a novel technique for analyzing randomized branching programs is needed.

Chapter 4

SLIDING WINDOW FUNCTIONS

Problems related to computing elementary statistics of input data have wide applicability and utility. Despite their usefulness, there are surprising gaps in our knowledge about the best ways to solve these simple problems, particularly in the context of limited storage space. Many of these elementary statistics can be easily calculated if the input data is already sorted but sorting the data in space S, requires time $T \in \Omega(n^2/S)$ [43, 34], a bound matched by the best comparison algorithms [123] for all $S \in O(n/\log n)$. It has not been clear whether exactly computing elementary statistical properties, such as frequency moments or element distinctness are as difficult as sorting when storage is limited.

4.1 Frequency Moments, Element Distinctness, and Order Statistics

Let $a = a_1 a_2 \dots a_n \in D^n$ for some finite set D. Frequency moments measure the diversity of symbols in a.

DEFINITION 4.1. *The* k^{th} frequency moment *of* a*,* $F_k(a)$*, as* $F_k(a) = \sum_{i \in D_a} f_i^k$*, where* f_i *is the frequency (number of occurrences) of symbol* i *in the string* a *and* D_a *is the set of symbols that occur in* a*. Therefore,* $F_0(a)$ *is the number of distinct symbols in* a *and* $F_1(a) = |a|$ *for every* a*.*

The element distinctness problem can be viewed as a decision-function version of F_0.

DEFINITION 4.2. *The* element distinctness *problem is a decision problem defined as:* $ED(a) = 1$ *if* $F_0(a) = |a|$ *and* 0 *otherwise. We write* ED_n *for the* ED *function restricted to inputs* a *with* $|a| = n$*.*

Order statistics reflect the landscape of the symbols of a. They are very simple to calculate if a is sorted.

DEFINITION 4.3. *The t^{th}* order statistic *of a, O_t, is the t^{th} smallest symbol in a. Therefore O_n is the maximum of the symbols of a and $O_{\lceil \frac{n}{2} \rceil}$ is the median.*

Element distinctness has been a particular focus of lower bound analysis. The first time-space tradeoff lower bounds for the problem apply to structured algorithms. Borodin, Razborov and Smolensky [44] gave a time-space tradeoff lower bound for computing ED on *comparison* branching programs of $T \in \Omega(n^{3/2}/S^{1/2})$ and, since $S \geq \log_2 n$, $T \in \Omega(n^{3/2}\sqrt{\log n}/S)$. Yao [155] improved this to a near-optimal $T \in \Omega(n^{2-\epsilon(n)}/S)$, where $\epsilon(n) = 5/(\ln n)^{1/2}$. Since these lower bounds apply to the average case for randomly ordered inputs, by Yao's lemma, they also apply to randomized comparison branching programs. These bounds also trivially apply to all frequency moments since, for $k \neq 1$, $ED(x) = n$ iff $F_k(x) = n$. This near-quadratic lower bound seemed to suggest that the complexity of ED and F_k should closely track that of sorting. For multi-way branching programs, Ajtai [8] showed that any linear time algorithm for ED must consume linear space. Moreover, when S is $n^{o(1)}$, Beame, Saks, Sun and Vee [32] showed a $T \in \Omega(n\sqrt{\log(n/S)/\log\log(n/S)})$ lower bound for computing ED. This is a long way from the comparison branching program lower bound and there has not been much prospect for closing the gap since the largest lower bound known for multi-way branching programs computing *any* single-output problem in P is only $T \in \Omega(n \log((n \log n)/S))$ [32].

4.2 Sliding Windows

Given the general difficulty of obtaining strong lower bounds for single-output functions, we consider the relative complexity of computing many copies of each of the functions at once and apply techniques for multi-output functions to make the comparison. Since we want to retain a similar input size to that of our original problems, we need to evaluate them on overlapping inputs.

A particularly natural circumstance in which one would want to evaluate many instances of a function on overlapping inputs occurs in the context of time series analysis. For many functions computed over sequences of data elements or data updates, it is useful to know the value of the function on many different intervals or *windows* within the sequence, each representing the recent history of the data at a given instant. In the case that an answer for every new element of the sequence is required, such computations have been termed *sliding-window* computations for the

associated functions [59]. In particular, we consider inputs of length $2n-1$ where the sliding-window task is to compute the function for each window of length n, giving n outputs in total. We write $F^{\boxplus n}$ to denote this sliding-window version of a function F.

DEFINITION 4.4. *Let D and R be two finite sets and $f : D^n \to R$ be a function over strings of length n. The operation $\boxplus$ takes the function f and returns a function $f^{\boxplus t} : D^{n+t-1} \to R^t$, defined by $f^{\boxplus t}(x) = (f(x_i \dots x_{i+n-1}))_{i=1}^{t}$.*

Because the operator $\boxplus$ produces a large number of outputs while less than doubling the input size, we concentrate on the case that $t = n$ and apply it to element distinctness, frequency moments and order statistics.

Many natural functions have been studied for sliding windows including entropy, finding frequent symbols, frequency moments and order statistics, which can be computed approximately in small space using randomization even in one-pass data stream algorithms [59, 23, 19, 107, 108, 51, 48]. Approximation is required since exactly computing these values in this online model can easily be shown to require large space. The interested reader may find a more comprehensive list of sliding-windows results by following the references in [48].

4.3 Contributions and Techniques

We show that the complexity gap between sorting and element distinctness cannot be closed. More precisely, we give a randomized multi-way branching program algorithm that for any space bound $S \in [c \log n, n]$ computes ED in time $T \in \tilde{O}(n^{3/2}/S^{1/2})$, significantly beating the lower bound that applies to comparison-based algorithms. Our algorithm for ED is based on an extension of Floyd's cycle-finding algorithm [98] (more precisely, its variant, Pollard's rho algorithm [125]). Pollard's rho algorithm finds the unique collision reachable by iterating a function $f : [n] \to [n]$ from a single starting location in time proportional to the size of the reachable set, using only a constant number of pointers. Variants of this algorithm have been used in cryptographic applications to find collisions in functions that supposedly behave like random functions [49, 134, 120].

More precisely, our new ED algorithm is based on a new deterministic extension of Floyd's algorithm to find *all* collisions of a function $f : [n] \to [n]$ reachable by iterating f from any one

of a set of k starting locations, using only $O(k)$ pointers and using time roughly proportional to the size of the reachable set. Motivated by cryptographic applications, [148] previously considered this problem for the special case of random functions and suggested a method using 'distinguished points', though the only analysis they gave was heuristic and incomplete. Our algorithm, developed independently, uses a different method, applies to arbitrary functions, and has a fully rigorous analysis.

Our algorithm for ED does not obviously apply to the computation of frequency moments, such as F_0, and so it is interesting to ask whether or not frequency moment computation is harder than that of ED and may be closer in complexity to sorting. We show that computing ED over n sliding windows only incurs a polylogarithmic overhead in time and space versus computing a single copy of ED. In particular, we can extend our randomized multi-way branching program algorithm for ED to yield an algorithm for $ED^{\boxplus n}$ that for space space $S \in [c \log n, n]$ runs in time $T \in \tilde{O}(n^{3/2}/S^{1/2})$.

In contrast, we prove strong time-space lower bounds for computing the sliding-window version of any frequency moment F_k for $k \neq 1$. In particular, the time T and space S to compute $F_k^{\boxplus n}$ must satisfy $T \in \Omega(n^2/S)$ and $S \geq \log n$. (F_1 is simply the size of the input, so computing its value is always trivial.) The bounds are proved directly for randomized multi-way branching programs which imply lower bounds for the standard RAM and word-RAM models, as well as for the data stream models discussed above. Moreover, we show that the same lower bound holds for computing just the parity of the number of distinct elements, $F_0 \bmod 2$, in each window. This formally proves a separation between the complexity of sliding-window $F_0 \bmod 2$ and sliding-window ED. These results suggest that in the continuing search for strong complexity lower bounds, $F_0 \bmod 2$ may be a better choice as a difficult decision problem than ED.

Our lower bounds for frequency moment computation hold for randomized algorithms even with small success probability $2^{-O(S)}$ and for the average time and space used by deterministic algorithms on inputs in which the values are independently and uniformly chosen from $[n]$. (For comparison with the latter average case results, it is not hard to show that over the same input distribution, ED can be solved with $\overline{T} \in \tilde{O}(n/\overline{S})$ and our reduction shows that this can be extended to $\overline{T} \in \tilde{O}(n/\overline{S})$ bound for $ED^{\boxplus n}$ on this input distribution.)

We complement our lower bound with a comparison-based RAM algorithm for any $F_k^{\boxplus n}$ that has $T \in \tilde{O}(n^2/S)$, showing that this is nearly an asymptotically tight bound, since it provides a general RAM algorithm that runs in the same time complexity. Since our algorithm for computing $F_k^{\boxplus n}$ is comparison-based, the comparison lower bound for F_k implied by [155] is not far from matching our algorithm even for a single instance of F_k.

It is interesting to understand how the complexity of computing a function F can be related to that of computing $F^{\boxplus n}$. To this end, we consider problems of computing the t^{th} order statistic in each window. For these problems we see the full range of relationships between the complexities of the original and sliding-window versions of the problems. In the case of $t = n$ (maximum) or $t = 1$ (minimum) we show that computing these properties over sliding windows can be solved by a comparison based algorithm in $O(n \log n)$ time and only $O(\log n)$ bits of space so there is very little growth in complexity. In contrast, we show that a $T \in \Omega(n^2/S)$ lower bound holds when $t = \alpha n$ for any fixed $0 < \alpha < 1$. Even for algorithms that only use comparisons, the expected time for errorless randomized algorithms to find the median in a single window is $\overline{T} \in \Omega(n \log \log_S n)$ [52] and there is an errorless randomized algorithm that precisely matches this bound [52]. Hence, these problems have a dramatic increase in complexity over sliding windows.

While our lower bounds apply to randomized branching programs, which allow the strongest non-explicit randomness, our randomized algorithms for element distinctness will only require a weaker notion, input randomness, in which the random string r is given as an explicit input to a RAM algorithm. For space-bounded computation, it would be preferable to only require the random bits to be available online as the algorithm proceeds.

4.4 Related Work

While sliding-windows versions of problems have been considered in the context of online and approximate computation, there is little research that has explicitly considered any such problems in the case of exact offline computation. One instance where a sliding-windows problem has been considered is a lower bound for generalized string matching due to Abrahamson [4]. This lower bound implies that for any fixed string $y \in [n]^n$ with n distinct values, $H_y^{\boxplus n}$ requires $T \cdot S \in \Omega(n^2/\log n)$ where decision problem $H_y(x)$ is 1 if and only if the Hamming distance between x

and y is n. This bound is an $\Omega(\log n)$ factor smaller than our lower bound for sliding-window $F_0 \bmod 2$.

4.5 Element Distinctness and Small-Space Collision-Finding

4.5.1 Efficient small-space collision-finding with many sources

Our approach for solving the element distinctness problem has at its heart a novel extension of Floyd's small space "tortoise and hare" cycle-finding algorithm [98]. Given a start vertex v in a finite graph $G = (V, E)$ of outdegree 1, Floyd's algorithm finds the unique cycle in G that is reachable from v. The out-degree 1 edge relation E can be viewed as a set of pairs $(u, f(u))$ for a function $f : V \to V$. Floyd's algorithm, more precisely, stores only two values from V and finds the smallest s and $\ell > 0$ and vertex w such such that $f^s(v) = f^{s+\ell}(v) = w$ using only $O(s + \ell)$ evaluations of f. The algorithm maintains two pointers, the "tortoise" and the "hare". Initially, both the tortoise and the hare are at vertex v. At each time step, we update their positions as follows: if the tortoise is at a vertex u, we move it to vertex $f(u)$; if the hare is at a vertex u, we move it to vertex $f(f(u))$. It is easy to see that the tortoise and hare meet after $O(s + l)$ steps. To find the two vertices v_1 and v_2 such that $f(v_1) = f(v_2) = w$, we re-position the tortoise to v and move both pointers one position at a time until they meet at w. This can be achieved in s additional steps.

We say that vertices $u \neq u' \in V$ are *colliding* iff $f(u) = f(u')$ and call $v = f(u) = f(u')$ a collision. Floyd's algorithm for cycle-finding can also be useful for finding collisions since in many instances the starting vertex v is not on a cycle and thus $s > 0$. In this case $i = f^{s-1}(v) \neq j = f^{s+\ell-1}(v)$ satisfy $f(i) = f(j) = w$, which is a collision in f, and the iterates of f produce a ρ shape (see Figure 4.1(a)). These colliding points may be found with minimal cost by also storing the previous values of each of the two pointers as Floyd's algorithm proceeds. The ρ shape inspired the name of Pollard's rho algorithm for factoring [125] and solving discrete logarithm problems [126]. This in turn is also the commonly associated name for the application of Floyd's cycle finding algorithm to the general problem of collision-finding.

Within the cryptographic community there is an extensive body of work building on these early advances of Pollard. There is also considerable research on cycle detection algorithms that use larger space than Floyd's algorithm and improve the constant factors in the number of edges (function

evaluations) that must be traversed to find the cycle (see for example [49, 134, 120] and references therein). In these applications, the goal is to find a single collision reachable from a given starting point as efficiently as possible.

The closest existing work to our problem tackles the problem of speeding up collision detection in random functions using parallelization [148]. The authors give a deterministic parallel algorithm for finding all collisions of a random function along with a heuristic analysis of its time and space performance. There are some similarities and crucial differences that a serialization of their algorithm would have to our solution. Their algorithm keeps a record of visits to predetermined vertices (so-called 'distinguished points') which allow the separate processes to determine quickly if they are on a path that has been explored before. Further, their heuristic argument suggests a bound of $O(n^{3/2}/S^{1/2})$ function iterations, though they were unable to supply a rigorous argument. This heuristic bound roughly matches the bounds we prove when we apply our deterministic algorithm for collision-finding to random hash functions applied to worst-case inputs for element distinctness, though the algorithm we present is different in detail and developed independently. It is an open question of considerable interest whether there exist deterministic solutions for worst case inputs with similar bounds.

Motivated by our application of efficiently solving element distinctness, we examine the time and space complexity of finding *all* colliding vertices, along with their predecessors, in the subgraph reachable from a possibly large set of k starting vertices, not just from a single start vertex. We will show how to do this using storage equivalent to only $O(k)$ elements of V and time roughly proportional to the size of the subgraph reachable from this set of starting vertices. Note that the obvious approach of running k independent copies of Floyd's algorithm in parallel from each of the start vertices does not solve this problem since it may miss collisions between different parallel branches (see Figure 4.1(b)), and it may also traverse large regions of the subgraph many times.

For $v \in V$, define $f^*(v) = \{f^i(v) \mid i \geq 0\}$ to be the set of vertices reachable from v and $f^*(U) = \bigcup_{v \in U} f^*(v)$ for $U \subseteq V$.

Theorem 4.1. *There is an $O(k \log n)$ space deterministic algorithm* COLLIDE_k *that, given $f : V \to V$ for a finite set V and $K = \{v_1, \ldots, v_k\} \subseteq V$, finds all pairs $(v, \{u \in f^*(K) | f(u) = v\})$ and runs in time $O(|f^*(K)| \log k \min\{k, \log n\})$.*

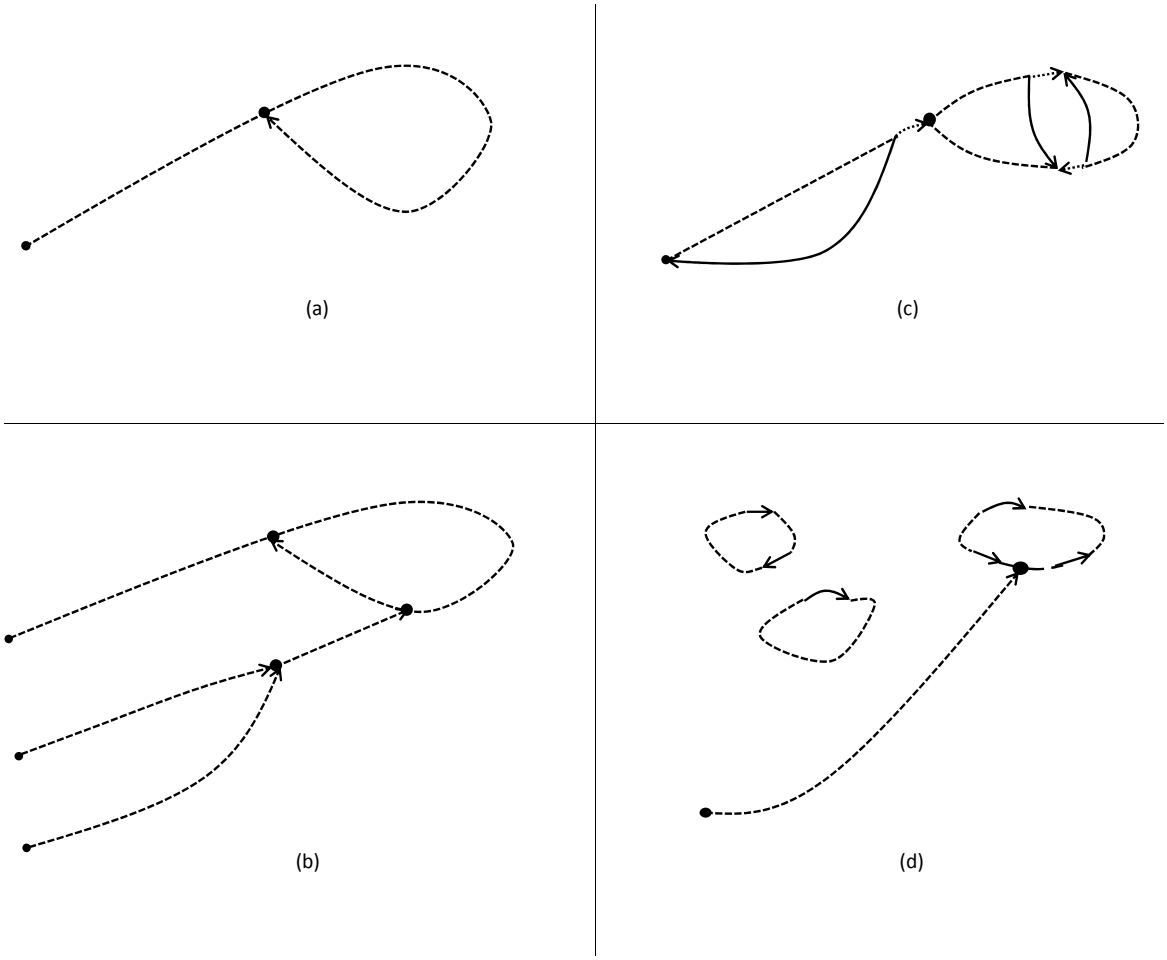

Figure 4.1: Collision-finding with multiple sources

Proof. We first describe the algorithm COLLIDE$_k$ (Algorithm 4.1): In addition to the original graph and the collisions that it finds, this algorithm will maintain a *redirection list* $R \subset V$ of size $O(k)$ vertices that it will provisionally redirect to map to a different location. For each vertex in R it will store the name of the new vertex to which it is directed. We maintain a separate list L of all vertices from which an edge of G has been redirected away and the original vertices that point to them.

The redirections for a single iteration of the algorithm are shown in Figure 4.1(c). The general situation for later iterations in the algorithm is shown in Figure 4.1(d).

Observe that in each iteration of the loop there is at most one vertex v where collisions can

Algorithm 4.1 COLLIDE$_k$

1: $R = \emptyset$.
2: **for** $j = 1, \ldots, k$ **do**
3: Execute Floyd's algorithm starting with vertex v_j on the graph G using the redirected out-edges for nodes from the redirection list R instead of f.
4: **if** the cycle found does not include v_j **then**
5: There must be a collision.
6: **if** this collision is in the graph G **then**
7: Report the collision v as well as the colliding vertices u and u', where u' is the predecessor of v on the cycle and u is the predecessor of v on the path from v_j to v.
8: **end if**
9: Add u to the redirection list R and redirect it to vertex v_j.
10: **end if**
11: Traverse the cycle again to find its length and choose two vertices w and w' on the cycle that are within 1 of half this length apart.
12: Add w and w' to the redirection list, redirecting w to $f(w')$ and w' to $f(w)$. This will split the cycle into two parts, each of roughly half its original length.
13: **end for**

occur and at most 3 vertices are added to the redirection list. Moreover, after each iteration, the set of vertices reachable from vertices $v_1, \ldots, v_j$ appear in a collection of disjoint cycles of the redirected graph. Each iteration of the loop traverses at most one cycle and every cycle is roughly halved each time it is traversed.

In order to store the redirection list R, we use a dynamic dictionary data structure of $O(k \log n)$ bits that supports insert and search in $O(\log k)$ time per access or insertion. We can achieve this using balanced binary search trees and we can improve the bound to $O(\sqrt{\log k / \log\log k})$ using exponential trees [18]. Before following an edge (i.e,, evaluating f), the algorithm will first check list R to see if it has been redirected. Hence each edge traversed costs $O(\log k)$ time (or $O(\sqrt{\log k / \log\log k})$ using exponential trees). Since time is measured relative to the size of the reachable set of vertices, the only other extra cost is that of re-traversing previously discovered edges. Since all vertices are maintained in cycles and each traversal of a cycle roughly halves its length, each edge found can be traversed at most $O(\min\{k, \log n\})$ times. □

As we have noted in the proof, with exponential trees the running time of the algorithm can be reduced to $O(|f^*(K)|\sqrt{\log k / \log\log k}\min\{k, \log n\})$, though the algorithm becomes significantly

more complicated in this case.

4.5.2 *A randomized $T^2 \cdot S \in \tilde{O}(n^3)$ element distinctness algorithm*

We will use collision-finding for our element distinctness algorithm. In this case the vertex set V will be the set of indices $[n]$, and the function f will be given by $f_{x,h}(i) = h(x_i)$ where h is a (random) hash function that maps $[m]$ to $[n]$.

Observe that if we find $i \neq j$ such that $f_{x,h}(i) = f_{x,h}(j)$ then either

- $x_i = x_j$ and hence $\mathrm{ED}(x) = 0$, or
- we have found a collision in h: $x_i \neq x_j$ but $h(x_i) = h(x_j)$;

We call x_i and x_j "pseudo-duplicates" in this latter case.

Given a parameter k, on input x our randomized algorithm will repeatedly choose a random hash function h and a random set K of roughly k starting points and then call the $\textsc{Collide}_k$ algorithm given in Theorem 4.1 on K using the function $f = f_{x,h}$ and check the collisions found to determine whether or not there is a duplicate among the elements of x indexed by $f^*_{x,h}(K)$. The space bound S of this algorithm will be $O(k \log n)$.

The running time of $\textsc{Collide}_k$ depends on $|f^*_{x,h}(K)|$, which in turn is governed by the random choices of h and K and may be large. Since $f^*_{x,h}(K)$ is also random, we also need to argue that if $ED(x) = 0$, then there is a reasonable probability that a duplicate in x will be found among the indices in $f^*_{x,h}(K)$. The following two lemmas analyze these issues.

Lemma 4.2. *Let $x \in [m]^n$. For $h : [m] \to [n]$ chosen uniformly at random and for $K \subseteq [n]$ selected by uniformly and independently choosing $2 \leq k \leq n/32$ elements of $[n]$ with replacement,* $\Pr[|f^*_{x,h}(K)| \leq 2\sqrt{kn}] \geq 8/9$.

Proof. We can run an experiment that is equivalent to selecting $f^*_{x,h}(K)$ as described in Algorithm 4.2. Observe that, when each new element i is chosen at random, the probability that a previous index is found (the while loop exits) is precisely $|I|/n$ and that $|I|$ increases by 1 per step,

Algorithm 4.2

Set $M = I = K = \emptyset$.
for $count = 1$ to k **do**
  Choose an element $i \in [n]$ uniformly at random and add it to K.
  while $i \notin I$ **do**
    Add i to I
    if $x_i \notin M$ **then**
      Add x_i to M
      Choose an element $i \in [n]$ uniformly at random.
    else
      Output ("duplicate found")
      break
    end if
  end while
end for
Output I.

except for the k steps when i was already in I or x_i was in M. View each of these random choices of i as a coin-flip with probability of heads being $|I|/n$.

The index set size $|I|$ at the j-th random choice is at least $j - k$ as we will have seen a duplicate in either I or M at most k times. Hence the probability of exiting the while loop is at least $(j-k)/n$. Therefore the expected number of while loop exits that have occurred when the t-th random choice is made is at least $\sum_{j=1}^{t-k} j/n$. Consider for a moment our experiment but with the bound on the for loop removed. Then, solving for t we get that the minimum number of random choices at which the expected number of while loop exits is at least $k/(1-\delta)$ is itself at least $\lceil(\sqrt{1+8kn/(1-\delta)}-1)/2+k\rceil \leq \sqrt{2kn/(1-\delta)}+k+1$. Let $\delta = 3/4$ and observe that $n > 32k$ implies that $\sqrt{8kn/3}+k+1 \leq 2\sqrt{kn}$. Our experiment terminates when count $> k$ and so we can now bound the number of random choices and hence $|f^*_{k,n}(K)|$. By treating the event of exiting of the while loop as independent coin flips with the lower bound probability of $(j-k)/n$ at random choice j and applying the Chernoff inequality, as explained in Chapter 2, we get that $\Pr[|f^*_{k,n}(K)| > 2\sqrt{kn}] \leq e^{-9k/8}$ as required to prove the lemma. □

Lemma 4.3. *Let $x \in [m]^n$ be such that $ED(x) = 0$. Then for $h : [m] \to [n]$ chosen uniformly at random and for $K \subseteq [n]$ selected by uniformly and independently choosing $2 \leq k \leq n/32$ elements of $[n]$ with replacement, $\Pr[|f^*_{x,h}(K)| \leq 2\sqrt{kn}$ and $\exists i \neq j \in f^*_{x,h}(K)$ such that $x_i = x_j] \geq$*

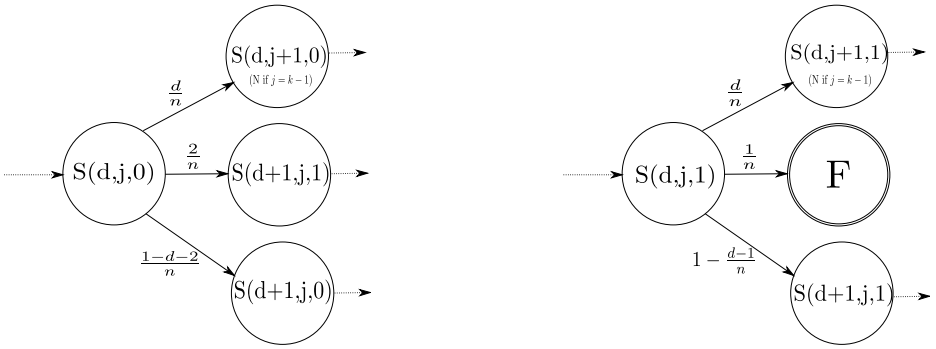

Figure 4.2: Transitions from the Markov chain in the proof of Lemma 4.3

$k/(18n)$.

Proof. We show this first for an x that has a single pair of duplicated values and then observe how this extends to the general case. Consider the sequence of indices selected in the experiment in the proof of Lemma 4.2 on input x. We define an associated Markov chain with a state $S(d, j, b)$ for $0 \leq d < n$, for $0 \leq j \leq k-1$, and for $b \in \{0, 1\}$, where state $S(d, j, b)$ indicates that there have been d distinct indices selected so far, j pseudo-duplicates, and b is the number of indices from the duplicated pair selected so far. In addition, the chain has two absorbing states, F, indicating that a duplicate has been found, and N, indicating that no duplicates have been found at termination.

Observe that the probability in state $S(d, j, b)$ that the next selection is a pseudo-duplicate is precisely d/n; hence it is also the probability of a transition from $S(d, j, b)$ to $S(d, j+1, b)$ if $j < k-1$, or to N if $j = k-1$. From state $S(d, j, 0)$ there is a $2/n$ chance of selecting one of the duplicated pair, so this is the probability of the transition from $S(d, j, 0)$ to $S(d+1, j, 1)$. Finally, from state $S(d, j, 1)$, the probability that the other element of the duplicated pair is found is precisely $1/n$ and so this is the probability of a transition to F. The remaining transition from state from $S(d, j, b)$ leads to state $S(d+1, j, b)$ with probability $1 - (d + 2 - b)/n$. See Figure 4.2.

By the result of Lemma 4.2, with probability at least $8/9$, when started in state $S(0, 0, 0)$ this chain reaches state F or N in at most $t^* = 2\sqrt{kn}$ steps. The quantity we wish to lower bound is the probability that the chain reaches state F in its first t^* steps. We derive our bound by comparing the probability that F is reached in t^* steps to the probability that either F or N is reached in t^* steps.

Consider all *special* transitions, those that increase j or b, or result in states F or N. In any walk on the Markov chain that reaches F or N, at most $k+1$ special transitions can be taken, and to reach N, at least k special transitions must be taken.

Fix any sequence $0 \leq d_1 \leq \ldots \leq d_{k+1} < t^*$. We say that a walk on the Markov chain is *consistent* with this sequence iff the sequence of starting states of its special transitions is a prefix of some sequence of states $S(d_1, *, *), \ldots, S(d_{k+1}, *, *)$. We condition on the walk being one of these consistent walks.

