Collision Finding with Many Classical or Quantum Processors

by

Stacey Jeffery

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Combinatorics and Optimization

Waterloo, Ontario, Canada, 2011

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

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

Abstract

In this thesis, we investigate the cost of finding collisions in a black-box function, a problem that is of fundamental importance in cryptanalysis. Inspired by the excellent performance of the heuristic rho method of collision finding, we define several new models of complexity that take into account the cost of moving information across a large space, and lay the groundwork for studying the performance of classical and quantum algorithms in these models.

Acknowledgements

I am deeply indebted to my supervisor, Dr. Michele Mosca, for introducing me to the subject of quantum information processing, and for the years of support and encouragement since. I'm also grateful to the reading committee, Dr. Ashwin Nayak and Dr. Ben Reichardt for the useful feedback.

In addition, I would like to thank Dr. Anne Broadbent for patiently mentoring me since I was an undergraduate student and teaching me most of what I know about research. I would also like to thank Dr. Frédéric Magniez for his advice and guidance over the course of my Masters study. I also thank and acknowledge Dr. Magniez for his collaboration on the result of Chapter 6.

I'm grateful to Dr. Dan Brown and Dr. Edlyn Teske for useful discussions and feedback on the results of Chapter 5.

I would like to thank my fellow students and officemates, in particular Jamie Sikora for explaining how to write a thesis, Robin Kothari for helpful comments on Chapter 7, and Sarvagya Upadhyay for pointing me in the direction of [5].

Finally, it's with the utmost gratitude that I thank my family and friends for their years of support, in particular my mother, Kelly macGregor, and my grandmothers, Helen Jeffery and Carolyn Weeks, for two and a half decades of love, advice and inspiration, as well as my partner, Moritz Ernst, for his love, undying support, and delicious cooking.

Table of Contents

Chapter 1

Introduction

The problem of finding collisions arises in the area of cryptanalysis in relation to breaking collision resistant hash functions. These functions are used in authentication schemes, and the ability to find a collision in a hash function upon which an authentication scheme is based is considered to compromise the security of the authentication scheme. It is thus of great practical importance to understand the complexity of the general collision finding problem, in order to, for instance, choose appropriate parameters.

In a recent commentary, [16], Bernstein conjectures that for the problem of collision finding, no quantum algorithm will ever beat the current classical methods in practice. We lay the groundwork for investigating this question and, in the doing so, develop several new models of computation that we hope will be of general interest.

The fastest method of collision finding in practice is the heuristic parallel rho method. It uses a number of processors proportional to the problem size. Several factors make the rho method achieve better performance in practice than other methods of collision finding which have better query complexity, or even better complexity in the standard RAM model. The first is that each processor is able to work with a very small amount of space. In practice, accessing data stored in a large space is costly, and this can't be avoided. The finite speed of light, which bounds the speed of information, and the holographic principle, which bounds the amount of information a physical region can store, mean that the cost of accessing an array must grow with the amount of information in the array. This motivates us to formally define a new model of complexity which counts the cost of information movement: locality-sensitive complexity.

The second practical advantage that the parallel rho method has is that there is minimal communication between processors throughout most of the computation. Communication

between many processors is costly for the same reason that information access in a large space is costly. This motivates us to define the parallel counterpart to the locality-sensitive model: the grid model, which contrasts other parallel models in that it counts the cost of communication between two processors by their distance from one another.

These issues are not only of practical interest, but also of fundamental theoretical interest. In the words of Aaronson and Ambainis:

> [I]f we are interested in the fundamental limits imposed by physics, then we should acknowledge that the speed of light is finite, and that a bounded region of space can store only a finite amount of information, according to the holographic principle [2].

Although we only consider collision finding and related problems in the new models, we certainly believe there is future work in investigating the complexity of other problems in these models.

This thesis is in three major parts. In Chapter 2 we will give necessary preliminaries and outline various common models of computation, culminating in the introduction of two new models, each available in three flavours: the locality-sensitive models (deterministic, randomized, and quantum) and the grid models (deterministic, randomized, and quantum). In Chapter 3, we will introduce the precise definitions of the problems we will be examining.

Chapters 4, 5, and 6 deal with algorithms. In Chapter 4, we will survey quantum and classical algorithms for collision finding and element distinctness in well-studied models and extend them, where applicable, to the new models. In Chapter 5, we will take a close look at the best known classical heuristic for collision finding (on one or many processors), Pollard's rho method. We will show that the only known rigorous analysis of this algorithm that does not require a random oracle assumption actually rests on a false assumption, and thus, the rho method really is merely heuristic (though we note that it appears to be an excellent heuristic in practice). Finally in Chapter 6, we will present a new algorithm that deterministically solves a limited version of element distinctness.

In Chapters 7 and 8, we tell the other side of the story: lower bounds. We begin by surveying the known techniques for quantum lower bounds in Chapter 7, before discussing how future work may apply them to our models in Chapter 8. In Chapter 8, we derive some trivial lower bounds for our new models and discuss the possibility of closing the remaining gaps using known techniques.

Chapter 2

Models of Computation

Computation is essentially the process of manipulating the physical state of some small subsystem of the universe in order to solve some mathematical problem. The state of any physical system evolves over time, according to the laws of physics. Computation is simply the process of taking control of this evolution so that a desired answer is, eventually, encoded in the state of the system.

A finite physical system can be in one of a finite number of states, each of which we consider as encoding some integer, represented by a binary string. We call the set of states the *state space*, Ω, and label the states as $\Omega = \{0,1\}^n$, for some n.

In this thesis, we will discuss *algorithms* to be carried out by some type of *computer* in order to solve some particular problem. An algorithm is a finite set of mathematical instructions comprised of certain operations of which the target computer is assumed to be capable. For instance, an algorithm for any type of deterministic computer should not include the instruction: "choose a uniform random integer from $\{0,\ldots,N\}$".

There is thus a need to define *models* of computation, so that when designing an algorithm, we know which instructions we may use. Along with this notion is the idea of efficiency: a basic operation in one model might be very costly in another. We therefore have, in each model, an idea of the cost of operations. Though a classical computer can, in theory, do everything a quantum computer can do, it may require exponentially greater resources.

A *deterministic* algorithm is a set of logical instructions for definite evolutions from one state in Ω to another. For every state, at every step of the algorithm, the state at any subsequent step is well-defined.

A *randomized* algorithm is the same as a deterministic algorithm, except that we assume the existence of an extra register, often called the *coin register*, which is reset to a random bit string in $\{0,1\}^r$ at every step. The next state of the computation register is a well-defined function of the contents of the coin register and current computation register, but the coin register's next state is independent of these.

Neither of these types of algorithms gives any notion of cost. For example, a deterministic algorithm for sorting a list could consist of the single instruction: "sort the list". This is a deterministic instruction for evolving the state, but we would not say that sorting has cost 1. We therefore have the notion of some basic set of operations, usually on a constant number of operands, and we count each of these as 1.

A *random access machine* (RAM) is a model of computation that closely resembles a standard computer. Operations are performed on values in an array of addressed memory. The cost of a basic operation, such as adding two numbers, can be counted in one of two ways. In the *uniform cost model*, basic operations are considered to have constant cost, whereas in the *logarithmic cost model*, basic operations have cost that is logarithmic in the size of the operands. In our case, we will be operating on things of size $\log N$, and ignoring $\log N$ factors, so there is little difference between these two models. Note however that accessing an arbitrary memory location takes constant time in this model, independent of the size of the memory. Another model of computation, the *circuit model*, counts the number of gates used. In this model, a memory access cost is logarithmic in the size of the memory.

In the problems we will be interested in, we can view the input as a *black-box* function $f : [N] \to [R]$. We can think of f as being a very large table of N values from $[R]$, indexed by $[N]$, so that $f(x)$ (which we could also denote $f[x]$ if we want to emphasize the tabular nature of f) is the x^{th} entry of f. In practice, f may be a more abstract function which we have some unspecified means of computing, however, we make no assumption about the structure of f.

One classic example is that $f : \mathcal{X} \times \mathcal{C} \to \{0,1\}$ is a verifier for some NP-complete language $\mathcal{L}$. We can efficiently compute f on any instance-certificate pair $(x,c) \in \mathcal{X} \times \mathcal{C}$, but given some $x \in \mathcal{X}$, determining whether or not there exists $c \in \mathcal{C}$ such that $f(x,c) = 1$ is thought to be very difficult (unless P=NP). Thus, even though we may have an efficient implementation of f, it can still take an exponential amount of time to establish certain properties of f.

A second example, related to the main topic of this thesis, is finding hash collisions. A hash function is a primitive that often comes up in cryptography (see [40] for a thorough overview). A hash function $h : \mathcal{X} \to \mathcal{Y}$ is an efficiently computable function where $|\mathcal{Y}| <<$

$|\mathcal{X}|$, so naturally there are many pairs $x, x' \in \mathcal{X}$ with $x \neq x'$ and $h(x) = h(x')$. However, a necessary property of a hash function in many applications is *collision resistance*, that is, finding such a pair x, x' is difficult. This is another example of a situation where we may have an efficiently computable function, but finding some property of that function, in this case, a pair that collides under the function, is believed to be difficult.

In these examples, and all those we will consider, we generally require exponential resources to solve a problem, so algorithms seek to minimize the growth constant (constant in the exponent) but do not hope to reduce the required resources to polynomial (in the input size, $\log N$). We are therefore not particularly concerned with any poly-logarithmic factors, such as the computation time of the input function f. We will use the notation $\tilde{\Omega}$, $\tilde{O}$, and $\tilde{\Theta}$ to indicate asymptotic growth up to poly-logarithmic factors.

The dominating cost in the black-box problems we consider is the number of times we must make use of the black-box f (each use is called a *query*), which we assume dwarfs the cost of a query itself. This leads to a notion of complexity called *query complexity*. For a deterministic algorithm A, we define the query complexity of A, $D(\mathsf{A})$, as the maximum over all f, number of queries made by A to to input f. Similarly, for a randomized algorithm A that outputs $\mathbf{F}(f)$ with probability at least $\frac{2}{3}$ on input f, we define $R(\mathsf{A})$ as the maximum, over all f, number of queries made to f.

We can then define the deterministic query complexity of a problem, $D(\mathbf{F})$, as the minimum, over all deterministic algorithms A that compute $\mathbf{F}$, $D(\mathsf{A})$, and the randomized query complexity, $R(\mathbf{F})$, as the minimum, over all randomized algorithms A that compute $\mathbf{F}$, $R(\mathsf{A})$.

When the inputs to $\mathbf{F}$ are considered as Boolean functions, that is, a query returns a single bit, then $D(\mathbf{F})$ is sometimes called the *decision tree complexity* and $R(\mathbf{F})$ the *randomized decision tree complexity*. In general we can view the decision tree complexity and query complexity as equivalent. An input $f : [N] \to [R]$ to a problem can be viewed as a vector in $[R]^N$ or a binary string in $\{0,1\}^{N \log R}$. A query $f(x)$ returns the x^{th} entry in the vector, and a *bit query* $f[i]$ returns the i^{th} bit in the string. We can simulate a bit query with a single query (by querying the vector entry containing that bit and discarding all but the desired bit) and we can simulate a query with $\log R$ bit queries. Depending on the setting, it may be more convenient to consider queries or bit queries.

2.1 Quantum Computation

In this section, we give a very brief introduction to quantum computation. For a more thorough introduction, see [29].

Whereas the state of a classical system can be represented as a binary string of some finite length n, a quantum system of size n can be in any *superposition* of these states. That is, the quantum state has some *amplitude* associated with each of the 2^n classical states, which we represent as a 2^n-dimensional complex vector. We fix an orthonormal basis for $\mathbb{C}^{2^n}$ and label its members with the 2^n binary strings of length n, or equivalently, the integers from 1 to N, where $N = 2^n$: $\{|1\rangle, |2\rangle, \ldots, |N\rangle\}$. We call this the *computational basis*. A quantum state is therefore of the form $\sum_{x\in[N]} \alpha_x |x\rangle$ for $\alpha_x \in \mathbb{C}$ and $[N] = \{1, \ldots, N\}$. If we *measure* the state, we will get some outcome $x \in [N]$. The probability of observing the outcome x is $|\alpha_x|^2$. We therefore have $\||\psi\rangle\|_2 = 1$ for all quantum states $|\psi\rangle$.

We can combine two systems $\mathcal{H}_1$ and $\mathcal{H}_2$ to get the system $\mathcal{H}_1 \otimes \mathcal{H}_2$. If the first system is in the state $|\psi_1\rangle$ and the second system is in the state $|\psi_2\rangle$, then the joint system is in the state $|\psi_1\rangle \otimes |\psi_2\rangle$, which we sometimes write as $|\psi_1\rangle|\psi_2\rangle$.

We may want to consider the state of a subsystem of the quantum system, which is difficult to do when the state is given as a vector. We can therefore consider a general state as a *density operator*, a positive semidefinite operator with trace 1. The density operator of the complete system is $\rho = |\psi\rangle\langle\psi|$, where $\langle\psi| = |\psi\rangle^\dagger$. Such a system is said to be in a pure state. To consider some part of a system, say $\mathcal{H}_1$ in the system $\mathcal{H}_1 \otimes \mathcal{H}_2$, we use the partial trace operator to *trace out* the subsystem on $\mathcal{H}_2$. Thus, if the joint system is in the state ρ, the state on $\mathcal{H}_1$ is $\rho_1 = \mathrm{Tr}_2(\rho)$. It may not be possible, in general, to write ρ_1 as $|\phi\rangle\langle\phi|$ for some vector ϕ. In that case, we say that the state is *mixed*. If $\mathrm{Tr}_1(\rho)$ and $\mathrm{Tr}_2(\rho)$ are mixed states (for some pure state ρ), then we say the two systems are *entangled*.

A quantum operator is a linear operation on the state space $\mathcal{H}$. More generally, since we need states to have norm 1, the operator must be unitary. A quantum circuit is a sequence of unitary operations on some initial state: $U_T U_{T-1} \ldots U_1 |\psi_{init}\rangle$.

To get classical information from a quantum state, we can perform a *measurement*. A measurement can be described by a set of projections $\{P_x : x \in \Sigma\}$ such that $\sum_x P_x = I$. The set Σ is called the set of *outcomes*. The outcome of the measurement on a state $|\psi\rangle$ will be $x \in \Sigma$ with probability $\langle\psi|P_x|\psi\rangle$. If the outcome of a measurement is x, then the state of the system after the measurement is the normalization of $P_x|\psi\rangle$.

2.2 Quantum Query Model

In the quantum analogue of a query to f, we need the operation to be unitary, and thus reversible. We could thus define a query operator with respect to a particular black-box, f, as:

$$\mathcal{O}'_f|x,z\rangle = |x, z \oplus f(x)\rangle$$

However, in the case where $f(x)$ is a single bit, it will usually be more convenient to think of a query as encoding the queried value in the phase, as:

$$\mathcal{O}_f|x\rangle = (-1)^{f(x)}|x\rangle$$

(Note that $\mathcal{O}_f$ is the same as $\mathcal{O}'_f$ when the second register contains $|-\rangle$).

Clearly this operator is self-inverse.

Note that the query operator $\mathcal{O}_f$ depends on the input f. Depending on the actual application, we may be able to construct a unitary for each input, but it may be desirable to think of the query operator as independent of the input. To this end, we extend our Hilbert space as follows. Let $\mathcal{H}_C$ be the computation space, spanned by vectors of the form $|x,w\rangle$ where x is the next query input and w is the workspace. Non-query operators act only on $\mathcal{H}_C$. Let $\mathcal{H}_I$ be the input space, spanned by vectors of the form $|f\rangle$, where f is the input function, which we can think of as being stored as a table (more generally, we may have a quantum circuit for computing values of f, but thinking of f as a large table allows us to abstract away from any of the details of this circuit). Non-query operators act as the identity on $\mathcal{H}_I$. Generally the input space will start in a basis state $|f\rangle$ (we will input a definite function), however in Section 7.2 we will see that it can be useful to consider the input as being in a superposition.

We can now think of a query operator as acting on $\mathcal{H}_I \otimes \mathcal{H}_C$ as follows:

$$\mathcal{O}|f,x,w\rangle = (-1)^{f(x)}|f,x,w\rangle$$

Note that the action of $\mathcal{O}$ on $\mathcal{H}_C$ is the same as that of $\mathcal{O}_f$ (when f is in the input register) so we can think of the query operator in either of these ways, depending on which is most convenient.

To construct a general circuit in the quantum query model, we simply interleave some number T of query operators with arbitrary unitaries, $U_0, \ldots, U_T$. The initial state is some arbitrary quantum state that is independent of the input, so we write:

$$|\psi_f^0\rangle := U_0|0\ldots0\rangle$$

Generally, we let $|\psi_f^t\rangle$ denote the state of $\mathcal{H}_C$ just before the $t+1^{\text{th}}$ query on input f, so:

$$|\psi_f^t\rangle = U_t\mathcal{O}_f|\psi_f^{t-1}\rangle$$

and final state:

$$|\psi_f^T\rangle = U_T\mathcal{O}_f U_{T-1}\ldots U_1\mathcal{O}_f U_0|0\ldots0\rangle$$

Measuring the (without loss of generality) rightmost bit of $|\psi_f^T\rangle$ produces the output. We say the circuit computes $\mathbf{F}$ with bounded error ϵ, for $\epsilon \in [0, \frac{1}{2})$, if the maximum probability over all valid inputs f that the outcome is not $\mathbf{F}(f)$ is ϵ. More formally, let A denote the circuit above. Then A computes $\mathbf{F}$ with bounded error if:

$$\max_f \left(1 - \sum_{|c\rangle \in \mathcal{S}_{\mathbf{F}(f)}} \left|\langle c|\psi_f^T\rangle\right|^2\right) \leq \epsilon$$

where $\mathcal{S}_{\mathbf{F}(f)}$ is the set of all computational basis states of $\mathcal{H}_C$ that have $\mathbf{F}(f)$ in the answer register.

We say that a quantum algorithm has query complexity $Q(\mathsf{A}) = T$ if it makes T uses of $\mathcal{O}$. The quantum query complexity of a problem is defined as $Q(\mathbf{F}) := \min Q(\mathsf{A})$ where the minimum is taken over all quantum algorithms A that compute $\mathbf{F}$ with bounded error.

2.3 Locality-Sensitive Models

For the problems we will be considering, we can reduce the time complexity by making use of an amount of space that depends polynomially on N. In a standard model of classical or quantum computation, accessing a position in this table would cost constant or logarithmic

time, which we consider negligible. We argue that this lack of cost is not well-motivated physically. In order to perform a random access to a table of physical size k, information must travel up to k units of distance, at a finite speed, due to the finite speed of light. We cannot reduce k arbitrarily, due to the holographic principle. These two properties are stated formally as:

Superluminal Signaling Assumption: Information cannot travel faster than the speed of light, which is finite.

Holographic Principle: The information contained in a region of space is upper bounded by 1 bit per Planck area of the surface area of the space [17].

The implication of these two physical limits is that a memory of size S has access time that scales as $\sqrt{S}$. This is because, since the entropy (information content) of a region of space is proportional to the *surface area*, the sum of the dimensions of a region of space storing S bits of information must be in $\Omega(\sqrt{S})$. The best we can do asymptotically is to store information in two dimensions. Even if we consider a three-dimensional space of *volume* S, the amount of information we can store in that space is actually limited by the *surface area* of that space, by the holographic principle. The surface area is maximized (asymptotically) by setting one of the dimensions of the space to 1 unit of memory (which is minimal in this case). It is thus reasonable to consider information as being stored in two dimensions. The girth of a two-dimensional space is minimized when that space is arranged in a disc of radius $\sqrt{S/\pi}$, but is still asymptotically minimal (and simpler to consider) when arranged in a $\sqrt{S} \times \sqrt{S}$ grid. We thus always assume that information is stored in a grid.

The bound on the speed of random access imposed by the finite speed of light and the holographic principle was pointed out by [14, 2]. They point out that, even in a grid of S qubits of memory, performing a single memory access on a superposition over all memory locations costs $O(\sqrt{S})$, since amplitude must traverse the grid. This consideration has come up in the classical world, in particularly in the study of *very large scale integration* (VLSI), which concerns the design of complicated circuits on very small chips. Optimal designs for such chips were studied in the 70s and 80s. These works have a similar spirit to our models in that a two-dimensional space is considered and the cost of communication across the space is taken into account. There are several area-time tradeoffs for particular problems that are very much in the spirit of what we would like to achieve, for instance [4, 50].

This stricter measure of complexity has also been discussed recently by Bernstein [16], who points out that certain algorithms requiring large space fail to account for the cost of access to that space. Until now, it has been practical to ignore this cost, since the

constants involved were relatively minuscule, however, as we move towards the physical limits of computation, we cannot ignore these fundamental physical principles. We thus define the following measures of complexity, in which random access to a table of size S costs $\Theta(\sqrt{S})$. We assume the memory is arranged in a grid.

The *locality-sensitive complexity* of an algorithm is a measure of complexity in which we count all non-query operations as in the RAM model (or quantum circuit model), with the exception that accessing a table of size S (or any other operation that necessarily requires communication across a space of area S) costs $\Theta(\sqrt{S})$. We assign each query operation a cost of 1. This implicitly makes the assumption that the query can be made in efficient (poly-logarithmic) resources (since we are ignoring poly-logarithmic factors).

The deterministic locality-sensitive complexity of a problem $\mathbf{F}$, $\overline{D}(\mathbf{F})$, is the minimum over all deterministic algorithms A that compute $\mathbf{F}$, locality-sensitive complexity of A.

Similarly, the randomized locality-sensitive complexity of a problem $\mathbf{F}$, $\overline{R}(\mathbf{F})$, is the minimum over all randomized algorithms A that compute $\mathbf{F}$, locality-sensitive complexity of A.

The quantum locality-sensitive complexity of a problem $\mathbf{F}$, $\overline{Q}(\mathbf{F})$, is the minimum over all quantum algorithms A that compute $\mathbf{F}$, locality-sensitive complexity of A.

Note that we immediately have $D(\mathbf{F}) \leq \overline{D}(\mathbf{F})$, $R(\mathbf{F}) \leq \overline{R}(\mathbf{F})$, and $Q(\mathbf{F}) \leq \overline{Q}(\mathbf{F})$.

2.4 Distributed Grid Models

We now discuss distributed locality-sensitive computation. We will consider many parallel processors, each with its own poly-logarithmic memory, arranged in a grid. The cost of communication between two processors will be proportional to the distance between them, so we will suppose that each processor can only communicate directly with its neighbours, which will cost $\Theta(1)$. The processors may be deterministic, randomized, or quantum.

This model is of fundamental physical interest, since it is similar to the way the universe actually performs computations; in a very parallel but local manner. One motivation for this model is the argument that a database of S quantum memory units is actually a collection of S computing units [16, 54], so it is justified to replace the notion of memory with that of processors. Another motivation is that the heuristic performance of the rho method of collision finding, discussed in Chapter 5, seems to have exceptionally good performance in a parallel model, even under our locality-sensitive constraints, and so we are interested in how the quantum counterpart might compete with this performance.