Assume that $k \geq 2$. In order to reach state F, there must be one special transition that changes b from 0 to 1 and another that leads to F. Consider the choices $(a, a') \in \binom{[k]}{2}$ of where these can occur among the first k special transitions. The conditional probability that a consistent special transition goes from $S(d_a, j_a, 0)$ to $S(d_a + 1, j_a, 1)$ is precisely $2/(d_a + 2) \geq 1/t^*$ and the conditional probability that it goes from $S(d_{a'}, j_{a'}, 1)$ to F is precisely $1/(d_{a'} + 1) \geq 1/t^*$. In particular, the conditional probability that two special transitions of these types occur is lower bounded by the probability that k Bernoulli trials with success probability $1/t^*$ yield at least two successes. This lower bound is at least

$$\begin{aligned}\binom{k}{2} 1/(t^*)^2 (1 - 1/t^*)^{k-2} &\geq [1 - (k-2)/t^*] \cdot \binom{k}{2}/(t^*)^2 \\ &\geq [1 - (k-2)/(2\sqrt{kn})]k(k-1)/(8kn) \geq k/(16n).\end{aligned}$$

Since the different sequences correspond to disjoint sets of inputs, conditioned on the event that F or N is reached in at most t^* steps, which occurs with probability at least $8/9$, the conditional probability that F is reached is $\geq k/(16n)$. Therefore the total probability that F is reached in at most t^* steps is at least $k/(18n)$, as required[1].

In the general case, we observe that at each step prior to termination, the probability of finding a pseudo-duplicate, given that j of them have previously been found, does not depend on x. On the other hand, additional duplicated inputs in x only increase the chance of selecting a first index that is duplicated, and can only increase the chance that one of its matching indices will be selected

[1]We have not tried to optimize these constant factors. Also, one can similarly do a separate sharper analysis when $k = 1$.

in subsequent choices. Therefore, finding a duplicate is at least as likely for x as it is in the above analysis. □

These lemmas, together with the properties of COLLIDE$_k$ yield the following theorem.

Theorem 4.4. *For any $\epsilon > 0$, and any S with $c \log n \leq S \leq n/32$ for some constant $c > 0$, there is a randomized RAM algorithm with input randomness computing ED_n with 1-sided error (false positives) at most ϵ, that uses space S and time $T \in O(\frac{n^{3/2}}{S^{1/2}} \log^{5/2} n \log(1/\epsilon))$. Further, when $S \in O(\log n)$, we have $T \in O(n^{3/2} \log(1/\epsilon))$.*

Proof. Choose $k \geq 2$ such that the space usage of COLLIDE$_k$ on $[n]$ is at most $S/2$. Therefore $k \in \Omega(S/\log n)$. The algorithm is as follows:

> On input x, run $(18n/k)\log(1/\epsilon)$ independent runs of COLLIDE$_k$ on different $f_{x,h}$, each with independent random choices of hash functions h and independent choices, K, of k starting indices, and each with a run-time cut-off bounding the number of explored vertices of $f^*_{x,h}(K)$ at $t^* = 2\sqrt{kn}$.
>
> On each run, check if any of the collisions found is a duplicate in x, in which case output $ED(x) = 0$ and halt. If none are found in any round then output $ED(x) = 1$.

The algorithm will never incorrectly report a duplicate in a distinct x and by Lemma 4.3, each run has a probability of at least $k/(18n)$ of finding a duplicate in an input x such that $ED(x) = 0$ so the probability of failing to find a duplicate in $(18n/k)\log(1/\epsilon)$ rounds is at most ϵ.

Using Theorem 4.1 each run of our bounded version of COLLIDE$_k$, requires run-time $O(\sqrt{kn} \log k \min\{k, \log n\})$ and hence the total runtime of the algorithm is $O(\sqrt{kn} \cdot n/k \cdot \log k \min\{k, \log n\} \log(1/\epsilon))$ which is $O(n^{3/2} \log(1/\epsilon))$ for k constant and $O(n^{3/2}/k^{1/2} \cdot \log^2 n \cdot \log(1/\epsilon))$ for general k. The claim follows using $k \in \Omega(S/\log n)$. □

4.6 Element Distinctness over Sliding Windows

The main result of this section shows that our randomized branching program for ED_n can even be extended to a $T \in \tilde{O}(n^{3/2}/S^{1/2})$ randomized branching program for its sliding windows version

$ED_n^{\boxplus n}$. We do this in two steps. We first give a deterministic reduction which shows how the answer to an element distinctness problem allows one to reduce the input size of sliding-window algorithms for computing $ED_n^{\boxplus m}$.

Lemma 4.5. *Let $n > m > 0$.*

(a) If $ED_{n-m+1}(x_m, \ldots, x_n) = 0$ then $ED_n^{\boxplus m}(x_1, \ldots, x_{n+m-1}) = 0^m$.

(b) If $ED_{n-m+1}(x_m, \ldots, x_n) = 1$ then define

i. $i_L = \max\{j \in [m-1] \mid ED_{n-j+1}(x_j, \ldots, x_n) = 0\}$ where $i_L = 0$ if the set is empty and

ii. $i_R = \min\{j \in [m-1] \mid ED_{n-m+j}(x_m, \ldots, x_{n+j}) = 0\}$ where $i_R = m$ if the set is empty.

Then

$$ED_n^{\boxplus m}(x_1, \ldots, x_{n+m-1}) = 0^{i_L}1^{m-i_L} \wedge 1^{i_R}0^{m-i_R} \wedge ED_{m-1}^{\boxplus m}(x_1, \ldots, x_{m-1}, x_{n+1}, \ldots, x_{n+m-1})$$

where each $\wedge$ represents bit-wise conjunction.

Proof. The elements $M = (x_m, \ldots, x_n)$ appear in all m of the windows so if this sequence contains duplicated elements, so do all of the windows and hence the output for all windows is 0. This implies part (a).

If M does not contain any duplicates then any duplicate in a window must involve at least one element from $L = (x_1, \ldots, x_{m-1})$ or from $R = (x_{n+1}, \ldots, x_{n+m-1})$. If a window has value 0 because it contains an element of L that also appears in M, it must also contain the rightmost such element of L and hence any window that is distinct must begin to the right of this rightmost such element of L. Similarly, if a window has value 0 because it contains an element of R that also appears in M, it must also contain the leftmost such element of L and hence any window that is distinct must end to the left of this leftmost such element of R. The only remaining duplicates that can occur in a window can only involve elements of both L and R. In order, the m windows contain the following sequences of elements of $L \cup R$: $(x_1, \ldots, x_{m-1})$, $(x_2, \ldots, x_{m-1}, x_{n+1})$, …, $(x_{m-1}, x_{n+1}, \ldots, x_{n+m-2})$, $(x_{n+1}, \ldots, x_{n+m-1})$. These are precisely the sequences for which $ED_{m-1}^{\boxplus m}(x_1, \ldots, x_{m-1}, x_{n+1}, \ldots, x_{n+m-1})$ determines distinctness. Hence part (b) follows. □

We use the above reduction in input size to show that any efficient algorithm for element distinctness can be extended to solve element distinctness over sliding windows at a small additional cost.

Lemma 4.6. *If there is an algorithm A that solve element distinctness, ED, using time at most $T(n)$ and space at most $S(n)$, where T and S are nondecreasing functions of n, then there is an algorithm A^* that solves the sliding-window version of element distinctness, $ED_n^{\boxplus n}$, in time $T^*(n)$ that is $O(T(n)\log^2 n)$ and space $S^*(n)$ that is $O(S(n)+\log^2 n)$. Moreover, if $T(n)$ is $\Omega(n^\beta)$for $\beta > 1$, then $T^*(n)$ is $O(T(n)\log n)$.*

If A is deterministic then so is A^*. If A is randomized with error at most ϵ then A^* is randomized with error $o(1/n)$. Moreover, if A has the obvious 1-sided error (it only reports that inputs are not distinct if it is certain of the fact) then the same property holds for A^*.

Proof. We first assume that A is deterministic. Algorithm A^* will compute the n outputs of $ED_n^{\boxplus n}$ in n/m groups of m using the input size reduction method from Lemma 4.5. In particular, for each group A^* will first call A on the middle section of input size $n-m+1$ and output 0^m if A returns 0. Otherwise, A^* will do two binary searches involving at most $2\log m$ calls to A on inputs of size at most n to compute i_L and i_R as defined in part (b) of that lemma. Finally, in each group, A^* will make one recursive call to A^* on a problem of size m.

It is easy to see that this yields a recurrence of the form

$$T^*(n) = (n/m)[cT(n)\log m + T^*(m)].$$

In particular, if we choose $m = n/2$ then we obtain $T^*(n) \le 2T^*(n/2) + 2cT(n)\log n$. If $T(n)$ is $\Omega(n^\beta)$ for $\beta > 1$ this solves to $T^*(n) \in O(T(n)\log n)$. Otherwise, it is immediate from the definition of $T(n)$ that $T(n)$ must be $\Omega(n)$ and hence the recursion for A^* has $O(\log n)$ levels and the total cost associated with each of the levels of the recursion is $O(T(n)\log n)$.

Observe that the space for all the calls to A can be re-used in the recursion. Also note that the algorithm A^* only needs to remember a constant number of pointers for each level of recursion for a total cost of $O(\log^2 n)$ additional bits.

We now suppose that the algorithm A is randomized with error at most ϵ. For the recursion based

on Lemma 4.5, we use algorithm A and run it $C = O(\log n)$ times on input $(x_m, \ldots, x_n)$, taking the majority of the answers to reduce the error to $o(1/n^2)$. In case that no duplicate is found in these calls, we then apply the noisy binary search method of Feige, Peleg, Raghavan, and Upfal [66] to determine i_L and i_R with error at most $o(1/n^2)$ by using only $C = O(\log n)$ calls to A. (If the original problem size is n we will use the same fixed number $C = O(\log n)$ of calls to A even at deeper levels of the recursion so that each subproblem has error $o(1/n^2)$.) There are only $O(n)$ subproblems so the final error is $o(1/n)$. The rest of the run-time analysis is the same as in the deterministic case.

If A has only has false positives (if it claims that the input is not distinct then it is certain that there is a duplicate) then observe that A^* will only have false positives. □

Combining Theorem 4.4 with Lemma 4.6 we obtain our algorithm for element distinctness over sliding windows.

Theorem 4.7. *For space $S \in [c \log n, n]$, $ED^{\boxplus n}$ can be solved in time $T \in O(n^{3/2} \log^{7/2} n / S^{1/2})$ time with 1-sided error probability $o(1/n)$. If the space $S \in O(\log n)$ then the time is reduced to $T \in O\big(n^{3/2} \log n\big)$.*

4.6.1 A fast average case algorithm for $ED^{\boxplus n}$ with alphabet $[n]$

When the input alphabet is chosen uniformly at random from $[n]$ there exists a much simpler 0-error sliding-window algorithm for $ED^{\boxplus n}$ that is efficient on average. The algorithm runs in $O(n)$ time on average using $O(\log n)$ bits of space. By way of contrast, under the same distribution, we prove an average case time-space lower bound of $\overline{T} \in \Omega(n^2/\overline{S})$ for $(F_0 \bmod 2)^{\boxplus n}$ in Section 4.7.

Theorem 4.8. *For input randomly chosen uniformly from $[n]^{2n-1}$, $ED^{\boxplus n}$ can be solved in average time $\overline{T} \in O(n)$ and average space $\overline{S} \in O(\log n)$.*

The simple method we employ is as follows. We start at the first window of length n of the input and perform a search for the first duplicate pair starting at the right-hand end of the window and going to the left. We check if a symbol at position j is involved in a duplicate by simply scanning all the symbols to the right of position j within the window. If the algorithm finds a duplicate in a

suffix of length x, it shifts the window to the right by $n - x + 1$ and repeats the procedure from this point. If it does not find a duplicate at all in the whole window, it simply moves the window on by one and starts again.

In order to establish the running time of this simple method, we will make use of the following birthday-problem-related facts.

Lemma 4.9. *Assume that we sample i.u.d. with replacement from $[n]$ with $n \geq 4$. Let X be a discrete random variable that represents the number of samples taken when the first duplicate is found. Then*

$$\Pr\left(X \geq n/2\right) \leq e^{-\frac{n}{16}}. \tag{4.1}$$

and

$$\mathbb{E}(X^2) \leq 4n. \tag{4.2}$$

Proof. For every $x \geq 1$ we have

$$\Pr(X \geq x) = \prod_{i=1}^{x-1}\left(1 - \frac{i}{n}\right) \leq \prod_{i=1}^{x-1} e^{-\frac{i}{n}} \leq e^{-\frac{x^2}{4n}}.$$

Inequality (4.1) now follows by substituting $x = n/2$ giving

$$\Pr\left(X \geq \frac{n}{2}\right) \leq e^{-\frac{n}{16}}.$$

To prove inequality (4.2), recall that for non-negative valued discrete random variables

$$\mathbb{E}(X) = \sum_{x=1}^{\infty} \Pr(X \geq x).$$

Observe that

$$\mathbb{E}(X^2) = \sum_{x=1}^{\infty} \Pr(X^2 \geq x) = \sum_{x=1}^{\infty} \Pr(X \geq \sqrt{x})$$

$$\leq \sum_{x=1}^{\infty} e^{-\frac{(\sqrt{x})^2}{4n}} \leq \int_{x=0}^{\infty} e^{-\frac{(\sqrt{x})^2}{4n}} = 4n.$$

□

We can now show the running time of our average case algorithm for $ED^{\boxplus n}$.

Proof of Theorem 4.8. Let U be a sequence of values sampled uniformly from $[n]$ with $n \geq 4$. Let M be the index of the first duplicate in U found when scanning from the right and let $X = n - M$. Let $W(X)$ be the number of comparisons required to find X. Using our naive duplicate finding method we have that $W(X) \leq X(X+1)/2$. It also follows from inequality (4.2) that $\mathbb{E}(W) \leq 4n$.

Let $R(n)$ be the total running time of our algorithm and note that $R(n) \leq n^3/2$. Furthermore the residual running time at any intermediate stage of the algorithm is at most $R(n)$.

Let us consider the first window and let M_1 be the index of the first duplicate from the right and let $X_1 = n - M_1$. If $X_1 \geq n/2$, denote the residual running time by $R^{(1)}$. We know from (4.1) that $\Pr(X_1 \geq n/2) \leq e^{-\frac{n}{16}}$. If $X_1 < n/2$, shift the window to the right by $M_1 + 1$ and find X_2 for this new window. If $X_2 \geq n/2$, denote the residual running time by $R^{(2)}$. We know that $\Pr(X_2 \geq n/2) \leq e^{-\frac{n}{16}}$. If $X_1 < n/2$ and $X_2 < n/2$ then the algorithm will terminate, outputting 'not all distinct' for every window.

The expected running time is then

$$\begin{aligned}\mathbb{E}(R(n)) &= E\left(W(X_1)\right) + E\left(R^{(1)}\right)\Pr\left(X_1 \geq \frac{n}{2}\right) \\ &\quad + \Pr\left(X_1 < \frac{n}{2}\right)\left[E\left(W(X_2)\middle| X_1 < \frac{n}{2}\right) + E\left(R^{(2)}\right)\Pr\left(X_2 \geq \frac{n}{2}\middle| X_1 < \frac{n}{2}\right)\right] \\ &\leq 4n + \frac{n^3}{2}e^{-\frac{n}{16}} + 4n + \frac{n^3}{2}e^{-\frac{n}{16}} \in O(n)\end{aligned}$$

The inequality follows from the followings three observations. We know trivially that $\Pr(X_1 < n/2) \leq 1$. Second, the number of comparisons $W(X_2)$ does not increase if some of the elements in a window are known to be unique. Third, $\Pr(X_2 \geq n/2 \wedge X_1 < n/2) \leq \Pr(X_2 \geq n/2) \leq e^{-\frac{n}{16}}$. □

We note that similar results can be shown for inputs uniformly chosen from the alphabet $[cn]$ for any constant c.

4.7 Frequency Moments over Sliding Windows

We now show a $T \in \Omega(n^2/S)$ lower bound for randomized branching programs computing frequency moments over sliding windows. This contrasts with our significantly smaller $T \in \tilde{O}(n^{3/2}/S^{1/2})$ upper bound from the previous section for computing element distinctness over sliding windows in this same model, hence separating the complexity of ED and F_k for $k \neq 1$ over sliding windows. Our lower bound also applies to $F_0 \bmod 2$.

4.7.1 A general sequential lower bound for $F_k^{\boxplus n}$ and $(F_0 \bmod 2)^{\boxplus n}$

We derive a time-space tradeoff lower bound for randomized branching programs computing $F_k^{\boxplus n}$ for $k = 0$ and $k \geq 2$. Further, we show that the lower bound also holds for computing $(F_0 \bmod 2)^{\boxplus n}$. (Note that the parity of F_k for $k \geq 1$ is exactly equal to the parity of n; thus the outputs of $(F_k \bmod 2)^{\boxplus n}$ are all equal to $n \bmod 2$.)

Theorem 4.10. *Let $k = 0$ or $k \geq 2$. There is a constant $\delta > 0$ such that any $[n]$-way branching program of time T and space S that computes $F_k^{\boxplus n}$ with error at most η, $0 < \eta < 1 - 2^{-\delta S}$, for input randomly chosen uniformly from $[n]^{2n-1}$ must have $T \cdot S \in \Omega(n^2)$. The same lower bound holds for $(F_0 \bmod 2)^{\boxplus n}$.*

Corollary 4.11. *Let $k = 0$ or $k \geq 2$.*

- *The average time $\overline{T}$ and average space $\overline{S}$ needed to compute $(F_k)^{\boxplus n}(x)$ for x randomly chosen uniformly from $[n]^{2n-1}$ satisfy $\overline{T} \cdot \overline{S} \in \Omega(n^2)$.*

- *For $0 < \eta < 1 - 2^{-\delta S}$, any η-error randomized RAM or word-RAM algorithm computing $(F_k)^{\boxplus n}$ using time T and space S satisfies $T \cdot S \in \Omega(n^2)$.*

Proof of Theorem 4.10. We derive the lower bound for $F_0^{\boxplus n}$ first. Afterwards we show the modifications needed for $k \geq 2$ and for computing $(F_0 \bmod 2)^{\boxplus n}$. For convenience, on input $x \in [n]^{2n-1}$, we write y_i for the output $F_k(x_i, \ldots, x_{i+n-1})$.

We use the general approach to derive lower bounds on multi-output branching programs as outlined in Theorem 2.6. Our choice of the probability distribution over the input and the subset of the outputs we consider is dictated by the following two issues.

The first issue is that if a path in a small program B' finds that certain values are equal, then the answers to nearby windows may be strongly correlated with each other; for example, if $x_i = x_{i+n}$ then $y_i = y_{i+1}$. Such correlations risk making the likelihood too high that the correct outputs are produced on a path. Therefore, instead of considering the total number of outputs produced, we reason about the number of outputs from positions that are not duplicated in the input and argue that with high probability there will be a linear number of such positions.

The second issue is that inputs for which the value of F_0 in a window happens to be extreme, say n — all distinct — or 1 — all identical — allow an almost-certain prediction of the value of F_0 for the next window. We will use the fact that under the uniform distribution, cases like these almost surely do not happen; indeed the numbers of distinct elements in every window almost surely fall in a range close to their mean and in this case the value in the next window will be predictable with probability bounded below 1 given the value in the previous ones. In this case we use the chain rule to compute the overall probability of correctness of the outputs.

We start by analyzing the likelihood that an output of F_0 is extreme.

Lemma 4.12. *Let a be chosen uniformly at random from $[n]^n$. Then the probability that $F_0(a)$ is between $0.5n$ and $0.85n$ is at least $1 - 2e^{-n/50}$.*

Proof. For $a = a_1 \dots a_n$ uniformly chosen from $[n]^n$,

$$\mathbb{E}[F_0(a)] = \sum_{\ell \in [n]} \Pr_a[\exists i \in [n] \text{ such that } a_i = \ell] = n[1 - (1 - 1/n)^n].$$

Hence $0.632n < (1 - 1/e)n < \mathbb{E}[F_0(a)] \leq 0.75n$. We use Doob martingales and the Azuma-Hoeffding inequality as explained in Chapter 2.4 to measure the concentration of the values of F_0. Define a Doob martingale D_t, $t = 0, 1, \dots, n$ with respect to the sequence $a_1 \dots a_n$ by $D_t = \mathbb{E}[F_0(a) \mid a_1 \dots a_t]$. Therefore $D_0 = \mathbb{E}[F_0(a)]$ and $D_n = F_0(a)$. Applying the Azuma-Hoeffding inequality, we have

$$\Pr_a[F_0(a) \notin [0.5n, 0.85n]] \leq \Pr_a[|F_0(a) - \mathbb{E}[F_0(a)| \geq 0.1n] \leq 2e^{-2\frac{(0.1n)^2}{n}} = 2e^{-n/50},$$

which proves the claim. □

We say that x_j is *unique in* x iff $x_j \notin \{x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_{2n-1}\}$.

Lemma 4.13. *Let x be chosen uniformly at random from $[n]^{2n-1}$ with $n \geq 2$. With probability at least $1 - 4ne^{-n/50}$,*

(a) all outputs of $F_0^{\boxplus n}(x)$ are between $0.5n$ and $0.85n$, and

(b) the number of positions $j < n$ such that x_j is unique in x is at least $n/24$.

Proof. We know from Lemma 4.12 and the union bound that part (a) is false with probability at most $2ne^{-n/50}$. For any $j < n$, let U_j be the indicator variable of the event that j is unique in x and $U = \sum_{j<n} U_j$. Now $\mathbb{E}(U_j) = (1 - 1/n)^{2n-2}$ so $\mathbb{E}(U) = (n-1)(1-1/n)^{2n-2} \geq n/8$ for $n \geq 2$. Observe also that this is a kind of typical "balls in bins" problem and so, as discussed in [63], it has the property that the random variables U_j are *negatively associated*; for example, for disjoint $A, A' \subset [n-1]$, the larger $\sum_{j \in A} U_j$ is, the smaller $\sum_{j \in A'} U_j$ is likely to be. Hence, it follows [63] that U is more closely concentrated around its mean than if the U_j were fully independent. It also therefore follows that we can apply a Chernoff bound, as explained in Chapter 2, directly to our problem, giving $\Pr[U \leq n/24] \leq \Pr[U \leq \mathbb{E}(U)/3] \leq e^{-2\mathbb{E}(U)/9} \leq e^{-n/36}$. We obtain the desired bound for parts (a) and (b) together by another application of the union bound. □

Correctness of a small branching program for computing outputs in π-unique positions

DEFINITION 4.5. *Let B' be an $[n]$-way branching program and let π be a source-sink path in B' with queries Q_π and answers $A_\pi : Q_\pi \to [n]$. An index $\ell < n$ is said to be* π-unique *iff either (a) $\ell \notin Q_\pi$, or (b) $A_\pi(\ell) \notin A_\pi(Q_\pi - \{\ell\})$.*

In order to measure the correctness of a small branching program, we restrict our attention to outputs that are produced at positions that are π-unique and upper-bound the probability that a small branching program correctly computes outputs of $F_0^{\boxplus n}$ at many π-unique positions in the input.

Let $\mathcal{E}$ be the event that all outputs of $F_0^{\boxplus n}(x)$ are between $0.5n$ and $0.85n$.

Lemma 4.14. *Let $r > 0$ be a positive integer, let $\epsilon \leq 1/10$, and let B' be an $[n]$-way branching program of height $q = \epsilon n$. Let π be a path in B' on which outputs from at least r π-unique positions*

are produced. For random x uniformly chosen from $[n]^{2n-1}$,

$$\Pr[\textit{these } r \textit{ outputs are correct for } F_0^{\boxplus n}(x), \mathcal{E} \mid \pi_{B'}(x) = \pi] \leq (17/18)^r.$$

Proof. Roughly, we will show that when $\mathcal{E}$ holds (outputs for all windows are not extreme) then, conditioned on following any path π in B', each output produced for a π-unique position will have only a constant probability of success conditioned on any outcome for the previous outputs. Because of the way outputs are indexed, it will be convenient to consider these outputs in right-to-left order.

Let π be a path in B', Q_π be the set of queries along π, $A_\pi : Q_\pi \to [n]$ be the answers along π, and $Z_\pi : [n] \to [n]$ be the partial function denoting the outputs produced along π. Note that $\pi_{B'}(x) = \pi$ if and only if $x_i = A_\pi(i)$ for all $i \in Q_\pi$.

Let $1 \leq i_1 < i_2 < \ldots < i_r < n$ be the first r of the π-unique positions on which π produces output values; i.e., $\{i_1, \ldots, i_r\} \subseteq \mathrm{dom}(Z_\pi)$. Define $z_{i_1} = Z_\pi(i_1), \ldots, z_{i_r} = Z_\pi(i_r)$.

We will decompose the probability over the input x that $\mathcal{E}$ and all of $y_{i_1} = z_{i_1}, \ldots, y_{i_r} = z_{i_r}$ hold via the chain rule. In order to do so, for $\ell \in [r]$, we define event $\mathcal{E}_\ell$ to be $0.5n \leq F_0^{(i)}(x) \leq 0.85n$ for all $i > i_\ell$. We also write $\mathcal{E}_0 \stackrel{\text{def}}{=} \mathcal{E}$. Then

$$\begin{aligned}
&\Pr[y_{i_1} = z_{i_1}, \ldots, y_{i_r} = z_{i_r},\ \mathcal{E} \mid \pi_{B'}(x) = \pi] \\
&= \Pr[\mathcal{E}_r \mid \pi_{B'}(x) = \pi] \\
&\qquad \times \prod_{\ell=1}^{r} \Pr[y_{i_\ell} = z_{i_\ell},\ \mathcal{E}_{\ell-1} \mid y_{i_{\ell+1}} = z_{i_{\ell+1}},\ \ldots,\ y_{i_r} = z_{i_r},\ \mathcal{E}_\ell,\ \pi_{B'}(x) = \pi] \\
&\leq \prod_{\ell=1}^{r} \Pr[y_{i_\ell} = z_{i_\ell} \mid y_{i_{\ell+1}} = z_{i_{\ell+1}},\ \ldots,\ y_{i_r} = z_{i_r},\ \mathcal{E}_\ell,\ \pi_{B'}(x) = \pi]. \qquad (4.3)
\end{aligned}$$

We now upper bound each term in the product in (4.3). Depending on how much larger $i_{\ell+1}$ is than i_ℓ, the conditioning on the value of $y_{i_{\ell+1}}$ may imply a lot of information about the value of y_{i_ℓ}, but we will show that even if we reveal more about the input, the value of y_{i_ℓ} will still have a constant amount of uncertainty.

For $i \in [n]$, let W_i denote the vector of input elements $(x_i, \ldots, x_{i+n-1})$, and note that $y_i = F_0(W_i)$; we call W_i the i^{th} window of x. The values y_i for different windows may be closely

related. In particular, adjacent windows W_i and W_{i+1} have numbers of distinct elements that can differ by at most 1 and this depends on whether the extreme end-points of the two windows, x_i and x_{i+n}, appear among their common elements $C_i = \{x_{i+1}, \ldots, x_{i+n-1}\}$. More precisely,

$$y_i - y_{i+1} = \mathbf{1}_{\{x_i \notin C_i\}} - \mathbf{1}_{\{x_{i+n} \notin C_i\}}. \tag{4.4}$$

In light of (4.4), the basic idea of our argument is that, because i_ℓ is π-unique and because of the conditioning on $\mathcal{E}_\ell$, there will be enough uncertainty about whether or not $x_{i_\ell} \in C_{i_\ell}$ to show that the value of y_{i_ℓ} is uncertain even if we reveal

1. the value of the indicator $\mathbf{1}_{\{x_{i_\ell+n} \notin C_{i_\ell}\}}$, and

2. the value of the output $y_{i_\ell+1}$.

We now make this idea precise in bounding each term in the product in (4.3), using $\mathcal{G}_{\ell+1}$ to denote the event $\{y_{i_{\ell+1}} = z_{i_{\ell+1}}, \ \ldots, \ y_{i_r} = z_{i_r}\}$.

$$\begin{aligned}
&\Pr[y_{i_\ell} = z_{i_\ell} \mid \mathcal{G}_{\ell+1}, \ \mathcal{E}_\ell, \ \pi_{B'}(x) = \pi] \\
&= \sum_{m=1}^{n} \sum_{b \in \{0,1\}} \Pr[y_{i_\ell} = z_{i_\ell} \mid y_{i_\ell+1} = m, \mathbf{1}_{\{x_{i_\ell+n} \notin C_{i_\ell}\}} = b, \mathcal{G}_{\ell+1}, \mathcal{E}_\ell, \pi_{B'}(x) = \pi] \\
&\qquad \times \Pr[y_{i_\ell+1} = m, \mathbf{1}_{\{x_{i_\ell+n} \notin C_{i_\ell}\}} = b \mid \mathcal{G}_{\ell+1}, \mathcal{E}_\ell, \pi_{B'}(x) = \pi] \\
&\leq \max_{\substack{m \in [0.5n, 0.85n] \\ b \in \{0,1\}}} \Pr[y_{i_\ell} = z_{i_\ell} \mid y_{i_\ell+1} = m, \mathbf{1}_{\{x_{i_\ell+n} \notin C_{i_\ell}\}} = b, \mathcal{G}_{\ell+1}, \mathcal{E}_\ell, \pi_{B'}(x) = \pi] \\
&= \max_{\substack{m \in [0.5n, 0.85n] \\ b \in \{0,1\}}} \Pr[\mathbf{1}_{\{x_{i_\ell} \notin C_{i_\ell}\}} = z_{i_\ell} - m + b \mid \\
&\qquad\qquad y_{i_\ell+1} = m, \mathbf{1}_{\{x_{i_\ell+n} \notin C_{i_\ell}\}} = b, \mathcal{G}_{\ell+1}, \mathcal{E}_\ell, \pi_{B'}(x) = \pi]
\end{aligned} \tag{4.5}$$

where the inequality follows because the conditioning on $\mathcal{E}_\ell$ implies that $y_{i_\ell+1}$ is between $0.5n$ and $0.85n$ and the last equality follows because of the conditioning together with (4.4) applied with $i = i_\ell$. Obviously, unless $z_{i_\ell} - m + b \in \{0, 1\}$ the probability of the corresponding in the maximum in (4.5) will be 0. We will derive our bound by showing that given all the conditioning in (4.5), the probability of the event $\{x_{i_\ell} \notin C_{i_\ell}\}$ is between $2/5$ and $17/18$ and hence each term in the product in (4.3) is at most $17/18$.