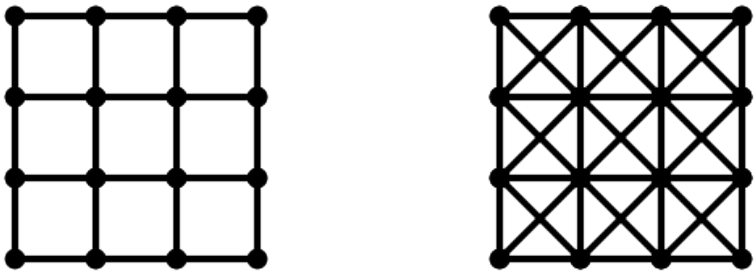

Figure 2.1: A 4×4 *grid* (left). We can also consider each vertex being connected to the nearest diagonal vertices (right). In either case, each vertex has a constant number of neighbours.

We suppose we have M processors arranged in a $\sqrt{M} \times \sqrt{M}$ grid. Each has local black-box access to the input f. In a single unit of time, each processor may: perform a query, perform a basic computation step, or send a message of size $\log N$ to one of its neighbours.

Consider running some algorithm A in this model. Let $T^{(i)}(\mathsf{A})$ be the number of operations performed by processor i. Define $T(\mathsf{A}) := \max_i T^{(i)}$. We can designate a distinguished processor that must output the correct answer when the algorithm terminates. No matter which processor we choose, there must be some processor at distance $\Theta(\sqrt{M})$ from the distinguished processor. If the algorithm has $T \in o(\sqrt{M})$, then the output of the distinguished processor is necessarily independent of some processors, since no communication from them can have reached the distinguished processor in this time. Thus, we should really consider the algorithm as having run on some smaller number of processors. We therefore always assume that $T \in \Omega(\sqrt{M})$, and to this end, we say that the complexity of A in this model is actually $\Theta(T + \sqrt{M})$ (T technically accounts for all communication costs, but the $+\sqrt{M}$ emphasizes the fact that T can't be less than $\sqrt{M}$). We can therefore do away with the distinguished processor formalism and suppose that each processor, or any one of the processors, must output the correct answer; it makes no difference asymptotically.

We can thus define our final measures of complexity. If we are working with deterministic processors, we can consider the *deterministic M-grid complexity* of a problem $\mathbf{F}$:

$$D^{\boxplus}_M(\mathbf{F}) := \min T(\mathsf{A}) + \sqrt{M}$$

where A is any deterministic grid algorithm that solves $\mathbf{F}$. From this, we get the *deterministic grid complexity*:

$$D^{\boxplus}(\mathbf{F}) = \min_M D^{\boxplus}_M(\mathbf{F})$$

If the processors are randomized, we have the *randomized M-grid complexity*:

$$R_M^{\boxplus}(\mathbf{F}) := \min T(\mathsf{A}) + \sqrt{M}$$

where A is taken over all randomized grid algorithms for $\mathbf{F}$, and the *randomized grid complexity*:

$$R^{\boxplus}(\mathbf{F}) = \min_M R_M^{\boxplus}(\mathbf{F})$$

Finally, when working with a grid of quantum processors, we have the *quantum M-grid complexity* of a problem:

$$Q_M^{\boxplus}(\mathbf{F}) := \min T(\mathsf{A}) + \sqrt{M}$$

where A is taken over all quantum grid algorithms for $\mathbf{F}$, and the *quantum grid complexity*:

$$Q^{\boxplus}(\mathbf{F}) := \min_M Q_M^{\boxplus}(\mathbf{F})$$

We assume that a quantum processor can implement any unitary in some universal gate set. In the quantum grid model, we allow quantum communication, so in one time step, a processor may send a quantum message of size $\log N$ qubits to one of its neighbours. We do allow for entanglement between processors, but not for free. We suppose the processors begin with their quantum memories in separable states, and entanglement between processors must be produced through quantum communication.

Several other paradigms of many-processor computation have been considered in the past.

An early model of parallel computing, the *PRAM model* [23], is the parallel analogue of the RAM model. In this model, some number of processors have random access to a shared memory. Communication between any two processors is accomplished through this shared memory. In an *ideal PRAM machine* there are an unlimited number of processors (similar to our situation, where the number of processors can grow with the problem) and unlimited shared random access memory. This model differs significantly from ours in that it allows all processors to have constant cost random access to the shared memory, not accounting for the distance the information must travel. In particular, two processors in this model may communicate in constant time, which is physically unrealistic.

However, there are interesting relationships between this model and decision tree complexity. In particular, Nisan showed [42] that the complexity of a Boolean function $\mathbf{F}$ in this model is $\Theta(\log D(\mathbf{F}))$, which he shows using block sensitivity (see Chapter 7).

Another related area is that of *distributed computing*. In this model, a network of independent machines is connected by some graph, and attempt to carry out a joint computation. It is similar to our model in the sense that each processor has its own memory

and can only communicate with its neighbours by sending messages. However the spirit of this paradigm is quite different from ours, in that the individual machines are generally considered to be quite separate, possibly belonging to different users, and the problems considered in this model are generally related to properties of the connection graph itself (for example, finding a shortest path). In addition, in this model there is no bound on the memory of individual machines.

There are many variations on these two broad ideas. For a thorough survey, see [37].

2.5 Summary of Models

The locality-sensitive models and grid models are related to their query counterparts, in the following ways:

Table 2.1: Relationships between models

$D_M^{\boxplus}(\mathbf{F}) \geq \frac{D(\mathbf{F})}{M} + \sqrt{M}$	$\overline{D}(\mathbf{F}) \geq D(\mathbf{F})$
$R_M^{\boxplus}(\mathbf{F}) \geq \frac{R(\mathbf{F})}{M} + \sqrt{M}$	$\overline{R}(\mathbf{F}) \geq R(\mathbf{F})$
$Q_M^{\boxplus}(\mathbf{F}) \geq \frac{Q(\mathbf{F})}{M} + \sqrt{M}$	$\overline{Q}(\mathbf{F}) \geq Q(\mathbf{F})$

These relationships produce trivial lower bounds in the new models, discussed more in Chapter 8.

Chapter 3

Oracle Problems

In this section, we give precise definitions of the problems we will be studying, including several variations on collision finding and related problems. We present several lemmas on the relationships between these problems.

Definition 3.0.1. *Given two problems,* $\mathbf{F}_N$ *and* $\mathbf{G}_N$*, we say* $\mathbf{F}_N$ poly-logarithmically reduces *to* $\mathbf{G}_N$ *if, given an algorithm* A *that solves* $\mathbf{G}_N$*, we can solve* $\mathbf{F}_N$ *using poly*$(\log N)$ *calls to* A *and poly*$(\log N)$ *other operations.*

$\mathbf{F}_N$ *and* $\mathbf{G}_N$ *are* poly-logarithmically equivalent *if* $\mathbf{F}_N$ *reduces to* $\mathbf{G}_N$ *and* $\mathbf{G}_N$ *reduces to* $\mathbf{F}_N$*.*

The following problem, often called *search* or *black-box search* is the most fundamental black-box problem.

Problem 1 $\mathbf{OR}_N$

Given a function $f : [N] \to \{0, 1\}$, return $x \in [N]$ such that $f(x) = 1$, or output that none exists. An element $x \in [N]$ such that $f(x) = 1$ is called a *marked element.*

As the name suggests, the problem of $\mathbf{OR}_N$ is poly-logarithmically equivalent to calculating $f(1) \vee \cdots \vee f(N)$. Certainly if we can compute $\mathbf{OR}_N$ as defined above, we can compute $f(1) \vee \cdots \vee f(N)$. Similarly, if we have a method A for computing $f(1) \vee \cdots \vee f(N)$, we can compute $\mathbf{OR}_N(f)$ using a logarithmic number of calls to A: we simply do a binary search of $[N]$ for an input that evaluates to 1.

We now introduce the main problem of interest: collision finding.

Problem 2 $\mathbf{COL}_N$

Given a 2-to-1 function $f : [N] \to [R]$, return a pair (x, y), $x \neq y$ such that $f(x) = f(y)$. Such a pair is called a *collision*. If (x, y) is a collision, we write $x \sim y$ or $y = \tilde{x}$.

We could, more generally, consider functions $f : X \to Y$ where $|X| = N$ and $|Y| = R$, however, in order to consider such an input and output from a black-box, we would need a fixed binary encoding of the elements of X and Y. We therefore consider all such functions as having domain $[N]$ and range $[R]$. Additionally, we assume that we can efficiently store elements of $[R]$, so we suppose that R is polynomial in N. Then $\log R \in \Theta(\log N)$. In practice it is most common to look for collisions in hash functions with $R << N$, so we generally need not be concerned with the size of R, since $[R] \subset [N]$ in this case, so we can even consider $f : [N] \to [N]$. In theory we may not want to make this assumption.

In the *comparison query* model, since we do not necessarily deal with any elements of Y, we do not assume they have some encoding, efficient or otherwise, however in this thesis we will not consider this model.

The problem $\mathbf{COL}_N$, is of a somewhat theoretical nature in that it assumes that the input is a 2-to-1 function, a very precise structure that may not often come up in practice. We therefore consider a broader class of functions that behave similarly.

Definition 3.0.2. *Fix an appropriate constant c. We say a function f is* almost strictly many-to-1 *if*

$$\left|\{y \in \mathrm{im}(f) : \left|f^{-1}(y)\right| = 1\}\right| \leq c \log N$$

That is, there are at most $c \log N$ elements in the image of f with a unique pre-image. We will refer to these unique pre-images as single *elements.*

Problem 3 $\widetilde{\mathbf{COL}}_N$

Given a function $f : [N] \to [R]$ that is *almost strictly many-to*-1, return a collision.

The notion of an almost strictly many-to-1 function captures k-to-1 functions for any constant integer $k > 1$, as well as functions that are similar to such a function in the sense that they deviate from such a function in a logarithmic number of positions. This is desirable, since we would like the definition to capture anything that is intuitively close to a k-to-1 function. More generally, the definition includes any function where *most* elements of $[N]$ are part of at least one collision pair. Intuitively, it includes 2-to-1 functions, functions that are very close to 2-to-1, and anything *easier* to find collisions in.

This broader class of functions shares some properties with 2-to-1 functions that algorithms discussed in Chapter 4 will make use of. In particular, a famous theorem, often called the *birthday paradox*, states that if we make about $\sqrt{N}$ uniform independent choices from a set of size N, then we will make the same choice more than once with constant probability. Querying a random element in $[N]$ is the same as choosing a random element from $\text{im}(f)$, though not necessarily uniformly. In Appendix A, we prove some variations of the birthday paradox that are used to prove the following two lemmas.

Lemma 3.0.3. *Suppose f is almost strictly many-to-1. Then the probability that a uniform random subset of $[N]$ of size $S \in O(\sqrt{N})$ contains a collision is $\Omega(\frac{S^2}{N})$. In particular, the probability that a random pair $(x, y) \in [N] \times [N]$ is a collision is $\Omega(\frac{1}{N})$.*

Proof. Let $\mathcal{S}$ be a set of size S. We will consider adding elements to $\mathcal{S}$ in the sequence $x_1, \ldots, x_S$.

Let X denote the number of single elements in $\mathcal{S}$. By Markov's inequality, we have:

$$\Pr[X \geq 1] \leq E[X] \leq \frac{S}{N} c \log N \in O\left(\frac{\sqrt{N} \log N}{N}\right)$$

So the probability that $\mathcal{S}$ has no single elements is at least $\frac{1}{2}$ (for sufficiently large N) so we will condition on this being true, to simplify our analysis.

Suppose there is no collision in $x_1, \ldots, x_k$. Then the probability that x_{k+1} collides with one of $x_1, \ldots, x_k$ is at least $\frac{k}{N}$, since each x_i has at least one (distinct) element in $[N]$ with which it collides. Thus, the probability that there is no collision in $\mathcal{S}$ is:

$$\begin{aligned}
\Pi_{k=1}^{S-1}\left(1 - \frac{k}{N}\right) &\leq \Pi_{k=1}^{S-1} e^{-\frac{k}{N}} \\
&= \exp\left(-\sum_{k=1}^{S-1} \frac{k}{N}\right) \\
&= \exp\left(-\frac{S(S-1)}{2N}\right) \\
&\leq 1 - \frac{S(S-1)}{4N}
\end{aligned}$$

Thus, the probability that there is a collision in $\mathcal{S}$ is at least $\frac{S(S-1)}{4N} \in \Omega(\frac{S^2}{N})$.

Setting $\mathcal{S} = \{x, y\}$, we have $\Pr[x \neq y \wedge f(x) = f(y)] \in \Omega(\frac{1}{N})$. □

Lemma 3.0.4. *Suppose f is almost strictly many-to-1. Fix a set $\mathcal{S} \subseteq [N]$ of size $S \geq c' \log N$ for $c' > c$, with no collision. The probability that a uniform random $X \in [N]$ collides with an element of $\mathcal{S}$ is at least $\Omega(\frac{S}{N})$.*

Furthermore, fix $y \in [N]$ such that y is not single. The probability that a uniform random $X \in [N]$ collides with y is $\Omega(\frac{1}{N})$.

Proof. Let $\mathcal{S}$ be a subset of $[N]$ of size $S \geq c' \log N$. Then it has at least $S - c \log N \geq \frac{c'-c}{c} S$ non-single elements. Each of these has at least one distinct element in $[N]$ that collides with it, so there are at least $\Theta(S)$ elements in $[N]$ that collide with an element in $\mathcal{S}$. The probability of choosing one is at least $\Omega(\frac{S}{N})$.

If y is not single then there is at least one element in $[N]$ that collides with it, so the probability of choosing such an element is $\Omega(\frac{1}{N})$. □

It seems that in practice, $\widetilde{\mathbf{COL}}_N$ is what we want to solve, however the nice structure of exactly 2-to-1 functions makes $\mathbf{COL}_N$ easier to work with. Certainly for lower bounds we can simply consider $\mathbf{COL}_N$, since the set of 2-to-1 functions is a subset of the set of almost strictly many-to-1 functions, so lower bounds for $\mathbf{COL}_N$ are lower bounds for $\widetilde{\mathbf{COL}}_N$. Intuitively, a nondeterministic algorithm that solves $\mathbf{COL}_N$ in worst case expected time T should also solve $\widetilde{\mathbf{COL}}_N$ in worst case expected time T, however it is possible that some algorithm, particularly a deterministic algorithm, may exploit the exact structure of 2-to-1 functions in a way that cannot be applied to almost strictly many-to-1 functions, so we cannot say that these problems are equivalent. However, since most of our analyses are based on the probability of some pair being a collision, they mostly apply to $\widetilde{\mathbf{COL}}_N$. Thus, while we will generally consider the problem $\mathbf{COL}_N$, we will note where an algorithm works just as well for $\widetilde{\mathbf{COL}}_N$.

A related problem to collision finding is that of element distinctness. It is of great theoretical interest, having been used to prove lower bounds on sorting. It is somehow a more general version of $\widetilde{\mathbf{COL}}_N$, since it places no restriction on the input function.

Problem 4 $\mathbf{ED}_N$

Given a function $f : [N] \to [R]$, determine whether or not it is 1-to-1. Equivalently, given a function $f : [N] \to [R]$ return a collision pair.

The problem $\widetilde{\mathbf{COL}}_N$ (and hence $\mathbf{COL}_N$) is a special case of $\mathbf{ED}_N$, but $\mathbf{ED}_N$ is strictly more difficult, since every allowed input to $\widetilde{\mathbf{COL}}_N$ has the property that it has many

collisions, whereas an input to $\mathbf{ED}_N$ may have only a constant number of collisions. Such inputs are intuitively the most difficult, and we therefore often think of inputs to this problem as having only a constant number of collisions.

We have defined $\mathbf{ED}_N$ in two equivalent ways. We now show that these are, in fact, equivalent.

Claim 3.0.5. *The following problems are poly-logarithmically equivalent:*

1. *Given a function $f : [N] \to [R]$, determine whether or not f is 1-to-1, and*
2. *Given a function $f : [N] \to [R]$, return a collision in f if there is one.*

Proof. Given a procedure for solving the second problem, we can clearly determine whether f is 1-to-1 in a single call.

Suppose we have a procedure A that outputs 1 if a function is 1-to-1 and 0 otherwise. We will show how to use this procedure to find a collision in a function $f : [N] \to [R]$. Let $n = \log N$. We can think of f as a function on $\{0,1\}^n$.

Let $\mathcal{S}(z) := \{w \in \{0,1\}^n : z \text{ is a prefix of } w\}$.

Input: A black-box $f : \{0,1\}^n \to [R]$.
Output: A collision pair (x, y)

---

1. **if** $\mathsf{A}(f) = 1$ **then** return 1-to-1
2. **if** $\mathsf{A}(f|_{\mathcal{S}(0)}) = 0$ **then** $x_1 := y_1 := 0$
3. **else if** $\mathsf{A}(f|_{\mathcal{S}(1)}) = 0$ **then** $x_1 := y_1 := 1$
4. **else** $x_1 := 0$, $y_1 := 1$
5. **for** $i := 2, \ldots, n$
   (a) **if** $\mathsf{A}(f|_{\mathcal{S}(x_1 \ldots x_{i-1}0) \cup \mathcal{S}(y_1 \ldots y_{i-1}0)}) = 0$ **then** $x_i := y_i := 0$
   (b) **else if** $\mathsf{A}(f|_{\mathcal{S}(x_1 \ldots x_{i-1}1) \cup \mathcal{S}(y_1 \ldots y_{i-1}1)}) = 0$ **then** $x_i := y_i := 1$
   (c) **else if** $\mathsf{A}(f|_{\mathcal{S}(x_1 \ldots x_{i-1}0) \cup \mathcal{S}(y_1 \ldots y_{i-1}1)}) = 0$ **then** $x_i := 0$, $y_i := 1$
   (d) **else** $x_i := 1$, $y_i := 0$
6. return x, y

Note that the above procedure makes $\Theta(\log N)$ calls to A. We now argue that after any step i, $x_1 \dots x_i$ and $y_1 \dots y_i$ are consistent with some collision pair. As a base case, consider x_1 and y_1. If $\mathsf{A}(f|_{S(b)}) = 0$ for some bit b, then there is a collision in f with $x, y \in \mathcal{S}(b)$ so there is some collision with $x_1 = y_1 = b$. Otherwise, there is no collision in which x and y both have the same bit, so each collision pair differ in the first bit. Therefore, we can set $x_1 = 0$ and $y_1 = 1$.

Consider step $i > 1$. If $\mathsf{A}(f|_{\mathcal{S}(x_1 \dots x_{i-1}b) \cup \mathcal{S}(y_1 \dots y_{i-1}b)}) = 0$ for some bit b, then if $x_1 \dots x_{i-1} = y_1 \dots y_{i-1}$ then clearly we have a collision pair where each of x and y have prefix $x_1 \dots x_{i-1}b$. If $x_1 \dots x_{i-1} \neq y_1 \dots y_{i-1}$, this could only happen if there is no collision pair with both x and y have prefix $x_1 \dots x_{i-1}$ or $y_1 \dots y_{i-1}$, so if there is a collision in this set, then it must be that exactly one of the pair has prefix $x_1 \dots x_{i-1}b$ and the other $y_1 \dots y_{i-1}b$.

If $\mathsf{A}(f|_{\mathcal{S}(x_1 \dots x_{i-1}b) \cup \mathcal{S}(y_1 \dots y_{i-1}b)}) = 1$ for both choices of b, then it must be the case that $\mathsf{A}(f|_{\mathcal{S}(x_1 \dots x_{i-1}b) \cup \mathcal{S}(y_1 \dots y_{i-1}\bar{b})}) = 0$ for some b, since $x_1 \dots x_{i-1}$ and $y_1 \dots y_{i-1}$ are prefixes of some collision pair. In this case, any collision pair from this set must have exactly one of the pair with prefix $x_1 \dots x_{i-1}b$ and the other with prefix $y_1 \dots y_{i-1}\bar{b}$, since, in particular, they may not have the same i^{th} bit, since this would have been caught by one of the first two branches.

The result follows. □

We now consider the relationships between the problems of element distinctness, collision finding, and black-box search. We first introduce some necessary definitions.

Definition 3.0.6. *A distribution ρ on $V \in \mathcal{X}^\ell$ is* uniform t-wise independent *if for any $k \leq t$, for any $\alpha \in \mathcal{X}^k$ and $\mathcal{I} \subseteq [\ell]$ with $|\mathcal{I}| = k$ we have:*

$$\Pr_\rho[V_\mathcal{I} = \alpha] = \frac{1}{|\mathcal{X}|^k}$$

That is, the marginal distribution on any subvector of length up to t is uniform.

A random function $f : [N] \to [R]$ is a random variable over all functions $[N] \to [R]$. A random function $f : [N] \to [R]$ is uniform t-wise independent *if $(f(1), \dots, f(N))$ is t-wise independent on $[R]^N$. A set of functions $\mathcal{F}$ such that a uniform random variable on $\mathcal{F}$ is uniform t-wise independent is called a* uniform t-wise independent family of functions.

The property of t-wise independence for some $t < \ell$ is referred to as limited independence.

Fact 3.0.7. *There exist uniform t-wise independent families of functions $\mathcal{F}$, such that each function in $\mathcal{F}$ can be stored using $\Theta(t \log N)$ space.*

The concept of limited independence will be of importance in several sections of this thesis, since it provides a form of randomness that can be stored efficiently. We now use it to show a relationship between $\mathbf{ED}_N$ and $\widetilde{\mathbf{COL}}_N$.

Lemma 3.0.8. *Suppose, for any positive integer N, we have an algorithm that solves $\mathbf{ED}_N$ using $T(N) \in \Theta(N^\epsilon)$ queries and S space. Then we can construct an algorithm that solves $\widetilde{\mathbf{COL}}_N$ using $\tilde{\Theta}(T(\sqrt{N})) \in \tilde{\Theta}(\sqrt{T(N)})$ queries and space $\tilde{\Theta}(S)$.*

Proof. Suppose we have an algorithm A that solves $\mathbf{ED}_N$ using $T(N) \in \Theta(N^\epsilon)$ queries. Note that A solves $\mathbf{ED}_{\sqrt{N}}$ in $T(\sqrt{N}) \in \Theta(\sqrt{N}^\epsilon) \in \Theta(\sqrt{N^\epsilon})$ queries. That is, $T(\sqrt{N}) \in \Theta(\sqrt{T(N)})$.

Let $f : [N] \to [R]$ be almost strictly many-to-1. We define a subset of $[N]$ of size approximately $c\sqrt{N}$ by $\mathcal{S} = \{h(1), \ldots, h(c\sqrt{N})\} = \{h_1, \ldots, h_k\}$ for some constant c and some uniform 4-wise independent $h : [N] \to [N]$. Since $\{f(h_1), \ldots, f(h_k)\}$ is a set of about $c\sqrt{N}$ (by Lemma A.0.1) 4-wise independent random variables, $f|_{\mathcal{S}}$ has at least 1 collision with high probability, by Lemma A.0.4, which is a variation of the birthday paradox.

We can therefore run A on $f \circ h : [\sqrt{N}] \to [N]$, costing $\Theta(\sqrt{T(N)})$. □

We now compare element distinctness and collision finding to search.