Membership of x_{i_ℓ} in C_{i_ℓ}: First note that the conditions $y_{i_\ell+1} = m$ and $\mathbf{1}_{\{x_{i_\ell+n} \notin C_{i_\ell}\}} = b$ together imply that C_{i_ℓ} contains precisely $m - b$ distinct values. We now use the fact that i_ℓ is π-unique and, hence, either $i_\ell \notin Q_\pi$ or $A_\pi(i_\ell) \notin A_\pi(Q_\pi - \{i_\ell\})$.

First consider the case that $i_\ell \notin Q_\pi$. By definition, the events $y_{i_\ell+1} = m$, $\mathbf{1}_{\{x_{i_\ell+n} \notin C_{i_\ell}\}} = b$, $\mathcal{E}_\ell$, and $\mathcal{G}_{\ell+1}$ only depend on x_i for $i > i_\ell$ and the conditioning on $\pi_{B'}(x) = \pi$ is only a property of x_i for $i \in Q_\pi$. Therefore, under all the conditioning in (4.5), x_{i_ℓ} is still a uniformly random value in $[n]$. Therefore the probability that $x_{i_\ell} \in C_{i_\ell}$ is precisely $(m - b)/n$ in this case.

Now assume that $i_\ell \in Q_\pi$. In this case, the conditioning on $\pi_{B'}(x) = \pi$ implies that $x_{i_\ell} = A_\pi(i_\ell)$ is fixed and not in $A_\pi(Q_\pi - \{i_\ell\})$. Again, from the conditioning we know that C_{i_ℓ} contains precisely $m - b$ distinct values. Some of the elements that occur in C_{i_ℓ} may be inferred from the conditioning – for example, their values may have been queried along π – but we will show that there is significant uncertainty about whether any of them equals $A_\pi(i_\ell)$. In this case we will show that the uncertainty persists even if we reveal (condition on) the locations of all occurrences of the elements $A_\pi(Q_\pi - \{i_\ell\})$ among the x_i for $i > i_\ell$.

Other than the information revealed about the occurrences of the elements $A_\pi(Q_\pi - \{i_\ell\})$ among the x_i for $i > i_\ell$, the conditioning on the events $y_{i_\ell+1} = m$, $\mathbf{1}_{\{x_{i_\ell+n} \notin C_{i_\ell}\}} = b$, $\mathcal{E}_\ell$, and $\mathcal{G}_{\ell+1}$, only biases the numbers of distinct elements and patterns of equality among inputs x_i for $i > i_\ell$. Further the conditioning on $\pi_{B'(x)} = \pi$ does not reveal anything more about the inputs in C_{i_ℓ} than is given by the occurrences of $A_\pi(Q_\pi - \{i_\ell\})$. Let $\mathcal{A}$ be the event that all the conditioning is true.

Let $q' = |A_\pi(Q_\pi - \{i_\ell\})| \leq q - 1$ and let $q'' \leq q'$ be the number of distinct elements of $A_\pi(Q_\pi - \{i_\ell\})$ that appear in C_{i_ℓ}. Therefore, since the input is uniformly chosen, subject to the conditioning, there are $m - b - q''$ distinct elements of C_{i_ℓ} not among $A_\pi(Q_\pi - \{i_\ell\})$, and these distinct elements are uniformly chosen from among the elements $[n] - A_\pi(Q_\pi - \{i_\ell\})$. Therefore, the probability that any of these $m - b - q''$ elements is equal to $x_{i_\ell} = A_\pi(i_\ell)$ is precisely $(m - b - q'')/(n - q')$ in this case.

It remains to analyze the extreme cases of the probabilities $(m-b)/n$ and $(m-b-q'')/(n-q')$ from the discussion above. Since $q = \epsilon n$, $q'' \leq q' \leq q - 1$, and $b \in \{0, 1\}$, we have the probability $\Pr[x_{i_\ell} \in C_{i_\ell} \mid \mathcal{A}] \leq \frac{m}{n-q+1} \leq \frac{0.85n}{n-\epsilon n} \leq \frac{0.85n}{n(1-\epsilon)} \leq 0.85/(1-\epsilon) \leq 17/18$ since $\epsilon \leq 1/10$. Similarly, $\Pr[x_{i_\ell} \notin C_{i_\ell} \mid \mathcal{A}] < 1 - \frac{m-q}{n} \leq 1 - \frac{0.5n-\epsilon n}{n} \leq 0.5 + \epsilon \leq 3/5$ since $\epsilon \leq 1/10$. Plugging in the

larger of these upper bounds in (4.3), we get:

$$\Pr[z_{i_1}, \ldots, z_{i_r} \text{ are correct for } F_0^{\boxplus n}(x),\ \mathcal{E} \mid \pi_{B'}(x) = \pi] \leq (17/18)^r,$$

which proves the lemma. □

Putting the Pieces Together We now combine the above lemmas. Suppose that $TS \leq n^2/4800$ and let $q = n/10$. We can assume without loss of generality that $S \geq \log_2 n$ since we need $T \geq n$ to determine even a single answer.

Consider the fraction of inputs in $[n]^{2n-1}$ on which B correctly computes $F_0^{\boxplus n}$. By Lemma 4.13, for input x chosen uniformly from $[n]^{2n-1}$, the probability that $\mathcal{E}$ holds and there are at least $n/24$ positions $j < n$ such that x_j is unique in x is at least $1 - 4ne^{-n/50}$. Therefore, in order to be correct on any such x, B must correctly produce outputs from at least $n/24$ outputs at positions $j < n$ such that x_j is unique in x.

For every such input x, by our earlier outline, one of the 2^S $[n]$-way branching programs B' of height q contained in B produces correct output values for $F_0^{\boxplus n}(x)$ in at least $r = (n/24)q/T \geq 20S$ positions $j < n$ such that x_j is unique in x.

We now note that for any B', if $\pi = \pi_{B'}(x)$ then the fact that x_j for $j < n$ is unique in x implies that j must be π-unique. Therefore, for all but a $4ne^{-n/50}$ fraction of inputs x on which B is correct, $\mathcal{E}$ holds for x and there is one of the $\leq 2^S$ branching programs B' in B of height q such that the path $\pi = \pi_{B'}(x)$ produces at least $20S$ outputs at π-unique positions that are correct for x.

Consider a single such program B'. By Lemma 4.14 for any path π in B', the fraction of inputs x such that $\pi_{B'}(x) = \pi$ for which $20S$ of these outputs are correct for x and produced at π-unique positions, and $\mathcal{E}$ holds for x is at most $(17/18)^{20S} < 3^{-S}$. By Proposition 4.13, this same bound applies to the fraction of all inputs x with $\pi_{B'}(x) = \pi$ for which $20S$ of these outputs are correct from x and produced at π-unique positions, and $\mathcal{E}$ holds for x is at most $(17/18)^{20S} < 3^{-S}$.

Since the inputs following different paths in B' are disjoint, the fraction of all inputs x for which $\mathcal{E}$ holds and which follow some path in B' that yields at least $20S$ correct answers from distinct runs of x is less than 3^{-S}. Since there are at most 2^S such height q branching programs, one of which

must produce $20S$ correct outputs from distinct runs of x for every remaining input, in total only a $2^S 3^{-S} = (2/3)^S$ fraction of all inputs have these outputs correctly produced.

In particular this implies that B is correct on at most a $4ne^{-n/50} + (2/3)^S$ fraction of inputs. For n sufficiently large this is smaller than $1 - \eta$ for any $\eta < 1 - 2^{-\delta S}$ for some $\delta > 0$, which contradicts our original assumption. This completes the proof of Theorem 4.10. □

Lower bound for $(F_0 \bmod 2)^{\boxplus n}$ We describe how to modify the proof of Theorem 4.10 for computing $F_0^{\boxplus n}$ to derive the same lower bound for computing $(F_0 \bmod 2)^{\boxplus n}$. The only difference is in the proof of Lemma 4.14. In this case, each output y_i is $F_0(W_i) \bmod 2$ rather than $F_0(W_i)$ and (4.4) is replaced by

$$y_i = (y_{i+1} + \mathbf{1}_{\{x_i \notin C_i\}} - \mathbf{1}_{\{x_{i+n} \notin C_i\}}) \bmod 2. \tag{4.6}$$

The extra information revealed (conditioned on) will be the same as in the case for $F_0^{\boxplus n}$ but, because the meaning of y_i has changed, the notation $y_{i_\ell+1} = m$ is replaced by $F_0(W_{i_\ell+1}) = m$, $y_{i_\ell+1}$ is then $m \bmod 2$, and the upper bound in (4.5) is replaced by

$$\max_{\substack{m \in [0.5n, 0.85n] \\ b \in \{0,1\}}} \Pr[\mathbf{1}_{\{x_{i_\ell} \notin C_{i_\ell}\}} = (z_{i_\ell} - m + b) \bmod 2 \mid F_0(W_{i_\ell+1}) = m, \mathbf{1}_{\{x_{i_\ell+n} \notin C_{i_\ell}\}} = b, \mathcal{G}_{\ell+1}, \mathcal{E}_\ell, \pi_{B'}(x) = \pi]$$

The uncertain event is exactly the same as before, namely whether or not $x_{i_\ell} \in C_{i_\ell}$ and the conditioning is essentially exactly the same, yielding an upper bound of $17/18$. Therefore the analogue of Lemma 4.14 also holds for $(F_0 \bmod 2)^{\boxplus n}$ and hence the time-space tradeoff of $T \cdot S \in \Omega(n^2)$ follows as before.

Lower Bound for $F_k^{\boxplus n}$, $k \geq 2$ We describe how to modify the proof of Theorem 4.10 for computing $F_0^{\boxplus n}$ to derive the same lower bound for computing $F_k^{\boxplus n}$ for $k \geq 2$. Again, the only difference is in the proof of Lemma 4.14. The main change from the case of $F_0^{\boxplus n}$ is that we need to replace (4.4) relating the values of consecutive outputs.

We will use the notation $F_k(j)$ (resp. $f_i^{(j)}$) to denote the k^{th} frequency moment (resp. the

frequency of symbol i) of the string in the window of length n starting at position j. For $k \geq 2$, we now have

$$y_i - y_{i+1} = \left[\left(f_{x_i}^{(i)}\right)^k - \left(f_{x_i}^{(i)} - 1\right)^k\right] - \left[\left(f_{x_{i+n}}^{(i+1)}\right)^k - \left(f_{x_{i+n}}^{(i+1)} - 1\right)^k\right]. \tag{4.7}$$

We follow the same outline as in the case $k = 0$ in order to bound the probability that $y_{i_\ell} = z_{i_\ell}$ but we reveal the following information, which is somewhat more than in the $k = 0$ case:

1. $y_{i_\ell+1}$, the value of the output immediately after y_{i_ℓ},

2. $F_0(W_{i_\ell+1})$, the number of distinct elements in $W_{i_\ell+1}$, and

3. $f_{x_{i_\ell+n}}^{(i_\ell+1)}$, the frequency of $x_{i_\ell+n}$ in $W_{i_\ell+1}$.

For $M \in \mathbb{N}$, $m \in [n]$ and $1 \leq f \leq m$, define $\mathcal{C}_{M,m,f}$ be the event that $y_{i_\ell+1} = M$, $F_0(W_{i_\ell+1}) = m$, and $f_{x_{i_\ell+n}}^{(i_\ell+1)} = f$. Note that $\mathcal{C}_{M,m,f}$ only depends on the values in $W_{i_\ell+1}$, as was the case for the information revealed in the case $k = 0$. As before we can then upper bound the ℓ^{th} term in the product given in (4.3) by

$$\max_{\substack{m\in[0.5n,0.85n]\\ M\in\mathbb{N}, f\in[m]}} \Pr[y_{i_\ell} = z_{i_\ell} \mid \mathcal{C}_{M,m,f}, \mathcal{G}_{\ell+1}, \mathcal{E}_\ell, \pi_{B'}(x) = \pi] \tag{4.8}$$

Now, by (4.7), given event $\mathcal{C}_{M,m,f}$, we have $y_{i_\ell} = z_{i_\ell}$ if and only if $z_{i_\ell} - M = \left[\left(f_{x_{i_\ell}}^{(i_\ell)}\right)^k - \left(f_{x_{i_\ell}}^{(i_\ell)} - 1\right)^k\right] - \left[f^k - (f-1)^k\right]$, which we can express as as a constraint on its only free parameter $f_{x_i}^{(i)}$,

$$\left(f_{x_{i_\ell}}^{(i_\ell)}\right)^k - \left(f_{x_{i_\ell}}^{(i_\ell)} - 1\right)^k = z_{i_\ell} - M - f^k + (f-1)^k.$$

Observe that this constraint can be satisfied for at most one positive integer value of $f_{x_{i_\ell}}^{(i_\ell)}$ and that, by definition, $f_{x_{i_\ell}}^{(i_\ell)} \geq 1$. Note that $f_{x_{i_\ell}}^{(i_\ell)} = 1$ if and only if $x_{i_\ell} \notin C_{i_\ell}$, where C_{i_ℓ} is defined as in the case $k = 0$. The probability that $f_{x_{i_\ell}}^{(i_\ell)}$ takes on a particular value is at most the larger of the

probability that $f_{x_{i_\ell}}^{(i_\ell)} = 1$ or that $f_{x_{i_\ell}}^{(i_\ell)} > 1$ and hence (4.8) is at most

$$\max_{\substack{m\in[0.5n,0.85n]\\ M\in\mathbb{N},f\in[m],c\in\{0,1\}}} \Pr[\mathbf{1}_{\{x_{i_\ell}\notin C_{i_\ell}\}} = c \mid \mathcal{C}_{M,m,f}, \mathcal{G}_{\ell+1}, \mathcal{E}_\ell, \pi_{B'}(x) = \pi]$$

We now can apply similar reasoning to the $k = 0$ case to argue that this is at most $17/18$: The only difference is that $\mathcal{C}_{M,m,f}$ replaces the conditions $y_{i_\ell+1} = F_0(W_{i_\ell+1}) = m$ and $\mathbf{1}_{\{x_{i_\ell+n}\notin C_{i_\ell}\}} = b$. It is not hard to see that the same reasoning still applies with the new condition. The rest of the proof follows as before.

4.7.2 A time-space efficient algorithm for $F_k^{\boxplus n}$

We now show that our time-space tradeoff lower bound for $F_k^{\boxplus n}$ is nearly optimal even for restricted RAM models.

Theorem 4.15. *There is a comparison-based deterministic RAM algorithm for computing $F_k^{\boxplus n}$ for any fixed integer $k \geq 0$ with time-space tradeoff $T \cdot S \in O(n^2 \log^2 n)$ for all space bounds S with $\log n \leq S \leq n$.*

Proof. We denote the i-th output by $y_i = F_k(x_i, \ldots, x_{i+n-1})$. We first compute y_1 using the comparison-based time $O(n^2/S)$ sorting algorithm of Pagter and Rauhe [123]. This algorithm produces the list of outputs in order by building a space S data structure D over the n inputs and then repeatedly removing and returning the index of the smallest element from that structure using a POP operation. We perform POP operations on D and keep track of the last index popped. We also will maintain the index i of the previous symbol seen as well as a counter that tells us the number of times the symbol has been seen so far. When a new index j is popped, we compare the symbol at that index with the symbol at the saved index. If they are equal, the counter is incremented. Otherwise, we save the new index j, update the running total for F_k using the k-th power of the counter just computed, and then reset that counter to 1.

Let $S' = S/\log_2 n$. We compute the remaining outputs in n/S' groups of S' outputs at a time. In particular, suppose that we have already computed y_i. We compute $y_{i+1}, \ldots, y_{i+S'}$ as follows:

We first build a single binary search tree for both $x_i, \ldots, x_{i+S'-1}$ and for $x_{i+n}, \ldots, x_{i+n+S'-1}$

and include a pointer $p(j)$ from each index j to the leaf node it is associated with. We call the elements $x_i, \ldots, x_{i+S'-1}$ the old elements and add them starting from $x_{i+S'-1}$. While doing so we maintain a counter c_j for each index $j \in [i, i+S'-1]$ of the number of times that x_j appears to its right in $x_i, \ldots, x_{i+S'-1}$. We do the same for $x_{i+n}, \ldots, x_{i+n+S'-1}$, which we call the new elements, but starting from the left. For both sets of symbols, we also add the list of indices where each element occurs to the relevant leaf in the binary search tree.

We then scan the $n - S'$ elements $x_{i+S'}, \ldots, x_{i+n-1}$ and maintain a counter $C(\ell)$ at each leaf ℓ of each tree to record the number of times that the element has appeared.

For $j \in [i, i+S'-1]$ we produce y_{j+1} from y_j. If $x_j = x_{j+n}$ then $y_{j+1} = y_j$. Otherwise, we can use the number of times the old symbol x_j and the new symbol x_{j+n} occur in the window $x_{j+1}, \ldots, x_{j+n-1}$ to give us y_{j+1}. To compute the number of times x_j occurs in the window, we look at the current head pointer in the new element list associated with leaf $p(j)$ of the binary search tree. Repeatedly move that pointer to the right if the next position in the list of that position is at most $n+j-1$. Call the new head position index ℓ. The number of occurrences of x_j in $x_{j+1}, \ldots, x_{S'}$ and $x_{n+1}, \ldots, x_{n+j}$ is now $c_j + c_\ell$. The head pointer never moves backwards and so the total number of pointer moves will be bounded by the number of new elements. We can similarly compute the number of times x_{j+n} occurs in the window by looking at the current head pointer in the old element list associated with $p(j+n)$ and moving the pointer to the left until it is at position no less than $j+1$. Call the new head position in the old element list ℓ'.

Finally, for $k > 0$ we can output y_{j+1} by subtracting $(1+c_j+c_\ell+C(p(j)))^k-(c_j+c_\ell+C(p(j))^k$ from y_j and adding $(1 + c_{j+n} + c_{\ell'} + C(p(j+n)))^k - (c_{j+n} + c_{\ell'} + C(p(j+n))^k$. When $k = 0$ we compute y_{j+1} by subtracting the value of the indicator $\mathbf{1}_{c_j+c_\ell+C(p(j))=0}$ from y_j and adding $\mathbf{1}_{c_{j+n}+c_{\ell'}+C(p(j+n))=0}$.

The total storage required for the search trees and pointers is $O(S' \log n)$ which is $O(S)$. The total time to compute $y_{i+1}, \ldots, y_{i+S'}$ is dominated by the $n - S'$ increments of counters using the binary search tree, which is $O(n \log S')$ and hence $O(n \log S)$ time. This computation must be done $(n-1)/S'$ times for a total of $O(\frac{n^2 \log S}{S'})$ time. Since $S' = S/\log n$, the total time including that to compute y_1 is $O(\frac{n^2 \log n \log S}{S})$ and hence $T \cdot S \in O(n^2 \log^2 n)$. □

4.8 Order Statistics over Sliding Windows

We first show that when order statistics are extreme, their complexity over sliding windows does not significantly increase over that of a single instance.

Theorem 4.16. *There is a deterministic comparison algorithm that computes $MAX_n^{\boxplus n}$ (equivalently $MIN_n^{\boxplus n}$) using time $T \in O(n \log n)$ and space $S \in O(\log n)$.*

Proof. Given an input x of length $2n-1$, we consider the window of n elements starting at position $\lceil \frac{n}{2} \rceil$ and ending at position $n + \lceil \frac{n}{2} \rceil - 1$ and find the largest element in this window naively in time n and space $O(\log n)$; call it m. Assume without loss of generality that m occurs between positions $\lceil \frac{n}{2} \rceil$ and n, that is, the left half of the window we just considered. Now we slide the window of length n to the left one position at a time. At each turn we just need to look at the new symbol that is added to the window and compare it to m. If it is larger than m then set this as the new maximum for that window and continue.

We now have all outputs for all windows that start in positions 1 to $\lceil \frac{n}{2} \rceil$. For the remaining outputs, we now run our algorithm recursively on the remaining $n + \lceil \frac{n}{2} \rceil$-long region of the input. We only need to maintain the left and right endpoints of the current region. At each level in the recursion, the number of outputs is halved and each level takes $O(n)$ time. Hence, the overall time complexity is $O(n \log n)$ and the space is $O(\log n)$. □

In contrast when an order statistic is near the middle, such as the median, we can derive a significant separation in complexity between the sliding-window and a single instance. This follows by a simple reduction and known time-space tradeoff lower bounds for sorting [43, 34].

Theorem 4.17. *Let P be a branching program computing $O_t^{\boxplus n}$ in time T and space S on an input of size $2n-1$, for any $t \in [n]$. Then $T \cdot S \in \Omega(t^2)$ and the same bound applies to expected time for randomized algorithms.*

Proof. We give lower bound for $O_t^{\boxplus n}$ for $t \in [n]$ by showing a reduction from sorting. Given a sequence s of t elements to sort taking values in $\{2, \ldots, n-1\}$, we create a $2n-1$ length string as follows: the first $n-t$ symbols take the same value of n, the last $n-1$ symbols take the same value

of 1 and we embed the t elements to sort in the remaining t positions, in an arbitrary order. For the first window, O_t is the maximum of the sequence s. As we slide the window, we replace a symbol from the left, which has value n, by a symbol from the right, which has value 1. The t^{th} smallest element of window $i = 1, \ldots, t$ is the i^{th} largest element in the sequence s. Then the first t outputs of $O_t^{\boxplus n}$ are the t elements of the sequence s output in increasing order. The lower bound follows from [43, 34]. As with the bounds in Corollary 4.11, the proof methods in [43, 34] also immediately extend to average case and randomized complexity. □

For the special case $t = \lceil \frac{n}{2} \rceil$ (median), we note that the best lower bound known for the single-input version of the median problem is $T \in \Omega(n \log \log_S n)$ derived in [52] for $S \in \omega(\log n)$ and this is tight for the expected time of errorless randomized algorithms.

4.9 Discussion and Open Questions

We derived a $TS \in \Omega(n^2)$ time-space tradeoff for sliding-window frequency moments that is tight up to logarithmic factors. For the element distinctness problem, we have shown that sliding windows increases the complexity of the single-window version by logarithmic factors. For the latter function, we have shown a new sharper $T \in \tilde{O}(n^{3/2}/S^{1/2})$ upper bound on randomized branching programs. Our algorithm is also implementable in similar time and space by RAM algorithms using input randomness. The impediment in implementing it on a RAM with online randomness is our use of truly random hash functions h.

It seems plausible that a similar analysis would hold if those hash functions were replaced by some $\log^{O(1)} n$-wise independent hash function such as $h(x) = (p(x) \bmod m) \bmod n$ where p is a random polynomial of degree $\log^{O(1)} n$, which can be specified in space $\log^{O(1)} n$ and evaluated in time $\log^{O(1)} n$. This would suffice for a similar $T \in \tilde{O}(n^{3/2}/S^{1/2})$ randomized RAM algorithm with the most natural online randomness. The difficulty in analyzing this is the interaction of the chaining of the hash function h with the values of the x_i. This is an open problem that comes up in cryptographic applications where it is desired that repeated applications of a hash function produce independent outputs.

It remains to be able to produce a time-space tradeoff separation for the single-output rather than between the windowed versions of ED and either F_k or $F_0 \bmod 2$. No better clasdical algorithm

than simply sorting the input is known for exactly computing frequency moments. In the context of quantum query complexity, we show another separation between the complexities of the ED and $F_0 \bmod 2$ problems. ED has quantum query complexity $\Theta(n^{2/3})$ (lower bound in [1] and matching quantum query algorithm in [15]). On the other hand, the lower bounds in Chapter 8.9 imply that $F_0 \bmod 2$ has quantum query complexity $\Omega(n)$. Moreover, we show that sliding-window $F_0 \bmod 2$ does not exhibit the same increase in complexity over single-window $F_0 \bmod 2$ it exhibits in classical settings. In fact, we construct a quantum algorithm for $(F_0 \bmod 2)^{\boxplus n}$ that runs in time $O(n^{3/2}\sqrt{n})$ and space $O(\log n)$. It remains to find time-space tradeoffs for $(F_0 \bmod 2)^{\boxplus n}$ on quantum computers.

Part II

ERROR CORRECTING CODES

Chapter 5

CODING THEORY PRELIMINARIES

The theory of error-correcting codes started with the seminal works of Claude Shannon in 1948 [135] and Richard Hamming in 1950 [81] and grew to become an essential tool in the fields of communication, cryptography and theoretical computer science. They have also been used in a wide variety of applications from correcting errors on DVDs and CDs to ensuring reliable satellite transmission. The basic function of an error-correcting code is to encode a message by adding redundancy, thus allowing for the recovery of errors introduced by the communication channel. The correcting ability of a code is determined by its *minimum distance*, a property that measures how similar the encoded messages are to each other; the efficiency of the code is measured by its *rate*, the ratio of the actual information encoded to the information transmitted over the channel. By adding more redundancy to the message, one will achieve better correction abilities but this would cause the efficiency of the code to decrease. One of the main goals in combinatorial coding theory is to find codes achieving a good tradeoff between the two opposing parameters, rate and minimum distance, as the message length tends to infinity. Codes that have the right balance between those parameters are called *asymptotically good codes*.

A code can be thought of as a set of strings, called *codewords* of a fixed length where the symbols are chosen from a fixed alphabet, often denoted Σ. When the alphabet is of size 2, the code is said to be binary. The length of a codeword is the *block length*, usually denoted n, while the message length is denoted m. Hence, the rate of the code is defined as the ratio of the message length to the block length. The minimum distance of the code is the minimum distance between any two codewords, where the distance is the number of positions where symbols differ. Encoding is the process by which a message is mapped to a codeword from the code. The decoding procedure takes a word that was received across the channel and recovers the codeword that was sent from the transmitting end of the channel.

There are two main models of channels that are used: *adversarial channels* or the *Hamming*

model where the errors can occur in any symbol of the codeword but there is an upper bound to the number of errors that can occur and *probabilistic errors channels* or the *Shannon model* where each symbol is the codeword is changed to another symbol with a certain fixed probability (strictly less than 1/2). The restrictions we enforce on the channels (maximum number of errors or a small error probability) is necessary to ensure recovery from errors. For example, allowing for any number of errors in the Hamming model can transform a transmitted codeword to another codeword in the code and it will not be possible to detect that any errors occurred. Therefore, the encoding and decoding procedures will depend on the type of channel used for transmission.

For probabilistic errors channels, Shannon [135] proved that there exists a set of encoding and decoding functions that encode a binary code of constant rate and recover the original message sent across a probabilistic error channel with success probability $1 - o(1)$ [1], where the last probability is over the random choice of the messages. Many explicit constructions of codes with constant rates were found, along with decoders that recover from errors with high probability, e.g. Low Density Parity Check Codes [70].

In the adversarial channels setting, the decoder has access only to the received word and since the errors can occur anywhere in it, the best one can achieve is to find the codeword that is closest in distance to the received word. Such a decoder is called a *nearest-neighbor decoder*. Intuitively, the larger the minimum distance of the code is, the more errors the nearest-neighbor decoder can correct since the codewords are far apart from each other. To establish the limitations of error correction on these channels, many bounds that relate the rate of the code to its minimum distance were found, starting with the Hamming bound [81], thus proving that asymptotically good codes exist. A slew of explicit constructions of asymptotically good codes followed, along with efficient decoding algorithms in settings where the number of errors allowed is upper bounded by half the minimum distance: Reed-Solomon codes, Forney's codes, and Justeseen's codes to name a few. For more information about those bounds and codes, the reader is referred to [113].

We focus on adversarial channels and study asymptotically good codes over them. One important factor in the construction of these codes is the efficiency of their encoders and decoders. Their efficiency is characterized by the amount of resources needed for these computations in terms of

[1] o(1) grows inversely exponential in the block length.

the block length of the code. For example, the naive encoder that uses a look-up table to find the codeword for each message would require space equal to the size of the code, and the decoder that compares each codeword to the received word to find the closest one would also run in time equal to the size of the code, which is exponential in the block length for an asymptotically good code. Efficient encoding and error recovery procedures are essential to guarantee the usability and practicality of a code, especially in real-time applications. But, how efficient can these procedure be for an asymptotically good code? In this thesis, we study how the efficiency of the encoding and error correction and detection relates the two main parameters of the code, the rate and the minimum distance.

We consider three main resources used by the algorithms: the running time and space used by the computation, and the number of positions in the input that are queried. In the remainder of this chapter, we survey some of the lower bounds and time-space tradeoffs known for asymptotically good codes and some of the results about sufficient conditions on codes for the efficient error detection and correction. We also present our contributions in each area. First, we start with some basic definitions and notions from coding theory.

5.1 Coding Theory Basic Notions

In this section, we present notions from coding theory that we will be using in the remaining of the thesis. We use $\mathbb{F}_q$ to denote the finite field of q elements, for any q power of a prime.

DEFINITION 5.1. *Let n be a positive integer. An* error correcting code *C is a subset of $\mathbb{F}_q^n$. The elements of C are called* codewords*, and n is called the* block length*. The* rate *of the code is given by $r = \frac{\log_q |C|}{n}$, where $|C|$ is the number of codewords in the code.*

In most communication settings, one has access to the original (unencoded) messages along with the injective encoding function (also called encoder) that maps each message to a codeword.

DEFINITION 5.2. *Let m, n be positive integers and let $E : \mathbb{F}_q^m \to \mathbb{F}_q^n$ be an injective function. The error correcting code C associated with E is the image of $\mathbb{F}_q^m$ by E in $\mathbb{F}_q^n$. m is called the* message length *and the rate of the code is given by $r = \frac{m}{n}$.*

Since channels introduce errors by changing symbols in a codeword, Hamming [81] defined a

distance that measures how many symbols were changed as the codeword is transmitted across the channel.

DEFINITION 5.3. *The* Hamming distance *between two vectors x and y in $\mathbb{F}_q^n$, denoted $\Delta(x, y)$ is defined as the number of $i \in [n]$ such that $x_i \neq y_i$ and the* fractional Hamming distance *between x and y, denoted $\delta(x, y)$, is defined as $\frac{\Delta(x,y)}{n}$. The* Hamming weight *of a vector x is its Hamming distance to the zero vector 0^n, denoted $w(x)$. The* minimum distance *of a code C is: $d = \min_{x \neq y \in C} \Delta(x, y)$ and the* relative minimum distance *is $\delta(C) = \frac{d}{n}$.*

DEFINITION 5.4. *let v be a vector in $\mathbb{F}_q^n$. The* distance of v to the code C, *denoted as $\delta(v, C)$, is defined as $\delta(v, C) = \min_{x \in C} \delta(v, x)$.*

The rate and the minimum distance are opposing parameters in the design of a code. A higher rate implies higher efficiency but it also implies less redundancy and hence a lower minimum distance. The goal in code design is to achieve both a constant rate and a constant relative minimum distance.

DEFINITION 5.5. *An* asymptotically good code C *is an error-correcting code with positive rate and a positive relative minimum distance. All other codes are called* asymptotically bad.

In this thesis, we are interested in codes that are vector spaces over $\mathbb{F}_q$. These codes are called *linear* codes.

DEFINITION 5.6. *A code C given by the encoder $E : \mathbb{F}_q^m \to \mathbb{F}_q^n$ is said to be* linear *if it is a linear subspace of $\mathbb{F}_q^n$ of dimension m. In that case, C is often denoted $C = [n, m, d]_q$, where d is the minimum distance. The* generator matrix *of a linear code is an $n \times m$ matrix of rank m, where the columns of G form a basis for C.* The dual code *of C is $C^\perp = \{y \in \mathbb{F}_q^n : \langle x, y \rangle = 0, \forall x \in C\}$, where $\langle x, y \rangle$ is the inner product of x and y. Hence, $dim(C) + dim(C^\perp) = n$.* The parity check matrix *H of C is the $n \times (n - m)$ generator matrix of its dual $C^\perp$, thus $H^T G = 0$, where H^T is the transpose of H.*

Therefore, the encoder of a linear code is given by the matrix-vector product $E(x) = Gx$, for $x \in \mathbb{F}_q^m$. Moreover, the minimum distance is given by $d = min_{x \neq 0 \in C} w(x)$ since C is closed under addition and scalar multiplication in $\mathbb{F}_q^n$.

To study the property of a code, it is useful sometimes to delete a subset of its codeword symbols.

DEFINITION 5.7. *Let $S \subseteq [n]$. The* punctured code at *S of a code C is the code resulting from projecting the codewords of C to S. It can be denoted $C|_S$ or $C^{-S'}$, where $S' = [n] \setminus S$, depending on the context.*

A code might have rows in its generator matrix that are identical. This would form a *repetition* in the code.

DEFINITION 5.8. *A* repetition *in a code C is a subset R of $[n]$ such that for all $c \in C$, $c_i = c_j$ for all $i, j \in R$. Hence, $dim(C|_R) = 1$.*

DEFINITION 5.9. *A* subcode *of C is a subspace of C over $\mathbb{F}_q$.*

We use *sparsity* to measure the size of the code in terms of the block length; a sparse code has a small rate. We also use *bias* to measure the concentration of the codeword weights around the minimum distance.

DEFINITION 5.10 (Sparsity and bias). *Let C be a code over the alphabet $\mathbb{F}_q$. C is said to be* $f(n)-$sparse *for a function $f(n) > 0$ if $|C| \leq f(n)$ and is said to be* ϵ-biased *if, for all $x \neq y \in C$, $1 - \frac{1}{q} - \epsilon \leq \delta(x, y) \leq 1 - \frac{1}{q} + \epsilon$.*

We analyze the complexity of some error detection and correction functions related to the code in terms of the number of queries to the received word made by an algorithm computing them. In this case, the algorithm queries the received word $v \in \mathbb{F}_q^n$ through an oracle that takes an input position $\ell \in [n]$ and returns v_ℓ. We are only interested in the number of queries the algorithm makes to the oracle and not its running time. This is relevant in scenarios where access to the received codeword is expensive in resources and it is desired to make as few queries a possible, ideally a constant number of queries. We discuss such scenarios in Section 5.2.

DEFINITION 5.11. *C is said to be* strongly k-locally testable *if there exists a constant $\epsilon > 0$ and a probabilistic algorithm T called the* tester*, that given oracle access to a vector $v \in \mathbb{F}_q^n$, queries the oracle at most k times and accepts every $v \in C$ with probability 1 and rejects every $v \notin C$ with probability at least $\epsilon \,.\, \delta(v, C)$.*

The term "strong" local testability is used to refer to the fact that the probability of rejection is proportional to the distance of the word from the code. Contrast that with the notion of "weak" local-testability where the rejection probability is upper bounded by a constant independent of the distance from the code. Note that strongly locally-testable codes are also weakly locally-testable.

DEFINITION 5.12. *C is said to be k-*self corrector*table if there exist constants $\tau > 0$ and $0 < \epsilon < \frac{1}{2}$ and a probabilistic algorithm SC called the* self corrector*, that given oracle access to a vector $v \in \mathbb{F}_q^n$ that is τ-close to a codeword $c \in C$ and an index $i \in [n]$, queries the oracle at most k times and computes c_i with probability at least $1 - \epsilon$, for some $\epsilon > 0$.*

Note that self-correctability is independent of the encoder used, since we are only interested in recovering a symbol from the closest codeword. Contrast that with the notion of "local-decodability" [90] where one wishes to recover a symbol from the message corresponding to the closest codeword under the encoder. This task requires knowledge of both the encoder and the messages. Moreover, if a linear code is self-correctable, it is also locally decodable since every linear code can be assumed to have the all the message bits appear in the codeword bits.

One of the bounds for codes that we will be using is the *Johnson bound* [87]. It derives an upper bound on the number of codewords that can be found in a ball around a codeword in the code, in terms of the radius of that ball. We present here a strengthened version of the bound due to Guruswami and Sudan.

Theorem 5.1 (The Johnson bound)**.** *[79] Let $C \subseteq \mathbb{F}_q^n$ be a (not necessarily linear) code with relative minimum distance δ. Let $e < \left(1 - \frac{1}{q}\right)\left(1 - \sqrt{1 - \frac{q}{q-1}\delta}\right) n$. Then the number of codewords of C in any Hamming ball of radius e in $\mathbb{F}_q^n$ is at most* $\min\left\{ n(q-1), \frac{\delta n^2}{\delta n^2 - 2e\left(n - \frac{qe}{2(q-1)}\right)} \right\}$.

5.2 Query Complexity of Error Detection and Correction

Although recovering the original message sent across the channel is always desirable, there are settings where error detection (and not correction) is also desirable. A decoder might be expensive in terms of computational complexity and it may be more efficient to check first whether the received word is close to some codeword or far from every codeword. In the latter case, one might choose to

simply ignore the received word since the decoder will not be able to recover the original message. Another setting where error detection is used is with communication where errors are rare and it is more efficient to resend a word received with errors than to correct that word. Error detection is another way to view membership testing in the code and hence it applies to scenarios where one wishes to check the presence of elements in a database in an efficient manner.

The complexity measure we are interested in for membership testing is the number of queries to the input such an algorithm makes. Namely, we will be looking at codes where a tester only considers a constant number of positions in the input. With such a small number of queries to make, the tester cannot be guaranteed to always return the correct answer, hence we relax the correctness of the algorithm to hold with high probability. This relaxation, along with the constant number of queries, guarantee an efficient tester that returns a correct answer most of the time. An error correcting code for which such a tester exists is called a *locally-testable* code.

Locally testable codes are an important tool in theoretical computer science. They are used in the constructions of Probabilistically Checkable Proofs (PCP) [20]. These are defined by a verifier that takes as input two strings, the first is an alleged theorem and the second is a claimed proof for that theorem. The verifier should accept the proof if the theorem is correct and reject any claimed proof otherwise. Moreover, the verifier should query the proof at very few locations. Locally testable codes are at the heart of many PCP constructions and the better rate they have, the more efficient these constructions can be. Since rate and minimum distance are also essential for efficient error detection and correction, this leads to the following natural question, first asked by Goldreich and Sudan in [75]:

Problem 5.2. *Are there any asymptotically good codes that are also locally testable?*

No affirmative or negative answer is yet known for this question. In this thesis, we generalize a result of Kaufman and Sudan [94] that states that random linear codes over large alphabets of small size in terms of their block length (i.e. small rate) are locally testable and conversely a random code with relatively large size (i.e larger rate) cannot be locally tested with a constant number of queries, with high probability.

We can also apply the constant number of queries requirement on algorithms that correct a received word in the following sense: given a received word and a position i, we wish to recover

the symbol at position i of the codeword that is closest to the received word. This is relevant in settings where one wishes to recover only a codeword symbol at a specific position, as opposed to recovering the whole codeword, which might require more computational resources. Similarly to the testing setting, the accuracy of the algorithm needs to be relaxed to hold most of the time. Codes that have such algorithms are called *self-correctable* codes. Note that the self-correctability property is independent of the encoding and requires no knowledge of the encoder or the messages.

Self-correctability is central in the design of (information-theoretic) Private Information Retrieval Schemes [56] where a user wishes to retrieve an item i from a database without revealing any information about i. One trivial way to hide this information is to send the complete database to the user every time she wishes to query the database. However, this would incur a communication complexity equal to the size of the database. One way to make the communication sublinear in the size of the database is to replicate its content among $k \geq 2$ servers. In that case, the user can query the different servers at different locations of the database and then compute the value of the symbol at i from the answers to the queries she receives from the servers. Using a code that is self-correctable with a constant number of queries to encode the different copies of the database keeps the number of servers constant. A rich literature of work uses self-correctable codes (and the similar locally-decodable codes) to construct Private Information Retrieval Schemes [24, 157, 90].

5.2.1 Previous work on local-testability and self-correctability

The field of local-testability for codes started with the seminal work of Blum, Luby and Rubinfeld [42], who showed that Hadamard codes are locally-testable with 3 queries. Hadamard codes are codes whose codewords all have weight $\frac{n}{2}$ but whose rate is inversely exponential in n. In an effort to find locally testable codes with larger rate, many families of codes were proven to be locally testable, e.g. *Reed-Muller* codes [93] and *dual-BCH* codes [91]. Both of those code families include Hadamard codes as a special case but they also include codes that have larger rates than Hadamard codes (inversely polynomial in n), while maintaining a linear minimum distance. In [94] (subsequently strengthened in [101]), Kaufman and Sudan generalized these previous results and found two sufficient conditions for the local-testability and self-correctability of binary linear codes: sparsity (the rate is inverse polynomial) and small-bias (all the codewords have relative weight close

to 1/2). Such codes include all the previously mentioned codes and because of their high minimum distance, their rate is not constant; in fact it is $O(\frac{\log n}{n})$. In [100], Kopparty and Saraf prove that sparse low-bias codes over any abelian group are locally testable. Although their results subsume the results of Kaufman and Sudan in [94] (except for removing the small bias property in the local testability case), we obtained our results in this thesis that appeared in [112] independently and before their work was published. Currently, the largest rate of a known locally testable code with linear minimum distance, is inverse poly-logarithmic in n and it is due to Dinur [61].

5.2.2 Contributions

We generalize the techniques of Kaufman and Sudan in [94] to prove the local-testability and self-correctability of random sparse linear codes over large alphabets. We prove the following:

Theorem 5.3. *Let $C \subseteq \mathbb{F}_q$ be a linear code. For every $t > 0$ and $\gamma > 0$, there exist constant $k_1, k_2 \leq \infty$ such that if C is n^t-sparse and $n^{-\gamma}$-biased, then C is strongly k_1-locally-testable and k_2-self-correctable.*

Since random linear codes of polynomial size satisfy the low-bias property with high probability, it follows that they are locally testable and self-correctable with a constant number of queries.

Corollary 5.4. *Let $t > 0$ and let $A_{n \times t \log_q n}$ be a matrix chosen at random by choosing $t \log n$ basis elements from $\mathbb{F}_q^n$ with replacement. Then there exists $k > 0$ such that the code $C \subseteq F_q^n$ given by the image of the encoder $E(m) = Am$ for $m \in \mathbb{F}_q^{t \log_q n}$ is k-locally-testable and k-self-correctable with probability at least $1 - O(n^{-1})$.*

On the other hand, we prove that random linear codes with size super-polynomial in the block length are not locally-testable nor self-correctable with a constant number of queries, with high probability.

Theorem 5.5. *Let $C \subseteq \mathbb{F}_q^n$ be a random linear code of size $q^{(\log n)^t}$. Then C cannot be locally tested with $o((\log n)^{t-1})$ queries.*

We prove these results in Chapter 6.

5.3 Time and Space Complexity of Encoding

Achieving fast and space-efficient encoding is a desirable feature of designing error-correcting codes but it does not necessarily go hand in hand with good rate or minimum distance. For example, encoders that simply repeat the bits of the message a constant number of times are evidently computationally efficient, and while their rate is also constant with respect to the block length, their relative minimum distance tends to 0 as $1/n$. Therefore, it is important to study how the efficiency of the encoder varies with its rate and minimum distance and to establish tradeoffs between them. When deriving time-space tradeoffs for codes, we consider the following question:

Problem 5.6. *How can we relate the growth of the minimum distance of a binary error correcting code to the computational complexity of its encoder?*

Relating the minimum distance to the complexity of encoders helps to identify asymptotically good codes, especially when estimating the minimum distance directly is a hard task. Many error-correcting codes have been constructed that have low time-space encoding complexity but their minimum distance was hard to bound directly from the code construction. This is the case of Turbo codes constructed by Berrou, Glavieux, and Thitimajshima in [40] and "Turbo-Like" codes constructed by Divsalar, Jin and McEliece [62]. These codes are obtained by concatenating encoders represented by low-memory deterministic state diagrams, with a permutation applied to the message bits. One example of these low-memory encoders are repetition codes that repeat each message bit a constant number of times. The low encoding complexity is crucial to the success of their efficient iterative decoding algorithms on probabilistic error channels, since they cycle through the different states of the encoder and and the low memory usage of the encoder guarantees that their number is low. The efficiency of "Turbo-Like" codes made them a popular and practical choice to use in various communication settings, however their minimum distance was proven by Bazzi and Mitter [26] to be sublinear through the (high) efficiency of their encoders, thus making them a poor choice for communication over adversarial channels.

We focus on linear codes in this thesis. Since linear codes can be encoded by multiplying the generator matrix with the message, any linear code with positive rate can be encoded in quadratic time and logarithmic space on a RAM, regardless of their minimum distance. There are explicit

codes that achieve faster encoding times, at the expense of larger space. *Reed-Solomon codes*, an asymptotically good family of codes based on polynomial evaluations over large fields, can be encoded in $O(n \log^2 n)$ time and $O(n)$ space since their generator matrix is the Vandermonde matrix. Moreover, there are few constructions of asymptotically good binary codes encodable in linear time and linear space [71, 142, 25, 78]. The only explicit construction is Spielman's construction of linear-time encodable and decodable codes [142]. These constructions achieve a time-space product of $TS \in O(n^2)$.

We use the multi-output branching program as our computational model for encoding linear codes. Hence, the branching program computes a matrix-vector product, where the matrix is the generator matrix of the code the vector, which is the input to the branching program, is the message to be encoded. Any code (not necessarily linear) can be encoded with a branching program with linear time and linear space: the branching program is a tree with q^m leaves, each producing the codeword symbols corresponding to a message in $\mathbb{F}_q^m$. The time-space product for that branching program is $TS \in O(m^2)$ regardless of the minimum distance. Therefore, the best time-space tradeoff one can obtain for the encoder of an asymptotically good code is $TS \in \Omega(n^2)$.

5.3.1 Previous work on time-space tradeoffs for code-related functions

The first relation between the minimum distance and time-space encoding complexity was derived by Bazzi and Mitter in [26] in the branching program model. In fact, they establish a general theorem that asserts that if the encoder is a binary branching program that uses linear time and sublinear space, then the minimum distance of the code cannot grow linearly with the block length when the rate is nonvanishing, which is a rather surprising result. Indeed, this result implies that the (popular) Turbo-codes, introduced by Berrou, Glavieux, and Thitimajshima in [40], are asymptotically bad codes since they have low time-space encoding complexity. In general, they prove that if $C : \{0,1\}^m \to \{0,1\}^n$ is a (not necessarily linear) asymptotically good code encodable by branching program B in space S, then then the time $T \in \Omega(n \frac{\log(n/S)}{\log\log(n/S)})$. They improve the lower bound on time to $T \in \Omega(n \log(n/S))$ when the branching program is oblivious. Their proof is based on the single-output functions technique to derive lower bounds, although the underlying branching program has multiple outputs. They modify Ajtai's proof for the lower bound of the Hamming

closeness problem [7] to derive their lower bounds.

Their result partially answers the question in Problem 5.6 since the time-space tradeoff they derive is non-trivial only when $T \in o(n \log n)$. No time-space tradeoff is known for general (or even linear) codes when the time is $T \in \Omega(n \log n)$. Bazzi and Mitter conjectured that $TS \in o(n^2)$ for the encoder imply a sublinear minimum distance.

Conjecture 5.7 (Bazzi-Mitter Conjecture)**.** *[26] Let $C : \{0,1\}^n \to \{0,1\}^m$, $m = rn$, for some $r > 0$, be a code encodable by branching program B in space S and time T such that $TS \in o(n^2)$. Then the minimum distance of C must be $o(n)$, i.e., C cannot be asymptotically good.*

Time-space complexities of other functions related to codes have also been investigated. Following the Bazzi-Mitter result, Santhi and Vardy [131] gave a quadratic time-space tradeoff for computing the *dual syndrome function* of a code. This function is defined as $G^T x$ for an input $x \in \mathbb{F}_q^n$ and G is the generator matrix of the code. It plays the role of an error detector for the dual code. Their result imply a $TS \in \Omega(n^2)$ time-space tradeoff for encoding a very special case of codes that are their own dual, called *self-dual codes*. Although these codes are interesting objects to study and have a rich mathematical theory, they are a small subset of general codes and have strong restrictions on their structure [127]. In [88], Jukna gave an $\Omega(n/k^2 4^k)$ lower bound on the space of a non-deterministic q-way (q is a prime greater than 2) branching program testing the membership of a vector in a code, when the time is $T = kn$, for some $k > 0$. Although this result gives non-trivial lower bound on the space when $k \in O(\log n)$, the function they compute does not relate to the encoder of the code.

5.3.2 Contributions

We describe an approach to derive a $TS \in \Omega(n^2)$ time-space tradeoff for branching programs encoding asymptotically good linear codes. We use Abrahamson's technique for matrix-vector product time-space tradeoffs and derive properties of the generator matrix that are sufficient to get an optimal $TS \in \Omega(n^2)$ lower bound. As a first step to establish these properties in the generator matrix of an asymptotically good code, we prove that any such asymptotically good code contains a code that has no low-dimension projections. We present these results in Chapter 7.

Chapter 6

LOCAL-TESTABILITY AND SELF-CORRECTABILITY OF q-ARY SPARSE LINEAR CODES

In this chapter, we consider the problem of error correction and detection for linear codes over large alphabets and study the number of queries to a received word that are sufficient for testing and correcting individual symbols. Local-testability and self-correctability have a lot of applications in theoretical computer science, namely in Probabilistically Checkable Proofs [20] and Private Information Retrieval Schemes [56]. All the known constructions of locally-testable and self-correctable codes have vanishing rate as the block length tend to infinity and it is not known whether such codes with positive rate exist.

In an effort to characterize locally-testable and self-correctable codes, Kaufman and Sudan [94] proved that sparse binary linear codes with small bias are locally-testable and self-correctable. These codes include all the previously known constructions. We generalize the techniques of Kaufman and Sudan in to prove that q-ary sparse codes with small bias are locally testable and self-correctable

6.1 Contributions and Techniques

We follow the proof strategy of Kaufman and Sudan. We use properties of q-ary Krawtchouk polynomials and the McWilliams identity to bound the weight distributions of duals of sparse codes with small bias. These properties of q-ary Krawtchouk polynomials are non-obvious and were only obtained after the recent work of Krasikov and Zarkh on zeros of discrete orthogonal polynomials in [102]. Thus, extending the results of [94] to q-ary codes requires a more detailed analysis of the underlying Krawtchouk polynomials and their properties. Another tool from coding theory that was important for our analysis is the Johnson bound, which upper bounds the number of codewords in a Hamming ball in $\mathbb{F}_q^n$ in terms of its radius.

Using the bounds we derive on the weight distributions, we obtain the local testability result:

Theorem 6.1. *Let $\mathbb{F}_q$ be the finite field of size q and let $C \subseteq \mathbb{F}_q$ be a linear code. For every $t > 0$ and $\gamma > 0$, there exists a constant $k = k_{q,t,\gamma} <> 0$ s.t. if C is n^t-sparse and $n^{-\gamma}$-biased, then C is strongly k-locally-testable.*

To obtain the self-correctability result, we apply the weight distribution bounds on punctured codes of the original code, where we puncture the code at the position where we want to correct and other related positions. In particular, we prove:

Theorem 6.2. *For every $t > 0$ and $\gamma > 0$, there exists a constant $k = k_{t,\gamma}$ such that if $C \subseteq \mathbb{F}_q^n$ is a n^t-sparse and $n^{-\gamma}$-biased, then C is k-self-correctable.*

The tester and corrector we will construct for our codes will randomly choose dual codewords of a specific weight. The received word will be queried at the non-zero positions of the dual codeword chosen. Hence the number of queries of these algorithms is equal to the weight of the dual codeword. The error probability of these algorithms can be formulated in terms of the number of codewords in the dual code (and in related codes) of specific weights. The sparsity and small bias of the code provide a good landscape of its weight distributions and the MacWilliams identity relates the weight distribution of the code to that of its dual and derives bounds on the weight distribution of the dual codewords, thus allowing us to upper bound the error probabilities.

We start by studying the Krawtchouk polynomials as they are essential for the MacWilliams identity and derive properties related to their roots.

6.2 q-ary Krawtchouk Polynomials and the MacWilliams Identity

Let $q \geq 2, k$ and n be positive integers such that $k, q \leq n$. The Krawtchouk polynomials are a family of polynomials of degree k, ranging from 0 to n, defined over the discrete domain $i = 0, \ldots n$.

DEFINITION 6.1. q-ary Krawtchouk polynomials $P_{k,q,n}(i)$ *are discrete polynomials of degree k on $i = 0, \ldots, n$, defined as:*

$$P_{k,q,n}(i) = \sum_{\ell=0}^{k} \binom{i}{\ell} \binom{n-i}{k-\ell} (-1)^{\ell} (q-1)^{k-\ell}.$$

We derive some properties of Krawtchouk polynomials that will be useful for our analysis.

Proposition 6.3 (Properties of q-ary Krawtchouk polynomials). *Let $P_{k,q,n}(i)$ be q-ary Krawtchouk polynomials of degree k on $i = 0, 1, \ldots, n$. Then:*

a. $P_{k,q,n}(0) = \binom{n}{k}(q-1)^k$.

b. [102]$P_{k,q,n}(i)$ *has k real roots lying between* $(1-\frac{1}{q})n - k(1-\frac{2}{q}) - \frac{2}{q}\sqrt{(q-1)k(n-k)}$ *and* $(1-\frac{1}{q})n - k(1-\frac{2}{q}) + \frac{2}{q}\sqrt{(q-1)k(n-k)}$.

c. $P_{k,q,n}(i) \leq \frac{q^k}{k!}[(1-\frac{1}{q})n - i]^k = \frac{(q-1)^k}{k!}\left(n - \frac{q}{q-1}i\right)^k$, *for all* $i \in [n]$.

d. Let $\mu_1 = (1-\frac{1}{q})n - k(1-\frac{2}{q}) - \frac{2}{q}\sqrt{(q-1)k(n-k)}$ *and* $\mu_2 = (1-\frac{1}{q})n - k(1-\frac{2}{q}) + \frac{2}{q}\sqrt{(q-1)k(n-k)}$.

(i) $|P_{k,q,n}(i)| \leq \frac{q^k}{k!}[(q-1)k(n-k)]^{k/2}$, *for* $\mu_1 \leq i \leq \mu_2$.

(ii) $P_{k,q,n}(i) \leq 0$, *for integer* i, $\mu_2 \leq i \leq n$ *and odd* k.

Proof. a. This follows directly from the definition of $P_{k,q,n}$.

b. Part b follows from Theorem 6 of the work of Krasikov and Karkh [102]. The parameter "q" they use corresponds to $(1 - 1/q)$ in our notation. Moreover, we ignore the additive terms they derive in the bounds on the roots.

c. Part c follows from part b and basic manipulations of $P_{k,q,n}$:

$$\begin{aligned}
P_{k,q,n}(i) &= \sum_{\ell=0}^{k} \binom{i}{\ell}\binom{n-i}{k-\ell}(-1)^\ell (q-1)^{k-\ell} \\
&= \sum_{\ell=0}^{k} \binom{k}{\ell}\frac{1}{k!}\cdot\frac{i!}{(i-\ell)!}\cdot\frac{(n-i)!}{(n-i-k-\ell)!}(-1)^\ell (q-1)^{k-\ell} \\
&\leq \frac{1}{k!}\sum_{\ell=0}^{k} \binom{k}{\ell} i^\ell (n-i)^{k-\ell}(-1)^\ell (q-1)^{k-\ell} \\
&= \frac{1}{k!}\sum_{\ell=0}^{k} \binom{k}{\ell}(-i)^\ell [(n-i)(q-1)]^{k-\ell} \\
&= \frac{1}{k!}[(n-i)(q-1) - i]^k \\
&= \frac{q^k}{k!}[(1-\frac{1}{q})n - i]^k
\end{aligned}$$

d. Part d.(i) follows by replacing i with μ_2 in the above equation. To get part d.(ii), observe that $P_{k,q,n}$ is positive at 0 and alternates signs between its roots. After the largest root, $P_{k,q,n}$ is negative since the number of roots is odd.

□

The value of a q-ary Krawtchouk polynomial evaluated at any $i < n/2$, can be upper bounded in terms of i and $P_{k,q,n}(0)$.

Lemma 6.4. *For every k, for sufficiently large n and for every $\delta < \frac{1}{2}$, $P_{k,q,n}(\delta n) \leq (1-\delta)^k P_{k,q,n}(0)$.*

Proof. For $\delta = 0$, the bound follows trivially. Assume $\delta > 0$. By Proposition 6.3, part c, $P_{k,q,n}(\delta n) \leq \frac{q^k}{k!}[(1-\frac{1}{q})n - \delta n]^k \leq (q-1)^k \frac{n^k}{k!}\left(1 - \frac{q}{q-1}\delta\right)^k$. Since $\frac{n^k}{k!} = \binom{n}{k} + O(n^{k-1})$ and $(1-\frac{q}{q-1}\delta)^k \leq \epsilon(1-\delta)^k$ for some $\epsilon < 1$, we get $P_{k,q,n}(\delta n) \leq (1-\delta)^k (q-1)^k \binom{n}{k} = (1-\delta)^k P_{k,q,n}(0)$, for large enough n. □

Next, we study the *weight distribution* of a subset of $\mathbb{F}_q^n$, as determined by the weight of its elements.

DEFINITION 6.2. *Let D be a subset of $\mathbb{F}_q^n$. For every $i \in [n]$, let B_i^D be the number of vectors in D of weight i. The* weight distribution *of D is given by the vector $\langle B_0^D, \ldots, B_n^D \rangle$.*

Using the sparsity and the bias of a q-ary linear code, we get the following immediate landscape on the weight distribution of the code.

Proposition 6.5. *Let C be an n^t-sparse linear code over $\mathbb{F}_q$ with $\delta(C) \geq 1 - \frac{1}{q} - n^{-\gamma}$, for some $t, \gamma > 0$. Then:*

(a) $B_0^C = 1$.

(b) $B_i^C = 0$ *for all integers* $i \in [1, (1-\frac{1}{q})n - n^{1-\gamma})$.

(c) $\sum_{i=0}^{C} B_i^C \leq n^t$.

(d) If C is $n^{-\gamma}$ biased, then $B_i^C = 0$ *for all integers* $i \in [n(1-\frac{1}{q}) + n^{1-\gamma} + 1, n]$.

Proof. Part (a) follows from the fact that C is linear and hence, it has the 0^n vector as a codeword. Part (b) is an immediate consequence of the minimum distance of C. Part (c) follows from that the sum of the B_i^C for all i is equal to the size of the code. Part (d) follows from definition of bias and from part (b). □

Finally, we state the MacWilliams identity which relates the weight distribution of a linear code to that of its dual in terms of the Krawtchouk polynomials.

Theorem 6.6 (MacWilliams Identity). *[114] Let C be a linear code over $\mathbb{F}_q$ of block length n. Then $B_k^{C^\perp} = \frac{1}{|C|}\sum_{i=0}^{n} B_i^C P_{k,q,n}(i)$, where $P_{k,q,n}(i)$ is the q-ary Krawtchouk polynomial of degree k.*

6.3 Weight distribution of duals of sparse codes

Using the properties of Krawtchouk polynomials, the MacWilliams identity, and the weight distribution of a linear code, the following claim generalizes the bound for binary codes of Kaufman and Sudan in [94] to $q-$ary linear codes.

Lemma 6.7. *For every $\gamma, \eta > 0, c, t < \infty$, if $k \geq (t + c + 1)/\gamma$, then for sufficiently large n and any n^t-sparse set $D \subseteq \mathbb{F}_q^n$, we have:*

(a) $\left|\sum_{i=(1-1/q)n-n^{1-\gamma}}^{(1-1/q)n+n^{1-\gamma}} B_i^D P_{k,q,n}(i)\right| \leq \eta \cdot n^{-c} \cdot P_{k,q,n}(0)$.

(b) If k is odd, $\sum_{i=(1-1/q)n-n^{1-\gamma}}^{n} B_i^D P_{k,q,n}(i) \leq \eta \cdot n^{-c} \cdot P_{k,q,n}(0)$.