Lemma 3.0.9. *Suppose, for any positive integer N, we have an algorithm for $\mathbf{OR}_N$ using space S and $T(N)$ queries. Then:*

1. *we can solve $\mathbf{ED}_N$ using space $\Theta(S)$ and $\Theta(T(N^2))$ queries, and*
2. *we can solve $\widetilde{\mathbf{COL}}_N$ using space $\Theta(S)$ and $\tilde{\Theta}(T(N))$ queries.*

Proof. Given a function $f : [N] \to [R]$, define $g : [N] \times [N] \to \{0, 1\}$ as follows:

$$g(x, y) = \begin{cases} 1 & \text{if } x \neq y \text{ and } f(x) = f(y) \\ 0 & \text{else} \end{cases}$$

We can find a collision in f by performing black-box search on g, using $T(N^2)$ queries to g. Each query to g costs 2 f-queries. Thus, we can find a collision in f using $2T(N^2)$ f-queries.

Given an almost strictly many-to-1 function $f : [N] \to [R]$ and some $z \in [N]$, define $g_z : [N] \to \{0, 1\}$ as follows:

$$g_z(x) = \begin{cases} 1 & \text{if } x \neq z \text{ and } f(x) = f(z) \\ 0 & \text{else} \end{cases}$$

We can find a collision in f by choosing $z \in [N]$ and performing black-box search on g_z. Since f is almost strictly many-to-1, there are at most $O(\log N)$ elements in $[N]$ that are not part of a collision. Thus, we need repeat this process at most $O(\log N)$ times. Each times costs $T(N)$ queries to g_z, each of which costs 1 query to f. Thus we can find a collision in f in at most $\Theta(T(N) \log N) \in \tilde{\Theta}(T(N))$. □

Sometimes, if we know something about the number of marked elements in the input, we can expect a search algorithm to terminate in fewer steps. That is, the number of queries may depend on k (which we may or may not need to know in advance), the Hamming weight of the input (number of marked elements) as well as N. We then denote the cost of the algorithm as $T(N, k)$ to reflect this. If we have no guarantees on the Hamming weight of the input, we have worst case query complexity $T(N) = T(N, 1)$. Using this new cost function, we can get a second type of reduction from $\widetilde{\mathbf{COL}}_N$ to $\mathbf{OR}_N$.

Lemma 3.0.10. *Suppose we have an algorithm for $\mathbf{OR}_N$ using space S and $T(N, k)$ queries, where k is the Hamming weight of the input. Then we can solve $\widetilde{\mathbf{COL}}_N$ using space $\Theta(S)$ and $\Theta(T(N^2, N))$ queries.*

Proof. This follows from the same reduction used for $\mathbf{ED}_N$ in the above proof, and the fact that an input to $\widetilde{\mathbf{COL}}_N$ has $\Omega(N)$ collision pairs. □

Chapter 4

A Survey of ED and Collision Algorithms

In this chapter we survey known algorithms for collision finding and element distinctness and relate them to the grid model, with the exception of the rho algorithm, which we discuss in great detail in Chapter 5.

We begin by outlining optimal algorithms for $\mathbf{OR}_N$ in each model of computation.

4.1 Search-Based Methods

Since $\mathbf{ED}_N$ and $\mathbf{COL}_N$ can be reduced to $\mathbf{OR}_N$ (Lemmas 3.0.9 and 3.0.10), we begin by describing optimal methods of solving $\mathbf{OR}_N$ in the deterministic, randomized, and quantum models, and their grid counterparts. Each of these methods yields methods for $\mathbf{ED}_N$ and $\mathbf{COL}_N$ respectively. We summarize the upper bounds implied by these algorithms in Table 4.1.

4.1.1 Deterministic Search

It is not difficult to see that any deterministic algorithm for $\mathbf{OR}_N$ has worst case query complexity N. The optimal deterministic algorithm is therefore to query everything in $[N]$ until a marked element is found. If there are k marked elements, then the worst case query complexity is $T(N, k) = N - k + 1$.

Space: $\log N$
Query Complexity: $D(\mathbf{OR}_N) \in \Theta(N - k)$

4.1.2 Randomized Search

A randomized algorithm for $\mathbf{OR}_N$ may *randomly* query independent elements of $[N]$ until a marked element is found, but it must still have worst case query complexity in $\Omega(N)$, the worst case being that of a single marked element. If there are k marked elements, the probability that a random query returns a marked element is $\frac{k}{N}$, and so the expected number of queries is $T(N, k) \in \Theta(\frac{N}{k})$.

Space: $\log N$
Query Complexity: $R(\mathbf{OR}_N) \in \Theta(\frac{N}{k})$

4.1.3 Quantum Search

Using a quantum processor to query in superposition, we can speed things up considerably. Quantum search is one of the poster-algorithms of quantum computing, achieving a quadratic speedup over any classical search method. It was first developed by Lov Grover [24] and later generalized and analyzed by [18, 19] to a process called amplitude amplification. Given a quantum process expressed as a unitary U such that measuring $\mathsf{U}|0\rangle$ yields a marked element with probability p and a black-box f which outputs 1 if and only if the input is marked, the amplitude amplification algorithm outputs a marked element with at least constant probability after $\Theta(\frac{1}{\sqrt{p}})$ queries to f.

Algorithm 1 AmplitudeAmp

Input: A black-box $f : [N] \to \{0, 1\}$ and a unitary U

1. Define unitary operator S by $S|0\rangle = -|0\rangle$ and $S|x\rangle = |x\rangle$ for $x \neq 0 \in [N]$
2. $|\psi^0\rangle := \mathsf{U}|0\rangle$
3. **for** $t := 1 \ldots \frac{1}{\sqrt{p}}$

   $|\psi^t\rangle := \mathsf{U}S\mathsf{U}^{-1}\mathcal{O}_f|\psi^{t-1}\rangle$

Given a process that succeeds with probability p, we can amplify the probability of success by repeating the process. The trick to getting the quantum speedup is to amplify the *amplitude* of the marked states, rather than the *probability* of getting a marked state.

Let $M = \{x \in [N] : f(x) = 1\}$ be the set of marked elements, and $\overline{M} = [N] \setminus M$ the set of unmarked elements. Let $|M\rangle$ be the normalized projection of $|\psi^0\rangle$ onto $\text{span}\{|x\rangle : x \in M\}$ and $|\overline{M}\rangle$ the normalized projection of $|\psi^0\rangle$ onto $\text{span}\{|x\rangle : x \in \overline{M}\}$. Then we have $|\langle M|\psi^0\rangle|^2 = p$, by assumption.

We can consider each iteration as two reflections, $\mathcal{O}_f$ and $\mathsf{U}S\mathsf{U}^{-1}$. Operator $\mathcal{O}_f$ has no effect on the magnitude of either the amplitude of $|M\rangle$ or the amplitude of $|\overline{M}\rangle$, it merely changes the relative phase by putting a -1 in front of $|M\rangle$. It is the reflection about $|M\rangle$: $I - 2|M\rangle\langle M|$. The second reflection is the reflection about $|\psi^0\rangle$, $I - 2|\psi^0\rangle\langle\psi^0|$. Let $\sin\frac{\theta_t}{2} = \langle M|\psi^t\rangle$, so the probability of measuring a marked state if we measure $|\psi^t\rangle$ is $\sin^2\frac{\theta_t}{2}$, and $p = \sin^2\frac{\theta_0}{2}$. We have:

$$
\begin{aligned}
|\psi^{t+1}\rangle &= \mathsf{U}S\mathsf{U}^{-1}\mathcal{O}_f|\psi^t\rangle \\
&= \mathsf{U}S\mathsf{U}^{-1}(|\psi^t\rangle - 2|M\rangle\langle M|\psi^t\rangle) = \mathsf{U}S\mathsf{U}^{-1}(|\psi^t\rangle - 2\sin\frac{\theta_t}{2}|M\rangle) \\
&= |\psi^t\rangle - 2\sin\frac{\theta_t}{2}|M\rangle - 2|\psi^0\rangle\langle\psi^0|\psi^t\rangle + 4\sin\frac{\theta_t}{2}|\psi^0\rangle\langle\psi^0|M\rangle \\
&= |\psi^t\rangle - 2\sin\frac{\theta_t}{2}|M\rangle - 2\left[\cos\frac{\theta_0}{2}\cos\frac{\theta_t}{2} + \sin\frac{\theta_0}{2}\sin\frac{\theta_t}{2}\right]|\psi^0\rangle + 4\sin\frac{\theta_t}{2}\sin\frac{\theta_0}{2}|\psi^0\rangle \\
&= |\psi^t\rangle - 2\sin\frac{\theta_t}{2}|M\rangle + 2\left[\sin\frac{\theta_0}{2}\sin\frac{\theta_t}{2} - \cos\frac{\theta_0}{2}\cos\frac{\theta_t}{2}\right]|\psi^0\rangle
\end{aligned}
$$

$$= |\psi^t\rangle - 2\sin\frac{\theta_t}{2}|M\rangle + \left[\cos\frac{\theta_t-\theta_0}{2} - \cos\frac{\theta_t+\theta_0}{2} - \cos\frac{\theta_t-\theta_0}{2} - \cos\frac{\theta_t+\theta_0}{2}\right]|\psi^0\rangle$$

$$= |\psi^t\rangle - 2\sin\frac{\theta_t}{2}|M\rangle - 2\cos\frac{\theta_t+\theta_0}{2}|\psi^0\rangle$$

Then we have:

$$\left|\langle M|\psi^{t+1}\rangle\right| = \left|\langle M|\psi^t\rangle - 2\sin\frac{\theta_t}{2}\langle M|M\rangle - 2\cos\frac{\theta_t+\theta_0}{2}\langle M|\psi^0\rangle\right|$$

$$= \left|\sin\frac{\theta_t}{2} - 2\sin\frac{\theta_t}{2} - 2\cos\frac{\theta_t+\theta_0}{2}\sin\frac{\theta_0}{2}\right|$$

$$= \left|\sin\frac{\theta_t}{2} + \sin\frac{\theta_t+\theta_0+\theta_0}{2} - \sin\frac{\theta_t+\theta_0-\theta_0}{2}\right|$$

$$= \left|\sin\frac{\theta_t+2\theta_0}{2}\right|$$

Every iteration, the angle increases by $\theta_0 = 2\sin^{-1}\sqrt{p}$, so the probability of measuring a marked state if we measure at step $t+1$ is $\sin^2(t\theta_0 + \frac{\theta_0}{2})$. This means that if we iterate too many times, the success probability actually starts to decrease again. To combat this, we can begin by assuming there are many marked elements and run the algorithm for a random number of iterations from some constant interval $[1, c]$. If this does not succeed, double the interval size and repeat [18, 19].

Note that if we input $\mathsf{U} = H^{\otimes n}$, the n-fold Hadamard gate, for $n = \log N$, then $\mathsf{U}|0\rangle$ is a uniform superposition over all elements in the basis $\{|x\rangle\}_{x\in[N]}$. When we measure $\mathsf{U}|0\rangle$ in the basis $\{|x\rangle\}_{x\in[N]}$, we get any outcome $x \in [N]$ with probability $\frac{1}{p} = \frac{1}{N}$. In particular, if there is a single $x \in [N]$ such that $f(x) = 1$, then this process is the standard quantum search algorithm of [24]. More generally, if there are k marked elements with respect to f, then measuring the uniform superposition gives a marked element with probability $\frac{k}{N}$. The algorithm therefore finds a marked element in expected $\sqrt{\frac{N}{k}}$ queries. This is optimal [15, 18, 55].

Space: $\log N$
Query Complexity: $Q(\mathbf{OR}_N) \in \Theta\left(\sqrt{\frac{N}{k}}\right)$

More generally, we can use AmplitudeAmp to reduce the number of iterations of any probabilistic process quadratically. We will see this tool come up in various algorithms throughout this thesis.

4.1.4 Deterministic Grid Search

A deterministic algorithm must query every element in the worst case, but it is possible to divide this work among M processors. For $j = 1, \ldots, M$, processor j queries the elements $(j-1)\frac{N}{M}+1, \ldots, j\frac{N}{M}$. If there are k marked items, then there must be at least one processor whose search space contains at least $\frac{k}{M}$ marked elements, so some processor is guaranteed to find a marked element after $\frac{N}{M} - \frac{k}{M}$ queries. Thus, the grid complexity of this algorithm is $T(N, k, M) = \Theta(\frac{N-k}{M} + \sqrt{M})$. The optimal number of processors is thus $M = (N-k)^{2/3}$, in which case $T(N, k) \in \Theta((N-k)^{1/3})$.

M-Grid Complexity: $D^{\boxplus}_M(\mathbf{OR}_N) \in \Theta(\frac{N-k}{M} + \sqrt{M})$ **Grid Complexity:** $D^{\boxplus}(\mathbf{OR}_N) \in \Theta((N-k)^{1/3})$ (when $M = (N-k)^{2/3}$)

4.1.5 Randomized Grid Search

We can similarly spread the required $\Theta(\frac{N}{k})$ randomized queries among M processors in the randomized model. In this setting, at each step, each processor makes a random query. At each step, there is $\Theta(\frac{Mk}{N})$ probability that a marked element will be found. Each processor thus needs to make about $\Theta(\frac{N}{Mk})$ queries before we can expect that one has found a marked element. This costs $T(N, k, M) \in \Theta(\frac{N}{Mk} + \sqrt{M})$, which is optimized when $M = \left(\frac{N}{k}\right)^{2/3}$, giving $T \in \Theta\left(\left(\frac{N}{k}\right)^{1/3}\right)$.

M-Grid Complexity: $R^{\boxplus}_M(\mathbf{OR}_N) \in \Theta(\frac{N}{Mk} + \sqrt{M})$ **Grid Complexity:** $R^{\boxplus}(\mathbf{OR}_N) \in \Theta\left(\left(\frac{N}{k}\right)^{1/3}\right)$ (when $M = \left(\frac{N}{k}\right)^{2/3}$)

4.1.6 Quantum Grid Search

We might try to similarly divide the $\sqrt{\frac{N}{k}}$ required quantum queries among M quantum processors, but according to a result of Zalka [55] (also see Section 7.2.2) this is not possible. The best we can do with many quantum processors is to divide the search space among M processors and have them perform their own independent quantum searches of their subspaces. More simply put (yet equivalently), as in the randomized case, we can have each processor do an independent search (of the entire search space), but only run for

some T steps. In the quantum case, after T queries, a processor has probability $\frac{kT^2}{N}$ of finding a marked element, so the probability that one of the M processors has found a marked element after T steps is about $\frac{kMT^2}{N}$. We thus set $T \in \Theta\left(\sqrt{\frac{N}{kM}}\right)$, that is, we run a quantum search on each processor for $\Theta\left(\sqrt{\frac{N}{kM}}\right)$ queries, taking time $\Theta\left(\sqrt{\frac{N}{kM}} + \sqrt{M}\right)$. This is optimized when $M = \sqrt{\frac{N}{k}}$, giving time $\Theta\left(\left(\frac{N}{k}\right)^{1/4}\right)$.

M-**Grid Complexity:** $Q_M^{\boxplus}(\mathbf{OR}_N) \in \Theta\left(\sqrt{\frac{N}{Mk}} + \sqrt{M}\right)$
Grid Complexity: $Q^{\boxplus}(\mathbf{OR}_N) \in \Theta\left(\left(\frac{N}{k}\right)^{1/4}\right)$ (when $M = \sqrt{\frac{N}{k}}$)

4.1.7 Application to Element Distinctness and Collision Finding

The following table summarizes the upper bounds in each of the six models for both $\mathbf{ED}_N$ and $\widetilde{\mathbf{COL}}_N$ obtained from reduction to search. Each of the single processor algorithms uses logarithmic space.

Table 4.1: Upper Bounds Based on Reduction to Search

	$\mathbf{ED}_N$	$\widetilde{\mathbf{COL}}_N$
D	$O(N^2)$	$\tilde{O}(N)$
R	$O(N^2)$	$O(N)$
Q	$O(N)$	$O(\sqrt{N})$
$D_M^{\boxplus}$	$O(\frac{N^2}{M} + \sqrt{M})$	$\tilde{O}(\frac{N}{M} + \sqrt{M})$
$D^{\boxplus}$	$O(N^{2/3})$	$\tilde{O}(N^{1/3})$
$R_M^{\boxplus}$	$O(\frac{N^2}{M} + \sqrt{M})$	$O(\frac{N}{M} + \sqrt{M})$
$R^{\boxplus}$	$O(N^{2/3})$	$O(N^{1/3})$
$Q_M^{\boxplus}$	$O(\frac{N}{\sqrt{M}} + \sqrt{M})$	$O(\sqrt{\frac{N}{M}} + \sqrt{M})$
$Q^{\boxplus}$	$O(N^{1/2})$	$O(N^{1/4})$

Since the algorithms described for search all use logarithmic space per processor, the query complexity is the same as the locality-sensitive complexity.

The deterministic upper bounds for collision finding are obtained from applying the reduction from Lemma 3.0.9, whereas the other upper bounds for collision finding can be obtained from either reduction.

4.2 Tabular Methods

In the single processor models, we can make use of extra space to exploit some of the structure of the problems of $\mathbf{ED}_N$ and $\mathbf{COL}_N$ that differentiate them from $\mathbf{OR}_{N^2}$. In particular, if we make and store S queries to the input, we have actually queried $\binom{S}{2}$ *pairs* of elements $x \neq y$. These pairs are not independent, but we can still gain something from considering a collection of S queries over simply querying $\frac{S}{2}$ pairs and discarding the result each time. In such a *tabular method*, we collect a table of S query-result pairs $(x, f(x))$ (using space $S \log N \in \tilde{\Theta}(S)$) and look for a collision pair in this table, by sorting the entries with respect to $f(x)$. We denote the cost of one such iteration by $\mathfrak{I}(S)$. Of course, in a locality-sensitive model, we must count the cost of accessing elements in this space, which will affect $\mathfrak{I}$. Here we describe tabular methods in each model, beginning with the single-processor ones, and give both query complexity and locality-sensitive complexity.

4.2.1 Deterministic Tabular Method

We can solve $\mathbf{ED}_N$ (or $\mathbf{COL}_N$) deterministically by querying every element in $[N]$, storing the results in a table of size N, and sorting the table to find any collision. This costs only N queries, but the time for inserting each query result in its sorted order is at least $\tilde{\Theta}(\sqrt{N})$, bringing the total cost up to $\tilde{\Theta}(N\sqrt{N})$.

Space: $\tilde{\Theta}(N)$
Query Complexity: $D(\mathbf{ED}_N) \in O(N)$
Locality-Sensitive Complexity: $\overline{D}(\mathbf{ED}_N) \in O(N^{3/2})$

4.2.2 Randomized Tabular Method

In the randomized tabular method, we collect a table of S uniform independent $(x_i, f(x_i))$ pairs, and search for a collision $f(x_i) = f(x_j)$ by sorting the pairs by the $f(x_i)$ entry. If the probability of finding a collision in a single iteration is p, the total cost is $\frac{1}{p}\mathfrak{I}$.

Algorithm 2 RandomizedTabularCollisionFinding

Input: A black-box $f : [N] \to [R]$

1. **for** $i := 1, \ldots, S$
   (a) choose $x_i \in_R [N]$
   (b) compute $f(x_i)$ and store $(x_i, f(x_i))$
2. Sort the table of $(x_i, f(x_i))$ pairs by $f(x_i)$
3. Output any collision in the computed values
4. If no collision was found, repeat from step 1

First consider $\widetilde{\mathbf{COL}}_N$. By Lemma 3.0.3 we need query only $\Theta(\sqrt{N})$ random elements before we have a very high chance of finding a collision in the set. We therefore consider $S \in O(\sqrt{N})$. A single iteration finds a collision with probability $p \in \Theta(\frac{S^2}{N})$ by Lemma 3.0.3. We therefore repeat the procedure about $\Theta(\frac{N}{S^2})$ times, for a cost of $\Theta(\frac{N}{S^2})\mathfrak{I}$. Each iteration costs S queries, as well as the time required for sorting. In the standard model, this cost is $S \log S$, so we have only logarithmic overhead. However, in a locality-sensitive view of the world, we must account for the distance each entry must travel to reach its sorted position. This makes our total number of operations per iteration $\Theta(S+\sqrt{S}S\log S)$, giving total $\tilde{\Theta}\left(S\sqrt{S}\frac{N}{S^2}\right) \in \tilde{\Theta}\left(\frac{N}{\sqrt{S}}\right)$.

Space: $\tilde{\Theta}(S)$ (for $S \in O(\sqrt{N})$)
Query Complexity: $R(\widetilde{\mathbf{COL}}_N) \in O(\frac{N}{S})$
Optimal Query Complexity: $R(\widetilde{\mathbf{COL}}_N) \in O(\sqrt{N})$ (when $S = \sqrt{N}$)
Locality-Sensitive Complexity: $\overline{R}(\widetilde{\mathbf{COL}}_N) \in \tilde{O}(\frac{N}{\sqrt{S}})$
Optimal Locality-Sensitive Complexity: $\overline{R}(\widetilde{\mathbf{COL}}_N) \in \tilde{O}(N^{3/4})$ (when $S = \sqrt{N}$)

The procedure works for $\mathbf{ED}_N$ as well, but in this case we may have as few as one collision. We thus consider $S \in O(N)$, and each iteration has probability $p \in \Omega(\frac{S^2}{N^2})$ of finding the collision, so $O(\frac{N^2}{S^2})$ iterations are required. The cost per iteration in various models is the same as in the case of $\widetilde{\mathbf{COL}}_N$: S queries, or $\tilde{\Theta}(S\sqrt{S})$ locality-sensitive time.

Space: $\tilde{\Theta}(S)$ (for $S \in O(N)$)
Query Complexity: $R(\mathbf{ED}_N) \in O(\frac{N^2}{S})$
Optimal Query Complexity: $R(\mathbf{ED}_N) \in O(N)$ (when $S = N$)
Locality-Sensitive Complexity: $\overline{R}(\mathbf{ED}_N) \in \tilde{O}(\frac{N^2}{\sqrt{S}})$
Optimal Locality-Sensitive Complexity: $\overline{R}(\mathbf{ED}_N) \in \tilde{O}(N^{3/2})$ (when $S = N$)

Note that since this algorithm is optimal for $\mathbf{ED}_N$ when we use space $S = N$, we end up with essentially the deterministic tabular method. It is interesting to note that randomness does not seem to help solve $\mathbf{ED}_N$ in this setting.

4.2.3 Quantum Tabular Methods and the Algorithm of Brassard, Høyer and Tapp

In the quantum analogue of RandomizedTabularCollisionFinding, we can simply use amplitude amplification on the process of finding a table with a collision in it. We thus only need $\frac{1}{\sqrt{p}}$ iterations, for a cost of $\frac{1}{\sqrt{p}}\mathfrak{I}$. For $\widetilde{\mathbf{COL}}_N$, this gives query complexity $O\left(S\sqrt{\frac{N}{S^2}}\right) \in O(\sqrt{N})$ and locality-sensitive complexity $\tilde{O}(\sqrt{SN})$, so we may as well set $S = O(1)$ and use logarithmic space in either setting. In that case we are simply using quantum search on $[N] \times [N]$, as in Section 4.1. For $\mathbf{ED}_N$, we have a similar situation. The query complexity is $O\left(S\sqrt{\frac{N^2}{S^2}}\right) \in O(N)$ and locality-sensitive complexity is $\tilde{O}(\sqrt{S}N)$. Again, we may as well set S to be constant, which gives us the search based method of Section 4.1.