Proof. We start with the proof for part (a).

$$
\begin{aligned}
&\left|\sum_{i=(1-1/q)n-n^{1-\gamma}}^{(1-1/q)n+n^{1-\gamma}} B_i^D P_{k,q,n}(i)\right| \\
&\leq \sum_{i=(1-1/q)n-n^{1-\gamma}}^{(1-1/q)n+n^{1-\gamma}} B_i^D \left|P_{k,q,n}(i)\right| \\
&\leq \max\left\{|P_{k,q,n}(i)| : i \in [(1-1/q)n - n^{1-\gamma}, (1-1/q)n + n^{1-\gamma}]\right\} \sum_{i=0}^{n} B_i^D \\
&\leq n^{(1-\gamma)k}\frac{q^k}{k!}\sum_{i=0}^{n} B_i^D \qquad \text{by Proposition 6.3, part c}
\end{aligned}
$$

$$
\begin{aligned}
&\leq n^{-t-c-1} n^k \frac{q^k}{k!} |D| && \text{by the definition of the } B_i\text{'s} \\
&\leq n^{-c} \cdot n^{k-1} \frac{q^k}{k!} \\
&\leq \eta \cdot n^{-c} \cdot (q-1)^k \binom{n}{k} && \text{for all } \eta = \eta_{q,\,k} > 0 \text{ and sufficiently large } n \\
&\leq \eta \cdot n^{-c} \cdot P_{k,q,n}(0).
\end{aligned}
$$

Part (b) of the lemma follows from Proposition 6.3, part d.(ii) since k is odd. □

We use the above lemma to bound the weight enumerators of $C^\perp$.

Lemma 6.8. *Let C be an n^t-sparse code in $\mathbb{F}_q^n$ with $\delta(C) \geq 1 - \frac{1}{q} - n^{-\gamma} - 2/n$. Then, for every $c, t, \gamma, \eta > 0$, there exists a $k_0 \geq (t + c + 1)/\gamma$ such that for every odd $k \geq k_0$,*

$$B_k^{C^\perp} \leq \frac{P_{k,q,n}(0)}{|C|}(1 + \eta \cdot n^{-c})).$$

If C is $n^{-\gamma}$-biased, then for every (odd and even) $k \geq k_0$,

$$B_k^{C^\perp} \in \frac{P_{k,q,n}(0)}{|C|} \left[1 - \eta \cdot n^{-c}, 1 + \eta \cdot n^{-c}\right].$$

Proof. By the MacWilliams Identity and for odd k, we have:

$$
\begin{aligned}
B_k^{C^\perp} &= \frac{1}{|C|} \sum_{i=0}^{n} B_i^C P_{k,q,n}(i) \\
&= \frac{P_{k,q,n}(0)}{|C|} + \frac{1}{|C|} \sum_{i=1}^{n} B_i^C P_{k,q,n}(i) \\
&= \frac{P_{k,q,n}(0)}{|C|} + \frac{1}{|C|} \sum_{i=(1-\frac{1}{q})n - n^{1-\gamma} - 2}^{n} B_i^C P_{k,q,n}(i) && \text{by Proposition 6.5, part (b)} \\
&\leq \frac{P_{k,q,n}(0)}{|C|} + \frac{1}{|C|}(\eta \cdot n^{-c} P_{k,q,n}(0)) && \text{by Lemma 6.7, part (b)} \\
&\leq \frac{P_{k,q,n}(0)}{|C|} \left(1 + \eta \cdot n^{-c}\right).
\end{aligned}
$$

When C is $n^{-\gamma}$-biased, we have:

$$\begin{aligned}
B_k^{C^\perp} &= \frac{P_{k,q,n}(0)}{|C|} + \frac{1}{|C|}\sum_{i=1}^{n} B_i^C P_{k,q,n}(i) \\
&= \frac{P_{k,q,n}(0)}{|C|} + \frac{1}{|C|}\sum_{i=(1-\frac{1}{q})n-n^{1-\gamma}-2}^{(1-\frac{1}{q})n+n^{1-\gamma}-2} B_i^C P_{k,q,n}(i) \qquad \text{by Proposition 6.5, part (d)} \\
&\in \left[\frac{P_{k,q,n}(0)}{|C|} - \frac{1}{|C|}(\eta \cdot n^{-c} P_{k,q,n}(0)), \frac{P_{k,q,n}(0)}{|C|} + \frac{1}{|C|}(\eta \cdot n^{-c} P_{k,q,n}(0))\right] \quad \text{by Lemma 6.7, part (a)} \\
&\in \frac{P_{k,q,n}(0)}{|C|}\left[1-\eta\cdot n^{-c}, 1+\eta\cdot n^{-c}\right].
\end{aligned}$$

□

6.4 Local Testing

As every dual codeword y of the code satisfies $\langle c, y\rangle = 0$ for every codeword in C, computing the inner product of the received word $v \in \mathbb{F}_q^n$ with a dual codeword can serve as a test to its membership in the code. Since the number of queries to v is restricted to k, we choose dual codewords with k non-zero symbols so that the inner product can be computed with only k queries to v. That is exactly how the tester T_k determines whether v is a codeword from C. Let T_k^v denote T_k with oracle access to v.

Algorithm 6.1 T_k

1: Choose y uniformly at random from $[C^\perp]_k$, where $[C^\perp]_k = \{y \in C^\perp, \text{ s.t. } w(y) = k\}$.
2: Accept if and only if $\langle y, v\rangle = 0$, where v is the word to which T_k has oracle access.

If $v \in C$, T_k^v accepts with probability 1. When $v \notin C$, we estimate the probability that T_k^v accepts. To that end, we consider a new code that is the linear span of C and v, $C||v = \bigcup_{\alpha=0}^{q-1}(C + \alpha v)$. The following proposition shows that when $v \notin C$, T_k^v accepts if and only y is in the dual code of $C||v$.

Lemma 6.9. *For $v \notin C$, let $Rej_k(v)$ be the probability that T_k^v rejects. Then,*

$$Rej_k(v) = 1 - \frac{B_k^{(C||v)^\perp}}{B_k^{C^\perp}}.$$

Proof. We show that if $v \notin C$, then T_k^v accepts if and only if $y \in (C||v)^\perp$. If T_k^v accepts, then $\langle y, v\rangle = 0$. Hence, for all $x \in C$ and $\alpha \in \mathbb{F}_q$, $\langle y, x\rangle = 0$ and $\langle y, x + \alpha v\rangle = \langle y, x\rangle + \alpha\langle y, v\rangle = 0$, and $y \in (C||v)^\perp$. If $y \in (C||v)^\perp$, then $\langle y, x + \alpha v\rangle = 0$, for all $x \in C$ and $a \in \mathbb{F}_q$, and thus T_k^v accepts. Since y is chosen uniformly at random from $[C^\perp]_k$, the claim follows. □

To prove the strong local testability of C, we now show that $Rej_k(v) \geq \epsilon\delta(v, C))$, for some $\epsilon > 0$. To that end, we need to upper bound $B_k^{(C||v)^\perp}$ and lower bound $B_k^{C^\perp}$. Lemma 6.8 gives an lower bound on $B_k^{C^\perp}$. The following two lemmas derive an upper bound on $B_k^{(C||v)^\perp}$: Lemma 6.10 gives a bound $B_k^{(C+\alpha v)^\perp}$ for each α and Lemma 6.12 combines those bounds to upper bound $B_k^{(C||v)^\perp}$. These lemmas generalize the results from [94] to codes over larger alphabets, however their proofs are more complicated and require a finer analysis.

Lemma 6.10. *Let $k, t, \gamma > 0$ be constants. Let $\gamma' \leq \gamma/2$. For sufficiently large n, let D be an n^t-sparse code in $\mathbb{F}_q^n$ of distance at least $1 - \frac{1}{q} - n^{-\gamma}$. Let $\delta \leq 1 - \frac{1}{q}$, $a = \max\{(1 - \frac{1}{q})n - n^{1-\gamma'} - \delta n,\ \delta n\}$, and $b = (1 - \frac{1}{q})n - n^{1-\gamma'}$. Then,*

$$\textstyle\sum_{i=a}^{b} P_{k,q,n}(i)B_i^D \leq 2(2q^2+q)P_{k,q,n}(0).\left(\min\left\{\left(1 - \frac{q}{q-1}\delta\right)^{k-2}, \left(\frac{2q}{(q-1)}\delta\right)^{k-2}\right\} + \left(\frac{2q}{(q-1)}n^{-\gamma}\right)^{k-2}\right).$$

Proof. For a code with minimum distance $n(1 - \frac{1}{q}) - n^{1-\gamma}$, the Johnson bound states that, for $i \leq n(1 - \frac{1}{q}) - n^{1-\gamma/2}$, the number of codewords in a ball of radius i about 0^n is at most $\frac{qn^2}{(n - \frac{q}{q-1}i)^2}$. Let $m_i = \frac{qn^2}{(n - \frac{q}{q-1}i)^2}$. Then, by the Johnson bound, $\sum_{j=0}^{i} B_j^D \leq m_i$, for all $i \leq b$. Using Proposition 6.3, part c, we get:

$$\begin{aligned}\sum_{i=a}^{b} P_{k,q,n}(i)B_i^D &\leq \frac{(q-1)^k}{k!}\sum_{i=a}^{b}\left(n - \frac{q}{q-1}i\right)^k B_i^D \\ &\leq \frac{(q-1)^k}{k!}\left(n - \frac{q}{q-1}a\right)^k m_a + \frac{(q-1)^k}{k!}\sum_{i=a+1}^{b}\left(n - \frac{q}{q-1}i\right)^k (m_i - m_{i-1}),\end{aligned}$$

where the second inequality follows from applying the fact below to $x_i = B_i^D$, $y_i = m_i - m_{i-1}$

and $z_i = \left(n - \frac{q}{q-1}i\right)^k$:

Fact 6.11. *For non negative integers $x_1, x_2, \ldots, x_\ell, y_1, y_2, \ldots, y_\ell$ and $z_1 \geq z_2 \geq \ldots \geq z_\ell$, if $\sum_{j=0}^{i} x_i \leq \sum_{j=0}^{i} y_i$ for all $i \in [\ell]$ then $\sum_{i=1}^{\ell} x_i z_i \leq \sum_{i=1}^{\ell} y_i z_i$.*

Replacing m_a by its value in the first term, we get

$$\begin{aligned}
\frac{(q-1)^k}{k!}\left(n - \frac{q}{q-1}a\right)^k m_a &\leq \frac{(q-1)^k}{k!}\left(n - \frac{q}{q-1}a\right)^k \times \frac{qn^2}{\left(n - \frac{q}{q-1}a\right)^2} \\
&\leq \frac{q(q-1)^k}{k!}n^2\left(n - \frac{q}{q-1}a\right)^{k-2}
\end{aligned}$$

Viewing m_i as a function $f(i) = \frac{qn^2}{(n - \frac{q}{q-1}i)^2}$ with first derivative f' and second derivative f'', we have $f(i) \leq f(i-1) + f'(i)$ since $f''(i) > 0$ for all $i \in [a, b]$. Hence, $m_i - m_{i-1} \leq \frac{2q^2n^2}{\left(n - \frac{q}{q-1}i\right)^3}$ and we have:

$$\begin{aligned}
\frac{(q-1)^k}{k!}\sum_{i=a+1}^{b}\left(n - \frac{q}{q-1}i\right)^k (m_i - m_{i-1}) &\leq \frac{(q-1)^k}{k!}\sum_{i=a+1}^{b}\left(n - \frac{q}{q-1}i\right)^k \frac{2q^2n^2}{\left(n - \frac{q}{q-1}i\right)^3} \\
&= \frac{2q^2n^2(q-1)^k}{k!}\sum_{i=a+1}^{b}\left(n - \frac{q}{q-1}i\right)^{k-3} \\
&\leq \frac{2q^2n^2(q-1)^k}{k!}(b-a)\left(n - \frac{q}{q-1}a\right)^{k-3} \\
&\leq \frac{2q^2n^2(q-1)^k}{k!}\left(\frac{q-1}{q}n - a\right)^{k-2} \\
&\leq \frac{2q^2n^2(q-1)^k}{k!}\left(n - \frac{q}{q-1}a\right)^{k-2}
\end{aligned}$$

Adding up the upper bounds, we get $\sum_{i=a}^{b} P_{k,q,n}(i)B_i^D \leq \frac{2q^2+qn^2(q-1)^k}{k!}\left(n - \frac{q}{q-1}a\right)^{k-2}$. Substituting for the value of a and using the crude bound $\frac{(q-1)^k n^k}{k!} \leq 2P_{k,q,n}(0)$, we get:

$\sum_{i=a}^{b} P_{k,q,n}(i)B_i^D \leq 2(2q^2+q)P_{k,q,n}(0).\min\left\{\left(1 - \frac{q}{q-1}\delta\right)^{k-2}, \left(\frac{q}{(q-1)}(\delta + n^{-\gamma})\right)^{k-2}\right\}$.

Using the convexity of the function $f(x) = x^{k-2}$ and the fact that for all $x, y, z > 0$, $\min\{x, y+z\} \leq \min\{x, y\} + z$, the lemma follows. □

We use the above two lemmas to upper bound the weight enumerators of the dual code of $C||v$:

Lemma 6.12. *For every $0 < c, t < \infty$ and $\gamma, \eta > 0$, there exists a k_0 such that if C is an n^t-sparse code of distance $\delta(C) \geq 1 - \frac{1}{q} - n^{-\gamma}$ and $v \in \mathbb{F}_q^n$ is δ-far from C, then, for any odd $k \geq k_0$,*

$$B_k^{(C||v)^\perp} \leq (1 - (1 - \frac{1}{q})\delta + \eta \cdot n^{-c}) \frac{P_{k,q,n}(0)}{|C|}.$$

Proof. Let $\gamma' = \gamma/2$. We will prove the lemma for $k_0 = \max\{k_1, k_2, 16(q^2+q)\}$, where k_1 is chosen to be large enough so that Lemma 6.7 applies and k_2 is the constant given by Lemma 6.8 as a function of t, c and γ.

By the MacWilliams Identity, we have

$$\begin{aligned} B_k^{(C||v)^\perp} &= \frac{1}{q|C|} \sum_{i=0}^{n} B_i^{(C||v)} P_{k,q,n}(i) \\ &= \frac{1}{q|C|} \sum_{i=0}^{n} B_i^{C} P_{k,q,n}(i) + \frac{1}{q|C|} \sum_{\alpha=1}^{q-1} \sum_{i=0}^{n} B_i^{(C+\alpha v)} P_{k,q,n}(i) \\ &= \frac{1}{q} B_k^{C^\perp} + \frac{1}{q|C|} \sum_{\alpha=1}^{q-1} \sum_{i=0}^{n} B_i^{(C+\alpha v)} P_{k,q,n}(i) \end{aligned}$$

By Lemma 6.8, we have $B_k^{C^\perp} \leq \frac{P_{k,q,n}(0)}{|C|}(1 + \frac{\eta}{q} \cdot n^{-c})$. Hence, it is enough to prove for every $\alpha = 1, 2, \ldots, q-1$,

$$\frac{1}{|C|} \sum_{i=0}^{n} B_i^{(C+\alpha v)} P_{k,q,n}(i) \leq (1 - \delta + \frac{\eta}{q} \cdot n^{-c}) \frac{P_{k,q,n}(0)}{|C|}.$$

Applying Lemma 6.7, part (b), to $(C+\alpha v)$, for every η we get $\sum_{i=n(1-\frac{1}{q})-n^{1-\gamma'}}^{n} B_i^{(C+\alpha v)} P_{k,q,n}(i) \leq \frac{\eta}{2q} \cdot n^{-c} \cdot P_{k,q,n}(0)$. Now, it suffices to prove $\sum_{i=0}^{n(1-\frac{1}{q})-n^{1-\gamma'}} B_i^{(C+\alpha v)} P_{k,q,n}(i) \leq (1 - \delta + \eta' n^{-c}) P_{k,q,n}(0)$, where $\eta' = \frac{\eta}{2q}$.

Since $\delta(C) \geq 1 - \frac{1}{q} - n^{-\gamma}$, by Proposition 6.5, part (b), $B_i^{C+\alpha v} = 0$ for every integer $i \in [0, n(1-\frac{1}{q}) - n^{1-\gamma'} - \delta n]$, except possibly for $i = \delta n$. This happens when $\delta n \leq n(1-\frac{1}{q}) - n^{1-\gamma'} - \delta n$ $\alpha v \in C + \alpha v$. In that case, $\alpha v \in (C + \alpha v)$ and since $\delta(v, C) \leq \delta$, there exists $c \in C$ such that $\delta(\alpha v, c) \leq \delta$. Hence $B_{\delta n}^{C+\alpha v} = 1$. Thus, we have

$$\sum_{i=0}^{n(1-\frac{1}{q})-n^{1-\gamma'}} B_i^{C+\alpha v} P_{k,q,n}(i) \leq P_{k,q,n}(\delta n) + \sum_{i=a}^{b} B_i^{C+\alpha v} P_{k,q,n}(i),$$

where $a = \max\{n(1-\frac{1}{q}) - n^{1-\gamma'} - \delta n, \delta n\}$ and $b = n(1-\frac{1}{q}) - n^{1-\gamma'}$. Applying Lemma 6.4 and using the bound in Lemma 6.10 for these values of a and b, we get:

$$\sum_{i=0}^{b} B_i^{C+\alpha v} P_{k,q,n}(i)$$
$$\leq P_{k,q,n}(0) \left((1-\delta)^k + 2(2q^2+q).\min\left\{ (1-\frac{q}{(q-1)}\delta)^{k-2}, \left(\frac{q}{q-1}\delta\right)^{k-2} \right\} + \left(\frac{2q}{(q-1)}n^{-\gamma}\right)^{k-2} \right).$$

To obtain the desired upper bound, first observe that $\left(\frac{2q}{(q-1)}n^{-\gamma}\right)^{k-2} \leq \frac{\eta}{2q} \cdot n^{-c}$, for every $\eta > 0$ and n sufficiently large, since $\gamma k \geq t + c + 1$. Next, we need to prove that for odd $k \geq k_0$, where k_0 will be specified below, we have:

$(1-\delta)^k + 2(2q^2+q).\min\left\{ \left(1-\frac{q}{q-1}\delta\right)^{k-2}, \left(\frac{q}{(q-1)}\delta\right)^{k-2} \right\} \leq (1-\delta)$, for $0 \leq \delta \leq 1 - 1/q$.

We consider the following cases:

- For $\delta \leq \frac{1}{40q^2}$ and $k \geq 4$, we have: $2(2q^2+q)\left(\frac{q}{q-1}\delta\right)^{k-2} \leq 2(2q^2+q)\left(\frac{q}{q-1}\delta\right)^{2} \leq 20q^2\delta^2 \leq \frac{1}{2}\delta$. Therefore $(1-\delta)^k + 2(2q^2+q)\left(\frac{q}{q-1}\delta\right)^{k-2} \leq (1-\delta)^2 + \frac{1}{2}\delta \leq 1-\delta$ since $\delta \leq \frac{1}{2}$.

- For $\frac{1}{40q^2} \leq \delta \leq 1 - \frac{1}{q}$ and $k \geq 240q^4 + 3$, we have:

$$\begin{aligned}
&(1-\delta)^k + 2(2q^2+q)\left(1-\frac{q}{q-1}\delta\right)^{k-2} \\
&= (1-\delta)^2 \cdot (1-\delta)^{k-2} + 2(2q^2+q)\left(1-\frac{q}{q-1}\delta\right)^{k-2} \\
&\leq (1-\delta)^2 \cdot (1-\delta)^{k-2} + 2(2q^2+q) \cdot (1-\delta)^{k-2} \qquad \text{since } (1-\delta)^{k-2} \geq \left(1-\tfrac{q}{q-1}\delta\right)^{k-2} \\
&= (1-\delta)^{k-2} \cdot \left((1+\delta^2-2\delta+2(2q^2+q)\right) \\
&\leq 6q^2(1-\delta)^{k-2} \qquad \text{since } 6q^2 \geq 2(2q^2+q)+1 \text{ for all } q \geq 2 \text{ and } \delta \geq 2.
\end{aligned}$$

  Therefore $(1-\delta)^k + 2(2q^2+q)\left(1-\frac{q}{q-1}\delta\right)^{k-2} \leq 6q^2(1-\delta)(1-\frac{1}{40q^2})^{k-3} \leq 6q^2(1-\delta)e^{-(k-3)/40q^2} \leq (1-\delta)6q^2e^{-6q^2} \leq (1-\delta)$ since $xe^{-x} \leq 1$ for all $x \geq 1$.

The lemma follows for $k_0 = \max\{\frac{t+c+1}{\gamma}, 240q^4 + 3\}$. □

Finally, we prove the main theorem for local testability for small-bias codes over large alphabets.

Proof of Theorem 6.1. Let $v \in \mathbb{F}_q^n$ such that $\delta(v, C) = \delta$. To prove local testability, we prove that $Rej_k(v) \geq (1 - \frac{1}{q})\delta$. Recall that $Rej_k(v) = 1 - \frac{B_k^{(C||v)^\perp}}{B_k^{C^\perp}}$. Given t and γ, let k be an odd integer greater than k_0 as given by Lemma 6.12. Therefore, we have $B_k^{(C||v)^\perp} \leq (1 - (1 - \frac{1}{q})\delta + \eta \cdot n^{-c})\frac{P_{k,q,n}(0)}{|C|}$. Moreover, from Lemma 6.8, we have $B_k^{C^\perp} \in \frac{P_{k,q,n}(0)}{|C|}[1 - \eta \cdot n^{-c}, 1 + \eta \cdot n^{-c}]$. Therefore, $Rej_k(v) = (1 - \frac{1}{q})\delta - \eta \cdot n^{-c}$. Using $c = 2$ and the fact that $\delta > \frac{1}{n}$, for sufficiently large n, $Rej_k(v) = (1 - \frac{3}{2q})\delta$ and the theorem follows. □

6.5 Self-correctability

Similarly to local testing, the self corrector $SC_k(i)$, has oracle access to a word $v \in \mathbb{F}_q^n$ and takes as input an index $i \in [n]$, uses a dual codeword y of weight k such that $y_i \neq 0$. As the inner product $\langle v, y\rangle$ is equal to 0 when $v \in C$, $SC_k(i)$ uses that equality to compute v_i. Hence, the resulting number of queries is $k - 1$.

Algorithm 6.2 $SC_k(i)$

1: Choose y uniformly at random from $[C^\perp]_{k,i}$, where $[C^\perp]_{k,i} = \{y \in C^\perp, \text{ s.t. } w(y) = k \text{ and } y_i \neq 0\}$.
2: Output $(-y_i)^{-1} \sum_{j \in [n]-\{i\} s.t. y_j \neq 0} v_j$, where v is the word to which SC_k has oracle access.

$SC_k(i)$ has oracle access to v such that $\delta(v, c) < \frac{1}{2k}$, for every $c \in C$. It makes $k - 1$ queries to v. Let $SC_k^v(i)$ denote $SC_k(i)$ with oracle access to v. If $v \in C$, $SC_k^v(i)$ correctly computes $c_i = v_i$ since $\langle v, y\rangle = 0$. Otherwise, it is incorrect if the errors in v occur at positions where $y_j \neq 0$. In order to upper bound the error probability, we need estimates on the sets of dual codewords that are non-zero at i and positions where the errors occur.

We start by estimating the probability that y chosen uniformly at random from $[C^\perp]_k$ has non-zero entries at indices i and j. Let $[C^\perp]_{k,i}$ be the set of duals codewords of weight k with a non-zero symbol at position i. $C^{-i} = \{\pi_{-i}(c), c \in C\}$, where $\pi_{-i}(c_1, \ldots, c_{i-1}, c_i, c_{i+1}, \ldots, c_n) = (c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_n)$. Hence, C_i is the code C punctured at position i.

Proposition 6.13. *Let* $\pi_{-i}^{-1}(c_1,\ldots,c_{i-1},c_{i+1},\ldots,c_n) = (c_1,\ldots,c_{i-1},0,c_{i+1},\ldots,c_n)$ *and* $\pi_{-i}^{-1}(D) = \{\pi_{-i}^{-1}(y) \mid y \in D\}$. *If* $\delta(C) \geq \frac{2}{n}$, *then* $[C^\perp]_{k,i} = [C^\perp]_k - \pi_{-i}^{-1}\left([(C^{-i})^\perp]_k\right)$ *and hence,* $|[C^\perp]_{k,i}| = |[C^\perp]_k| - |[(C^{-i})^\perp]_k|$.

Proof. $[C^\perp]_k - [C^\perp]_{k,i}$ is the set of dual codewords of weight k and a 0 at position i. Let $y \in [C^\perp]_k$. If $y_i = 0$, then $\pi_{-i}^{-1}(y_{-i}) = y$. Moreover, $\pi_{-i}(y) \in (C^{-i})^\perp$ since for all $c \in C$, $\langle \pi_{-i}(c), \pi_{-i}(y)\rangle = \langle c, y\rangle - c_i y_i = 0$. Hence $\{\pi_{-i}(y) \mid y \in [C^\perp]_k$ such that $y_i = 0\} \subseteq [(C^{-i})^\perp]_k$. To finish the proof, we show that $(C^{-i})^\perp = \{\pi_{-i}(y) \mid y \in C^\perp s.t.\ y_i = 0\}$.

$(C^{-i})^\perp$ is the dual of C^{-i}, hence $|(C^{-i})^\perp| = \frac{q^{n-1}}{|C^{-i}|}$. Since $\delta(C) \geq \frac{2}{n}$, the homomorphism between C^{-i} and C has only 0^n in its kernel. Then $|C^{-i}| = |C|$ and $|(C^{-i})^\perp| = \frac{q^{n-1}}{|C|}$ since $\delta(C) \geq \frac{2}{n}$. Thus, $|(C^{-i})^\perp| = \frac{1}{q}|C^\perp|$. On the other hand, $|\{\pi_{-i}(y) \mid y \in C^\perp$ such that $y_i = 0\}| \geq \frac{1}{q}|C^\perp|$ since π_{-i} is a homomorphism between $C^\perp$ and $(C^\perp)^{-i}$. Therefore $(C^{-i})^\perp = \{\pi_{-i}(y) \mid y \in C^\perp$ such that $y_i = 0\}$ and the lemma follows. □

Extending the puncturing to two indices $i \neq j$, let $C^{-\{i,j\}}$ be the projection of C on $[n] - \{i,j\}$ and let $\pi_{-\{i,j\}}$ be that projection. Using the inclusion-exclusion principle and applying proposition 6.13 to C^{-j} , we get:

Proposition 6.14. *For every* $i \neq j$, *if* $\delta(C) \geq \frac{3}{n}$, *then* $[C^\perp]_{k,\{i,j\}} = [C^\perp]_k - \pi_{-i}^{-1}([(C^{-i})^\perp]_k) - \pi_{-j}^{-1}([(C^{-j})^\perp]_k) + \pi_{-\{i,j\}}^{-1}([(C^{-\{i,j\}})^\perp]_k)$. *Hence,* $|[C^\perp]_{k,\{i,j\}}| = |[C^\perp]_k| - |[(C^{-i})^\perp]_k| - |[(C^{-j})^\perp]_k| + |[(C^{-\{i,j\}})^\perp]_k|$.

Using what we know about weight distributions of the above dual codes from lemma 6.8, we derive the probability that $y_j \neq 0$ when y is chosen at random from $[C^\perp]_{k,i}$.

Lemma 6.15. *For every* $\gamma, \eta > 0$ *and* $0 < c, t < \infty$, *there exists* k *such that for sufficiently large n, if* $C \subseteq \mathbb{F}_q^n$ *is an* n^t*-sparse,* $n^{-\gamma}$*-biased linear code, then for every* $i \neq j \in [n]$, $\Pr_{y \in_U [C^\perp]_{k,i}}[y_j \neq 0] = \frac{|[C^\perp]_{k,\{i,j\}}|}{|[C^\perp]_{k,i}|} \leq \frac{(k-1)}{n-1} + \eta \cdot n^{-c}$.

Proof. First, note that $\Pr_y[y_j \neq 0] = \frac{|[C^\perp]_{k,\{i,j\}}|}{|[C^\perp]_{k,i}|}$, where y is chosen uniformly at random from $[C^\perp]_{k,i}$ Using the above two propositions, we calculate the size of those sets. C, C^{-i}, C^{-j} and $C^{-\{i,j\}}$ are n^t-sparse, $n^{-\gamma}$-biased codes, with respective block lengths n, $n-1$, $n-1$ and $n-2$ and relative minimum distances $\delta(C), \geq \delta(C) - 1/n$, $\geq \delta(C) - 1/n$ and $\geq \delta(C) - 2/n$, respectively.

Note also that they all have the same size. Picking k large enough to apply Lemma 6.8 to these codes, we have for all $\eta_1 > 0$:

$$|[C^{\perp}]_k| \in \frac{(q-1)^k}{|C|}\left(\left[\binom{n}{k} - \eta_1 \cdot n^{-c}, \binom{n}{k} + \eta_1 \cdot n^{-c}\right]\right)$$

$$|[(C^{-i})^{\perp}]_k| \in \frac{(q-1)^k}{|C|}\left(\left[\binom{n-1}{k} - \eta_1 \cdot n^{-c}, \binom{n-1}{k} + \eta_1 \cdot n^{-c}\right]\right)$$

$$|[(C^{-j})^{\perp}]_k| \in \frac{(q-1)^k}{|C|}\left(\left[\binom{n-1}{k} - \eta_1 \cdot n^{-c}, \binom{n-1}{k} + \eta_1 \cdot n^{-c}\right]\right)$$

$$|[(C^{-\{i,j\}})^{\perp}]_k| \in \frac{(q-1)^k}{|C|}\left(\left[\binom{n-2}{k} - \eta_1 \cdot n^{-c}, \binom{n-2}{k} + \eta_1 \cdot n^{-c}\right]\right)$$

Now we use Proposition 6.14 to obtain:

$$|[C^{\perp}]_{k,i}| \geq \frac{(q-1)^k}{|C|}\left(\binom{n-1}{k-1} \cdot (1 - \eta_2 \cdot n^{-c})\right)$$

and

$$|[C^{\perp}]_{k,\{i,j\}}| \leq \frac{(q-1)^k}{|C|}\left(\binom{n-2}{k-1}\frac{(k-1)}{(n-k)} \cdot (1 - \eta_3 \cdot n^{-c})\right),$$

for all $\eta_2, \eta_3 > 0$. Therefore, $\frac{|[C^{\perp}]_{k,\{i,j\}}|}{|[C^{\perp}]_{k,i}|} \leq \frac{(k-1)}{n-1} + \eta \cdot n^{-c}$, for all $\eta > 0$. □

Now, we prove the main lemma that bounds the error probability of $SC_k^v(i)$.