A more clever tabular method, developed by Brassard, Høyer and Tapp [20], fills a table and then quantum searches for an element that collides with some table element. This reduces the query cost of each iteration to $\mathfrak{I} = 1$, after an initial setup cost of $\mathfrak{S} = S$ queries.

Algorithm 3 BHT

Input: A black-box $f : [N] \to [R]$

1. **for** $i := 1, \ldots, S$
   (a) Choose $x_i \in_R [N]$
   (b) Compute $f(x_i)$ and store $(x_i, f(x_i))$ in the table
2. Sort the table of $(x_i, f(x_i))$ pairs by $f(x_i)$
3. Output any collision in the computed values, or else continue
4. Define the function $h : [N] \to \{0, 1\}$ by $h(x) = 1$ if and only if $\exists i$ such that $x \neq x_i$ and $f(x) = f(x_i)$
5. Quantum search for x such that $h(x) = 1$
6. Compute $f(x)$ and find $(x', f(x))$ in the table
7. Output (x, x')

If p is the probability that a randomly chosen x collides with some element in the table, then the total cost is $\mathfrak{S} + \frac{1}{\sqrt{p}}\mathfrak{I}$. First consider $\widetilde{\mathbf{COL}}_N$. If the table has a collision, we are done, so assume it contains S unique images. If we select a random element, the probability that it collides with an element in the table is $p \in \Theta(\frac{S}{N})$ for almost strictly many-to-1 functions by Lemma 3.0.4. Thus the total cost is $\Theta\left(\mathfrak{S} + \sqrt{\frac{N}{S}}\mathfrak{I}\right)$.

For query complexity, we have $\mathfrak{S} = S$ and $\mathfrak{I} = 1$, so the total cost is $\Theta\left(S + \sqrt{\frac{N}{S}}\right)$, which is optimized at $\Theta(N^{1/3})$ queries by $S = N^{1/3}$. This query complexity is optimal [35, 9]. Even if we include a random access analysis of the initial sorting of $S = N^{1/3}$ elements, it merely adds a logarithmic factor. Similarly, the cost of checking if a single element collides with an element in the sorted table is just $\log S$, again only adding a logarithmic factor.

In a locality-sensitive model, however, we count the sorting in the initial step as costing $\mathfrak{S} = \sqrt{S}S \log S \in \tilde{\Theta}(S^{3/2})$, and each iteration now costs $\mathfrak{I} \in \tilde{\Theta}(\sqrt{S})$, since we potentially have to compare the new element with the furthest table element. This gives a total cost in the locality-sensitive model of $\tilde{\Theta}(S^{3/2} + \sqrt{N})$. From the perspective of this model,

the optimal space is $S = \Theta(1)$, which is the same as choosing an element and quantum searching for a collision with it in the single processor case (using the reduction to search from Lemma 3.0.9).

Query Complexity: $Q(\widetilde{\mathbf{COL}}_N) \in O\left(S + \sqrt{\frac{N}{S}}\right)$
Optimal query complexity: $Q(\widetilde{\mathbf{COL}}_N) \in O(N^{1/3})$ (when $S = N^{1/3}$)
Locality-Sensitive Complexity: $\overline{Q}(\widetilde{\mathbf{COL}}_N) \in O(\sqrt{N})$

For element distinctness, this method is less effective, even from a query complexity perspective. If there is just a single collision in f, then the table only contains an element in the collision pair with probability $p' \in \Theta(\frac{S}{N})$. Conditioned on one collision pair element being in the table, the probability that a random element collides with a table element is $p \in \Theta(\frac{1}{N})$. The cost is thus $(\mathfrak{S} + \frac{1}{\sqrt{p}}\mathfrak{I})\frac{1}{\sqrt{p'}}$. This gives a query complexity of $\Theta(\sqrt{NS} + \frac{N}{\sqrt{S}})$ and locality-sensitive complexity of $\tilde{\Theta}(S\sqrt{N} + N)$, which is $\tilde{\Theta}(N)$ if we set $S \in \Theta(1)$.

Query Complexity: $Q(\mathbf{ED}_N) \in O\left(\sqrt{SN} + \frac{N}{\sqrt{S}}\right)$
Optimal query complexity: $Q(\mathbf{ED}_N) \in O(N^{3/4})$ (when $S = \sqrt{N}$)
Locality-Sensitive Complexity: $\overline{Q}(\mathbf{ED}_N) \in \tilde{O}(N)$

4.2.4 Randomized Grid Tabular Method

In the grid models, each processor is restricted to logarithmic space, however, across all M processors, the machine has a total of $\tilde{\Theta}(M)$ space. We thus replace the notion of space with that of processors, which allows us to perform parallel queries, as well as parallel sorting (for example, [34]), which costs only $\Theta(\sqrt{M})$ steps of computation and communication. Thus, the algorithm is essentially the same as RandomizedTabularCollisionFinding (with $S = M$) but with an iteration cost of only $\mathfrak{I} = \sqrt{M}$, and an added $\sqrt{M}$ for communicating the final answer. The total cost of $\widetilde{\mathbf{COL}}_N$ is thus $\Theta(\mathfrak{I}\frac{1}{p} + \sqrt{M}) = \Theta(\sqrt{M}\frac{N}{M^2} + \sqrt{M}) = \Theta(\frac{N}{M^{3/2}} + \sqrt{M})$.

M**-Grid Complexity:** $R^{\boxplus}_M(\widetilde{\mathbf{COL}}_N) \in O\left(\frac{N}{M^{3/2}} + \sqrt{M}\right)$
Grid Complexity: $R^{\boxplus}(\widetilde{\mathbf{COL}}_N) \in O(N^{1/4})$ (when $M = \sqrt{N}$)

Similarly, for $\mathbf{ED}_N$ we have a cost of $\Theta\left(\mathfrak{J}\frac{1}{p}+\sqrt{M}\right) \in \Theta\left(\sqrt{M}\frac{N^2}{M^2}+\sqrt{M}\right) \in \Theta\left(\frac{N^2}{M^{3/2}}+\sqrt{M}\right)$.

M-**Grid Complexity:** $R_M^{\boxplus}(\mathbf{ED}_N) \in O\left(\frac{N^2}{M^{3/2}}+\sqrt{M}\right)$
Grid Complexity: $R^{\boxplus}(\mathbf{ED}_N) \in O\left(\sqrt{N}\right)$ (when $M=N$)

Note that in both cases we expect to find a collision in the first iteration (or after a constant number of iterations): For $\mathbf{COL}_N$ we have $M=\sqrt{N}$, so $p=\Theta(1)$, and for $\mathbf{ED}_N$ we have $M=N$, so we are actually just querying all elements and sorting them, which works deterministically as well. This is somewhat interesting, because it means that for $\mathbf{ED}_N$, we gain no advantage from allowing randomness unless we have restricted space.

Grid Complexity: $D^{\boxplus}(\mathbf{ED}_N) \in O\left(\sqrt{N}\right)$ (when $M=N$)

4.2.5 Quantum Grid Tabular Method

Since the optimal randomized grid algorithms use enough space that we expect to find a collision in a constant number of iterations, we actually get no improvement over the randomized grid with a basic tabular method, since for $p \in \Theta(1)$ we get $\frac{1}{p} \in \Theta(\frac{1}{\sqrt{p}})$.

To be more precise, if we quantum search for a table of size M that contains a collision, we have cost $\Theta\left(\frac{1}{\sqrt{p}}\mathfrak{J}+\sqrt{M}\right) \in \Theta\left(\frac{1}{\sqrt{p}}\sqrt{M}+\sqrt{M}\right)$. In the case of collision finding, this gives a cost of $\Theta\left(\sqrt{\frac{N}{M^2}}\sqrt{M}+\sqrt{M}\right) \in \Theta\left(\frac{\sqrt{N}}{\sqrt{M}}+\sqrt{M}\right)$ which is optimized at $\Theta\left(N^{1/4}\right)$ when $M=\sqrt{N}$, just as in the randomized algorithm, since you cannot find a table with a collision in faster than constant time.

Similarly, for element distinctness, we have cost $\Theta\left(\sqrt{\frac{N^2}{M^2}}\sqrt{M}+\sqrt{M}\right) \in \Theta\left(\frac{N}{M^{1/2}}+\sqrt{M}\right)$ which is optimized at $\Theta(N)$ by $M=N$.

Remark 4.2.1. *In cases with fewer than optimal processors, the quantum tabular method does do better than the randomized version.*

M-**Grid Complexity:** $Q_M^{\boxplus}(\widetilde{\mathbf{COL}}_N) \in O\left(\frac{\sqrt{N}}{\sqrt{M}}+\sqrt{M}\right)$
Grid Complexity: $Q^{\boxplus}(\widetilde{\mathbf{COL}}_N) \in O(N^{1/4})$ (when $M=N^{1/2}$)

M-**Grid Complexity:** $Q_M^{\boxplus}(\mathbf{ED}_N) \in O\left(\frac{N}{M^{1/2}} + \sqrt{M}\right)$
Grid Complexity: $Q^{\boxplus}(\mathbf{ED}_N) \in O(\sqrt{N})$ (when $M = N$)

Remark 4.2.2. *It is somewhat surprising that the quantum tabular method in the grid setting does no better than a simple quantum grid search. This method does take advantage of the structure of the problem, collecting many values and then comparing them all by sorting, but the cost of this comparison negates any benefit over just having each processor search the space* $[N] \times [N]$ *for a collision pair.*

As in the single processor case, we could try to use the method of Brassard Høyer, and Tapp, but this method actually does not parallelize well. We would have a cost of $\sqrt{M}\frac{1}{\sqrt{p}} + \sqrt{M}$, where p is the probability that a random element collides with a table, but note that this is the same cost as the basic quantum tabular method on the grid, above, except with a smaller p. The problem is, at each iteration, rather than having each processor make a new query, we have just one processor make a new query, which costs just as much, but is less useful.

4.3 Markov Chains and Quantum Walks

In this section we give a tight upper bound on the quantum query complexity of $\mathbf{ED}_N$, due to Ambainis [8], which makes use of the theory of quantum walks, the quantum analogue of a random walk. We begin by giving some necessary definitions and results in the area of Markov chains. For a thorough introduction to the subject, see, for example, [38].

A *Markov chain* is a random process for transitioning between states in some finite space Ω, in which the state at time $t+1$, denoted X_{t+1}, may depend on the state at time t, but is independent of all previous states. Such a chain is represented by a stochastic *transition matrix* $P \in \mathbb{R}^{\Omega\times\Omega}$, where $P[x, y] = \Pr[X_{t+1} = y | X_t = x]$. If the initial state of the system is a random variable on Ω with distribution ρ_0, then the state at time t will have distribution $\rho_0 P^t$.

Definition 4.3.1. *To measure the closeness of two distributions,* ρ_1 *and* ρ_2*, we will use the* total variation distance*:*

$$\|\rho_1 - \rho_2\|_{TV} := \max_{A\subseteq\Omega} |\rho_1(A) - \rho_2(A)| = \frac{1}{2}\sum_{x\in\Omega} |\rho_1(x) - \rho_2(x)|$$

Definition 4.3.2. *The mixing time of a Markov chain P is given by:*

$$t_{mix}(\varepsilon, P) := \min\{t : \sup_{\rho_0} \left\|\rho_0 P^t - \pi\right\|_{TV} \leq \varepsilon\}$$

where π is the stationary distribution of P.

Fact 4.3.3. *Let P be an irreducible, reversible, aperiodic Markov chain, with eigenvalue gap δ. Then*

$$t_{mix}(\varepsilon, P) \leq (\frac{1}{\delta} - 1) \log \frac{1}{2\varepsilon}$$

(See, for example, [38]).

Definition 4.3.4. *For an irreducible Markov chain P with stationary distribution π, the* time-reversal *of P is the Markov chain P^* defined from P by:*

$$P^*[x, y] = \frac{\pi(y)}{\pi(x)} P[y, x]$$

for all states x and y.

A random walk on a graph G is a Markov chain with transitions matrix:

$$P[x, y] = \begin{cases} \frac{1}{\deg(x)} & \text{if } (x, y) \in E(G) \\ 0 & \text{else} \end{cases}$$

For example, we can view the standard tabular method in the randomized single processor model as being a walk on the complete graph with vertex set given by all tables of size S, with self-loops added, which we denote by $K(S)$. That is, we start with a random state in the set of all tables of size S, and then at each step, we choose another state, uniformly at random from all possible states, and check if it is marked, by sorting and looking for a collision. The random walk on a complete graph with self-loops is a symmetric, ergodic Markov chain.

More generally we can construct a random walk on any graph with vertex set consisting of the space we want to search, and consider some of the vertices marked. We can consider simulating this walk until we land on a marked vertex.

There are three costs associated with a Markov process-based algorithm. As before, we have the setup cost $\mathfrak{S}$, which is the cost of constructing an initial state from the desired initial distribution, and storing it in some convenient way, in our case, in sorted order. We

also have the checking cost, $\mathfrak{C}$, the cost of checking whether the current state is marked, and the update cost, $\mathfrak{U}$, the cost of transitioning to a new state and storing it in some convenient way, again in our case in sorted order. Notice that we can move some cost between the updating process and the checking process: by taking more care in the update process we may facilitate the process of checking.

We have the following relationship between Markov chains and search algorithms:

Theorem 4.3.5. *Let P be a symmetric, ergodic Markov chain on state space Ω with eigenvalue gap δ. Let p be the proportion of the state space that is marked. Then there is a randomized algorithm that finds a marked element in expected time $O(\mathfrak{S} + \frac{1}{p}(\frac{1}{\delta}\mathfrak{U} + \mathfrak{C}))$.*

The algorithm is as follows:

Algorithm 4 GenericWalkAlgorithm

Input: A black-box $f : \Omega \to \{0, 1\}$ such that $f(v) = 1$ if and only if v is marked

1. Construct and store an initial state
2. **for** $i := 1, \ldots, \frac{1}{p}$
   (a) **for** $j := 1, \ldots, \frac{1}{\delta}$
      i. Simulate a step of P and update to the new state
   (b) Check if the current state is marked, and if so, output

We have that P is reversible and ergodic, so the mixing time is $t_{mix}(\epsilon, P) \leq (\frac{1}{\delta} - 1) \log \frac{1}{2\epsilon}$. Thus, after $\frac{1}{\delta}$ steps, the distribution of the chain is within a constant of uniform. Therefore, each time the algorithm executes step 2b the current state is nearly a uniform independent sample, and so has probability $\Theta(p)$ of being marked. We therefore run the outer loop approximately $\Theta(\frac{1}{p})$ times.

For example, consider again the random walk on $K(S)$. The complete graph with self-loops has eigenvalue gap $\delta = 1$, since its only nonzero eigenvalue is 1. The probability that a random vertex is marked (contains a collision pair) is $p \in \Theta(\frac{S^2}{N})$ for almost strictly many-to-1 functions, so we get an algorithm for $\widetilde{\mathbf{COL}}_N$ with total cost $O(\mathfrak{S} + \frac{N}{S^2}(\mathfrak{U} + \mathfrak{C}))$ since both setup and update/checking consist of filling a table of S $(x, f(x))$ pairs, we get a cost of $O(\frac{N}{S^2}\mathfrak{I})$, just as in RandomizedTabularMethod. Similarly, the analysis works for $\mathbf{ED}_N$.

We now briefly introduce the quantum analogue of a random walk, a *quantum walk*. For a more thorough introduction, see [30].

In a quantum walk on a graph G, we think of walking on the edges rather than on the vertices. In this way, we have some distribution over pairs (x, y), where we can consider being at vertex y, and having just come from vertex x (or vice versa). Of course, in the quantum version of random walks, we have the quantum analogue of a probability distribution, which is a superposition. Our states $|X_0\rangle, |X_1\rangle, \ldots$ look like $|X_i\rangle = \sum_{(x,y)\in E(G)} \alpha_{x,y}|x, y\rangle$.

If we are searching on a particular graph, we begin in a superposition corresponding to the distribution $\pi(x) = \frac{\deg(x)}{\sum_{y\in V(G)} \deg(y)}$ (or for a more general Markov chain P, π is the stationary distribution), $\sum_{x\in V(G)} \sqrt{\pi(x)}|x, 0\rangle$, and progressively move the amplitude onto the marked vertices (note the similarity to AmplitudeAmp in that we begin in a very mixed state and progressively move amplitude onto the marked states). The setup cost $\mathfrak{S}$ is the cost of constructing this initial state.

There are two types of walk "steps" a quantum walk takes. In the first type, we mix the second register over the neighbours of the vertex in the first register:

$$W_1|x\rangle|0\rangle = |x\rangle \sum_{y\in V(G)} \sqrt{P[x, y]}|y\rangle$$

In the second walk operator, we mix the first register over the neighbours of the vertex in the second register, but this time according to the time-reversed Markov chain P^*:

$$W_2|0\rangle|y\rangle = \sum_{x\in V(G)} \sqrt{P^*[y, x]}|x\rangle|y\rangle$$

A single step of the quantum walk is given by the operation $W_2 S_2 W_2^{-1} W_1 S_1 W_1^{-1}$, where $S_1|x\rangle|0\rangle = -|x\rangle|0\rangle$ and $S_2|0\rangle|y\rangle = -|0\rangle|y\rangle$. The update cost, $\mathfrak{U}$, is therefore the cost of two applications each of W_1 and W_2.

Finally, the checking cost, $\mathfrak{C}$, is the cost of implementing the operation:

$$\mathcal{O}|x\rangle|y\rangle = -|x\rangle|y\rangle$$

if and only of x is marked (notice how we "mark" marked states in the same way as AmplitudeAmp).

The following theorem, which is a generalization of a result of Ambainis in [8], is due to [39] (see also [47]):

Theorem 4.3.6. *Let P be a reversible ergodic Markov chain on state space Ω with eigenvalue gap δ. Let p be the proportion of the state space that is marked. Then there is a quantum algorithm that finds a marked element in expected time $O(\mathfrak{S} + \frac{1}{\sqrt{p}}(\frac{1}{\sqrt{\delta}}\mathfrak{U} + \mathfrak{C}))$.*

The algorithm is simply the quantum walk analogue of Algorithm 4.3.

Consider the quantum walk on $K(S)$. For $\mathbf{ED}_N$, we have $p \in \Theta(\frac{S^2}{N^2})$. The complete graph with self-loops has eigenvalue gap $\delta = 1$. In terms of query cost, the setup costs S queries, the update costs S queries, and the checking requires no extra queries, so we have total cost:

$$\begin{aligned} Q(\mathbf{ED}_N) &= O\left(\mathfrak{S} + \frac{1}{\sqrt{p}}\left(\frac{1}{\sqrt{\delta}}\mathfrak{U} + \mathfrak{C}\right)\right) \\ &= O\left(S + \frac{N}{S}S\right) = O(S + N) \end{aligned}$$

This is optimized by using $S \in \Theta(1)$, which just gives a standard quantum algorithm for $\mathbf{OR}_{N^2}$, just as in the standard quantum tabular method. We actually lost some of our quantum advantage in this setting by having $\delta = 1$, and thus $\delta = \sqrt{\delta}$. One advantage of quantum walks over classical random walks is that they can mix quadratically faster, but we have chosen a walk with constant mixing time, so we have not made use of this advantage. The quantum random walk algorithm of Ambainis improves this shortcoming by walking on a graph that takes longer to mix, but has cheaper transitions. The key is that in a single transition, just one element in the set is changed, and the rest of the set stays the same. This costs just one query. The graph representing this process is the well-studied Johnson graph.

Definition 4.3.7. *A Johnson graph, $J(S, N)$ has vertex set $V := \{\mathcal{S} \subseteq [N] : |\mathcal{S}| = S\}$ and edge set $E = \{(\mathcal{S}, \mathcal{T}) : |\mathcal{S} \cap \mathcal{T}| = S - 1\}$.*

Fact 4.3.8. *The eigenvalue gap of $J(N, S)$ is in $\Theta(\frac{1}{S})$.*

The explicit algorithm is as follows:

Algorithm 5 QuantumWalkED

Input: A black-box $f : [N] \to [R]$

1. Construct the state $\sum_{\mathcal{S}\subseteq[N],|\mathcal{S}|=S} |\mathcal{S}\rangle|0\rangle$, where $\mathcal{S}$ is stored in sorted order
2. **for** $i := 1, \ldots, \frac{1}{p}$
   (a) **for** $j := 1, \ldots, \frac{1}{\delta}$
      i. Perform one step of the quantum walk on $J(N, S)$
   (b) Apply the check operator $\mathcal{O}$

The query complexity, in the case of $\mathbf{ED}_N$ is:

$$Q(\mathbf{ED}_N) \in O\left(\mathfrak{S} + \frac{1}{\sqrt{p}}\left(\frac{1}{\sqrt{\delta}}\mathfrak{U} + \mathfrak{C}\right)\right)$$

$$\in O\left(S + \frac{N}{S}\left(\sqrt{S} + 0\right)\right) \in O\left(S + \frac{N}{\sqrt{S}}\right)$$

This is optimized when $S = N^{2/3}$, giving $Q(\mathbf{ED}_N) \in O(N^{2/3})$.

Even in the RAM model, non-query operations only add a logarithmic overhead. There are some subtle details involving quantum data structures, but roughly speaking, the cost of making the first S queries and sorting them is $\mathfrak{S} \in \Theta(S \log S)$, the cost of querying a new element and putting it in its sorted position is $\Theta(\log S)$, and the cost of checking the sorted data structure for a collision is $\Theta(\log S)$.

However, in a locality-sensitive model, the costs are much higher: we have $\mathfrak{S} \in \Theta(S\sqrt{S}\log S)$, $\mathfrak{U} \in \Theta(\sqrt{S})$. We cannot avoid these extra costs, because checking if an element collides with any of the elements in a table of size S will require information to traverse the table, costing at least $\Theta(\sqrt{S})$. The total cost is therefore:

$$\tilde{O}\left(S\sqrt{S} + \frac{N}{S}\left(\sqrt{S}\sqrt{S}\right)\right) \in \tilde{O}(S\sqrt{S} + N)$$

This is optimized when we use only logarithmic space, in which case we are simply doing a random walk on pairs in $[N] \times [N]$ looking for a marked element. This is a method of quantum search that is equivalent to regular quantum search.

Query Complexity: $Q(\mathbf{ED}_N) \in O\left(S + \frac{N}{\sqrt{S}}\right)$
Optimal Query Complexity: $Q(\mathbf{ED}_N) \in O(N^{2/3})$ (when $S = N^{2/3}$)
Locality-Sensitive Complexity: $\overline{Q}(\mathbf{ED}_N) \in O(N)$

This is an improvement in terms of query complexity, and is actually tight in this regard [35, 9]. However, in the locality-sensitive model, it is no better than reduction to quantum search.

Ambainis' method also works for $\widetilde{\mathbf{COL}}_N$, with the same complexity as the Brassard-Høyer-Tapp method. In this case we have $p \in \Theta(\frac{S^2}{N})$, so the query cost is $O\left(S + \sqrt{\frac{N}{S^2}}\sqrt{S}\right) \in O\left(S + \sqrt{\frac{N}{S}}\right)$, which optimizes to $O(N^{1/3})$ queries when $S = N^{1/3}$. In the locality-sensitive model, we have total cost $O\left(S\sqrt{S} + \sqrt{\frac{N}{S^2}}\left(\sqrt{S}\sqrt{S}\right)\right) \in O(S\sqrt{S}+\sqrt{N})$, which is optimized with logarithmic space, in which case we are simply doing a random walk on $[N] \times [N]$ looking for a collision pair, which is, again, just another reduction to quantum search.

Applications to the quantum grid model face the same short-comings as with BHT. Changing one element costs the same as changing all elements, so we may as well refresh the whole table at each step, which gives us exactly the quantum tabular method.

4.4 Summary

The following table summarizes the best upper bounds in each model for $\mathbf{ED}_N$ and $\widetilde{\mathbf{COL}}_N$:

Table 4.2: Upper Bounds

	$\mathbf{ED}_N$		$\widetilde{\mathbf{COL}}_N$	
D	$\Theta(N)$		$\Theta(N)$	$\lozenge$
R	$\Theta(N)$		$\Theta(\sqrt{N})$	
Q	$\Theta(N^{2/3})$		$\Theta(N^{1/3})$	
$\bar{D}$	$O(N^{3/2})$		$O(N)$	$\lozenge$
$\bar{R}$	$O(N^{3/2})$		$O(N^{3/4})$	
$\bar{Q}$	$O(N)$	$\lozenge$	$O(\sqrt{N})$	$\lozenge$
$D^{\boxplus}_M$	—		$\tilde{O}(\frac{N}{M}+\sqrt{M})$	
$D^{\boxplus}$	$O(N^{1/2})$		$\tilde{O}(N^{1/3})$	$\lozenge$
$R^{\boxplus}_M$	$O(\frac{N^2}{M^{3/2}}+\sqrt{M})$		$O(\frac{N}{M^{3/2}}+\sqrt{M})$	
$R^{\boxplus}$	$O(N^{1/2})$		$O(N^{1/4})$	
$Q^{\boxplus}_M$	$O(\frac{N}{\sqrt{M}}+\sqrt{M})$		$O(\frac{\sqrt{N}}{\sqrt{M}}+\sqrt{M})$	
$Q^{\boxplus}$	$O(N^{1/2})$	$\lozenge$	$O(N^{1/4})$	$\lozenge$

The $\lozenge$ symbol indicates a bound that is no improvement over reduction to search. One pattern to note is that the new models — locality-sensitive and grid — seem to be particularly bad for these problems in the quantum case. For both problems, the known methods do not seem to work any better than a reduction to search in these new models.

It is somewhat surprising that the quantum grid methods do no better than the randomized grid methods. However, what is perhaps more surprising is the algorithm we describe in the next section, which, though a heuristic, appears to do *better* in the randomized grid model than any known quantum method.

Chapter 5

The Rho Method

The fastest collision finding algorithm in practice today is the parallel rho method of van Oorschot and Wiener [51], which is a parallelization of the rho method of collision finding, first introduced by Pollard [44]. The parallel version of this algorithm fits perfectly into the classical grid model, because during most of the computation there is no communication between processors.

Heuristically, the algorithm requires $\Theta\left(\frac{\sqrt{N}}{M}\right)$ queries on each of M log-space processors, and has communication cost $\Theta(\sqrt{M})$, so that the grid complexity is (heuristically) $R_M^{\boxplus}(\mathsf{rho}) \in \Theta\left(\frac{\sqrt{N}}{M} + \sqrt{M}\right)$, which is optimized when $M = N^{1/3}$, giving $R^{\boxplus}(\mathsf{rho}) \in \Theta(N^{1/6})$. However, the expected number of queries has never been rigorously proven. Several attempts at analysis have been made, and there are mounds of evidence that the rho method (and variations) is an *excellent* heuristic, but each analysis requires a random oracle, which would blow up the space required by each processor from $\Theta(\log N)$ to $\Theta(N)$, and correspondingly, increase memory access cost in the grid model. The only analysis (of which we are aware) that does not require a random oracle [26] is based on an assumption about limited independent functions, which we show in Section 5.2.1 not to be true in general. Thus, as of now, the rho method is merely a heuristic, albeit, a very good one.

The layout of this chapter is as follows. We begin by describing the most basic version of the parallel rho method rho and analyse its expected performance on a random 2-to-1 input, showing that in this case it does yield a time-processor tradeoff of $\Theta\left(\frac{\sqrt{N}}{M} + \sqrt{M}\right)$ for $\mathbf{COL}_N$. We can also consider applying this method to the more general problem of $\mathbf{ED}_N$. We show in Section 5.1.2 that on randomized inputs with just a single collision (the hardest cases of $\mathbf{ED}_N$) rho returns a collision in expected time $\Theta\left(\frac{N}{M} + \sqrt{M}\right)$. In

Section 5.2 we describe an elegant variation of the rho method [26, 25] used to solve the discrete logarithm problem, but we show in Section 5.2.1 that the assumption upon which the analysis is based is not generally true.

5.1 Basic Algorithm

In the context of the rho method, it is necessary to consider functions $f : [N] \to [N]$, because we will be composing f with itself many times. This is reasonable, since in applications, we will generally be looking for collisions in hash functions with smaller range than domain. Additionally, if the outputs of f are larger than N, it is not unreasonable in this context to simply hash them back down using some suitable uniform limited independent hash (see Definition 3.0.6).

The rho method works by using the input function itself, f, to walk through the space $[N]$ until the path collides with itself. To visualize this, we use the notion of a function graph.

Definition 5.1.1. *Let $f : [N] \to [N]$. The function graph, G_f, is a directed graph with vertex set $[N]$ and edge set $\{x \to f(x) : x \in [N]\}$.*

For any function f, every vertex in G_f will have out-degree 1. If f is a 2-to-1 function, every vertex will have in-degree 2 or 0. If f is a permutation, every vertex will have in-degree 1, and so G_f will consist of one or more cycles.

Walking along G_f, we have exactly one possible next vertex from vertex x: $f(x)$. Thus the walk is necessarily deterministic. Since G_f is finite, no matter where we begin the walk, we will eventually hit a vertex that we have already seen. At this point, since the walk is deterministic, we will cycle. Assuming that the path did not collide with its first point, it will be shaped like the Greek letter rho (ρ), with a *tail* and a *cycle* (see Figure 5.1).

There are a number of techniques for detecting when a cycle has occurred and recovering the collision that only require keeping track of $\Theta(\log N)$ bits, including the starting point, the number of steps taken, and the current point. We now outline the method of *distinguished points* [21] used in the parallelization of the rho method.

Let $\mathcal{D}$ be a random subset of $[N]$ that is just large enough so that we expect to see a constant number of points in $\mathcal{D}$ over the course of our walk (This requires some idea of how long it will be before we cycle. We discuss this in more detail later). We need some succinct way of determining whether a point is in $\mathcal{D}$, so we suppose it is defined by some

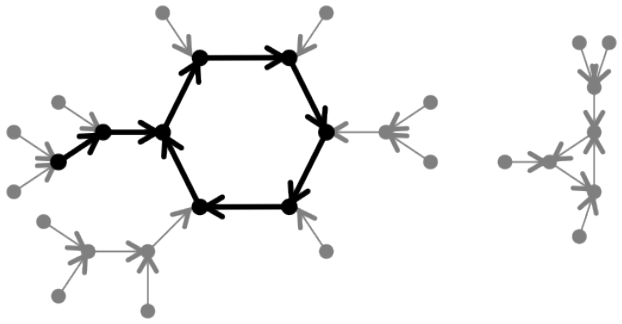

Figure 5.1: A function graph of a 2-to-1 function. A possible path through the graph is shown in bold.

uniform k-wise independent function $h : [N] \to [\frac{N}{|\mathcal{D}|}]$ with $x \in \mathcal{D}$ if and only if $h(x) = 1$, and $k \in O(\log N)$.

A point in $\mathcal{D}$ is said to be *distinguished*. In addition to keeping track of the starting point, the number of steps, and the current point, we also record every distinguished point we see, and at which step we first see it. Let d be the first distinguished point we see twice, say at steps ℓ_1 and ℓ_2. Then the cycle has length $\sigma = \ell_2 - \ell_1$, and so we can find the collision as follows: return to the starting point x_0 and run σ steps to get to x_σ. Keep this path and start a new path at x_0. Run the two paths in parallel until the first τ such that $f(x_\tau) = f(x_{\sigma+\tau})$. Then $(x_\tau, x_{\sigma+\tau})$ is a collision.

We could instead start two paths from distinct starting points x and y and run them until they have both hit the same distinguished point. The result will look like the greek letter lambda (λ), rather than rho, leading some to call this a *lambda method*. This method can be parallelized to an arbitrary number of processors. We outline this method in detail now.

Let $[M]$ index M processors arranged in a grid. Each processor $j \in [M]$ has three registers: $(R^j_{init}, R^j_{curr}, R^j_{count})$, the initial register, the current register, and the count register. Though $x_i^{(j)}$ will be used to denote the value considered by processor j at the i^{th} iteration of phase 1, note that the processor does not keep track of these values, it only stores the contents of the three registers at any given time (and implicitly, the function h that defines $\mathcal{D}$). The algorithm is as follows:

Algorithm 6 rho

Input: A black-box $f : [N] \to [N]$

Phase 1 Each processor $j \in [M]$ does the following:

1. Choose $x_0^{(j)}$ uniformly at random from $[N]$ and initialize memory as $(R^j_{init}, R^j_{curr}, R^j_{count})_j \leftarrow (x_0^{(j)}, x_0^{(j)}, 0)$
2. **while** $R^j_{curr} \notin \mathcal{D}$ **and** $R^j_{count} < max$
   - Compute $x_{i+1}^{(j)} := f(x_i^{(j)}) = f(R^j_{curr})$ (where $i = R^j_{count}$)
   - Store $R^j_{curr} \leftarrow x_{i+1}^{(j)}$
   - $R^j_{count}++$

Phase 2 The processors perform a parallel sort of their triples $(R_{init}, R_{curr}, R_{count})$ by the R_{curr} value. Each processors compares its R_{curr} with those of its neighbours. If two processors j and $j+1$ have the same value for R_{curr}, they perform Phase 3. If a processor has a distinguished point that does not coincide with any of its neighbours, it returns.

Phase 3 If processors j and k have coinciding distinguished points $x_{t_j}^{(j)} = x_{t_k}^{(k)}$ with $t_j \leq t_k$ then they behave as follows:

1. Processor k computes $x_{t_k - t_j}^{(k)} := f^{t_k - t_j}(x_0^{(k)}) = f^{t_k - t_j}(R^k_{init})$ and stores it in R^k_{curr}
2. Processor j resets R^j_{curr} to $x_0^{(j)} := R^j_{init}$
3. **if** $x_{t_k - t_j}^{(k)} = x_0^{(j)}$ **then** return (no actual collision occurred between j and k)
4. **while** $f(R^k_{curr}) \neq f(R^j_{curr})$
   - $R^j_{curr} \leftarrow f(R^j_{curr})$
   - $R^k_{curr} \leftarrow f(R^k_{curr})$
5. Output (R^j_{curr}, R^k_{curr})

Suppose, for a moment, that f is a random mapping. Then each processor is randomly moving through the space $[N]$ until a distinguished point is found, so for an expected number of steps proportional to $\frac{N}{|\mathcal{D}|}$. The processor does not explicitly store all the points it samples during this process, however, they can be recalculated from $x_0^{(j)}$. In addition,

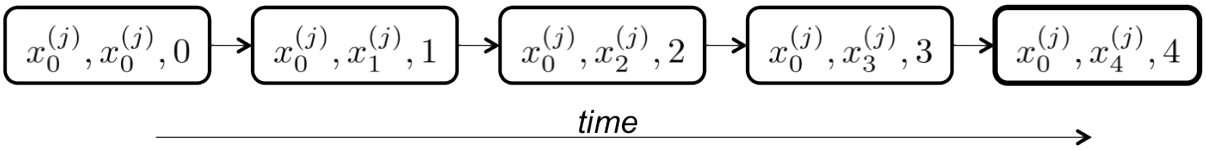

Figure 5.2: **Phase 1**. The state of $(R^j_{init}, R^j_{curr}, R^j_{count})$ for processor j over time. $x_i^{(j)} = f(x_{i-1}^{(j)}$ for $i \geq 1$.

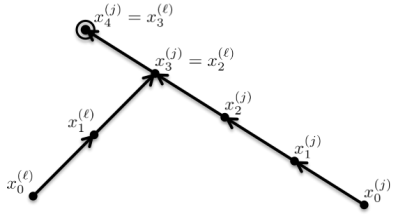

Figure 5.3: Two paths through G_f, starting from $x_0^{(j)}$ and $x_0^{(\ell)}$. If the two paths collide, they will end at the same distinguished point.

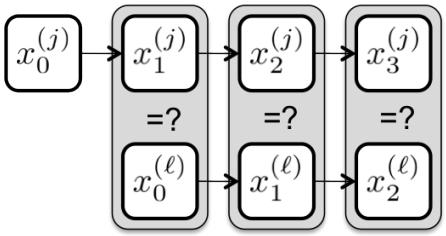

Figure 5.4: **Phase 3**. Two processors that have ended on the same distinguished point reiterate side by side until they find the first coinciding values: the collision point.

if some processor j sees some point x at some step, and processor k sees its partner $\tilde{x}$ at some (possibly different) step, then processor j and k will continue on the same path, eventually hitting the same distinguished point (probably at different times). Thus, if there is a collision in the entire set of sampled points, then there will be two processors stopped at the same distinguished point (except in the rare case where a distinguished point does not occur in time and the processors stop after max steps) and the collision will be found. (Note: the above algorithm also does not detect when a path collides with *itself*, that is, when one processor has some $x_k^{(j)} = x_\ell^{(j)}$ for $k \neq \ell$. We can easily build in a check for this by combining it with the single processor version of the rho method).

Thus, if f is almost strictly many-to-1, we need to look at about $\sqrt{N}$ points before we can expect to find a collision, so we set $|\mathcal{D}| = M\sqrt{N}$, and $max = c\frac{\sqrt{N}}{M}$ for some appropriately large constant c. We call this instance of the rho algorithm rho_{col}. Each processor runs for $O\left(\frac{\sqrt{N}}{M}\right)$ steps in phases 1 and 3. Phase 2 takes $\Theta(\sqrt{M})$ steps using [34]. Thus, the runtime of the algorithm is $R_M^{\boxplus}(\mathsf{rho}) \in \Theta\left(\frac{\sqrt{N}}{M} + \sqrt{M}\right)$. This is optimized when $M = N^{1/3}$, giving $R^{\boxplus}(\mathsf{rho}) \in \Theta(N^{1/6})$.

If we know f has only a constant number of collisions we can set $|\mathcal{D}| \in \Theta(M)$, $max = c\frac{N}{M}$. We call this rho_{ed}. In this case we can expect each processor to traverse $\Theta\left(\frac{N}{M}\right)$ elements, so in total we can expect to look at most of the elements in $[N]$. This gives $R_M^{\boxplus}(\mathsf{rho}) \in \Theta\left(\frac{N}{M} + \sqrt{M}\right)$ which is optimized when $M = N^{2/3}$, giving $\Theta(N^{1/3})$.

If we have no guarantees on the number of collisions in f we can suppose it has many and set $|\mathcal{D}| \in \Theta(N)$, run the algorithm, and if it does not find a collision, halve the size of $|\mathcal{D}|$ and repeat, until we find a collision. Each such iteration costs $\Theta\left(\frac{N}{|\mathcal{D}|M}\right)$. In the worst case, where f has only a constant number of collisions, we may not find a collision until we set $|\mathcal{D}| \in \Theta(M)$, $max = c\frac{N}{M}$, so the total cost is at most $\sum_{i=0}^{\log N} \frac{N}{M}\frac{2^i}{N} = \frac{1}{M}\sum_{i=0}^{\log N} 2^i \in O\left(\frac{N}{M}\right)$.

The function f is not a random function, however, but rather, the input function. In most of the remainder of this chapter, we discuss whether or not rho succeeds in returning a collision pair.

5.1.1 Probabilistic Analysis for $\mathbf{COL}_N$

Define $\mathcal{F}_N^{(2)}$ to be the set of functions $f : [N] \to [N]$ that are 2-to-1. In this section, we will show that if f is chosen uniformly at random from $\mathcal{F}_N^{(2)}$, then rho_{col} succeeds with at least constant probability.

If f were a completely random function, we could simply use the standard birthday paradox argument directly, however f is simply uniform over the restricted set of functions $\mathcal{F}_N^{(2)}$, so we have to be careful. However, we will see that the argument for why we expect to find a collision is nearly identical to the proof of Lemma 3.0.3.

It's not difficult to see the following:

Lemma 5.1.2. *Let f be any 2-to-1 function. Then $\mathcal{F}_N^{(2)} = \{\pi_1 \circ f \circ \pi_2 : \pi_1, \pi_2 \in S_N\}$.*

Fix a reference function $\chi \in \mathcal{F}_N^{(2)}$ as follows: $\chi(x) = \lceil \frac{x}{2} \rceil$. Then any function in $f \in \mathcal{F}_N^{(2)}$ can be constructed from χ by $f = \pi_1 \circ \chi \circ \pi_2$ for (non-unique) $\pi_1, \pi_2 \in S_N$. This gives the following:

Lemma 5.1.3. *If π_1 and π_2 are chosen uniformly and independently from S_N, then $\pi_1 \circ \chi \circ \pi_2$ is a uniform random function from $\mathcal{F}_N^{(2)}$.*

Proof. Let $\mathcal{S}_f := \{(\pi_1, \pi_2) \in S_N^2 : f = \pi_1 \circ \chi \circ \pi_2\}$. The probability that $\pi_1 \circ \chi \circ \pi_2 = f$ is $\frac{|\mathcal{S}_f|}{N!N!}$. We will show that the size of S_f is the same for each f.

For any $f, g \in \mathcal{F}_N^{(2)}$, we can show a one-to-one mapping from $\mathcal{S}_f$ to $\mathcal{S}_g$. Let $\sigma, \tau \in S_N$ be such that $g = \sigma \circ f \circ \tau$. Consider the one-to-one mapping $(\pi_1, \pi_2) \mapsto (\sigma \circ \pi_1, \pi_2 \circ \tau)$. Certainly if $f = \pi_1 \circ \chi \circ \pi_2$ then $g = \sigma \circ f \circ \tau = \sigma \circ \pi_1 \circ \chi \circ \pi_2 \circ \tau$. Thus $|\mathcal{S}_f| \leq |\mathcal{S}_g|$. By the same argument, $|\mathcal{S}_g| \leq |\mathcal{S}_f|$ so $|\mathcal{S}_f| = |\mathcal{S}_g|$. □

We can therefore think of choosing the random function f as follows. Begin with an empty table of N entries, indexed by $[N]$. Each time f is queried on some x, check the x entry of the table, and if it contains an entry, return that entry. Otherwise, choose a new entry at random in the following manner: each time take some $y \in [N]$ without replacement (π_2), and put $\lceil \frac{y}{2} \rceil$ in the table (χ). Once the table is full, we can apply a random π_1 to each table element to get the random function f. Note that this does not change which pairs of elements are collisions (that is, (x, y) is a collision with respect to $\chi \circ \pi_2$ if and only if it is a collision with respect to $\pi_1 \circ \chi \circ \pi_2$).

Theorem 5.1.4. *Let f be a random function from $\mathcal{F}_N^{(2)}$. Then the probability that rho_{col} returns a collision in f is bounded away from 0 by a constant.*

Proof. Consider the sequence of random variables:

$$(Y_i)_{i=1}^r = (X_1^{(1)}, X_1^{(2)}, \ldots, X_1^{(M)}, X_2^{(1)}, \ldots, X_2^{(M)}, \ldots, X_{T_M}^{(M)})$$

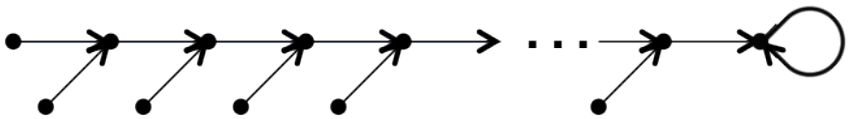

Figure 5.5: A 2-to-1 function with bad structure for rho.

and suppose we query f in the order indicated by this sequence. Every time we add an entry to the table, we're choosing a uniform random $x \in [N]$, without replacement, and we have a collision as soon as we have two elements x and y in the table such that $\lceil \frac{x}{2} \rceil = \lceil \frac{y}{2} \rceil$.

Suppose there is no collision in $Y_1, \ldots, Y_k$. Then the probability that $Y_{k+1} = Y_i \pmod{\frac{N}{2}}$ for some $Y_i \in \{Y_1, \ldots, Y_k\}$ is exactly $\frac{k}{N-k} \geq \frac{k}{N}$. Thus, the probability that there is no collision in $Y_1, \ldots, Y_r$ is at most:

$$\Pi_{k=1}^{r-1} \left(1 - \frac{k}{N}\right) \leq 1 - \frac{r(r-1)}{4N}$$

where the inequality is obtained as in Lemma 3.0.3. In rho_{col}, we have $r \in \Theta(\sqrt{N})$ with at least constant probability, so the probability that we have a collision is at least:

$$\Theta\left(\frac{\sqrt{N}(\sqrt{N}-1)}{4N}\right) \in \Theta(1)$$

□

The algorithm rho_{col} can expect to find a collision when the expectation is taken over the input, but there are certain inputs where the algorithm has low probability of finding a collision in $\Theta\left(\frac{\sqrt{N}}{M}\right)$ steps. If we let it run longer, for worst case inputs, it would be expected to take up to $\Theta\left(\frac{N}{M}\right)$ steps to find a collision. One such worst-case input is as follows:

$$f(x) = \begin{cases} x+1 & \text{if } x < N-1 \text{ even} \\ x+2 & \text{if } x < N-1 \text{ odd} \\ N & \text{if } x = N-1 \text{ or } x = N \end{cases}$$

The function graph of f (Figure 5.5) illustrates why this function is not well-suited to rho. If we have two processors starting at x_1 and x_2 respectively, with $x_1 < x_2$, it will take $\Theta(x_2 - x_1)$ steps for the path starting at x_1 to collide with the path starting at x_2.

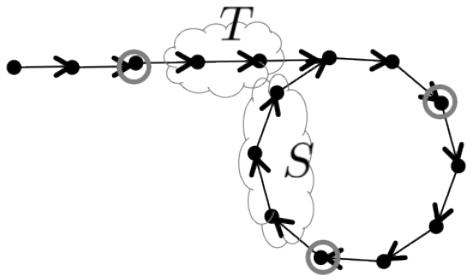

Figure 5.6: The rho of a function. Distinguished points are circled.