Lemma 6.16. *For every $t > 0$, $\gamma, \eta > 0$, there exists a constant $k = k_{t,\gamma} > 0$ such that: if C is an n^t-sparse, $n^{-\gamma}$-biased linear code in $\mathbb{F}_q^n$ and $v \in \mathbb{F}_q^n$ is τ-close to C, then for every $i \in [n]$, $\Pr[SC_k^v(i) \neq c_i] \leq k\tau + \eta/n$.*

Proof. Choose k large enough to apply Lemma 6.15 with $c = 2$. Hence, for $y \in_U [C^{\perp}]_{k,i}$ and $i \neq j$, $\Pr[y_j \neq 0] \leq \frac{k-1}{n-1} + \eta \cdot n^{k-2}$.

Let E be the set of errors in v, i.e. $E = \{j \in [n] | v_j \neq c_j\}$. Since v is τ-close to C, then $|E| \leq \tau n$. Let S_y be the set of non-zero symbols in y, i.e. $S_y = \{j \in [n] | y_j \neq 0\}$. $SC_k^v(i)$ will err only if the errors on v line up with some non-zero symbol of y, and hence only if $E \cap S_y \neq \phi$. Therefore, $\Pr_{y \in [C^{\perp}]_{k,i}}[SC_k^v(i) \neq c_i] = \Pr_y[E \cap S_y \neq \phi] = \Pr_y[\exists j \in E \text{ such that } j \in S_y] \leq |E| \max_{j \in E} \Pr_y[y_j \neq 0] \leq k\tau + \eta/n$. □

Choosing τ to be strictly less than $\frac{1}{2k}$, the probability of error of SC_k^v is hence strictly less than 1/2. The self-correctibilty of sparse small-biased linear codes in $\mathbb{F}_q^n$ (Theorem 6.2) directly follows.

6.6 Applications to random linear codes

In this section, we study the bias of random linear codes with $O(n^t)$ codewords. We prove that such code have low bias and hence are locally-testable and self-correctable. We start by computing the probability that a random linear code of dimension $t \log_q n$ has low bias. Using the Azuma-Hoeffdding inequality applied to a Doob's martingale, we get the following easy fact.

Proposition 6.17. *Let $A_{n \times t \log_q n}$ be a matrix chosen at random by choosing $t \log n$ basis elements from $\mathbb{F}_q^n$ with replacement. Then the code $C \subseteq F_q^n$ given by the image of the encoder $E(m) = Am$ for $m \in \mathbb{F}_q^{t \log_q n}$ has bias at most $\sqrt{\frac{4t \log n}{n}}$ with probability at least $1 - n^{-t}$.*

Proof. To upper bound the probability that the bias is at most $\sqrt{\frac{4t \log n}{n}}$, it is enough to upper bound the probability: $\Pr\left[\exists m \in \{0,1\}^{t \log n} \text{ s.t. } (w(Am) - n(1 - \frac{1}{q})) \geq n\sqrt{\frac{4t \log n}{n}}\right] \leq n^t \Pr[(w(Am) - n(1 - \frac{1}{q})) \geq n\sqrt{\frac{4t \log n}{n}}]$, where $w(x)$ is the number of 1's in x. Let $a_1, a_2, \ldots, a_{t \log n}$ denote the rows of A. We define a Doob martingale, as explained in Chapter 2.4, w_i, $i = 0, 1, \ldots, n$ with respect to the sequence $a_1, a_2, \ldots, a_n$ by $w_i = \mathbb{E}[w(\langle a_1, m\rangle, \langle a_2, m\rangle, \ldots, \langle a_i, m\rangle | a_1, a_2, \ldots, a_t)]$. Therefore $w_0 = \mathbb{E}[w(x)] = n(1 - \frac{1}{q})$ for x chosen uniformly at random from $\mathbb{F}_q^n$ and $w_n = w(Am)$. Therefore by applying the Azuma-Hoeffdding inequality, we have $\Pr_A[(w(Am) - n(1 - \frac{1}{q})) \geq n\sqrt{\frac{4t \log n}{n}}] = \Pr_A[|w_n - w_0| \geq \frac{2tn \log n}{\sqrt{n}}] \leq n^{-2t}$ and the proposition follows.

□

Using Theorems 6.1 and 6.2, we prove the local-testability and self-correctability of sparse random linear codes.

Corollary 6.18. *Let $t > 0$ and let $A_{n \times t \log_q n}$ be a matrix chosen at random by choosing $t \log n$ basis elements from $\mathbb{F}_q^N$ with replacement. Then there exists $k > 0$ such that the code $C \subseteq F_q^n$ given by the image of the encoder $E(m) = Am$ for $m \in \mathbb{F}_q^{t \log_q n}$ is k-locally-testable and k-self-correctable with probability at least $1 - O(n^{-1})$ in n.*

On the other hand, if C is a random linear code of superpolynomial size, then it cannot be tested with a constant number of queries as shown by the following theorem:

Theorem 6.19. *Let $C \subseteq \mathbb{F}_q^n$ be a random linear code of size $q^{(\log n)^t}$. Then C cannot be locally tested with $o((\log n)^{t-1})$ queries.*

The proof is a simple generalization of that of the same theorem for binary linear codes in [94].

6.7 Discussion and Open Questions

We proved that sparse codes with small bias over large alphabets are locally testable and self-correctable. We used properties of the generalized Krawtchouk polynomials and some basic results from coding theory like the McWilliams identity and the Johnson bound. As random sparse linear codes have low-bias, they are also locally-testable and self-correctable.

The next step is to relax the small bias condition, while maintaining a relative minimum distance close to $1 - \frac{1}{q}$. Kaufman and Sudan were able to remove the small bias condition (thus allowing codewords for arbitrarily large weight) for local testability in the case of sparse binary codes with relative minimum distance at least $\frac{1}{2} - n^{-\gamma}$ in [94]. This result is tight as shown in [37], where a construction is given for a code of linear minimum distance and dimension $1.1 \log n$ that not locally testable with $o(\log n)$ queries. Even in the binary case, removing the small bias condition for self-correctability of sparse codes with large distance is still an open problem.

Since our results were published, local testability was studied for *affine-invariant* codes. A linear code is *affine-invariant* if the coordinates of the code can be viewed as a vector space and the code is invariant under affine transformations of the coordinates. The interest in affine-invariance comes from the fact that most (explicitly constructed) locally-testable codes (e.g. Reed-Muller codes) are affine-invariant. Furthermore, the local-testability of these codes is a result of their affine-invariance. Building on previous work [76, 92], Ben-Sasson, Ron-Zewi and Sudan [36] proved that sparse affine-invariant codes over any field of arbitrary size are locally testable. Despite these results, it is still not clear whether affine-invariance is necessary for local-testability.

Chapter 7

SUFFICIENT CONDITIONS FOR OPTIMAL TIME-SPACE TRADEOFFS FOR ENCODERS

The results in this chapter describe an approach based on Abrahamson's matrix-vector product time-space tradeoff [3, 5], to derive optimal time-space tradeoffs for encoding asymptotically good binary error-correcting codes. As a first step towards that time-space tradeoff, we describe a procedure to remove low-dimension projections from these codes. We hope that these results are useful for future research.

7.1 From Branching Programs to Rigid Matrices

Let $E : \{0,1\}^m \rightarrow \{0,1\}^n$ be an encoder for an asymptotically good binary linear code, and B a branching program computing E. As we are restricting ourselves to binary linear codes, the encoder can be viewed as a matrix-vector product. The generator matrix G associated with E is an $n \times m$ matrix of rank m.

Recall from Theorem 2.6 that in order to derive a $TS \in \Omega(n^2)$ time-space tradeoff for B, it is enough to prove that $pr^L_{E,\mathcal{U}}(h,k) \leq c^{-k}$ for $|L| = \Theta(n)$, $h = \Theta(m)$, $k \leq \frac{|L|h}{T}$, c a constant and $\mathcal{U}$ the uniform distribution over $\{0,1\}^m$. $pr^L_{E,\mathcal{U}}(h,k)$ is the maximum probability that a branching program of height at most h correctly produces k bits of the codeword $E(x)$ at positions from the subset L of output positions, for a random input $x \in \{0,1\}^m$. To establish the upper bound on $pr^L_{E,\mathcal{U}}(h,k)$, we define a class of "rigid" matrices and prove that the upper bound holds if the generator matrix G belongs to that class. Note that this notion of rigidity is not the same as the classical notion of matrix rigidity of Valiant [147] where one counts the number of entries in a matrix that need to be changed to reduce the rank of the matrix. Our notion of rigidity involves changing complete rows or columns to 0 and not individual entries

DEFINITION 7.1. *Let G be an $n \times m$ matrix and let $\gamma \leq 1$ be a positive constant and k and h be positive such that $k + h \leq m$. G is (h,k,γ)-rigid iff there exist a subset $L \subseteq [m]$ of size at least k*

such that for all $R \subseteq L$ and for all $U \subset [n], |U| \geq m - h$, rank$(G_{R\times U}) \geq \gamma|R|$, where $(G_{R\times U})$ is the submatrix of G formed by the rows in R and columns in U.

Definition 7.1 generalizes that of c-ok matrices described in [5]. The main difference is allowing some rows to be removed from the matrix G. This is necessary when the generator is of the form $\begin{pmatrix} I_m \\ Z \end{pmatrix}$, when I_m is the $m \times m$ identity matrix. In that case, for any subset of the rows in the first m rows, a subset of the columns can be removed such that the rank of the resulting submatrix is 0.

The following theorem generalizes the proof technique used in [5] to derive lower bounds for matrix-vector products.

Theorem 7.1. *Let $\gamma \leq 1$ be a positive constant and k, h, m and n be positive such that $k + h \leq m$. Let G be an $n \times m$ (h, k, γ)-rigid matrix and $L \subseteq [n]$ be the subset of rows as in Definition 7.1 and $E : \{0,1\}^m \to \{0,1\}^n$ be the function such that $E(x) = Gx$. Let p and q be positive integers such that and $p \leq h$ and $q \leq k$. Then $pr^L_E(p, q) \leq 2^{-\gamma q}$.*

Proof. Since G is a (h, k, γ)-rigid matrix, then $|L| \geq k$ and for all $R \subseteq L$ and $U \subset [m], |U| \geq m - h$, we have $rank(G_{R\times U}) \geq \gamma|R|$.

Let P be a depth p branching program that q-solves x. Let π be the computation of P on x and denote its outputs $v_1, v_2, \ldots, v_q$. Each output bit v_i correspond to a linear equation in x since P computes a subset of the outputs of Gx. Hence, the q outputs give a linear system $Yx = v$, where Y consists of q rows in L. Moreover, the p queries to x along π define another linear system $I^{(m)}_p x = u$ where $I^{(m)}_p$ is an $p \times m$ submatrix of the $m \times m$ identity matrix. Combining the 2 linear systems, we get a new linear system $Ax = w$, where $A = \begin{pmatrix} I^{(n)}_p \\ Y \end{pmatrix}$ is an $(p + q) \times n$ matrix and $w = \begin{pmatrix} u \\ v \end{pmatrix} \in \{0,1\}^{(p+q)}$.

$I^{(m)}_p$ has exactly $m - p$ all-zeroes columns. Consider the corresponding columns in A and look at the submatrix of A consisting of these columns and the last q rows. Call this submatrix A'. A' is a submatrix of G. Since G is (h, k, γ)-rigid matrix , $(m - p) \geq (m - h)$ and $q \leq k$, then A' has rank at least γq.

Any γq linearly independent columns of A', along with the remaining p columns of $I_p^{(n)}$, form linearly independent columns of A. Hence, A has rank at least $p+\gamma q$. There are at most $2^{m-rank(A)}$ vectors $x \in \{0,1\}^n$ that are solutions to the linear system $Ax = w$. Hence, there are at most $2^{n-p-\gamma q}$ such solutions x and the probability that π q-solves a random input is at most $2^{-p-\gamma q}$. Since P has exactly 2^p distinct computations, the probability that P q-solves a random input is at most $2^{-\gamma q}$. □

Therefore, it is enough to prove that the generator matrix G of an asymptotically good binary linear code is (h,k,γ)-rigid matrix for some value of k, h and γ.

Next, we present one possible direction to establish the rigidity requirements for the generator matrix. We remove from the generator matrix any rows that have low rank. The rank of these rows will only decrease if we consider only a subset of the columns of the matrix. The next step to establish the time-space tradeoff is to lower bound the rank of any subset of the remaining rows when a subset of the columns is removed from the matrix.

7.2 Removing Low-Dimension Projections from Asymptotically Good Codes

As a first step towards establishing the rigidity of a generator matrix, we prove that an asymptotically good code contains a code C' such that for every subset $S \in [n]$, the projection $C'|_S$ has dimension at least $\gamma|S|$ for some γ that depends on the rate and relative minimum distance of the original code C. Moreover C' is asymptotically good.

Let $0 < \alpha < 1$ be any positive constant and $\epsilon > 0$ be a constant such that $\epsilon \leq \frac{\alpha m}{n}$. We can assume without loss of generality that the code has a unique subset $R \subseteq [n]$ such that $dim(C|_R) < \epsilon|R|$. Otherwise, we take the union of the different sets.

Algorithm 7.1 iteratively punctures the code at the locations where the low-dimension subset is 0 until no such subset remains. The final code C' is an asymptotically good code, with block length, dimension and minimum distance, all linear in n.

The final code C' is an asymptotically good code, with block length, dimension and minimum distance all linear in n.

Claim 7.2. *Let $n > 0$ be a positive integer and let $0 < \alpha, \delta, r < 1$ be positive. Let $C : \{0,1\}^m \rightarrow$*

Algorithm 7.1 REMOVE-LOW-DIM

1: Let $C' = C$.
2: **while** there exists a subset R of such that $dim(C'|_R) < \epsilon|R|$ **do**
3: Let $C'|_{R=0}$ be the subcode of C' such that all the bits with indices in R are 0's.
4: Puncture $C'|_{R=0}$ at the indices of R and denote this code C'.
5: **end while**
6: If there exists any remaining repetitions R in C', puncture at all the positions of R except one position.

$\{0,1\}^n$ *be linear code with* $m = nr$ *and minimum distance* $d = \delta n$ *and* $0 < \epsilon < \alpha r$ *a positive constant. Then there exists a subset* $S \subseteq [n]$ *such that the code* C_S*, resulting from projecting a subcode of* C *on the indices of* S*, has dimension* $m' \geq (1-\alpha)m$*, block length* $n' = |S|$ *and relative minimum distance* $\delta' \geq \epsilon\delta$*, no repetitions and no low dimension projections, i.e. for every subset* $R \subseteq [n']$*,* $dim((C_S)|_R) \geq \epsilon|R|$.

Proof. After each iteration of the while loop, the minimum distance of C' is maintained at d since the codewords in C' had a value of 0 at the indices punctured in line 4. Moreover, the dimension decreases by at most $\epsilon|R|$ and the block length decreases by R. Now we upper-bound the number of executions of the while loop. Let P be the union of the sets R considered. Since $|P| \leq n$, the dimension of the final code is at least $m - \epsilon|P| \geq m - (\frac{\alpha m}{n})n \geq (1-\alpha)m$. Then, the while loop will be executed at most αm times. The last step in the algorithm ensures that no repetitions of size at most $\frac{1}{\epsilon}$ remains (those are the only low dimension subsets R whose dimension is at least $\epsilon|R|$). □

Given claim 7.2 we propose the following conjecture to get our desired time-space tradeoff.

Conjecture 7.3. *Let* $n > 0$ *be a positive integer and let* $0 < \delta, r < 1$ *be positive. Let* $C : \{0,1\}^m \to \{0,1\}^n$ *be linear code with* $m = nr$ *and minimum distance* $d = \delta n$*. Moreover, assume that* C *has no repetitions and no low dimension projections. Then for any generator matrix* G *of* C*, there exists positive constants* α, β *and* γ *such that* G *is* $(\alpha m, \beta n, \gamma)$*-rigid.*

7.2.1 Application: Removing repetitions from locally-testable and self-correctable codes

We apply Algorithm 7.1 to remove repetitions from locally-testable codes.

Assume that the code C is strongly k-locally testable. Let T be its tester with parameter η. Hence, for $v \in \mathbb{F}_2^n$, T^v accepts with probability 1 if $v \in C$; otherwise it rejects with probability at least $\eta \cdot \delta(v, C)$.

Moreover, if C is k-self-correctable, we denote its corrector by SC with parameters P and β. Hence if $\delta(v, C) \leq P$, $SC^v(i)$ correctly computes c_i with probability at least $1 - \beta$, where c is the closest codeword in C to v. We modify Algorithm 7.1 as follows:

Algorithm 7.2 REMOVE-REP

1: Let $C' = C$.
2: **while** there exists a repetition R of size at least $\frac{n}{\alpha m}$ **do**
3: Let $C'|_{R=0}$ be the subcode of C' such that all the bits with indices in R are 0's.
4: Puncture $C'|_{R=0}$ at the indices of R and denote this code C'.
5: **end while**
6: If there exists any remaining repetitions R in C', puncture at all the positions of R except one position.

We run Algorithm 7.2 on C. We prove that if C is asymptotically good, then so is C'. Moreover C' is locally-testable and self-correctable.

Claim 7.4. *Let $n > 0$ be a positive integer and let $0 < \alpha, \delta, r < 1$ be positive. Let $C : \{0,1\}^m \to \{0,1\}^n$ be linear code with $m = nr$ and minimum distance $d = \delta n$ and $0 < \epsilon < \alpha r$ a positive constant. Then there exists a subset $S \subseteq [n]$ such that the code C_S, resulting from projecting C on the indices of S, has dimension $m' \geq (1 - \alpha)m$, block length $n' = |S|$ and relative minimum distance $\delta' \geq \epsilon\delta$, and no repetitions. Moreover, if C is strongly k-locally-testable and k-self-correctable, then so is C'.*

Proof. With a similar analysis as for Claim 7.2, C' is an asymptotically good code. Let $n' = \Theta(n)$ denote its block length and $S \in [n]$ be such that $C_S = C'$.

Let $v' \in \mathbb{F}_2^{n'}$ and let $v \in \mathbb{F}_2^n$ such that: $v_S = v'$, $v_i = 0$ if i was punctured in step 4 of the algorithm, and $v_i = v'_j$ if i was punctured in the last step and $j \in [n']$ is the remaining position from the corresponding punctured repetition.

First we prove that C' is strongly k-locally-testable. We construct a tester T' for C', with parameter η'. $T'^{v'}$ runs T^v. When T queries v at position $i \in [n]$, T' queries the corresponding

position in v', as per the construction of v above. If $v' \in C'$, $v \in C$, T will accept and so will T'. If v' is at distance $\delta(v', C')$ of C', then v is at distance $\delta(v, C) \geq \frac{\delta(v', C')n'}{n}$. T^v rejects with probability at least $\epsilon\delta(v, C) \geq \frac{\epsilon\delta(v', C')n'}{n}$. Therefore $T'^{v'}$ rejects with probability at least $\epsilon'\delta(v', C')$, where $\epsilon' = \frac{\epsilon\delta' n'}{n}$. Hence C' is strongly k-locally-testable since $n' \geq (1 - \alpha)m$.

Now, we construct a corrector SC' for C, with parameters τ' and β'. For $\ell \in [n']$, $SC'^{v'}(\ell)$ runs $SC^v(j)$ where $j \in [n]$ is the position of symbol v'_ℓ in v. When SC queries v at position $i \in [n]$, SC' queries the corresponding position in v', as per the construction of v above. If $v' \in C'$, $v \in C$, SC will correctly produce $v_j = v'_\ell$ and so will SC'. If v' is at distance $\delta(v', C') \leq \tau'$ of C', the only errors in v will occur from repeating the error bits of v' at the positions punctured in the last step of Algorithm 7.2. Since these repetitions have size at most $\frac{\alpha n}{r}$, then v is at distance $\delta(v, C) \leq \frac{\tau' n'}{\alpha m}$. For SC^v to correctly compute c_j, $\delta(v, C)$ should be at most τ. Hence setting $\tau' = \frac{\alpha m \tau}{n}$ satisfies this condition. Also, note that $c_j = c'_\ell$, where c' is the closest codeword in C' to v'. Therefore $SC'^{v'}(\ell)$ computes c'_ℓ with probability $1 - \beta$ and $\beta' = \beta$ and C' is k-self-correctable.

□

Remark The motivation behind removing repetitions from locally-testable and self-correctable codes is that assuming that such codes have no repetitions leads to simpler arguments [41]. Note that in our analysis, it is essential that the original code is asymptotically good for the final code to be locally-testable and self-correctable. It remains an open problem whether we can construct a locally-testable or self-correctable code from the original code when its rate goes to 0 as the block length goes to infinity.

7.3 Comparison to Subsequent Work

Recently, Gál, Hansen, Koucký, Pudlák and Viola [69] gave lower bounds on the number of wires in a circuit encoding asymptotically good codes. They derive superlinear lower bounds for constant depth and give non-explicit constructions that satisfy these bounds. Moreover, they give a linear upper bound on the number of wires when the depth is slowly growing. Namely, they prove the following.

Theorem 7.5. *[69] For any positive constants ρ and δ, the number of wires in circuits of depth d*

with arbitrary gates computing (possibly non-linear) codes of rate ρ and relative minimum distance δ is lower bounded by a function $f_d(n)$ given by the following:

1. *if $d = 2$, $f_2(n) \in \Omega(n(\log n/\log\log n)^2)$.*

2. *if $d = 3$, $f_3(n) \in \Omega(n \log\log n)$.*

3. *if $d = 2k$ or $d = 2k+1$, for some integer k, then $f_d(n) \in \Omega(n\lambda_k(n))$.*

Moreover, these lower bounds are achieved by non-explicit circuits using only XOR gates encoding linear asymptotically good codes.

Although the authors use a different model of computation, they derive interesting properties of generator matrices of codes that are relevant to time-space tradeoffs for the branching program model. They prove that the underlying graph in a circuit computing an error correcting code of relative minimum distance $\delta > 0$ satisfies the following property: For every susbset X of the input nodes and a random subset Y of the output nodes such that $|X| = |Y|$, the expected number of vertex-disjoint paths between X and Y is at least $\delta|X|$. One important property of the generator matrix of a code is that the number of vertex-disjoint paths between X and Y is simply the rank of the submatrix $G_{X\times Y}$ of the generator matrix G that the circuit is computing. This property follows from Menger's theorem (one of its versions is stated in [69, Theorem 7]). Therefore, one concludes the following statement: if G is a generator matrix of a linear code $C \in \{0,1\}^n$, then the expected rank of the submatrix $G_{X\times Y}$ is at least $\delta|X|$, where X is an arbitrary subset of the message bits and Y is chosen uniformly at random from the output bits such $|X| = |Y|$.

The statement above does not prove the rigidity of the generator matrix as outlined in our approach since we want the statement to hold for every Y with no restriction on its size. But we can formulate our condition in terms of the property of the underlying graph of any circuit computing G: There exists a subset L in the output nodes (of size linear in the block length) such that for every subset $Y \in L$ and every subset X of the input nodes of size αm, the number of vertex-disjoint paths between X and Y is at least $\delta|Y|$, for some constants $\alpha, \delta > 0$. Finding such a subset L would prove Conjecture 7.3.

Part III

QUANTUM COMPUTING

Chapter 8

QUANTUM LOWER BOUNDS

Quantum computers are models of computation that use quantum mechanical effects exhibited by nature. Richard Feynman [67] first introduced them as universal machines capable of simulating physics and hypothesized that quantum computers are able to perform computational tasks that their classical counterparts are not able to do. The physics and engineering communities have been trying to construct scalable quantum computers. Meanwhile, computer scientists have been studying the computational limits and strengths of quantum computation. A quantum Turing machine was defined [60, 39, 139] that creates a *superposition of inputs* (as vectors) and at each time step, it is in a *superposition of states*, each representing a computation on a specific input. A series of quantum algorithms that exhibited speed-ups over their classical (even probabilistic) counterparts were constructed. One of the first notable ones was Simon's algorithm [139]. Simon gave an expected polynomial time quantum algorithm that finds the *xor-mask* of a function f, which is a string s such that for all $x, y \in \{0,1\}^n$ inputs to f, $f(x) = f(y)$ if and only if $x = y \oplus s$. Moreover, Simon showed that any probabilistic classical algorithm that obtains the evaluation of f on inputs through an oracle, requires more than $2^{n/4}$ queries to the oracle to determine whether s is equal to 0 or not, with probability at least $1/2 + 2^{-n/2}$. In a breakthrough, Shor [138] gave a quantum algorithm for integer factorization and discrete logarithm, two problems to which no efficient classical algorithms currently exist, by using Simon's period finding algorithm and a host of clever new ideas.

A careful examination of Simon's and Shor's algorithms shows that a main theme in the quantum computation is to construct a quantum operation that is a function of the input only, then to apply it to a superposition of states constructed by the algorithm. The remaining operations are independent of the input and applied to the superposition of states at that time step. Motivated by that fact, Beals, Buhrman, Cleve, Mosca and de Wolf [27] formally defined the *quantum query model*, where a quantum algorithm computes a function by making queries to its input which is provided as a

quantum oracle. The *quantum query complexity* of the function is the maximum number of queries a quantum algorithm needs to make to its input in order to evaluate the function (either with certainty or with high probability). In a classical setting, the number of queries required for a problem is the depth of a decision tree computing that problem. One well-known example where quantum computers have an advantage over their classical counterparts in terms of queries to the input is search. Grover [77] gave a quantum algorithm that finds a position with value 1 in a n-bit string using only $O(\sqrt{n})$ queries to the input, while any classical deterministic algorithm requires n queries to the input. It was subsequently shown that this upper bound is tight [38, 12].

It is believed that the power of quantum computers comes from their ability to simulate computations on a superposition of inputs and use a structure in the problem to carry those computations coherently. Such a structure can be reflected in restricting the inputs to a subset of the input space that satisfy a certain property. Those problems are called *promise* problems and the function computed is called a *partial* function: the algorithm is required to give a correct answer only when the input satisfies the property promised. Many speed-ups were attained for partial functions over classical computers. Ambainis [15] gave an $O(n^{2/3})$-query quantum algorithm for element distinctness and, Brassard, Høyer and Tapp [47] gave an $O(n^{1/3})$-query quantum algorithm for the collision problem. Starting with Grover's $O(\sqrt{n})$ quantum query algorithm for computing the OR of n bits, and continuing through recent work on read-once formula evaluation [65, 17, 128, 16], quantum query algorithms have been shown to provide polynomial speed-ups over classical query algorithms for computing many total functions. On the other hand, some functions proved to be hard for quantum computers, e.g. the parity function which requires $\frac{n}{2}$ queries to the input [27].

In this chapter, we consider problems where a certain structure is promised in the input and prove that, even when these problems are easy on some classical computational models, a quantum algorithm computing such a problem in the quantum query model requires a linear number of queries to the input. The problem we consider is deciding whether a function provided as input is surjective. It can be computed by a constant-depth polynomial-size circuit. Our lower bound on the quantum query complexity generalizes to a lower bound on the formula size of the problem we consider, improving almost quadratically on the previously known lower bound for that problem.

We also study the quantum query complexity of functions on sliding windows and consider the

problem of computing the parity of the number of distinct elements in a string of length n. This problem is a special case of the problem we consider above. We show that quantum algorithms have an advantage over classical ones for sliding-window functions. We do so by constructing an $O(n^{3/2})$-time, $O(\log n)$-space quantum algorithm that computes $(F_0 \bmod 2)^{\boxplus n}$, thus beating the $TS \in \Omega(n^2)$ classical time-space tradeoff we derived for the same function in Chapter 4.

8.1 The Quantum Query Model

A quantum algorithm works by applying a series of unitary operators on a quantum state. Mathematically, a pure quantum state $|\psi\rangle$ is a d-dimensional unit vector from the complex space $\mathbb{C}^d$, associated with the standard inner product. A quantum operator is a linear operator $U : \mathbb{C}^d \to \mathbb{C}^d$ satisfying $|det(U)| = 1$ and $UU^\dagger = I$ where I is the identity operator and $U^\dagger$ is the Hermitian conjugate of U.

Let D and R be two finite sets and let n be positive. In the quantum query model, we wish to compute a function $f : S \to R$ where S is a subset of a space D^n, and we access its input x through queries to each of its n positions. The memory of a quantum algorithm can be represented by 3 quantum registers:

1. the input register, which holds the input x,
2. the query register, which holds an integer $i \in [n]$, and
3. the working memory register, which holds a state $|j, z\rangle$, where $j \in D$ and $|z\rangle$ is arbitrary.

Hence, the state of the quantum algorithm is described as $|i, j, z\rangle$. The algorithm computes the function f by applying a series of unitary operations from two different types. The first type is the oracle operator O_x, which maps the state $|i, j, z\rangle$ to the state $e^{\frac{2\pi i}{|D|} j x_i}|i, j, z\rangle$, thus allowing the algorithm to query the input x at position $i \in [n]$[1]. Note that the input and the state are left unchanged by this operator. The second type of operator consists of arbitrary fixed operators U_t that are independent of the input and act only on the states $|i\rangle$ and $|z\rangle$. We define the quantum query algorithm as a sequence of these operations.

[1] A variation of the oracle operator stores x_i in the state $|i, (j + x_i) \bmod |D|, z\rangle$. The results holds for both models as they are equivalent.

DEFINITION 8.1. *A T-query quantum algorithm having oracle access to an input x is represented as the sequence $A = U_T O_x U_{T-1} \cdots U_1 O_x U_0$ applied to the initial state $|\phi_x^0\rangle|0\rangle$. The output of the algorithm is chosen according to a probability distribution resulting from measuring the final state $|\phi_x^{2T}\rangle$. A* measurement *(in the computational basis) is a set of projectors Π_b, $b \in R$ such that the algorithm outputs b with probability $||\Pi_b|\phi_x^{2T}\rangle||^2$.*

The complexity measure we are interested in the number of queries needed to compute a function f on any input x.