The expected shortest distance between two paths is $\Theta\left(\frac{N}{M}\right)$, so it is very unlikely that two paths collide, since they have expected length $\Theta\left(\frac{\sqrt{N}}{M}\right)$.

The natural solution is to attempt to randomize the input. For instance, if we compose the input function with a random permutation π, $f \circ \pi$ will have a random 2-to-1 structure. However, storing an actual random permutation would take space $N \log N$, and we need the permutation to be repeatable, so we cannot just use and discard a random value at each step. We will discuss this dilemma further, but first we give a randomized analysis of rho_{ed}.

5.1.2 Probabilistic Analysis for $\mathbf{ED}_N$

We now show that the element distinctness version of the algorithm, rho_{ed}, succeeds with at least constant probability on a random worst case input. Though any function is an allowed input to $\mathbf{F}_N$, a random function actually has many more collisions than a worst-case input, which has just one collision. We discuss some of the structure of the input that the algorithm attempts to exploit, and show that, for the expected structure of a *worst case* input, the algorithm finds a collision if there is one.

Define $\mathcal{F}_N^{(1)}$ to be the set of functions $f : [N] \to [N]$ with exactly one collision pair. These are the most difficult instances of $\mathbf{ED}_N$.

The function graph of $f \in \mathcal{F}_N^{(1)}$ has N vertices, each of which has out-degree 1. Exactly one vertex has in-degree 0, exactly one vertex has in-degree 2, and the remaining vertices have in-degree 1. Such a graph is a set of 0 or more directed cycles and a *rho*, which is a directed cycle with a directed path (called the *tail*) ending at one of the points in the cycle.

For $f \in \mathcal{F}_N^{(1)}$, define $\rho(f)$ to be the *rho size* of f, or the number of vertices in the rho of the function graph of f. Similarly, define the tail size, $\tau(f)$, to be the number of vertices in the tail of the rho of f and $\sigma(f)$ the number of vertices in the cycle. Note that $\rho(f) = \sigma(f) + \tau(f)$. For this analysis it will be sufficient, and hopefully illuminating, to analyse the structure of a random function in $\mathcal{F}_N^{(1)}$. Intuitively, if the tail and cycle are both large enough that at least one processor is expected to begin on each, then at least one processor can be expected to begin on each within an appropriate distance of the collision.

Let F be a uniform random variable on $\mathcal{F}_N^{(1)}$. We wish to consider $E[\sigma(F)]$ and $E[\tau(F)]$. It is not difficult to see that $E[\sigma(F)] = E[\tau(F)] = \frac{1}{2}E[\rho(F)]$: given a fixed rho size s, the conditional distribution of $\tau(F)$ is uniform on $1, \ldots, s-1$, so the conditional expectation is $\frac{s}{2}$. We thus now consider $E[\rho(F)]$.

Theorem 5.1.5. $E[\rho(F)] = \frac{2}{3}N + \frac{2}{3}$

Proof. For a fixed rho size s, there are $\binom{N}{s}$ ways to choose the elements in rho, $s!$ ways to order them, and $s-1$ possible shapes for the rho (corresponding to the possible collision values). There are $(N-s)!$ ways to configure the non-rho elements, which are just a permutation on $N-s$ elements. Thus, there are $\binom{N}{s}s!(N-s)!(s-1) = N!(s-1)$ functions in $\mathcal{F}_N^{(1)}$ with rho size s.

We have $\left|\mathcal{F}_N^{(1)}\right| = \sum_{s=2}^{N} N!(s-1) = N!\sum_{s=1}^{N-1} s = N!\frac{N(N-1)}{2}$, so the probability that F has rho size s is $\Pr[\rho(F) = s] = \frac{N!(s-1)}{N!N(N-1)/2} = \frac{2(s-1)}{N(N-1)}$.

Thus:

$$E[\rho(F)] = \sum_{s=2}^{N} s \Pr[\rho(F) = s] = \sum_{s=2}^{N} \frac{2s(s-1)}{N(N-1)}$$

$$= \frac{2}{N(N-1)}\left(\sum_{s=2}^{N} s^2 - \sum_{s=2}^{N} s\right)$$

$$= \frac{2}{N(N-1)}\left(\frac{N(N+1)(2N+1)}{6} - \frac{N(N+1)}{2}\right)$$

$$= \frac{1}{N-1}\left(\frac{(N+1)(2N+1) - 3(N+1)}{3}\right)$$

$$= \frac{(N+1)(2N-2)}{3(N-1)}$$

$$= \frac{(N+1)2}{3}$$

$$= \frac{2}{3}N + \frac{2}{3}$$

So the expected rho size of F is about $\frac{2}{3}N$. □

Corollary 5.1.6. $E[\tau(F)] = \frac{N}{3} + \frac{1}{3}$

Corollary 5.1.7. *Fix some constant ϵ, with $0 < \epsilon \leq \frac{1}{9}$. Then* $\Pr[\tau(F) \leq \epsilon\frac{N}{3}] \leq \frac{3}{4}$.

Proof. By Markov's inequality, we have:

$$\Pr[N - \tau \geq a] \leq \frac{E[N - \tau]}{a}$$

$$\Pr[\tau \leq N - a] \leq \frac{N - E[\tau]}{a}$$

Let $a = (1 - \epsilon)N$. Then:

$$\Pr[\tau \leq \epsilon N] \leq \frac{N - \frac{N}{3} - \frac{1}{3}}{(1 - \epsilon)N} \leq \frac{2}{3(1 - \epsilon)} \leq \frac{2}{3(8/9)} = \frac{3}{4}$$

□

We now show that rho_{ed} finds a collision with constant probability.

Let $\mathcal{T}_f$ be the set of tail points that occur *after* the last distinguished point on the tail of f. Let $\mathcal{S}_f$ be the set of points that occur *after* the last distinguished point on the rho-cycle of f before the collision, and *before* the collision itself (see Figure 5.6).

The algorithm rho_{ed} will find the collision in $f \in \mathcal{F}_N^{(1)}$ if and only if the following conditions are met:

1. There exists $i \in [M]$ such that $x_0^{(i)}$ is in $\mathcal{S}_f$ and the distance from $x_0^{(i)}$ to the first distinguished point after the collision is at most max.
2. There exists $j \in [M]$ such that $x_0^{(j)}$ is in $\mathcal{T}_f$ and the distance from $x_0^{(j)}$ to the first distinguished point after the collision is at most max.

We will now consider the sizes of the sets $\mathcal{S}_f$ and $\mathcal{T}_f$. These values will depend on both f and $\mathcal{D}$, the pseudo-random set of distinguished points of size M. Since F is uniform random, it does not matter how $\mathcal{D}$ is defined for this particular analysis. We thus define a distinguished point as any point $x \leq M$. This gives $\delta = \frac{M}{N}$ as the fraction of points that are distinguished.

Theorem 5.1.8. *With constant probability,* $|\mathcal{T}_F| \in \Theta\left(\frac{N}{M}\right)$.

Proof. First, by Corollary 5.1.7, with at least constant probability, $\tau(F) \in \Theta(N)$. Let Z_i denote the tail point at distance i from the collision, and consider the set $Z_1, \ldots, Z_{cN/M}$. The expected number of distinguished points in this set is:

$$E[Y] = \sum_{i=1}^{cN/M} P(Z_i \leq M) = c\frac{N}{M}\frac{M}{N} = c$$

Since the number of distinguished points in $Z_1, \ldots, Z_{cN/M}$ has a hypergeometric distribution, we know that

$$Var[Y] = \frac{c\frac{N}{M}M(N - cN/M)(N - M)}{N^2(N-1)} = c\frac{N - M - cN/M + c}{N-1}$$

Thus, by Chebyshev's inequality, the probability that Y is within some constant c_1 of $E[Y] = c$ is at least:

$$c_2 = 1 - \frac{Var[Y]}{c_1^2} = 1 - c\frac{N - M - cN/M + c}{c_1^2(N-1)} \geq 1 - \frac{c}{c_1^2}$$

Let $c_3 = c + c_1$. The probability that these Y distinguished points will all fall in the set $Z_{\frac{c}{2}N/M}, \ldots, Z_{cN/M}$ conditioned on $|Y - c| \leq c_1$ (and so $Y \leq c_3$) is at least:

$$\frac{\binom{cN/M - c_3}{\frac{c}{2}N/M}}{\binom{cN/M}{\frac{c}{2}N/M}} = \Pi_{i=0}^{c_3-1}\frac{\frac{c}{2}N/M - i}{cN/M - i} \geq \Pi_{i=0}^{c_3-1}\frac{1}{2}\frac{N/M - \frac{2i}{c}}{N/M - \frac{i}{c}} \geq \Pi_{i=0}^{c_3-1}\frac{1}{2}\frac{1}{2} = \frac{1}{4^{c_3}} =: c_4$$

Thus, the probability that $\frac{c}{2}N/M \leq |\mathcal{T}_f| \leq cN/M$ is at least $c_2c_4 = 1 - o(1)$, so with at least constant probability, $|\mathcal{T}_f| \in \Theta(N/M)$. □

An identical argument shows that, with at least constant probability, $|\mathcal{S}_F| \in \Theta\left(\frac{N}{M}\right)$, and the distance from the collision to the next distinguished point is in $\Theta\left(\frac{N}{M}\right)$. Therefore, with constant probability, the algorithm finds the collision. If we choose an input uniformly at random from $\mathcal{F}_N^{(1)}$, after a constant number of rounds of the algorithm we can expect to find the collision with high probability.

5.2 The Cayley Rho Algorithm

Several variations of rho have been analysed, mostly in its application to the discrete logarithm problem. The first, and only (that we are aware) formal analysis that does not require a random oracle assumption is an elegant variation of the original rho method for discrete log [26, 25]. Their main idea is to compose the standard walk with a limited independence walk on a *random Cayley graph*. In this section we will outline the main ideas of [26, 25], and then show that the assumption upon which their analysis is based is not generally true.

Given an additive group G, and a *generating set* $\mathcal{S} \subseteq G$, the Cayley graph of G with generating set $\mathcal{S}$, $\mathcal{G}(G, \mathcal{S})$ is the graph with vertex set G and edge set $\{x \to x + g : x \in G, g \in \mathcal{S}\}$. A *random Cayley graph* is a Cayley graph with a randomly chosen generating set. We will always have $G = \mathbb{Z}_N$. A Cayley graph can be stored by simply storing $\mathcal{S}$, so we will have $d := |\mathcal{S}| \in \Theta(\log N)$. In this section, we will consider N as a prime so that we can use the field structure of $\mathbb{Z}_N$. We have been considering N as even, so that we can easily consider 2-to-1 functions, but this is not particularly necessary: we can easily extend a a function $f : [N] \to [N]$ to a function $f' : [N'] \to [N']$ where N' is the next even number (or prime) by defining $f'(x) = x$ when $x > N$. If f is almost strictly many-to-1, f' will be as well.

For the purpose of general collision finding (as opposed to collision finding for a specific application, such as factoring or discrete log) we need to compose the input function f with some sufficiently random operator π. The operator π must be repeatable, so that once two paths collide, they remain on the same path, so we can't, for example, simply choose a random step at each point and then discard that random choice: we must store all randomness used so we can repeat our choices. However, we have space limitations as well, so clearly π cannot be completely random.

The ability to efficiently store limited independent functions (see Definition 3.0.6) makes them seem like a possible candidate for randomizing the input in some way. In order to randomize the walk of rho, we might try to compose it with some pseudorandom walk. Where we have a labelling of the edges, as in a Cayley graph, we can think of a random walk in the following way. At the i^{th} step, we choose some random string r_i, and choose which edge to take, as a function of r_i. Of course, in an actual random walk, we need the r_i to be mutually independent, however, for our walk, we don't want this, since we want to choose the same edge any time we're at a particular vertex. A random oracle models this situation perfectly, since the only dependence between the choices r_i is that if we are at vertex v at both time i and time j, then $r_i = r_j$. However, as stated above, what we need is some efficiently storable function.

If the states in the path of the random walk are pairwise independent, then we can argue that the events $X_i = X_j$ are pairwise independent, and apply a birthday paradox argument as in Lemma A.0.2. Notice that this is not the same as the choices at each step being pairwise independent. To see this, consider the following example. Suppose we want to take a limited-independence random walk on the complete graph with self-loops added, $\overline{K}_N$, using pairwise independent choices. We will consider $\overline{K}_N$ as the Cayley graph on the additive group $\mathbb{Z}_N$ with generating set $\mathbb{Z}_N$. We therefore walk by choosing a random integer $\pmod N$ and adding it to the current element, so our choice at step i is some $r_i \in \mathbb{Z}_N$. The following sequence is pairwise independent:

$$r_1 \in_R \mathbb{Z}_N$$

$$r_2 \in_R \mathbb{Z}_N$$

$$r_i = N - r_{i-1} - r_{i-2} \pmod N \forall i > 2$$

The (r_i) are certainly pairwise independent, but consider the path of the walk, $X_0, X_1, \ldots$ The X_i are not pairwise independent: for any i, j we have $X_i = X_{i+3j}$.

We now outline the walk of [26, 25]. The walk is defined for finding collisions in a specific function used for solving discrete log, but the goal of constructing a pseudo-random walk that is deterministic but has pairwise independent points in the path is the same, so we could generalize to finding collisions in arbitrary functions by composing an input f with a random walk on a random Cayley graph as follows.

Let $\mathcal{G}(\mathbb{Z}_N, \mathcal{S})$ be the Cayley graph with generating set $\mathcal{S} := \{g[1], \ldots, g[d]\}$ chosen uniformly at random from subsets of $\mathbb{Z}_N$ of size $d \in \Theta(\log N)$.

Let $h : [N] \to [d]$ be a uniform t-wise independent hash for some $t \in \Theta(\log N)$. This function is how we will make our choices at each step. Define one step in the walk as:

$$X_{i+1} = f(X_i) + g[h(f(X_i))]$$

This is just the composition of f and the function $\omega(x) = x + g[h(x)]$, the group action defined by g_k where k is a function of the input. Until h receives the same input twice (that is, until we see the same $f(x)$ twice and thus have a probable collision) the sequence of h values we see at each step, $r_1, r_2, \ldots$, is exactly a t-wise independent sequence of choices. We wish to upper bound the expected time before a collision occurs between two paths, or that a single path collides with itself (having multiple paths with different starting points can only decrease dependence) so we are actually concerned with the distribution of points

before a collision. Thus, the goal is to show that for t-wise independent choices $r_1, r_2, \ldots$ from $[d]$ the following walk will collide with itself within $\tilde{\Theta}(\sqrt{N})$ expected number of steps:

$$X_{i+1} = f(X_i) + g[r_i]$$

This walk is the composition of f with the limited independence walk given by $\bar{\omega}(x) = x + g[r_i]$. If we can show that the X_i are pairwise independent, we can apply a birthday argument.

Since our random choices, $\mathcal{S}$ and $r_1, r_2, \ldots$ are independent of the input, we will analyze the pseudorandom walk function $\bar{\omega}$ without composition with f. This will allow us to use the elegant analysis of [26, 25], which takes advantage of the Abelian structure of $\mathbb{Z}_N$, and intuitively, composition with a fixed f shouldn't affect the properties of the walk. Of course, we will be now looking at a collision in $\bar{\omega}$, but it's simple to redefine collision from $X_i = X_j$ to $X_i \in \{X_j, \tilde{X}_j\}$. However, we will ultimately be showing that this analysis falls slightly short of working.

So with all this in mind, we now consider the limited independence walk given by:

$$X_{i+1} = \bar{\omega}(X_i) = X_i + g[r_i]$$

where $r_1, \ldots$ is a t-wise independent sequence and $g[1], \ldots, g[d]$ are chosen uniformly at random from $\mathbb{Z}_N$.

Consider any path produced by the limited independence walk: $X_0, X_1, \ldots, X_\ell$. We have $X_\ell = X_0 + \sum_{i=1}^{\ell} g[r_i]$. We could thus represent the point X_ℓ by the starting point X_0 and the *choice vector* $r \in [d]^\ell$. Since $\mathbb{Z}_N$ is Abelian, we could have gotten from X_0 to X_ℓ by adding the $g[r_i]$ in *any* order. Thus if we are only interested in the point X_ℓ, and not how we got there, we can represent it by the starting point X_0 and the d-dimensional vector γ given by $\gamma_j = |\{i \in [\ell] : r_i = j\}|$. We called γ the *type* of X_ℓ. If the starting point is X_0, we have $X_\ell = X_0 + \sum_{j=1}^{d} \gamma_j g[j]$. Note that a state X_k does not usually have a unique type, but a type γ does correspond to a unique state X (given a fixed starting point X_0).

We can assume without loss of generality that the starting point is $X_0 = 0$. This should not make a difference, since the set $\mathcal{S}$ is random, so the first step adds a uniform random element of $\mathbb{Z}_N$.

Lemma 5.2.1. *Let X_ℓ be the endpoint of a path of length $\ell > 0$ and type γ. Then for all $\alpha \in [N]$ we have $\Pr_{\mathcal{S}}[X_\ell = \alpha] = \frac{1}{N}$.*

Proof. We have $X_\ell = \sum_{i=1}^{d} \gamma[i]g[i]$. Since $\ell > 0$, we have at least one entry in γ positive, so suppose $\gamma[k] > 0$ for some k. Then:

$$\Pr_{\mathcal{S}}[X_\ell = \alpha] = \Pr_{\mathcal{S}}\left[\sum_{i=1}^{d} \gamma[i]g[i] = \alpha\right]$$

$$= \frac{1}{N^d}\left|\{\vec{g} \in [N]^d : \sum_{i=1}^{d} \gamma[i]g[i] = \alpha\}\right|$$

$$= \frac{1}{N^d}\left|\left\{\vec{g} \in [N]^d : g[k] = \gamma[k]^{-1}\left(\alpha - \sum_{i \neq k} \gamma[i]g[i]\right)\right\}\right|$$

since N prime and $\gamma[k] \neq 0$

$$= \frac{1}{N^d} N^{d-1} = \frac{1}{N}$$

□

Consider any two states on the path, X_i and X_j, with types γ_1 and γ_2. We are interested in whether or not these points are independent. The following lemma is a generalization of a lemma in [26].

Lemma 5.2.2. *Let $\mathcal{X} = \{X_{i_1}, \ldots, X_{i_k}\}$ be a set of path endpoints and $\{\gamma_1, \ldots, \gamma_k\}$ a set of types such that X_{i_j} is the point corresponding to γ_j. Then $\{X_{i_1}, \ldots, X_{i_k}\}$ is a set of mutually independent random variables (in the uniform distribution over $\mathcal{S}$) if and only if $\{\gamma_1, \ldots, \gamma_k\}$ is a linearly independent set over $\mathbb{Z}_N^d$.*

Proof. We have:

$$\begin{bmatrix} \gamma_1^T \\ \gamma_2^T \\ \vdots \\ \gamma_k^T \end{bmatrix} \begin{bmatrix} g[1] \\ g[2] \\ \vdots \\ g[d] \end{bmatrix} = \begin{bmatrix} X_{i_1} \\ X_{i_2} \\ \vdots \\ X_{i_k} \end{bmatrix}$$

We can think of this as a system of linear equations in $\mathcal{S} = \{g[1], \ldots, g[d]\}$. The matrix $[\gamma_i[j]]$ is a $k \times d$ coefficient matrix.

Suppose $\mathcal{X}$ is a set of uniform mutually independent (in $\mathcal{S}$) random variables. Then $(X_{i_1}, \ldots, X_{i_k})$ should be uniform on $[N]^k$. Since $\mathcal{S}$ is chosen uniformly from $[N]^d$, this

means there should be the same number of solutions $\mathcal{S}$ for any choice of $\mathcal{X}$. That is, the number of solutions to the linear system should not depend on $\mathcal{X}$.

Suppose the γ_i are not linearly independent. Then for some choice of $\mathcal{X}$ there will be no solutions $\mathcal{S}$, and for some there will. Thus $\mathcal{X}$ will not be uniformly distributed over $[N]^k$.

Suppose γ_i are linearly independent. Then regardless of $\mathcal{X}$ there are N^{d-k} solutions, so the number of solutions is independent of $\mathcal{X}$.

□

Suppose we have paths of type $\gamma_1, \ldots, \gamma_k$ with endpoints $X_1, \ldots, X_k$, where $\gamma_k = c_1\gamma_1 + \cdots + c_{k-1}\gamma_{k-1}$ for constants $c_1, \ldots, c_{k-1} \in \mathbb{Z}_N$. Then we have $X_k = c_1X_1 + \cdots + c_{k-1}X_{k-1}$, so the X_i are not mutually independent. More specifically, if $\gamma_2 = c\gamma_1$ then $X_2 = cX_1$, so X_1 and X_2 are completely correlated.

Note that if two distinct path types γ_1 and γ_2 are linearly dependent in $\mathbb{Z}_N^d$, then we have $\gamma_1 = c\gamma_2 \pmod N$ for some $c \in \mathbb{Z}_N^d$. Since we are considering paths of length $\Theta(\sqrt{N})$, we can suppose $\|\gamma_i\|_1 < c\sqrt{N}$ for $i = 1, 2$ and some constant c. Further, suppose that no entry $\gamma_i[j]$ is greater than $\frac{1}{c}\sqrt{N}$ (types with a single large entry will be rare, since the choices r_i are uniform t-wise independent). We have:

$$\begin{aligned}\|\gamma_1\|_1 \gamma_2 &= \left\|c^{-1}\gamma_2\right\|_1 c\gamma_1 \pmod N \\ &= c^{-1}c \|\gamma_2\|_1 \gamma_1 \pmod N \\ &= \|\gamma_2\|_1 \gamma_1 \pmod N\end{aligned}$$

And since both norms are less than $c\sqrt{N}$, and all entries are at most $\frac{1}{c}\sqrt{N}$, the products are all less than N, and thus:

$$\|\gamma_1\|_1 \gamma_2 = \|\gamma_2\|_1 \gamma_1$$

Thus let $c_1\gamma_1 = c_2\gamma_2$ with $\gcd(c_1, c_2) = 1$. Define the *vector gcd* of a vector $v \in \mathbb{Z}^d$ as $\gcd(v[1], \ldots, v[d])$. We have $c_2 | c_1\gamma_1[i]$ for each $i \in [d]$, and since $\gcd(c_1, c_2) = 1$, we must have $c_2 | \gamma_1[i]$ for all $i \in [d]$. Thus the vector gcd of γ_1 is at least c_2. If $c_2 = 1$ then $\gamma_2 = c_1\gamma_1$, so the vector gcd of γ_2 is at least c_1, and $c_1 \neq 1$, since $\gamma_1 \neq \gamma_2$. Thus, at least one of γ_1 and γ_2 have vector gcd greater than 1.

In order to say that the probability of two path types being linearly dependent is small, [25] makes the following assumption:

VGCD assumption: Let r be chosen from $[d]^\ell$ by a t-wise independent distribution ρ. Let vgcd denote the gcd of all entries in a vector. If $\ell > \Omega(d)$, then

$$\Pr_\rho[\mathsf{vgcd}(r) > 1] < \frac{1}{8N} < \frac{1}{\Theta(2^d)}$$

This number theoretic statement is true when r is uniform random, however, limited independence is never to be trusted. In the following section we show the existence of t-wise independent distribution ρ which has $\mathsf{vgcd} > 1$ with high probability when $\ell \in O(t^3)$.

We first note that this is not a proof that the Cayley rho algorithm does not work. It is possible (though likely not easy to prove) that the particular limited independent distribution used in the Cayley rho algorithm does not suffer from this shortcoming. Additionally, it may be possible to show the independence of the states in the walk without this assumption, though likely more complicated. Finally, though it may be difficult to prove that a limited independent distribution really does yield pairwise independent states in the walk, this assumption about the distribution used seems to be as a good a heuristic assumption as a random oracle assumption, since any algorithm that assumes a random oracle will use a pseudo-random function in its place in practice (or else end up using an unreasonable amount of space to store a random function). It is assumed that the pseudo-random function used will behave sufficiently randomly with respect to the problem and input that its lack of actual randomness will not matter. From this perspective, the vgcd assumption seems to be just as valid, in the sense that the limited independent function used in practice is unlikely to be constructed so as to be especially bad for this application.

5.2.1 VGCD is False for some r

Consider a t-wise independent *choice vector* $r \in [d]^\ell$ with distribution ρ. For such an r, we may consider the d-dimensional vector $\gamma = \gamma(r)$ where $\gamma_j = |\{i \in [\ell] : r_i = j\}|$ for each $j \in [d]$. We can also define the random variable $\mathsf{vgcd} = \mathsf{vgcd}(r) = \gcd(\gamma(r))$. This is best illustrated through an example.

Example 5.2.3. *Let $d = 4$, $\ell = 12$, $r = (1, 2, 4, 2, 2, 4, 1, 1, 1, 4, 1, 1)$. Then $\gamma(r) = (6, 3, 0, 3)$ and $\mathsf{vgcd}(r) = 3$.*

Consider the vgcd assumption when $d = 2$. Simply choose $r_1, \ldots, r_{\ell-1}$ uniformly and independently from $[d]$ and set $r_\ell = \sum_{i=1}^{\ell-1} r_i \pmod 2$. This distribution is uniform $\ell-1$-wise independent and always yields an even number of 1s and an even number of 0s whenever ℓ is even, so the vgcd assumption certainly isn't true in this case.

For more general values of d we will also prove the existence of a similar counter example to this assumption. We will always assume ℓ is even.

We will consider $\Pr[2|\mathsf{vgcd}] \leq \Pr[\mathsf{vgcd} > 1]$, the probability that every choice from $[d]$ is made an even number of times. To this end, we introduce one final random variable: $b = b(r) := \gamma(r) \pmod 2$, the *parity vector* which records the parity of each entry of γ.

Let $\mathcal{S}_\ell := \{r \in [d]^\ell : b(r) = \vec{0}\}$. Choosing r uniformly from $\mathcal{S}_\ell$ is actually already quite close to t-wise independent for $t \in \Theta(d)$, $\ell \in \Omega(d^2)$. We will show this formally in Theorem 5.2.10, and then show that there exists a similar distribution that is perfectly t-wise independent, and yields even vgcd with probability less than a constant away from 1 whenever $\ell \in \Omega(d^3)$ in Theorem 5.2.15.

More formally, we prove the following:

Theorem 5.2.4. *For even ℓ, there exists a t-wise independent distribution ρ_ℓ on $[d]^\ell$ such that*

$$\Pr_{\rho_\ell}[b = \vec{0}] > 1 - 2^{\Theta(t \log d \log \ell + d) - \Theta(\ell/d)}$$

That is, with high probability (when $\ell \in \Omega(d^3)$, $t \in \Theta(d)$), each element of $[d]$ is chosen an even number of times.

Definition 5.2.5. *We let $n_k(b) := \left|\{r \in [d]^k : b(r) = b\}\right|$ denote the number of choice vectors of length k with parity vector b. Note that $n_\ell(\vec{0}) = |\mathcal{S}_\ell|$.*

$p_k(b) := \frac{n_k(b)}{d^k}$ denotes the proportion of all choice vectors of length k that have parity vector b. If a choice vector of length k is chosen uniformly at random, then p_k is the probability distribution on the parity vector.

Consider the distribution μ_ℓ on $[d]^\ell$ given by:

$$\mu_\ell(r) = \begin{cases} \frac{1}{n_\ell(\vec{0})} & \text{if } r \in \mathcal{S}_\ell \\ 0 & \text{else} \end{cases}$$

This is the uniform distribution on $\mathcal{S}_\ell$, and so certainly $\Pr_{\mu_\ell}[\mathsf{vgcd} > 1] = 1$. We also have, for all $i \in [\ell]$ and all $\alpha \in [d]$, $\Pr_{\mu_\ell}[r_i = \alpha] = \frac{1}{d}$, so the marginal distribution of each r_i is uniform on $[d]$. We wish to know how close μ_ℓ is to t-wise independent. That is, for $\vec{\alpha} \in [d]^t$ and $\vec{i} \in [\ell]^t$, how close is $\Pr_{\mu_\ell}[r_{i_1} = \alpha_1 \wedge \cdots \wedge r_{i_t} = \alpha_t]$ to $\frac{1}{d^t}$?

We have

$$\Pr_{\mu_\ell}[r_{i_1} = \alpha_1 \wedge \cdots \wedge r_{i_t} = \alpha_t] = \frac{|\{r \in \mathcal{S}_\ell : r_{i_1} = \alpha_1 \wedge \cdots \wedge r_{i_t} = \alpha_t\}|}{|\mathcal{S}_\ell|}$$

Let $I = \{i_1, \ldots, i_t\}$ and r_I denote the vector $(r_{i_1}, \ldots, r_{i_t})$.

$$|\{r \in S_\ell : r_I = \vec{\alpha}\}| = \left|\{r \in [d]^\ell : r_I = \vec{\alpha} \wedge b(r) = \vec{0}\}\right|$$

$$= \left|\{r \in [d]^\ell : r_I = \vec{\alpha} \wedge b(r_{[\ell]\setminus I}) = b(\vec{\alpha})\}\right|$$
$$= \left|\{u \in [d]^{\ell-t} : b(u) = b(\vec{\alpha})\}\right|$$
$$= n_{\ell-t}(b(\vec{\alpha}))$$

Thus:

$$\Pr_{\mu_\ell}[r_I = \vec{\alpha}] = \frac{n_{\ell-t}(b(\vec{\alpha}))}{n_\ell(\vec{0})} = \frac{p_{\ell-t}(b(\vec{\alpha}))d^{\ell-t}}{p_\ell(\vec{0})d^\ell} = \frac{p_{\ell-t}(b(\vec{\alpha}))}{p_\ell(\vec{0})}\frac{1}{d^t}$$

We are therefore interested in whether, and how quickly, the distribution p_k converges to uniform as k increases. We will use the random walk on a Boolean hypercube to investigate this.

The Distribution of b as a Random Walk on the Boolean Hypercube

In this section, we use some well-known theory of random walks to show that the distribution of $b(r)$, for $r \in [d]^k$, converges rapidly to uniform on $\mathcal{B}_{k \pmod 2} := \{b \in \{0,1\}^d : |b| = k \pmod 2\}$ as k increases.

As we choose uniform independent $r_i \in [d]$, we can keep track of the state $b_\tau := b(r[1,\ldots,\tau])$ at time τ. We note that at step τ, we flip the r_τ^{th} bit of $b_{\tau-1}$ to obtain b_τ.

The random variables $b_1, b_2, \ldots$ are a well-studied Markov chain: the random walk on the Boolean hypercube of dimension d. The Boolean hypercube is the graph with vertices $\{0,1\}^d$ and edges between two vertices if they have Hamming distance 1.

Unfortunately, this random walk does not always converge to stationarity, since the graph is bipartite. In particular, if the starting distribution has support consisting of strings of even parity, then the distribution after an even number of steps will be supported by even strings, and the distribution after an odd number of steps will be supported by odd strings. We can easily resolve this issue by studying the convergence of a slightly different random walk.

Consider the random walk on the Boolean hypercube of dimension $d-1$ with self-loops added. That is, the walk given by the transition matrix P, defined:

$$P[x,y] = \begin{cases} \frac{1}{d} & \text{if } x = y \\ \frac{1}{d} & \text{if } \Delta(x,y) = 1 \\ 0 & \text{else} \end{cases}$$

We can view the state of the walk at time τ as representing the first $d-1$ bits in b_τ. We note that this entirely characterizes the state b_τ, since the d^{th} bit of b_τ can be computed

from the parity of τ and the parity of the first $d-1$ entries of b_τ. The probability of remaining at the same state represents the probability of flipping the d^{th} bit (which leaves the first $d-1$ bits unchanged, but changes the parity of τ).

Define $\tilde{p_k}$ as the restriction of p_k to $\mathcal{B}_{k \pmod 2}$ (which is the support of p_k). We can view $\tilde{p_k}$ as a distribution on $\{0,1\}^{d-1}$.

This immediately gives:

Lemma 5.2.6. *Let ρ_0 be the distribution on $\{0,1\}^{d-1}$ with $\rho_0(\vec{0}) = 1$. Then*

$$\tilde{p_k} = \rho_0 P^k$$

Lemma 5.2.7. $t_{mix}(\varepsilon, P) \leq (\frac{d}{2} - 1) \log \frac{1}{2\varepsilon}$

Proof. We have P irreducible, reversible, and aperiodic, so:

$$t_{mix}(\varepsilon, P) \leq \left(\frac{1}{\delta} - 1\right) \log \frac{1}{2\varepsilon}$$

where δ is the eigenvalue gap of P. We can calculate the eigenvalue gap as $\frac{2}{d}$, giving:

$$t_{mix}(\varepsilon, P) \leq \left(\frac{d}{2} - 1\right) \log \frac{1}{2\varepsilon}$$

□

Lemma 5.2.8. *The uniform distribution is the unique stationary distribution for P.*

Proof. Since P is an irreducible random walk on a finite state space, its unique stationary distribution is given by:

$$\pi(x) = \frac{\deg(x)}{\sum_{y \in \{0,1\}^{d-1}} \deg(y)}$$

for all $x \in \{0,1\}^{d-1}$ (see, for example, [38]). Thus:

$$\pi(x) = \frac{d}{d2^{d-1}} = 2^{1-d}$$

□

Corollary 5.2.9. *Let μ be the uniform distribution on $\mathcal{B}_k$. Then $\|p_k - \mu\|_{TV} \leq 2^{-\frac{k}{d}}$.*

Proof. Note that μ is exactly the uniform distribution on $\{0,1\}^{d-1}$: the set $\mathcal{B}_k$ can be rewritten as $\mathcal{B}_k = \{(s,a) : s \in \{0,1\}^{d-1}, a = |s| + k \pmod 2\}$.

We have $\tilde{p_k} = \rho_0 P^k$. Thus if $k \geq t_{mix}(\varepsilon, P)$, then $\|\tilde{p_k} - \mu\|_{TV} \leq \varepsilon$ and so $\|p_k - \mu\|_{TV} \leq \varepsilon$, and so if $k \geq (\frac{d}{2} - 1)\log\frac{1}{2\varepsilon}$, then $\|p_k - \mu\|_{TV} \leq \varepsilon$. Solving for ε, we get $\|p_k - \mu\|_{TV} \leq 2^{-\frac{k}{d}}$. □

Theorem 5.2.10. *Consider the distribution μ_ℓ on $[d]^\ell$ given by the uniform distribution on $\mathcal{S}_\ell$. For any set of $I = \{i_1, \ldots, i_t\} \subseteq [\ell]$, the marginal distribution on r_I is close to uniform in the following precise sense:*

$$\forall \alpha \in [d]^t, \text{ we have } \left|\Pr[r_{i_1} = \alpha_1 \wedge \cdots \wedge r_{i_t} = \alpha_t] - \frac{1}{d^t}\right| \leq d^{-t} 2^{\frac{t-\ell}{d}+d+1}$$

Proof. We have $\Pr[v_I = \alpha] = \frac{|\{v \in S_\ell : v_I = \alpha\}|}{|S_\ell|}$. The number of vectors in S_ℓ with $v_I = \alpha$ is exactly the number of ways of choosing $v_{\bar{I}} \in [d]^{\ell-t}$ so that $b(v_{\bar{I}}) = b(\alpha)$ (since then $b(v) = \vec{0}$). This is exactly $n_{\ell-t}(b(\alpha))$. So:

$$\Pr[v_I = \alpha] - \frac{1}{d^t} = \frac{n_{\ell-t}(b(\alpha))}{n_\ell(\vec{0})} - \frac{1}{d^t}$$

$$= \frac{p_{\ell-t}(b(\alpha))d^{\ell-t}}{p_\ell(\vec{0})d^\ell} - \frac{1}{d^t} = \frac{1}{d^t}\left(\frac{p_{\ell-t}(b(\alpha))}{p_\ell(\vec{0})} - 1\right)$$

By Corollary 5.2.9 we have $\|p_k - \mu\|_{TV} \leq 2^{-\frac{k}{d}}$ so in particular, for any b, $\left|p_k(b) - \frac{1}{2^{d-1}}\right| \leq 2^{-\frac{k}{d}}$. Thus:

$$\frac{2^{1-d} - 2^{-\frac{\ell-t}{d}}}{2^{1-d} + 2^{-\frac{\ell}{d}}} - 1 \leq \frac{p_{\ell-t}(b)}{p_\ell(\vec{0})} - 1 \leq \frac{2^{1-d} + 2^{-\frac{\ell-t}{d}}}{2^{1-d} - 2^{-\frac{\ell}{d}}} - 1$$

$$\frac{2^{1-d} - 2^{-\frac{\ell-t}{d}} - 2^{1-d} - 2^{-\frac{\ell}{d}}}{2^{1-d} + 2^{-\frac{\ell}{d}}} \leq \frac{p_{\ell-t}(b)}{p_\ell(\vec{0})} - 1 \leq \frac{2^{1-d} + 2^{-\frac{\ell-t}{d}} - 2^{1-d} + 2^{-\frac{\ell}{d}}}{2^{1-d} - 2^{-\frac{\ell}{d}}}$$

$$\left|\frac{p_{\ell-t}(b)}{p_\ell(\vec{0})} - 1\right| \leq \frac{2^{-\frac{\ell-t}{d}} + 2^{-\frac{\ell}{d}}}{2^{1-d} - 2^{-\frac{\ell}{d}}} \leq \frac{2^{\frac{t}{d}} + 1}{2^{\frac{\ell}{d}-d+1} - 1} \leq \frac{2^{t/d+1}}{2^{\ell/d-d}}$$

where, for the last inequality, we assume $\ell > d^2$. This gives:

$$\left|\Pr[v_I = \alpha] - \frac{1}{d^t}\right| \leq \left|d^{-t}\left(\frac{p_{\ell-t}(b)}{p_\ell(\vec{0})} - 1\right)\right| \leq d^{-t} 2^{\frac{t-\ell}{d}+d+1}$$

□

For the remainder of this chapter, we let $\varepsilon = d^{-t}2^{\frac{t-\ell}{d}+d+1}$ denote the deviation from independence of μ_ℓ.

Almost t-wise independence to $\frac{t}{\log d}$-wise independence

We now crudely apply a theorem of Alon, Goldreich, and Mansour [5] to show that if we have an almost t-wise independent distribution on $[d]^\ell$, then there is a nearby distribution that is $\frac{t}{\log d}$-wise independent. (Their main result is to show that, in general, given an *almost* k-wise independent distribution on $\{0,1\}^\ell$, the nearest k-wise independent distribution is not as close as they would like, however, it is close enough for our purposes).

We first present relevant definitions. A distribution ρ over $\{0,1\}^n$ is *k-bitwise independent* if it is k-wise independent under the usual definition. It follows that a distribution over $[d]^\ell$ is k-bitwise independent if it is k-bitwise independent as a distribution over $\{0,1\}^{\ell \log d}$.

The *bias* of a distribution ρ with respect to a set $I \subset [n]$ is given by:

$$bias_I(\rho) := 2 \Pr\left[\sum_{i \in I} X_i = 0 \pmod 2\right] - 1$$

A distribution is k-bitwise independent if and only if it has bias 0 with respect to every subset of size at most k.

We now give some useful facts about the relationship between k-wise and k-bitwise independence.

Claim 5.2.11. *If a distribution ρ is uniform k-wise independent on $[d]^\ell$, then it is uniform k-bitwise independent on $\{0,1\}^{\ell \log d}$.*

Proof. Consider any k indices in $[\ell \log d]$. They are from at most k entries in $[\ell]$, and so they are independent, by the uniformity and k-wise independence of ρ. □

Claim 5.2.12. *If a distribution is uniform k-bitwise independent on $\{0,1\}^{\ell \log d}$, then it is uniform $\frac{k}{\log d}$-wise independent on $[d]^\ell$.*

Proof. Any $\frac{k}{\log d}$ entries in $[\ell]$ are made up of k bit-indices in $[\ell \log d]$, and therefore have a uniform marginal distribution. □

The theorem we will apply, taken from [5], is as follows:

Theorem 5.2.13. *Let ρ be a distribution over $\{0,1\}^n$ such that the maximum bias on any non-empty subset of size at most k is at most ϵ. Then there is a k-bitwise independent distribution ρ' such that $\|\rho - \rho'\|_{TV} \leq n^k\epsilon$*

The theorem is proven by simply showing that a simple modification of a distribution decreases the bias of a subset of size at most k to 0 without increasing the bias of any other subset, and that this modification moves the distribution by at most ϵ with respect to total variation distance. The main result of [5] is actually to show that there is not generally a *closer* k-bitwise independent distribution, but this is good enough for our purposes. Note that the theorem implies that $\|\rho - \rho'\|_1 \leq 2n^k\epsilon$.

Lemma 5.2.14. *The distribution μ_ℓ is almost t-bitwise independent over $\{0,1\}^{\ell \log d}$, with maximum bias $\epsilon < d^t\varepsilon$.*

Proof. We have:

$$\epsilon = 2 \max_{I \in [\ell \log d], |I| \leq t} \Pr\left[\sum_{i \in I} X_i = 0 \pmod 2\right] - 1$$

We can see that $\Pr[\sum_{i\in I} X_i = 0]$ has a maximum with each $i \in I$ in a different element of the vector over $[d]^\ell$. This is because of the uniformity of μ_ℓ: within an element, the marginal distribution of the sum of any bits will be uniform on $\{0,1\}$. Thus, we can let $j_1, \ldots, j_t$ be the indices in $[\ell]$ of the elements that have a bit of I in them, and let i_c be the index within the j_c^{th} element that is in I:

$$\Pr\left[\sum_{i \in I} X_i = 0\right]$$

$$\leq \sum_{\vec{\alpha} \in [d]^{t-1}} \Pr\left[r_{j_t} \in \{v \in [d]^\ell : v_{i_t} = \sum_{c=1}^{t-1} \alpha_{c,i_c} \pmod 2\} \wedge r_{j_1} = \alpha_1 \wedge \cdots \wedge r_{j_{t-1}} = \alpha_{t-1}\right]$$

Let $\mathcal{T}_{\vec{\alpha}} := \{v \in [d]^\ell : v_{i_t} = \sum_{c=1}^{t-1} \alpha_{c,i_c} \pmod 2\}$:

$$\Pr\left[\sum_{i \in I} X_i = 0\right] \leq \sum_{\vec{\alpha} \in [d]^{t-1}} \sum_{\beta \in \mathcal{T}_{\vec{\alpha}}} \Pr[r_{i_t} = \beta \wedge r_{i_1} = \alpha_1 \wedge \cdots \wedge r_{i_{t-1}} = \alpha_{t-1}]$$

$$\leq \sum_{\vec{\alpha} \in [d]^{t-1}} \sum_{\beta \in \mathcal{T}_{\vec{\alpha}}} \left(\frac{1}{d^t} + \varepsilon\right)$$

$$\leq d^{t-1}\frac{d}{2}\left(\frac{1}{d^t}+\varepsilon\right)=\frac{1}{2}+\frac{d^t\varepsilon}{2}$$

Plugging in, we get:

$$\epsilon \leq 2\left(\frac{1}{2}+\frac{d^t\varepsilon}{2}\right)-1=d^t\varepsilon$$

□

By theorem 5.2.13, there is a t-bitwise independent distribution ρ_ℓ on $\{0,1\}^{\ell \log d}$ with $\|\rho_\ell - \mu_\ell\|_1 \leq 2\|\rho_\ell - \mu_\ell\|_{TV} \leq 2(\ell \log d)^t d^t \varepsilon$. By claim 5.2.12, ρ_ℓ is $\frac{t}{\log d}$-wise independent on $[d]^\ell$.

Note that $\Pr_{\rho_\ell}[2|\mathsf{vgcd}] = \langle \rho_\ell, \mathcal{S}_\ell\rangle = \langle \mu_\ell, \mathcal{S}_\ell\rangle + \langle \rho_\ell - \mu_\ell, \mathcal{S}_\ell\rangle = 1 + \langle \rho_\ell - \mu_\ell, \mathcal{S}_\ell\rangle$ (here we identify a set with its characteristic vector). We have $|\langle \rho_\ell - \mu_\ell, \mathcal{S}_\ell\rangle| \leq \|\rho_\ell - \mu_\ell\|_1$ since $\mathcal{S}_\ell$ is just a vector of 1s and 0s. This, combined with the above result, give us the following:

Theorem 5.2.15. *ρ_ℓ is a $\frac{t}{\log d}$-wise independent distribution on $[d]^\ell$ with $\Pr_{\rho_\ell}[\mathsf{vgcd} > 1] \geq 1 - 2^{t\log\ell + t\log d + t\log\log d + t/d + d + 2 - \ell/d}$.*

Proof.

$$\begin{aligned}
\|\rho_\ell - \mu_\ell\|_1 &\leq 2(\ell\log d)^t d^t \varepsilon \\
&\leq 2(\ell d \log d)^t 2^{\frac{t-\ell}{d}+d+1} && \text{by theorem 5.2.10} \\
&= 2^{(\log\ell + \log d + \log\log d)t} 2^{\frac{t-\ell}{d}+d+2} \\
&= 2^{t\log\ell + t\log d + t\log\log d + t/d + d + 2 - \ell/d}
\end{aligned}$$

So $\Pr_{\rho_\ell}[\mathsf{vgcd} > 1] > \Pr_{\rho_\ell}[2|\mathsf{vgcd}] \geq 1 - 2^{t\log\ell + t\log d + t\log\log d + t/d + d + 2 - \ell/d}$. □

We can set $t \approx d\log d$ to get $d \approx \log N$-wise independence, as desired. Though this is good enough, we conjecture that a more careful analysis could remove these unsatisfying $\log d$ terms. However we still have that for $\ell \in \Omega(d^3)$, there is a high probability that $\mathsf{vgcd} > 1$:

$$\begin{aligned}
\Pr_{\rho_{d^3}}[\mathsf{vgcd} > 2] &> 1 - 2^{d\log d\log d^3 + d\log d\log d + d\log d\log\log d + d\log d/d + d - d^3/d + 2} \\
&= 1 - 2^{4d\log^2 d + d\log d\log\log d + \log d + d - d^2 + 2} \\
&\geq 1 - 2^{cd\log^2 d - d^2}
\end{aligned}$$

This means that in a walk of length $\log^3 N$ we cannot guarantee pairwise independence of the states in the walk.