DEFINITION 8.2. *The ϵ-error quantum query complexity of $f : S \to R$ is the minimum number of queries made by a quantum algorithm (having oracle access to its input) to compute $f(x)$ with probability at least $1 - \epsilon$ for all x.*

8.2 AC^0

As an alternative to Turing machines, one might consider logic circuits as a computational model for deciding languages and computing functions. Circuits are a *nonuniform* model of computation, in the sense that different circuits are needed for different input lengths, as opposed to a Turing machine where the same machine can be used to compute a function on all input lengths. Shannon [136] was the first to propose circuit complexity as a measure of the hardness of a function. In circuits, the relevant complexity measures are the number of gates needed to construct the circuit and the number of steps needed to evaluate a circuit such that gate evaluations occur in parallel whenever their inputs are available.

DEFINITION 8.3. *A boolean circuit with n inputs and m outputs over basis $\{AND, OR, NOT\}$ is a directed acyclic graph with n input nodes (in-degree 0) and m output nodes (out-degree 0) and intermediary nodes, called* gates*, labeled with AND, OR or NOT. The in-degree of the gates is called* fan-in*. The circuit computes the m outputs by evaluating the gates from the input nodes to the output nodes. The* depth *of the circuit is the length of the longest path between an input node and an output node. The* size *of the circuit is the number of gates in the graph.*

Circuit lower bounds have been used to derive lower bounds on the Turing machine complexity of functions (namely the running time). The connection between Turing machine complexity and

circuit complexity have been studied in [133, 124]: any Turing machine running in time $O(T(n))$ can be simulated by a circuit of size $O(T(n)\log_n)$, where n is the input size. Unfortunately, circuit lower bounds proved very hard to obtain, hence complexity theorists defined restricted classes of circuits in the hope that more structure in circuits would lead to stronger lower bounds. One of these classes is AC^0.

DEFINITION 8.4. AC^0 *is the class of circuits with OR, AND, and NOT gates of unbounded fan-in, constant depth and size polynomial in the input length.*

AC^0 describes a natural class of circuits of constant depth that are used in digital circuit design applications. A depth-2 AC^0 circuit is simply a DNF or CNF formula. Such formulas are essential for circuit minimization heuristics in engineering applications. Moreover, the constant depth and polynomial size make AC^0 circuits an efficient computational model for implementing boolean functions on physical circuit boards.

The computational power of AC^0 circuits (and variations of them) has been extensively studied. One of the most notable results limiting their power is Furst, Saxe and Sipser's [68] and Ajtai's [6] result showing that computing the parity of an n-bit binary string requires an exponential size circuit and hence PARITY is not in AC^0. In this thesis, we consider problems that have AC^0 circuits and prove that these problems require almost linear query complexity on quantum computers, thus proving that AC^0 is hard class of functions for quantum computers.

8.3 Previous Work on Quantum Query Complexity

The quantum query complexity of a number of problems in AC^0 has been thoroughly investigated. The first quantum query lower bounds in [38, 27] showed that Grover's algorithm for the OR is asymptotically optimal using a method called the *polynomial method*. In [12], Ambainis introduced the adversary method to prove an $\Omega(\sqrt{n})$ lower bound on the query complexity of an AND of ORs. Motivated in part by questions of the security of classical cryptographic constructions with respect to quantum adversaries, the *collision problem* of determining whether a function is 1-to-1 or 2-to-1 on its domain has been one of the most celebrated examples in quantum query complexity. Aaronson and Shi [1] proved an $\Omega(n^{1/3})$ lower bound on the query complexity of the collision problem with range $3n/2$, which yields an $\Omega(n^{2/3})$ lower bound on the element distinctness problem with range

$\Theta(n^2)$. The element distinctness problem is equivalent to the general case of testing whether a function is 1-to-1 (without the promise of being 2-to-1 in case that it is not 1-to-1). For the r-to-1 versus 1-to-1 problem, for integer $r \geq 2$, Aaronson and Shi also proved a lower bound of $\Omega((n/r)^{1/3})$ when the range size is $3n/2$. All of these problems can be computed in AC^0 and these query complexities are asymptotically optimal by results of Brassard et al. [47] and Ambainis [15]. Ambainis also was able to reduce the range size requirement for both lower bounds [13], as was Kutin [104] for the collision problem, independently, by different means.

Three main techniques have been developed to derive lower bounds on quantum query complexity. The first is the polynomial method of Beals et al. [27]; the second is the adversary method introduced by Ambainis [12, 141]. The polynomial method uses the fact that the probability of a quantum algorithm succeeding is an approximating polynomial for the function that it is computing. On the other hand, Ambainis' adversary method is based on tracking the state of the quantum algorithm on "difficult" inputs. In a recent body of work [86, 129, 130, 109], a third method has been developed as an extension of the adversary method; this method characterizes the query complexity of functions. This can be formulated either as a version of the adversary method that includes negative weights [86] (adding positive weights does not improve Ambainis' original adversary method), as a certain measure on span programs computing the function [129], or as the optimum of a semi-definite program maximizing the spectral norm of the various *adversary matrices* of the function [130, 109]. An adversary matrix of a function is a real symmetric matrix indexed by pairs of inputs such that if the function evaluates to the same value on a pair of inputs, then the corresponding entry in the adversary matrix is set to 0. Our proof uses the original adversary method [12] and does not require the machinery developed in the subsequent work.

The methods above have been used to show that some fairly simple functions do not benefit from significant improvement in query complexity using quantum queries. For example, since the parity of n bits requires degree $\Omega(n)$ to be approximated by a multivariate polynomial its quantum query complexity also must be linear. However, parity is still a somewhat complex function since it cannot even be approximated in AC^0, the class of decision problems solvable by Boolean circuits with unbounded fan-in, constant depth, and polynomial size. Can the simple functions in AC^0 always be computed using $O(n^\beta)$ queries for some $\beta < 1$? We show that this is not the case.

8.4 Contributions

We show that any quantum algorithm deciding whether an input function f from $[n]$ to $[n]$ is 2-to-1 or almost 2-to-1 requires $\Theta(n)$ queries to f. The same lower bound holds for determining whether or not a function f from $[2n-2]$ to $[n]$ is surjective. These results yield a nearly linear $\Omega(n/\log n)$ lower bound on the quantum query complexity of AC^0. The best previous lower bound known for any AC^0 function was the $\Omega((n/\log n)^{2/3})$ bound given by Aaronson and Shi's $\Omega(n^{2/3})$ lower bound for the element distinctness problem [1].

In the original collision problem, the main question was whether or not the input function is 1-to-1, where at most $O(n^{1/3})$ queries suffice for the promise versions of the problem and at most $O(n^{2/3})$ queries suffice even in the general case. We show that if one is concerned with whether or not a function is precisely 2-to-1, the number of queries required is substantially larger.

More precisely, we show an asymptotically tight linear lower bound for determining whether or not a function is either precisely 2-to-1 or *almost 2-to-1* in that the function is 2-to-1 except for exactly two inputs that are mapped 1-to-1. This also implies an $\Omega(n/\log n)$ lower bound on the Boolean quantum query complexity of AC^0, thus substantially improving the previous best lower bound of $\Omega((n/\log n)^{2/3})$ given by the results of Aaronson and Shi.

Using the original adversary method developed by Ambainis [12] we prove the following theorem:

Theorem 8.1. *Let n be a positive even integer, $\mathcal{F}$ be the set of functions f from $[n]$ to $[n]$ that are either 2-to-1 or almost 2-to-1 and $\phi : \mathcal{F} \to \{0,1\}$ be a Boolean function on $\mathcal{F}$ such that $\phi(f) = 1$ iff f is 2-to-1. Then any quantum algorithm deciding ϕ requires $\Omega(n)$ queries.*

The problem of determining whether a function is a 2-to-1 or almost 2-to-1 function is a special case of determining whether or not a function from $[2n-2]$ to $[n]$ is surjective. Therefore, we obtain the following lower bound.

Corollary 8.2. *Let n be a positive even integer and $\mathcal{G}$ be the set of functions from $[2n-2]$ to $[n]$. Define the function* ONTO $: \mathcal{G} \to \{0,1\}$ *by* ONTO$(f) = 1$ *if and only if f is surjective. Then any quantum algorithm computing* ONTO *with probability at least $2/3$ requires $\Omega(n)$ queries.*

We also obtain a lower bound on the quantum query complexity of $F_0 \bmod 2$ since under the promise that the function is either 2-to-1 or almost 2-to-1, determining which type it is reduces to finding the parity of the number of distinct elements in its range.

Corollary 8.3. *Any quantum algorithm computing* $F_0 \bmod 2$ *with probability at least* $2/3$ *requires* $\Omega(n)$ *queries.*

Since each value $f(i)$ can be encoded using $\log_2 n$ bits, the entire input is $n \log_2 n$ bits long. It is easy to see that determining whether such a function given by these bits is 2-to-1 or not is a simple AC^0 problem. Since each query of one of the bits of $f(i)$ is weaker than querying all of $f(i)$, we immediately obtain lower bounds on the query complexity of AC^0.

Corollary 8.4. *The quantum query complexity of* AC^0 *is* $\Omega(n/\log n)$.

We also derive a $\Omega(\frac{n^2}{\log n})$ lower bound on the formula size deciding the same problem using the relation between quantum query complexity and formula size lower bound [106], thus improving the previously known lower bound using Khrapchenko's method [95] almost quadratically.

Moreover, we study a special case of our problem, computing the parity of F_0 for a n-symbol string chosen from $[n]$, in the sliding window model and give an $O(n^{3/2})$-time, logarithmic space quantum algorithm for it. In the light of the classical time-space tradeoffs derived for the parity of F_0 in Chapter 4, this proves quantum algorithms have an advantage over classic ones for sliding-window functions.

8.5 The Quantum Adversary Method

We will use the adversary method as developed by Ambainis [12] to derive lower bounds on the quantum query complexity of Boolean functions. Below, we describe the motivation behind the method and give a sketch of the proof after stating it. Since the proof of the method will not be used in the thesis, this is meant to give a high-level explanation of the method. For a complete analysis, the reader is referred to [12].

8.5.1 Motivation

Consider a quantum algorithm computing a function ϕ with probability at least $1-\epsilon$, for some $\epsilon > 0$. We use the symbol f to denote the input to ϕ. The main motivation behind the method is to assume that an adversary has control over the oracle and holds a quantum state that is in a superposition of some subset S of the inputs $\sum_{f \in S} \alpha_f |f\rangle$.

The algorithm starts in a state that is independent of the input (typically the $|0\rangle$ state). At the beginning of the algorithm, the oracle part and the algorithm part of the resulting state $|\psi_{start}\rangle = \sum_{f \in S} \alpha_f |f\rangle|0\rangle$ are not *entangled*. Entanglement is a quantum notion that measure how dependent two states are: knowing the outcome of a measurement on one state completely determines the outcome of the second state.

As the computation progresses, the algorithm applies unitary transformations corresponding to the U operators in Definition 8.1. These operators only modify the algorithm part of the state. The oracle applies query transformations corresponding to the O operators in Definition 8.1. These operators entangle the algorithm part with the oracle part of the state. The algorithm succeeds if and only if the algorithm part and oracle part of the final state $|\psi_{end}\rangle = \sum_{f \in S} \alpha_f |f\rangle|\psi_f\rangle$ become highly entangled: any intermediary state in the computation holds different answers for different inputs; the high entanglement requirement of the final state guarantees that an a measurement $|\psi_f\rangle$ produces the answer $\phi(f)$ that corresponds to $|f\rangle$, with probability at least $1 - \epsilon$.

Hence, by showing that each query transformation cannot increase the entanglement of the state by a large amount, Ambainis [12] obtains a lower bound on the number of queries needed by the algorithm. Namely, the high entanglement of the final state requires that for $f, g \in S$ such that $\phi(f) \neq \phi(g)$, the inner product (where each state is viewed as a vector in some basis) $\langle \psi_f | \psi_g \rangle$ is at most $2\sqrt{\epsilon(1-\epsilon)}$. Therefore, by proving that after each query, the entanglement between the two parts of the state does not increase by a large amount, we obtain a lower bound on the number of queries of the algorithm.

To apply the quantum adversary method, one needs to choose adversarially a subset S of the inputs that evaluate differently on ϕ, such that it is hard for the algorithm to differentiate between them. The α's in the superposition of the inputs in S are typically chosen to be uniform over each subset of S that evaluate to a specific value of ϕ.

Theorem 8.5 (Ambainis [12]). *Let $\mathcal{F}$ be the set of functions f from $[n]$ to $[n']$ and $\phi : \mathcal{F} \to \{0,1\}$ be a Boolean function. Let A and B be two subsets of $\mathcal{F}$ such that $\phi(f) \neq \phi(g)$ if $f \in A$ and $g \in B$. If there exists a relation $R \subset A \times B$ such that:*

(a) For every $f \in A$, there exist at least m different functions $g \in B$ such that $(f,g) \in R$.

(b) For every $g \in B$, there exist at least m' different functions $f \in A$ such that $(f,g) \in R$.

(c) For every $x \in [n]$ and $f \in A$, there exist at most l different functions $g \in B$ such that $(f,g) \in R$ and $f(x) \neq g(x)$.

(d) For every $x \in [n]$ and $g \in B$, there exist at most l' different functions $f \in A$ such that $(f,g) \in R$ and $f(x) \neq g(x)$.

Then any quantum algorithm computing ϕ with probability at least $1-\epsilon$ requires $\Omega\left(\sqrt{\frac{mm'}{ll'}}\right)$ queries, for $0 < \epsilon < 1/2$.

Sketch of the proof of the adversary method. The intuition behind the proof is that the algorithm needs to differentiate between at least $\max(|A|m, |B|m')$ pairs of functions in R by the end. However, after each query to an $x \in [n]$, the algorithm learns only about functions that differ on x, of which there are not many for such functions f and g.

The adversarial subset is $S = A \cup B$ and the input superposition $\frac{1}{\sqrt{2|A|}} \sum_{f \in A} |f\rangle + \frac{1}{\sqrt{2|B|}} \sum_{g \in B} |g\rangle$. A progress function W_t is used to measure entanglement after each query: $W_t = \sum_{(f,g) \in R} |\alpha_f^* \alpha_g| \langle \psi_f^{2t} | \psi_g^{2t} \rangle$. Therefore, $\left(W_0 - W^T\right) \geq (1 - 2\sqrt{\epsilon(1-\epsilon)})\sqrt{mm'}$.

To upper bound $(W_t - W_{t+1})$, notice that the query to x at time t changes the sum in W_t only for $(f,g) \in R$ such that $f(x) \neq f(g)$. Moreover, since the range of ϕ is of size 2, the query oracle just multiplies by -1 the sum terms corresponding to $(f,g) \in R$ such that $f(x) \neq f(g)$. Therefore, $W_t - W_{t+1}$ can be upper bounded by $\sum_{(f,g) \in R, f(x) \neq g(x)} 2|\alpha_f^* \alpha_g|$. Using Cauchy-Schwartz's inequality, this sum is upper bounded by $\sqrt{ll'}$. □

8.6 Quantum Query Lower Bounds for the 2-TO-1-VS-ALMOST-2-TO-1 *Function*

Let n be a positive even integer and $f : [n] \to [n']$ for $n' > n/2$ be a function. We say that f is *2-to-1* if every point in the image either has exactly two pre-images or has no pre-images in $[n]$. We

say f is *almost 2-to-1* if f is 2-to-1 on $n-2$ points in the domain and 1-to-1 on the remaining 2 points, and every point has at most two pre-images.

We consider the following problem:

2-TO-1-VS-ALMOST-2-TO-1 Problem: Given a positive even integer n, integer $n' \geq n/2+1$, and a function $f : [n] \to [n']$ that is either 2-to-1 or almost 2-to-1, determine whether f is 2-to-1.

Let $\mathcal{F}$ be the set of functions from $[n]$ to $[n']$ that are either 2-to-1 or almost 2-to-1 and let $\phi : \mathcal{F} \to \{0,1\}$ be a Boolean function such that $\phi(f) = 1$ if and only if f is 2-to-1. We prove, using the adversary method [12], that the quantum query complexity of ϕ is $\Omega(n)$.

Theorem 8.1. *Let n be a positive even integer, $n' \geq n/2+1$, and $\mathcal{F}$ be the set of functions f from $[n]$ to $[n']$ that are either 2-to-1 or almost 2-to-1. Let $\phi : \mathcal{F} \to \{0,1\}$ be a Boolean function such that $\phi(f) = 1$ iff f is 2-to-1. Then any quantum algorithm computing ϕ with probability at least $2/3$ requires $\Omega(n)$ queries.*

Proof. We use the adversary method. Let A be the set of functions in $\mathcal{F}$ that are 2-to-1 and B be the set of functions that are almost 2-to-1. Obviously, $\phi(A) = 1$ and $\phi(B) = 0$. We denote the distance between two functions f and g by

$$d(f,g) = |\{i \in [n] | f(i) \neq g(i), f, g \in \mathcal{F}\}|.$$

For a function $g \in B$, we denote the two input points in $[n]$ that g maps injectively by $s(g) = \{s_1, s_2\}$ where $s_1 < s_2$. That is, $g(s_1) \neq g(s_2)$ and $g(i) \notin \{g(s_1), g(s_2)\}$ for all $i \notin s(g)$. Consider the following relation $R \subseteq A \times B$:

$$R = \{(f,g), f \in A, g \in B, d(f,g) = 2, f(s_1) = g(s_1) \text{ and } f(s_2) = g(s_2)\}.$$

In other words, the relation consists of pairs of functions in $A \times B$ that differ on exactly 2 points, neither of which is one of the injectively mapped points of the function in B.

Claim 8.6. *If $(f,g) \in R$ and $x, y \notin s(g) = \{s_1, s_2\}$ where $x \neq y$ are the points where f and g differ then $\{f(x), f(y)\} = \{g(s_1), g(s_2)\}$ and $g(x) = g(y) = z$, where $z \in n'$ is not in the range of f.*

Since $f(s_1) = g(s_1) \neq g(s_2) = f(s_2)$, and $g(i) \notin \{g(s_1), g(s_2)\}$ for every $i \notin s(g)$, we must have $\{f(x), f(y)\} = \{g(s_1), g(s_2)\}$, otherwise f will not pair any input with s_1 or s_2 and hence not be 2-to-1. Since $x, y \notin s(g)$, it must be that x and y are paired inputs of g. However, since f is 2-to-1 and pairs up the four points $\{x, y, s_1, s_2\}$ all other points must be paired with each other by both f and g since g agrees with f on these other points. Hence the only points to which x and y can be paired by g are each other, and therefore $g(x) = g(y) = z$. z cannot be in the range of f since f and g differ on exactly 2 points. The claim follows.

We now check the four properties of R:

1. Let f be a 2-to-1 function in A. Choose any two points $x, y \in [n]$ such that $f(x) \neq f(y)$. For each of the $n' - n/2$ points $z \in [n']$ not in the image of f we can define an almost 2-to-1 function $g \in B$ such that $(f, g) \in R$ by having g agree with f on all but inputs $\{x, y\}$ and setting $g(x) = g(y) = z$. There are $n(n-2)/2$ choices of the pair of x and y and $n' - n/2$ choices of z, each of which produces a distinct g, for a total of $m = n(n-2)(2n' - n)/4$ distinct $g \in B$ with $(f, g) \in R$.

2. Let g be an almost 2-to-1 function in B. Choose some ordered pair of inputs $(x, y) \notin s(g)$ such that $g(s_1) \neq g(x) = g(y) \neq g(s_2)$. Define a 2-to-1 function $f \in A$ such that $(f, g) \in R$ as follows: Let f agree with g on all inputs outside $\{x, y\}$. Define $f(x) = g(s_1)$ and $f(y) = g(s_2)$. There are $m' = n - 2$ choices of the ordered pair (x, y) since there $n/2 - 1$ pairs of matched inputs for g and 2 ways to order each pair.

3. Fix a function $f \in A$ and a point $x \in [n]$. Let $g \in B$ be such that $(f, g) \in R$ and $f(x) \neq g(x)$. Then f and g should differ an another point $y \in [n]$. By the claim, $f(x) \neq f(y)$ and $g(x) = g(y)$. There are precisely $n - 2$ choices of y. For each choice of y, the choice of g is determined by the value $z = g(x) = g(y)$ which must not be in the image of f. (Indeed each such choice yields such a valid g.) Hence there are precisely $n' - n/2$ choices of z. So, in total there are precisely $l = (n-2)(2n' - n)/2$ choices of g with $(f, g) \in R$ and $f(x) \neq g(x)$.

4. Fix a function $g \in B$ and a point $x \in [n]$. Let $f \in A$ be such that $(f, g) \in R$ and $f(x) \neq g(x)$. Then f and g should differ on another point $y \in [n]$. (We must have $x \notin s(g)$, otherwise

there is no such f.) By the claim we must have $g(x) = g(y)$. Therefore there is only one such choice for y. Moreover by the claim we must have $\{f(x), f(y)\} = \{f(s_1), f(s_2)\}$ which yields precisely $l' = 2$ such functions f.

It follows that $\sqrt{\frac{m \cdot m'}{l \cdot l'}} = \sqrt{n(n-2)}/2$. Therefore the quantum query complexity of ϕ is $\Omega(n)$. □

8.7 A Near Optimal Query Lower Bound for AC^0

To establish the lower bound on the query complexity of AC^0, we apply Theorem 8.1 to a range of size $n' = n/2 + 1$. In that case, observe that any almost 2-to-1 function is surjective but any 2-to-1 function is not. Therefore since $n = 2n' - 2$ we immediately obtain the following corollary.

Corollary 8.2. *Let n be a positive even integer and $\mathcal{G}$ be the set of functions from $[2n-2]$ to $[n]$. Define the function* ONTO $: \mathcal{G} \to \{0,1\}$ *by* ONTO$(f) = 1$ *if and only if f is surjective. Then any quantum algorithm computing* ONTO *with probability at least $2/3$ requires $\Omega(n)$ queries.*

Viewing the function f as a string of length n, the number of distinct elements in that string is n or $n-1$, depending whether f is surjective or not. Hence, we get the following corollary for the parity of F_0.

Corollary 8.3. *Any quantum algorithm computing $F_0 \bmod 2$ with probability at least $2/3$ requires $\Omega(n)$ queries.*

To get the bound on the quantum query complexity of AC^0, we construct a polynomial-size constant-depth circuit for ONTO.

Corollary 8.4. AC^0 *has worst-case quantum query complexity $\Omega(N/\log N)$ on N-bit functions.*

Proof. The lower bound in Corollary 8.2 is an asymptotically optimal $\Omega(n)$ for ONTO function. We show how it can be computed in AC^0:

We represent the input $f : [n] \to [n]$ by $N = n \log_2 n$ bits, $f_{i\ell}$ where $f_{i\ell}$ represents the ℓ-th bit

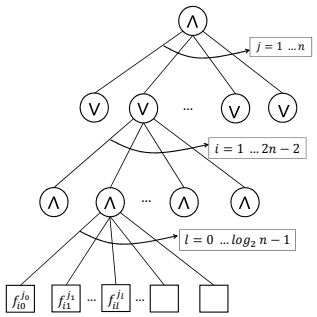

Figure 8.1: AC^0 circuit computing the ONTO function. The input function the input is represented by $n \log_2 n$ bits, $f_{i\ell}$ where $f_{i\ell}$ represents the ℓ-th bit of $f(i)$. $f_{i\ell}^{j_\ell}$ is $f_{i\ell}$ if $j_\ell = 1$ and $\neg f_{i\ell}$ if $j_\ell = 0$.

of $f(i)$. The following is an explicit polynomial-size constant-depth formula that computes ONTO:

$$\bigwedge_{j\in[n]} \bigvee_{i\in[2n-2]} \bigwedge_{\ell=0}^{\log_2 n-1} f_{i\ell}^{j_\ell} \ ,$$

where j_ℓ is the ℓ-th bit of the binary encoding of j, and x^{j_ℓ} is x if $j_\ell = 1$ and $\neg x$ if $j_\ell = 0$. The inner AND of the formula computes whether f_i is equal to j. Each OR on level 2 corresponds to a $j \in [n]$ computes whether there exists $i \in [2n-2]$ such that $f_i = j$. The top AND computes whether each j in the range of f has a pre-image i, i.e $f_i = j$. This formula is only depth 3 and size $O(n^2 \log n)$, as illustrated in Figure 8.1.

The lower bound of $\Omega(n)$ from Corollary 8.2 is $\Omega(N/\log N)$ as required.

It is interesting to note the importance of the domain size in this problem. Testing surjectivity for functions from $[n]$ to $[n]$ is equivalent to testing whether two elements from the domain are mapped to the same point in the range and thus has query complexity $\Theta(n^{2/3})$, given by the element distinctness problem [1]. □

8.8 Formula Size Lower Bound for the ONTO *Function*

A *boolean formula* is a boolean circuit whose underlying graph is a tree. The *size* of the formula is the number of leaves in the graph and for a boolean function f, the *formula size* is the smallest size of a formula computing f. The main difference between a formula and a circuit is that the fan-out

of a gate is restricted to be 1 for a formula, while it can be unbounded for a circuit. This structure allows to get larger lower bounds on the size of formulae as opposed to circuit lower bounds.

We derive a lower bound on the formula size on the ONTO function by applying the relation between the adversary method bound and formula size established in [106]. The authors proved that the bound derived from Ambainis' adversary method is a lower bound on the square root on the size of a formula computing the corresponding function. Moreover, they generalize the lower bound to randomized formulae approximating the function. A *ϵ-error randomized formula* is defined, similarly to a randomized branching program, as a distribution over deterministic formulae, such that any formula chosen according to that distribution is correct with probability $1 - \epsilon$, for every input x. In particular, they prove the following:

Theorem 8.7. *[106] Let $S \in \{0,1\}^n$ and $f : S \to \{0,1\}$. If $L^\epsilon(f)$ is the randomized formula size of f and Adv(f) is the bound derived using the basic adversary method, then $L^\epsilon(f) \geq (1 - \epsilon)Adv(f)$, for any $\epsilon < 1/2$.*

Using Corollary 8.2, we get the following corollary.

Corollary 8.8. *The ϵ-error randomized formula size of* ONTO *is in $\Omega\left((1-\epsilon)\frac{n^2}{\log^2 n}\right)$, for any $0 < \epsilon < 1/2$.*

The best bound one gets from previously known techniques for deriving lower bounds is $\Omega(n)$ by using Khrapshenko's method [95]. It is also important that having a range size almost half the domain size is essential to get such a large lower bound compared to Khrapshenko's method.

8.9 Parity of Sliding-Window F_0

The 2-TO-1-VS-ALMOST-2-TO-1 problem can be viewed as a special case of the $F_0 \bmod 2$ problem we study in Chapter 4: if the function f is 2-to-1, then the number of distinct elements in the input is $\frac{n}{2}$, otherwise it is $\frac{n}{2} + 1$ and hence the parity would be different in each case. To complement our classical time-space tradeoff of the sliding window version of the parity of F_0, we study the complexity $(F_0 \bmod 2)^{\boxplus n}$ in the quantum setting. We use Grover's quantum algorithm for search and Klauck's quantum algorithm [96] for sorting n integers in $TS \in O(n^{3/2} \log n)$ complexity.

Recall that the input to $(F_0 \bmod 2)^{\boxplus n}$ is a string x of length $2n-1$ and the outputs are denoted $y_1, y_2, \ldots, y_n$. The difference between two consecutive outputs is governed by the following equation:

$$y_i - y_{i+1} = \mathbf{1}_{\{x_i \notin C_i\}} - \mathbf{1}_{\{x_{i+n} \notin C_i\}}. \tag{8.1}$$

The following algorithm for $(F_0 \bmod 2)^{\boxplus n}$ runs in time $O(n^{3/2})$ and and uses logarithmic space. To compute the first output y_1 of $F_0^{\boxplus n}$, we run Klauck's sorting algorithm in time $n^{3/2}$ and space $\log n$. To compute the remaining outputs, for each output y_i, we use Grover's search algorithm to find the left and right borders of y_i, x_i and x_{i+n-1} respectively, in their common elements C_i. This allows us to evaluate the indicators in Equation 8.1 and compute y_i from the value of y_{i-1}. Since Grover's search uses logarithmic space and runs in time $\sqrt{n}$, the time complexity of the overall algorithm is $O(n^{3/2})$ and its space complexity is $O(\log n)$.

8.10 Discussion and Open Questions

Using the quantum adversary method, we derived linear lower bounds on the quantum query complexity of multiple functions in AC^0. We also obtained upper bounds on the complexity of sliding-window frequency moments computed by quantum algorithms. Our results motivate studying the following questions.

8.10.1 Approximate degree of boolean functions

While we have shown that there is a function $\phi : [n]^n \to \{0,1\}$ in AC^0 requiring linear quantum query complexity, when this is encoded using Boolean inputs it requires $\Omega(n \log n)$ bits and thus the quantum query lower bound is lower than the number of input bits by an $O(\log n)$ factor. (Note that this lower bound holds even when the query algorithm is able to read the group of $\log_2 n$ bits surrounding each query bit at no extra cost.) This does rule out a polynomial speed-up but it would be nice to obtain a linear lower bound for the Boolean case.

Also, our results do nothing to close the gap in the lower bound on the approximate degree of AC^0 functions. The approximate degree of a function can be used to derive lower bounds on the

communication complexity of closely related functions [137]. Obtaining tighter lower bounds on the approximate degree of AC^0 functions hold promise of tighter lower bounds on the communication complexity of functions in AC^0.

In [14], Ambainis used a standard recursive construction to obtain a Boolean function with quantum query complexity larger than its approximate degree by a small polynomial amount, thus giving a separation between query complexity and approximate degree. Determining the approximate degree of the 2-TO-1-VS-ALMOST-2-TO-1 or ONTO functions, would yield either a much larger degree lower bound for AC^0 or a much larger gap between approximate degree and quantum query complexity than is currently known.

8.10.2 Quantum Time-space tradeoffs in sliding windows

In the single window version, we showed that the parity of F_0 requires a linear number of queries in the quantum query model. The quantum algorithm we describe in Section 8.9 implies that we cannot match the classical time space tradeoff for the sliding-window parity of F_0 we derive in Chapter 4. We conjecture that the right time-space tradeoff is $TS \in \Omega(n^{3/2})$. This would match the quantum time-space tradeoff derived in [96] for sorting n numbers on an oblivious quantum computer. Their lower bound technique divides an oblivious quantum algorithm computing the function into smaller algorithms with few queries to the input that produce a small subset of the outputs. Then, they prove that the probability of success of the algorithm with few queries in computing these outputs is exponentially small in their number. Unfortunately, the basic adversary method lower bound for $(F_0 \bmod 2)$ becomes trivial when the success probability of the algorithm is exponentially small. Deriving lower bounds on the query complexity when the success probability is exponentially small is essential for the above technique to work.

Chapter 9

CONCLUSION

In this thesis, we have studied time-space tradeoffs for different problems on different computational models. We gave the first separation between general branching programs and randomized oblivious ones and established the largest time-space tradeoff known for a randomized model of computation. Moreover, we separated the element distinctness problem from that of computing frequency moments by considering the sliding-window versions of these functions and, on one hand, deriving an optimal $TS = \Omega(n^2)$ time-space tradeoff for frequency moments on general branching programs, and, on the other hand, constructing a $T^2S = \tilde{O}(n^3)$ algorithm for element distinctness.

In the area of error-correcting codes, we derived sufficient conditions for linear codes over large alphabets to be locally-testable and self-correctable, thus proving that sparse random codes have these local properties with high probabilities. For asymptotically good linear codes, we described sufficient properties of their generator matrices such that any encoder of these codes satisfy a time-space tradeoff of $TS = \Omega(n^2)$. We also proposed an approach to establish these properties for any such code and as a first step, proved that every such code contains an asymptotically good code with no low-dimension projections.

Finally, we turned to quantum computers and constructed a function in AC^0 that requires an almost linear number of queries in the quantum query model. This lower bound gives a separation between the element distinctness and the parity of F_0 in the quantum query model. Moreover, it implies almost quadratic formula size lower bounds on the ONTO function. For sliding-window functions, we gave an $n^{3/2}$-time, poly-logarithmic space algorithm for the sliding-window parity of F_0, thus beating the classical time-space tradeoff for that problem.

Below, we summarize some of the interesting directions for future work, inspired by the results of the thesis.

9.1 Direction for Further Study

There are many questions that are still unanswered in the area of time-space tradeoffs and quantum query lower bounds. These are some of the open problems that we discussed throughout of the thesis.

9.1.1 Lower bounds for oblivious randomized branching programs

Oblivious randomized branching programs are probability distributions over branching programs such that the probability distribution of the sequence of input positions read is independent of the input. Contrast this with randomized oblivious branching programs where the sequence of input positions read is fixed for each branching program in the support of the distribution. Oblivious randomized algorithms provide protection in cryptographic settings from adversaries that have the ability to observe which input positions are queried by the internal algorithms of the communicating parties. The obliviousness guarantees that the adversary does not learn any information about the input. Ajtai [9], and independently Damgård, Meldgaard, and Nielsen [58], gave a simulation of any RAM algorithm by a randomized oblivious one with only a poly-logarithmic factor overhead in both time and space. The techniques we use in Chapter 3 to derive time-space tradeoffs do not work for oblivious randomized algorithm since applying Yao's minimax principle in that case removes the randomness that is essential for the obliviousness property of these branching programs. Hence, novel techniques for deriving lower bounds on randomized computation are needed.

9.1.2 Separation between single-window $F_0 \bmod 2$ and ED

It would be interesting to get a separation between $F_0 \bmod 2$ and ED to mirror the separation in sliding windows and in the quantum query model. The only lower bounds known for F_0 are the ones derived for ED that we have previously described: a $TS = \Omega(n^{2-o(1)})$ time-space tradeoffs on comparison branching programs [155] and $T = \Omega(n\sqrt{\log(n/S)/\log\log(n/S)})$ tradeoff on general branching programs [32]. No lower bound on classic models is known for $F_0 \bmod 2$. An algorithm for $F_0 \bmod 2$ that has better complexity than sorting the input would give a separation between the the sliding-window and single-window versions of frequency moments. A lower bound

that matches the $T^2S \in \tilde{O}(n^3)$ we derive on ED would give a separation between the two.

9.1.3 Optimal Time-Space Tradeoffs for Encoders

The final step in the approach we outline in Chapter 7 is to prove Conjecture 7.3: If C is an asymptotically good code with no low-dimension projections, then any arbitrary puncturing of a subcode of C, obtained by fixing some message bits to 0, has dimension linear in its block length. A graph-theoretic analysis similar to the one in [69] might do the trick where one considers a depth-2 encoding circuit and tries to estimate the rank of a submatrix in the generator matrix based on the number of vertex-disjoint paths between the inputs and outputs of the circuit.

9.1.4 Approximate Degree of the ONTO *function*

The ϵ-approximate degree of a boolean function is the smallest degree of a polynomial approximating the function on every boolean input within a range of $\pm\epsilon$. The approximate degree of a function can be used to derive lower bounds on the communication complexity of closely related functions [137, 54]. Moreover, the quantum query complexity is an upper bound on the approximate degree [27]. That is a direct result of the polynomial method for obtaining lower bounds on quantum query complexity, first developed in [27]. The largest lower bound on the approximate degree of a function in AC^0 is $\Omega(n^{2/3})$ derived in [1]. Obtaining tighter lower bounds on the approximate degree of AC^0 functions holds the promise of tighter lower bounds on the communication complexity of functions in AC^0. Hence, finding the approximate degree of the ONTO function would either lead to a larger lower bound on the approximate degree of a function in AC^0, or give a bigger separation between the quantum query complexity and the approximate degree than the one derived in [14].

BIBLIOGRAPHY

[1] S. Aaronson and Y. Shi. Quantum lower bounds for the collision and the element distinctness problems. *Journal of the ACM*, 51(4):595–605, 2004.

[2] H. Abelson. Lower bounds on information transfer in distributed computations. *Journal of the ACM*, 27:384–392, 1980.

[3] K. R. Abrahamson. Time-space tradeoffs for branching programs contrasted with those for straight-line programs. In *27th Annual Symposium on Foundations of Computer Science*, pages 402–409, Toronto, Ontario, Oct. 1986. IEEE. See [5].

[4] K. R. Abrahamson. Generalized string matching. *SIAM Journal on Computing*, 16(6):1039–1051, 1987.

[5] K. R. Abrahamson. Time–space tradeoffs for algebraic problems on general sequential models. *Journal of Computer and System Sciences*, 43(2):269–289, Oct. 1991.

[6] M. Ajtai. Σ_1^1-formulae on finite structures. *Annals of Pure and Applied Logic*, 24:1–48, 1983.

[7] M. Ajtai. Determinism versus non-determinism for linear time RAMs with memory restrictions. *Journal of Computer and System Sciences*, 65(1):2–37, Aug. 2002.

[8] M. Ajtai. A non-linear time lower bound for Boolean branching programs. *Theory of Computing*, 1(1):149–176, 2005.

[9] M. Ajtai. Oblivious RAMs without cryptographic assumptions. In *Proceedings of the Forty-Second Annual ACM Symposium on Theory of Computing*, pages 181–190, Cambridge, Ma, June 2010.

[10] E. W. Allender. A note on the power of threshold circuits. In *30th Annual Symposium on Foundations of Computer Science*, pages 580–584, Research Triangle Park, NC, Oct. 1989. IEEE.

[11] N. Alon and W. Maass. Meanders and their applications in lower bounds arguments. *Journal of Computer and System Sciences*, 37:118–129, 1988.

[12] A. Ambainis. Quantum lower bounds by quantum arguments. *Journal of Computer and System Sciences*, 64:750–767, 2002.

[13] A. Ambainis. Polynomial degree and lower bounds in quantum complexity: Collision and element distinctness with small range. *Theory of Computing*, 1:37–46, 2005.

[14] A. Ambainis. Polynomial degree vs. quantum query complexity. *Journal of Computer and System Sciences*, 72:220–238, 2006.

[15] A. Ambainis. Quantum walk algorithm for element distinctness. *SIAM Journal on Computing*, 37(1):210–239, 2007.

[16] A. Ambainis. Quantum algorithms for formula evaluation. Technical Report arXiv:1006.3651, arxiv, 2010.

[17] A. Ambainis, A. M. Childs, B. Reichardt, R. Spalek, and S. Zhang. Any AND-OR formula of size N can be evaluated in time $N^{1/2+o(1)}$ on a quantum computer. In *Proceedings 48th Annual Symposium on Foundations of Computer Science*, pages 363–372, Berkeley, CA, Oct. 2007. IEEE.

[18] A. Andersson and M. Thorup. Dynamic ordered sets with exponential search trees. *Journal of the ACM*, 54(3):13:1–40, 2007.

[19] A. Arasu and G. S. Manku. Approximate counts and quantiles over sliding windows. In *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Database Systems*, pages 286–296, 2004.

[20] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, May 1998.

[21] K. Azuma. Weighted sums of certain dependent random variables. *Tohoku Math. Journal*, 19(3):357–367, 1967.

[22] L. Babai, N. Nisan, and M. Szegedy. Multiparty protocols, pseudorandom generators for logspace, and time-space trade-offs. *Journal of Computer and System Sciences*, 45(2):204–232, Oct. 1992.

[23] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the Twenty-First Annual ACM Symposium on Principles of Database Systems*, pages 1–16, 2002.

[24] O. Barkol, Y. Ishai, and E. Weinreb. On locally decodable codes, self-correctable codes, and t-private pir. *Algorithmica*, 58:831–859, 2010.

[25] L. Bazzi, M. Mahdian, and D. A. Spielman. The minimum distance of turbo-like codes. *IEEE Transactions on Information Theory*, 55(1):6–15, 2009.

[26] L. Bazzi and S. K. Mitter. Endcoding complexity versus minimum distance. *IEEE Transactions on Information Theory*, 51(6):2103–2112, 2005.

[27] R. Beals, H. Buhrman, R. Cleve, M. Mosca, and R. de Wolf. Quantum lower bounds by polynomials. *Journal of the ACM*, 48(4):778–797, 2001.

[28] P. Beame, R. Clifford, and W. Machmouchi. Element distinctness, frequency moments, and sliding windows. *Electronic Colloquium on Computational Complexity (ECCC)*, 19, 2012.

[29] P. Beame, T. S. Jayram, and M. Saks. Time-space tradeoffs for branching programs. *Journal of Computer and System Sciences*, 63(4):542–572, Dec. 2001.

[30] P. Beame and W. Machmouchi. Making branching programs oblivious requires superlogarithmic overhead. In *IEEE Conference on Computational Complexity*, pages 12–22, 2011.

[31] P. Beame and W. Machmouchi. The quantum query complexity of AC^0. *Quantum Information & Computation*, 12(7-8):670–676, 2012.

[32] P. Beame, M. Saks, X. Sun, and E. Vee. Time-space trade-off lower bounds for randomized computation of decision problems. *Journal of the ACM*, 50(2):154–195, 2003.

[33] P. Beame and E. Vee. Time-space tradeoffs, multiparty communication complexity, and nearest-neighbor problems. In *Proceedings of the Thirty-Fourth Annual ACM Symposium on Theory of Computing*, pages 688–697, Montreal, Quebec, Canada, May 2002.

[34] P. W. Beame. A general time-space tradeoff for finding unique elements. *SIAM Journal on Computing*, 20(2):270–277, 1991.

[35] R. Beigel and J. Tarui. On ACC. *Computational Complexity*, 4:350–366, 1994.

[36] E. Ben-Sasson, N. Ron-Zewi, and M. Sudan. Sparse affine-invariant linear codes are locally testable. In *Annual Symposium on Foundations of Computer Science*, pages 561–570, 2012.

[37] E. Ben-Sasson and M. Viderman. Low rate is insufficient for local testability. In *APPROX/RANDOM'10*, pages 420–433, 2010.

[38] C. H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani. Strengths and weaknesses of quantum computing. *SIAM Journal on Computing*, 26(5):1510–1523, 1997.

[39] E. Bernstein and U. Vazirani. Quantum complexity theory. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 11–20, 1993.

[40] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon limit error-correcting coding and decoding: turbo codes. In *Proceedings of the International Conference on Communication*, pages 1064–1070, 1993.

[41] A. Bhattacharyya, Z. Dvir, A. Shpilka, and S. Saraf. Tight lower bounds for 2-query lccs over finite fields. In *FOCS'11: Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science, 2011.*, pages 638–647, 2011.

[42] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. In *STOC '90: Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 73–83, 1990.

[43] A. Borodin and S. A. Cook. A time-space tradeoff for sorting on a general sequential model of computation. *SIAM Journal on Computing*, 11(2):287–297, May 1982.

[44] A. Borodin, F. E. Fich, F. Meyer auf der Heide, E. Upfal, and A. Wigderson. A time-space tradeoff for element distinctness. *SIAM Journal on Computing*, 16(1):97–99, Feb. 1987.

[45] A. Borodin, M. J. Fischer, D. G. Kirkpatrick, N. A. Lynch, and M. Tompa. A time-space tradeoff for sorting on non-oblivious machines. *Journal of Computer and System Sciences*, 22(3):351–364, June 1981.

[46] A. Borodin, A. A. Razborov, and R. Smolensky. On lower bounds for read-k times branching programs. *Computational Complexity*, 3:1–18, Oct. 1993.

[47] G. Brassard, P. Høyer, and A. Tapp. Quantum algorithm for the collision problem. *SIGACT News (Cryptology Column)*, 28:14–19, 1997.

[48] V. Braverman, R. Ostrovsky, and C. Zaniolo. Optimal sampling from sliding windows. *Journal of Computer and System Sciences*, 78(1):260–272, 2012.

[49] R. P. Brent. An improved Monte Carlo factorization algorithm. *BIT Numerical Mathematics*, 20:176–184, 1980.

[50] R. E. Bryant. Symbolic Boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, 24(3):283–316, 1992.

[51] A. Chakrabarti, G. Cormode, and A. McGregor. A near-optimal algorithm for computing the entropy of a stream. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 328–335, 2007.

[52] T. M. Chan. Comparison-based time-space lower bounds for selection. *ACM Transactions on Algorithms*, 6(2):26:1–26:16, Apr. 2010.

[53] A. K. Chandra, M. L. Furst, and R. J. Lipton. Multi-party protocols. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, pages 94–99, Boston, MA, Apr. 1983.

[54] A. Chattopadhyay. Discrepancy and the power of bottom fan-in in depth-three circuits. In *Proceedings 48th Annual Symposium on Foundations of Computer Science*, pages 449–458, Berkeley, CA, Oct. 2007. IEEE.

[55] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Math. Stat.*, 23:493–509, 1952.

[56] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *Journal of the ACM*, 45(6):965–981, 1998.

[57] S. A. Cook and R. A. Reckhow. Time bounded random access machines. *Journal of Computer and System Sciences*, 7(4):354–375, 1973.

[58] I. Damgård, S. Meldgaard, and J. B. Nielsen. Perfectly secure oblivious ram without random oracles. In *Proceedings of the 8th conference on Theory of cryptography*, TCC'11, pages 144–163, 2011.

[59] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, 31(6):1794–1813, 2002.

[60] R. Deutsch. Quantum computational networks. *Proceedings of the Royal Society of London, Series A, Mathematical and Physical Sciences*, pages 73–90, 1989.

[61] I. Dinur. The pcp theorem by gap amplification. *Journal of the ACM*, 54(3):12, 2007.

[62] D. Divsalar, H. Jin, and R. J. McEliece. Coding theorems for turbo-like codes. In *Proceedings of the 36th Allerton Conference on Communication, Control and Computing*, pages 201–210, 1998.

[63] D. P. Dubhashi and A. Panconesi. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2012.

[64] P. W. Dymond, F. E. Fich, N. Nishimura, P. Ragde, and W. L. Ruzzo. Pointers versus arithmetic in PRAMs. In *Proceedings, Structure in Complexity Theory, Eighth Annual Conference*, pages 239–252, San Diego, CA, May 1993. IEEE.

[65] E. Farhi, J. Goldstone, and S. Gutmann. A quantum algorithm for the Hamiltonian NAND tree. *Theory of Computing*, 4:291–299, 2008.

[66] U. Feige, P. Raghavan, D. Peleg, and E. Upfal. Computing with noisy information. *SIAM Journal on Computing*, 23(5):1001–1018, 1994.

[67] R. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21:467–488, 1982.

[68] M. L. Furst, J. B. Saxe, and M. Sipser. Parity, circuits, and the polynomial-time hierarchy. In *22nd Annual Symposium on Foundations of Computer Science*, pages 260–270, Nashville, TN, Oct. 1981. IEEE.

[69] A. Gál, K. A. Hansen, M. Koucký, P. Pudlák, and E. Viola. Tight bounds on computing error-correcting codes by bounded-depth circuits with arbitrary gates. In *STOC '12: Proceedings of the 44th symposium on Theory of Computing*, pages 479–494, 2012.

[70] R. G. Gallager. *Low Density Parity Check Codes*. PhD thesis, MIT, 1960.

[71] S. I. Gelfand, R. L. Dobrushin, and P. M. S. On the complexity of coding. In *Proceedings of the Second International Symposium on Information Theory*, pages 177–184, 1973.

[72] M. Goldmann and J. Håstad. On the power of small-depth threshold circuits. In *Proceedings 31st Annual Symposium on Foundations of Computer Science*, pages 610–618, St. Louis, MO, Oct. 1990. IEEE.

[73] O. Goldreich. Towards a theory of software protection and simulation by oblivious RAMs. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, pages 182–194, New York, NY, May 1987.

[74] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM*, 43(3):431–473, 1996.

[75] O. Goldreich and M. Sudan. Locally testable codes and pcps of almost-linear length. In *FOCS '02: Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002.*, pages 13–22, 2002.

[76] E. Grigorescu, T. Kaufman, and M. Sudan. Succinct representation of codes with applications to testing. In *APPROX-RANDOM*, pages 534–547, 2009.

[77] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, STOC '96, pages 212–219, 1996.

[78] V. Guruswami and W. Machmouchi. Explicit interleavers for a repeat accumulate accumulate (raa) code construction. In *Proceedings of the IEEE International Symposium on Information Theory*, pages 1968–1972, 2008.

[79] V. Guruswami and M. Sudan. Extensions to the johnson bound. Technical report, Manuscript, 2001.

[80] P. Hall. On representatives of subsets. *J. London Math. Soc.*, 10:26–30, 1935.

[81] R. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29:147–160, 1950.

[82] J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Transactions of the American Mathematical Society*, 117:285–306, 1965.

[83] O. Herfindhal. *Concentration in the U.S. Steel Industry*. PhD thesis, Columbia University, 1950.

[84] A. Hirscham. National power and the structure of foreign trade. *Publication of the Bureau of Business and Economic Research*, 1945.

[85] W. Hoeffding. Probability inequalities for sums of bounded random variables, 1963.

[86] P. Høyer, T. Lee, and R. Spalek. Negative weights make adversaries stronger. In *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*, pages 526–535, San Diego, CA, June 2007.

[87] S. M. Johnson. A new upper bound for error-correcting codes. *IRE Transactions on Information Theory*, pages 203–207, 1962.

[88] S. Jukna. A nondeterministic space-time tradeoff for linear codes. *Inf. Process. Lett.*, 109(5):286–289, 2009.

[89] S. Jukna. *Boolean Function Complexity: Advances and Frontiers*, volume 27 of *Algorithms and Combinatorics*. Springer-Verlag, 2012.

[90] J. Katz and L. Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *STOC '00*, pages 80–86, 2000.

[91] T. Kaufman and S. Litsyn. Almost orthogonal linear codes are locally testable. In *Proceedings 46th Annual Symposium on Foundations of Computer Science*, pages 317–326, Pittsburgh, PA, Oct. 2005. IEEE.

[92] T. Kaufman and S. Lovett. New extension of the weil bound for character sums with applications to coding. In *Annual Symposium on Foundations of Computer Science*, pages 788–796, 2011.

[93] T. Kaufman and D. Ron. Testing polynomials over general fields. In *Proceedings 45th Annual Symposium on Foundations of Computer Science*, pages 413–422, Rome, Italy, Oct. 2004. IEEE.

[94] T. Kaufman and M. Sudan. Sparse random linear codes are locally decodable and testable. In *Proceedings 47th Annual Symposium on Foundations of Computer Science*, pages 590–600, Berkely, CA, Oct. 2006. IEEE.

[95] V. Khrapchenko. Complexity of the realization of a linear function in the case of π-circuits. *Math. Notes Acad. Sciences*, 9:21–23, 1971.

[96] H. Klauck. Quantum time-space tradeoffs for sorting. In *35th ACM Symposium on Theory of Computing*, pages 69–76, 2003.

[97] D. E. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming*. Addison-Wesley, 1969.

[98] D. E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, 1971.

[99] D. E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, 1973.

[100] S. Kopparty and S. Saraf. Tolerant linearity testing and locally testable codes. In *APPROX '09 / RANDOM '09*, pages 601–614, 2009.

[101] S. Kopparty and S. Saraf. Local list-decoding and testing of random linear codes from high-error. In *Proceedings of the Forty-Second Annual ACM Symposium on Theory of Computing*, pages 417–426, Cambridge, Ma, June 2010.

[102] I. Krasikov and A. Zarkh. On zeros of discrete orthogonal polynomials. *J. Approx. Theory*, 156(2):121–141, 2009.

[103] G. Kuperberg, S. Lovett, and R. Peled. Probabilistic existence of rigid combinatorial structures. In *Proceedings of the 44th symposium on Theory of Computing*, pages 1091–1106, 2012.

[104] S. Kutin. Quantum lower bound for the collision problem with small range. *Theory of Computing*, 1(1):29–36, 2005.

[105] T. W. Lam and W. L. Ruzzo. Results on communication complexity classes. *Journal of Computer and System Sciences*, 44(2):324–342, Apr. 1992.

[106] S. Laplante, T. Lee, and M. Szegedy. The quantum adversary method and classical formula size lower bounds. In *Proceedings of the Twentieth Annual IEEE Conference on Computational Complexity*, pages 76–90, 2005.

[107] L. K. Lee and H. F. Ting. Maintaining significant stream statistics over sliding windows. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 724–732, 2006.

[108] L. K. Lee and H. F. Ting. A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Database Systems*, pages 290–297, 2006.

[109] T. Lee, R. Mittal, B. Reichardt, R. Špalek, and M. Szegedy. Quantum query complexity of state conversion. In *Proceedings of the 52nd Annual Symposium on Foundations of Computer Science*, pages 344–353, Palm Springs, CA, Oct. 2011. IEEE.

[110] M. Luby and C. Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2):373–386, 1988.

[111] O. B. Lupanov. On the problem of realization of symmetric Boolean functions by contact schemes. *Probl. Kibernet*, 15:85–99, 1965. In Russian.

[112] W. Machmouchi. Local-testability and self-correctability of q-ary sparse linear codes. In *Proceedings of IEEE International Symposium on Information Theory (ISIT)*, June 2010.

[113] F. MacWilliams and N. Sloane. *The Theory of Error-Correcting Codes*. North Holland Publishing Co., 1977.

[114] F. J. MacWilliams. Error-correcting codes for multiple-level transmission. *The Bell System Technical Journal*, pages 281–308, 1961.

[115] Y. Mansour, N. Nisan, and P. Tiwari. The computational complexity of universal hashing. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*, Baltimore, MD, May 1990.

[116] W. Masek. A fast algorithm for the string editing problem and decision graph complexity. Master's thesis, Massachusetts Institute of Technology, 1976.

[117] D. Muller. Complexity in electronic switching circuits. *IRE Transactions on Electronic Computers*, EC-5(1):15–19, 1956.

[118] J. I. Munro and M. S. Paterson. Selection and sorting with limited storage. In *19th Annual Symposium on Foundations of Computer Science*, pages 253–258, Ann Arbor, MI, Oct. 1978. IEEE.

[119] M. Naor and O. Reingold. On the construction of pseudorandom permutations: Luby-rackoff revisited. *J. Cryptology*, 12(1):29–66, 1999.

[120] G. Nivasch. Cycle detection using a stack. *Information Processing Letters*, 90(3):135–140, 2004.

[121] E. Okol'nishnikova. On the hierarchy of nondeterministic branching k-programs. In *Fundamentals of Computation Theory: 11th International Conference, FCT '97*, volume 1102 of *Lecture Notes in Computer Science*, pages 376–387, Krakow, Poland, 1997. Springer-Verlag.

[122] R. Ostrovsky. Efficient computation on oblivious RAMs. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*, pages 514–523, Baltimore, MD, May 1990.

[123] J. Pagter and T. Rauhe. Optimal time-space trade-offs for sorting. In *Proceedings 39th Annual Symposium on Foundations of Computer Science*, pages 264–268, Palo Alto, CA, Nov. 1998. IEEE.

[124] N. J. Pippenger and M. J. Fischer. Relations among complexity measures. *Journal of the ACM*, 26(2):361–381, Apr. 1979.

[125] J. M. Pollard. A Monte Carlo method for factorization. *BIT Numerical Mathematics*, 15(3):331–334, 1975.

[126] J. M. Pollard. Monte Carlo methods for index computation $(\mathrm{mod}\, p)$. *Mathematics of Computation*, 32(143):918–924, 1978.

[127] E. Rains and N. J. A. Sloane. Self-dual codes. *The Handbook of Coding Theory*, pages 177–294, 1998.

[128] B. W. Reichardt. Faster quantum algorithm for evaluating game trees. Technical Report arXiv:0907.1623, arxiv, 2009.

[129] B. W. Reichardt. Span programs and quantum query complexity: The general adversary bound is nearly tight for every Boolean function. In *Proceedings of the 50th Annual Symposium on Foundations of Computer Science*, pages 544–551, Atlanta, GA, 2009. IEEE.

[130] B. W. Reichardt. Reflections for quantum query algorithms. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 560–569, San Francisco, CA, Jan. 2011. Society for Industrial and Applied Mathematics.

[131] N. Santhi and A. Vardy. Minimum distance of codes and their branching program complexity. In *2006 IEEE International Symposium on Information Theory*, pages 1490–1494. IEEE, July 2006.

[132] M. Sauerhoff. A lower bound for randomized read-k-times branching programs. In *(STACS) 98: 15th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1373 of *Lecture Notes in Computer Science*, pages 105–115, Paris, France, Feb. 1998. Springer-Verlag.

[133] J. E. Savage. Computational work and time on finite machines. *Journal of the ACM*, 19(4):660–674, Oct. 1972.

[134] R. Sedgewick, T. G. Szymanski, and A. C.-C. Yao. The complexity of finding cycles in periodic functions. *SIAM Journal on Computing*, 11(2):376–390, 1982.

[135] C. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 1948.

[136] C. Shannon. The synthesis of two-terminal switching circuits. *The Bell System Technical Journal*, 28:59–98, 1949.

[137] A. A. Sherstov. The pattern matrix method. *CoRR*, abs/0906.4291, 2009.

[138] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.

[139] D. R. Simon. On the power of quantum computation. *SIAM Journal on Computing*, 26:116–123, 1994.

[140] E. Simpson. Measurement of diversity. *Nature*, 163:688–688, 1949.

[141] R. Špalek and M. Szegedy. All quantum adversary methods are equivalent. *Theory of Computing*, 2(1):1–18, 2006.

[142] D. A. Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Transactions on Information Theory*, 42(6):1723–1731, 1996.

[143] J. S. Thathachar. On separating the read-k-times branching program hierarchy. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 653–662, Dallas, TX, May 1998.

[144] M. Tompa. *Time-space tradeoffs for straight-line and branching programs*. PhD thesis, University of Toronto, 1978.

[145] M. Tompa. Time-space tradeoffs for computing functions, using connectivity properties of their circuits. *Journal of Computer and System Sciences*, 20:118–132, Apr. 1980.

[146] A. M. Turing. On computable numbers with an application to the entscheidungs problem. *Proc. London Math. Soc.*, 2(42):230–265, 1936. A correction, *ibid.*, **43**, pp. 544–546.

[147] L. Valiant. Graph-theoretic arguments in low-level complexity. In *The 6th Symposium on Mathematical Foundations of Computer Science*, 1977.

[148] P. C. van Oorschot and M. J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12(1):1–28, 1999.

[149] S. Žák. An exponential lower bound for one-time-only branching programs. In M. P. Chytil and V. Koubek, editors, *Mathematical Foundations of Computer Science 1984: Proceedings, 11th Symposium*, volume 176 of *Lecture Notes in Computer Science*, pages 562–566, Praha, Czechoslovakia, Sept. 1984. Springer-Verlag.

[150] I. Wegener. On the complexity of branching programs and decision trees for clique functions. *Journal of the ACM*, 35:461–471, 1988.

[151] I. Wegener. *Branching Programs and Binary Decision Diagrams: Theory and Applications*. Society for Industrial and Applied Mathematics, 2000.

[152] D. Williams. *Probability with Martingales*. Cambridge University Press, 1991.

[153] A. C. Yao. Probabilistic computations: Toward a unified measure of complexity. In *18th Annual Symposium on Foundations of Computer Science*, pages 222–227, Providence, RI, Oct. 1977. IEEE.

[154] A. C. Yao. Some complexity questions related to distributive computing. In *Conference Record of the Eleventh Annual ACM Symposium on Theory of Computing*, pages 209–213, Atlanta, GA, Apr.-May 1979.

[155] A. C. Yao. Near-optimal time-space tradeoff for element distinctness. In *29th Annual Symposium on Foundations of Computer Science*, pages 91–97, White Plains, NY, Oct. 1988. IEEE.

[156] A. C.-C. Yao. On acc and threshold circuits. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, pages 619–627, 1990.

[157] S. Yekhanin. Towards 3-query locally decodable codes of subexponential length. *Journal of the ACM*, 55(1):1–16, Feb. 2008.

[158] Y. Yesha. Time-space tradeoffs for matrix multiplication and the discrete Fourier transform on any general sequential random-access computer. *Journal of Computer and System Sciences*, 29:183–197, 1984.