Remark 5.2.16. *A similar argument works for more general primes. Simply replace the random walk on a Boolean hypercube with the random walk on the torus.*

5.3 Rho Methods and Quantum Computing

Though there does not appear to be a rigorous analysis showing that the rho algorithm works in expected $\Theta(\sqrt{N})$ time on a single processor, or as little as $\Theta(N^{1/6})$ on $N^{1/3}$ processors, there are numerous heuristic analyses that suggest that, in practice, the algorithm does this well [48, 49, 41, 32, 31].

We now address the natural question: can a quantum computer improve this algorithm. There are several reasons to believe that it cannot. First of all, the nature of this algorithm is very serial, whereas a quantum computer gains its speedup mainly from its ability to do operations in superposition, which are very parallel in nature. A quantum computer could not speed up the process of iterating f without exploiting some structure of f.

One possible idea would be to use quantum search to find an optimal starting configuration, thereby requiring fewer steps. We show now that this approach yields no speedup.

Consider M processors iterating for T steps. If T is not sufficiently large, this process will have sub-constant probability of finding a collision, but we can run the process multiple times and eventually we'll expect to find a collision. We can quadratically reduce the number of times we must run this process by simply using amplitude amplification on the process of finding a starting configuration that yields a collision with T steps. It is not surprising that it is optimal to simply make T large enough that we expect to find a collision in 1 iteration of the process, since otherwise we are throwing away all the computation we've done so far and starting from scratch. Nevertheless, we show the simple calculations that prove that this is in fact the case.

If we run the algorithm for T steps on M processors, we have probability $p \in \Theta\left(\frac{(MT)^2}{N}\right)$ of finding a collision (under some assumption) since we look at about TM different elements. One such iteration costs $T + \sqrt{M}$, and using amplitude amplification, we will run this about $\frac{1}{\sqrt{p}}$ times before we expect to find a collision, so the total cost is:

$$\sqrt{\frac{N}{(MT)^2}}(T + \sqrt{M}) + \sqrt{M} = \frac{\sqrt{N}}{M} + \frac{\sqrt{N}}{\sqrt{M}T} + \sqrt{M}$$

This is optimized by making T as large as possible, so that we expect to find a collision in the first iteration, with no need for quantum search.

Similarly for $\mathbf{ED}_N$, the probability that one iteration finds a collision is $\Theta\left(\frac{(MT)^2}{N^2}\right)$, so we get total cost:

$$\frac{N}{MT}(T + \sqrt{M}) + \sqrt{M} = \frac{N}{M} + \frac{N}{\sqrt{M}T} + \sqrt{M}$$

Once again, we may as well make T as large as we like, so we need only a single iteration and quantum search offers no advantage.

Chapter 6

A New Deterministic Algorithm for the Hardest Instances of ED_N

In this section we outline a new algorithm, which is joint work with Frédéric Magniez, that solves a very restricted case of element distinctness. We assume that the input function has exactly one collision, and that both the domain and the range are $[N]$. It requires $\Theta(\log^2 N)$ space and $\Theta(N \log N)$ locality-sensitive time on a single processor. It can be efficiently parallelized with deterministic grid complexity $\tilde{\Theta}\left(\frac{N}{M} + \sqrt{M}\right)$, which is optimized at $\tilde{\Theta}(N^{1/3})$ when $M = N^{2/3}$. We begin by outlining the algorithm, and then we briefly discuss the relationship to streaming models of computation.

We will consider integers in $[N]$ as $\log N$ bit strings, and so we will temporarily let $[N] = \{0, \ldots, N-1\}$, for convenience, for the duration of this section. For $x \in [N]$, denote by $x[i]$ the i^{th} bit of x. As usual, we also use this notation to index vectors: $v[i]$ denotes the i^{th} entry of $\vec{v}$. Finally, we say a string x (or vector) is *consistent* with $c[i]$ for some string (vector) c if $x[i] = c[i]$.

Consider a function $f : [N] \to [N]$ with *exactly* one collision. Let m denote the tail endpoint, which is the unique point with no preimage in f. Let r denote the collision point. The sequence $K := (f(x))_{x=0}^{N-1}$ contains every point in $[N] \setminus \{m, r\}$ exactly once, r exactly twice, and m exactly 0 times. We wish to collect statistics on this set in a way that gives us information on m and/or r, using an amount of space that is logarithmic in N. In particular, if we could find r, we could find the collision pair, which is exactly the pair of preimages of r in f, using N evaluations of f by simply streaming through K and recording any value that maps to r. We note the relation to the streaming model.

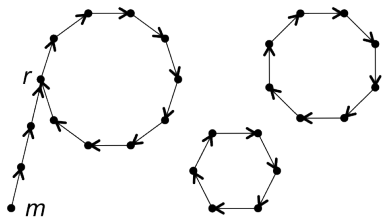

Figure 6.1: A function graph of a function with exactly one collision.

If we stream through the sequence K, we can collect some statistics on the values we see, provided these statistics don't require too much space to store. One simple set of statistics is the number of elements in the sequence with i^{th} bit 1 (resp. 0). In every bit where r and m differ, we will count $\frac{N}{2}+1$ values of r_i for the i^{th} bit, and $\frac{N}{2}-1$ values of $\bar{r_i} = m_i$, so we will learn r_i and m_i exactly. On the other hand, when $r_i = m_i$ we will learn nothing. This can be expected to happen in about half the bits given randomized input (or on fixed input if we randomize the statistics we collect, from $\langle e_i, x\rangle$ for $e_i[i] = 1$, $e_i[j] = 0$ for $j \neq i$, to $\langle s_i, x\rangle$ for randomly chosen strings s_i), but in particular, in every case it must happen in at least one bit, since $r \neq m$. This already gives a randomized $\Theta(\log N)$ space algorithm with runtime $\tilde{\Theta}(N\sqrt{N})$.

As we learn bits of r, we can narrow our search space by ignoring all elements of $[N]$ not consistent with the known bits of r. In particular, once we have learned a single bit of r in the manner described above, say $r[k]$, we can construct a subsequence of K consisting of the $\frac{N}{2}$ elements of $[N]$ that are consistent with $r[k]$. This new sequence, K', will contain r twice, and will contain every other point in $[N]$ that is consistent with $r[k]$ exactly once. Of course, it will not contain m; it shouldn't contain m, since the known bit of r is one where r and m differ, so m will not be consistent with that bit.

If we ignore the k^{th} bit of each item, K' is the sequence (up to reordering): $(0, \ldots, \frac{N}{2}-1, \tilde{r})$, where $\tilde{r}$ is just r when we ignore the k^{th} bit. We thus have every $\log N - 1$ bit string, plus a second appearance of $\tilde{r}$. We can thus learn $\tilde{r}$ by bitwise xoring the items in the sequence, since for $i \neq k$, $\sum_{x\in K'} x[i] \pmod 2 = \sum_{x\in[N/2]} x[i] + r[i] \pmod 2 = r[i]$. We can thus learn every bit of $\tilde{r}$, which is exactly every unknown bit of r. Once we have r, we can find and output the collision pair. The full algorithm is below.

Algorithm 7

Input: A black-box $f : [N] \to [N]$ with exactly one collision.

1. $\vec{d_0} := \vec{0}$, $\vec{d_1} := \vec{0}$
2. **for** $x := 0, \ldots, N-1$
3. $\quad y := f(x)$
4. $\quad$ **for** $i := 1, \ldots, \log N$
5. $\quad\quad$ **if** $y[i] = 0$ **then** $d_0[i] := d_0[i] + 1$
6. $\quad\quad$ **else** $d_1[i] := d_1[i] + 1$
7. $k := \min\{i \in \{1, \ldots, \log N\} : d_0[i] \neq d_1[i]\}$
8. $r := 0$
9. **if** $d_0[k] > d_1[k]$ **then** $r[k] := 0$
10. **else** $r[k] := 1$
11. **for** $x := 0, \ldots, N-1$
12. $\quad y := f(x)$
13. $\quad$ **if** $y[k] = r[k]$
14. $\quad\quad$ **for** $i := 1, \ldots, \log N$ such that $i \neq k$ **do** $r[i] := r[i] \oplus y[i]$
15. $C := \emptyset$
16. **for** $x := 0, \ldots, N-1$ **if** $f(x) = r$ **then** $C := C \cup \{x\}$
17. return C

First we set counters to 0. Each of d_0 and d_1 has $\log N$ entries, one for each bit position of the items in the sequence. d_0 counts the number of 0s encountered in each position and d_1 the number of 1s. The first loop performs this counting. It streams through the sequence $K = (f(x))_{x=0}^{N-1}$ and for each bit position i, checks if $f(x)[i]$ is 0 or 1 and increments the

correct counter.

We know that once this process has finished there must be at least one i such that $d_1[i] \neq d_0[i]$. We call the first such position k. We now know $r[k]$ exactly, and we know also that $m[k] \neq r[k]$.

The second loop streams through K' by simply streaming through K and ignoring all items for which $f(x)[k] \neq r[k]$. For each bit $i \neq k$, $r[i]$ is the xor of all i^{th} bits in the items of K'. Finally, in the third loop we search for preimages of r, which we now know. If $f(x) = r$, we add x to the set C, which will necessarily contain two items by the end of the loop, since r has two preimages. These two items are exactly the collision pair.

This process uses $\Theta(\log^2 N)$ space, since the counters $\vec{d_0}$ and $\vec{d_1}$ each have $\log N$ entries of size $\log N$.

The first and second loops cost $\Theta(N \log N)$ time each. The fourth loop costs $\Theta(N)$. Thus, the entire process costs $\Theta(N \log N)$ time.

Parallelization

The above algorithm can be carried out by M processors in $\Theta\left(\frac{N}{M} \log N + \sqrt{M}\right)$ time.

Consider a grid of M processors, each with $\Theta(\log^2 N)$ space. We can efficiently parallelize the above algorithm by dividing the space $[N]$ evenly among the M processors. For each of the three major loops, every processor performs the loop, restricting to its own search space. When every processor is finished the loop, answers are propagated back to some main processor. For instance, in the case of the first loop, some processor a might receive some $\vec{d_0^{(b)}}$ from one of its neighbours (possibly more), which it adds entry-wise to its own $\vec{d_0^{(a)}}$ before passing the result on to another of its neighbours. This propagation takes $\Theta(\sqrt{M})$ time, but each loop now takes $\Theta\left(\frac{N}{M} \log N\right)$ time, for a total of $\Theta\left(\frac{N}{M} \log N + \sqrt{M}\right)$ time. The optimal setting of M is thus $M = N^{2/3}$, giving a runtime $\Theta(N^{1/3} \log N)$.

This algorithm makes use of the particular structure of the input function in a way that cannot be directly applied to general f. It is somewhat interesting that this algorithm manages to solve *only* the hardest instances of $\mathbf{ED}_N$.

The algorithm illustrates an interesting relationship between solving query problems in limited space and the streaming model of computation, in which a global function must be computed on inputs received in sequence with limited storage space. A single log-space processor with black-box access to the input may be strictly more powerful, since it can

access the input adaptively, in any order, however in the problems we consider, one could argue that there is no advantage to querying adaptively.

The parallel between these two settings is of interest, and likely not fully explored. It will be interesting future work to see what else can be gained by considering the problems in this way.

Chapter 7

A Survey of Quantum Lower Bound Methods and their Application to Collision Finding and Element Distinctness

In this chapter, we survey the major methods of proving lower bounds for quantum query complexity. In Chapter 8 we will discuss the prospects for applying each to the locality-sensitive and grid models.

It may be useful to reflect here on what exactly a query lower bound tells us about a problem. A query lower bound on collision finding, for instance, does not tell us that, given a function f, we will not be able to find a collision in f with fewer than T queries. A query bound tells us that if we hope to solve the problem in fewer than T steps (and so fewer than T queries) we will need to exploit something about the structure of f: we cannot simply treat it as a black-box.

There are two main techniques for deriving quantum query lower bounds, both extensions of classical techniques: the polynomial method [12], and the adversary method [6].

There are several equivalent variations of what we will call the *weighted adversary method* as well as a strictly stronger version, called the *negative weights adversary method*. The negative weights adversary lower bound is a semidefinite program whose optimal solution is a tight lower bound for any Boolean problem, however finding this optimal solution may be incredibly difficult for a given problem.

To lower bound the problems we consider, it suffices, and is often much easier, to lower bound a related decision problem, which is a Boolean function $\mathbf{F} : [R]^N \to \{0,1\}$. For $\mathbf{ED}_N$ there is the equivalent problem of determining whether or not the input function is 1-to-1, and for $\mathbf{COL}_N$ there is the problem of determining whether an input function is 1-to-1 with the promise that it is either 1-to-1 or 2-to-1. Therefore, in a slight abuse of notation, we will, for the duration of this section, consider $\mathbf{ED}_N$ and $\mathbf{COL}_N$ to be these related decision problems. Though the methods we will discuss can be extended to non-Boolean functions, we will only consider their application to Boolean functions.

The problems we consider have inputs $f : [N] \to [R]$, which can be viewed as vectors in $[R]^N$, or equivalently, as binary strings of length $\tilde{N} := N \log R$. We thus consider each of our problems as a Boolean function $\mathbf{F} : [R]^N \to \{0,1\}$ or $\mathbf{F} : \{0,1\}^{\tilde{N}} \to \{0,1\}$ as appropriate. Throughout this chapter, $f[i]$ will denote the i^{th} bit of $f \in \{0,1\}^{\tilde{N}}$, and $f(x)$ will denote the x^{th} entry of $f \in [R]^N$.

To make a lower bound technique immediately applicable to a broad class of problems, it is often useful to relate it to some well-studied measure of the hardness of a problem. Then for any problem to which we can apply this measure, we get an immediate lower bound as a function of this measure. We now present some such measures to which we will relate the lower bound techniques.

Certificate Complexity

Definition 7.0.1. *Let $\mathbf{F} : [R]^N \to \{0,1\}$ be a Boolean function. A* 1-certificate *is an assignment $C : \mathcal{S} \to [R]$ for $\mathcal{S} \subseteq [N]$ such that $\mathbf{F}(f) = 1$ whenever f is consistent with C. The size of C is $|\mathcal{S}|$. We can similarly define a* 0-certificate*.*

The certificate complexity, $\mathsf{C}_f(\mathbf{F})$, of $\mathbf{F}$ on f *is the size of the smallest $\mathbf{F}(f)$-certificate that agrees with f. The* certificate complexity $\mathsf{C}(\mathbf{F})$ of $\mathbf{F}$ *is the maximum $\mathsf{C}_f(\mathbf{F})$ over all f. The* 1-certificate complexity $\mathsf{C}^{(1)}(\mathbf{F})$ of $\mathbf{F}$ *is the maximum $\mathsf{C}_f(\mathbf{F})$ over all f with $\mathbf{F}(f) = 1$. The* 0-certificate complexity of $\mathbf{F}$ *is similarly defined.*

Example 7.0.2. *$\mathsf{C}(\mathbf{OR}_N) = N$, since $\mathsf{C}_f(\mathbf{OR}_N) = N$ for $f = 0$, since changing any of its entries changes $\mathbf{F}(f)$ from 0 to 1. So the smallest $\mathbf{F}(f) = 0$-certificate that agrees with f has $\mathcal{S} = [N]$; in other words, the certificate is all of f.*

On the other hand, $\mathsf{C}^{(1)}(\mathbf{OR}_N) = 1$, since any $C : \{i\} \to \{0,1\}$ with $C(i) = 1$ is a 1-certificate for $\mathbf{OR}_N$, and every f such that $\mathbf{F}(f) = 1$ is consistent with some such certificate, since it has some $f[i] = 1$.

Example 7.0.3. $\mathsf{C}(\mathbf{PARITY}_N) = \mathsf{C}^{(0)}(\mathbf{PARITY}_N) = \mathsf{C}^{(1)}(\mathbf{PARITY}_N) = N$, *since* $N-1$ *of the bits of* f *never determine* $\mathbf{F}(f)$.

Example 7.0.4. $\mathsf{C}(\mathbf{ED}_N) = N$, *since* $\mathsf{C}_f(\mathbf{F}) = N$ *for any* f *that has no collision. Given any assignment* C *of size less than* N *that is consistent with a* 1*-to-*1 *function, there are several functions consistent with* C *that are not* 1*-to-*1 *(set the unassigned entry to the same value as any of the* $N-1$ *assigned entries).*

However, $\mathsf{C}^{(1)}(\mathbf{ED_N}) = 2$, *since a* 1*-certificate for any* f *with a collision* (x, y) *is just the assignment* $C : \{x, y\} \to \mathcal{X}$ *given by* $C(x) = C(y) = f(x)$.

Similarly, the certificate complexity of $\mathbf{COL}_N$ *is* $\mathsf{C}(\mathbf{COL}_N) = \frac{N}{2} + 1$, *though it has* 1*-certificate complexity* $\mathsf{C}^{(1)}(\mathbf{COL}_N) = 2$.

It is a notable feature of the problems we are interested in that they have constant 1-certificate complexity.

Block Sensitivity

Definition 7.0.5. *Let* $\mathbf{F} : \{0,1\}^{\tilde{N}} \to \{0,1\}$ *be a total Boolean function,* $f \in [R]^N \equiv \{0,1\}^{\tilde{N}}$, *and* $B \subseteq [\tilde{N}]$. *Let* f^B *denote the vector obtained from* f *by flipping all bits with indices in* B. *We say that* $\mathbf{F}$ *is* sensitive to B on f *if* $\mathbf{F}(f) \neq \mathbf{F}(f^B)$. *The* block sensitivity, $\mathsf{bs}_f(\mathbf{F})$, *of* $\mathbf{F}$ *on* f *is the maximum* t *such that there exist disjoint* $B_1, \ldots, B_t$ *such that* $\mathbf{F}$ *is sensitive to each* B_i *on* f. *The block sensitivity* $\mathsf{bs}(\mathbf{F})$ *of* $\mathbf{F}$ *is the maximum* $\mathsf{bs}_f(\mathbf{F})$ *over all* f.

Given a function $\mathbf{F} : [R]^N \to \{0,1\}$, we can consider a very similar, possibly more natural notion. Let $B \subset [N]$. Say that $\mathbf{F}$ is sensitive to B on f if there exists $g \in [R]^N$ such that $\mathbf{F}(f) \neq \mathbf{F}(g)$ and $f(x) = g(x)$ if and only if $x \notin B$. If we define the block sensitivity using this notion of sensitivity, its value will differ by at most a factor of $\log R$. It is therefore not particularly important which notion we employ.

Example 7.0.6. $\mathsf{bs}(\mathbf{OR}_N) = N$, *since* $\mathsf{bs}_f(\mathbf{OR}_N) = N$ *for* $f = 0$ *(with* $B_x = \{x\}$ *for* $x = 1, \ldots, N$*).*

Example 7.0.7. $\mathsf{bs}(\mathbf{MAJORITY}_N) = \frac{N}{2}$, *since* $\mathsf{bs}_f(\mathbf{MAJORITY}_N) = \frac{N}{2}$ *for any* f *with* $|f| = \frac{N}{2}$ *(with* $B_x = \{x\}$ *for each* x *with* $f(x) = 1$*).*

Example 7.0.8. *Let* f *be a* 1*-to-*1 *function with image* $[N] \subseteq [R]$. *Then flipping any bit in the last* $\log N$ *bits of any entry in* f *will create a collision. Thus* $\mathsf{bs}(\mathbf{ED}_N) \geq N \log N$.

Lemma 7.0.9. *[42] Let* $\mathbf{F} : \{0,1\}^{\tilde{N}} \to \{0,1\}$. *Then* $\mathsf{bs}(\mathbf{F}) \leq \mathsf{C}(\mathbf{F}) \leq \mathsf{bs}(\mathbf{F})^2$.

Proof. If $C : \mathcal{S} \to \{0,1\}$ is a $\mathbf{F}(f)$-certificate, and B is a block to which f is sensitive, then since $\mathbf{F}(f) \neq \mathbf{F}(f^B)$ it can't be that $B \cap \mathcal{S} = \emptyset$, so $\mathcal{S}$ must be as large as any set of disjoint blocks to which f is sensitive. Thus $\mathsf{bs}_f(\mathbf{F}) \leq \mathsf{C}_f(\mathbf{F})$ for all f, so $\mathsf{bs}(\mathbf{F}) \leq \mathsf{C}(\mathbf{F})$.

Let $B_1, \ldots, B_b$ be disjoint minimal sets of indices with $b = \mathsf{bs}_f(\mathbf{F}) \leq \mathsf{bs}(\mathbf{F})$. Define $C : \bigcup_{i=1}^{b} B_i \to \{0,1\}$ by $C(j) = f[j]$.

Suppose C is not an $\mathbf{F}(f)$-certificate. Then let f' be consistent with C but have $\mathbf{F}(f') \neq \mathbf{F}(f)$. Define B_{b+1} so that $f' = f^{B_{b+1}}$. Then $\mathbf{F}$ is sensitive to B_{b+1} under f, and B_{b+1} is disjoint from $B_1, \ldots, B_b$, since f and f' are both consistent with C, so they agree with all indices in $\bigcup_{i=1}^{b} B_i$. This contradicts the maximality of b. Thus, C is an $\mathbf{F}(f)$-certificate.

Suppose $|B_i| > \mathsf{bs}(\mathbf{F}) \geq \mathsf{bs}_{f^{B_i}}(\mathbf{F})$ for some i. Since $|B_i|$ is minimal, we must have that flipping any $j \in B_i$ in f^{B_i} takes $\mathbf{F}(f^{B_i})$ back to $\mathbf{F}(f)$. Thus $\mathbf{F}$ is sensitive to $\{j\}_{j \in B_i}$, so $\mathsf{bs}_{f^{B_i}}(\mathbf{F}) \geq |B_i|$, which is a contradiction. Thus $|B_i| \leq \mathsf{bs}(\mathbf{F})$ for all i.

Putting these together, we have an $\mathbf{F}(f)$-certificate C with size $\sum_{i=1}^{b} |B_i| \leq \mathsf{bs}(\mathbf{F})^2$. Thus $\mathsf{C}(\mathbf{F}) \leq \mathsf{bs}(\mathbf{F})^2$. □

7.1 The Polynomial Method

The Quantum Polynomial Method [12] extends the Classical Polynomial Method (see, for example, [13]) for proving lower bounds in the decision tree model to the quantum query model.

Let $\mathbf{F} : \{0,1\}^{\tilde{N}} \to \{0,1\}$ be a Boolean function. Then $\mathbf{F}$ can be *represented* by an $\tilde{N}$-variate polynomial $p \in \mathbb{R}[x_1, \ldots, x_{\tilde{N}}]$ — that is, $\mathbf{F}(f) = p(f)$ for all $f \in \{0,1\}^{\tilde{N}}$. We define $\deg(\mathbf{F})$ to be the minimum degree of any polynomial that represents $\mathbf{F}$.

More generally, we say that $p \in \mathbb{R}[x_1, \ldots, x_{\tilde{N}}]$ *approximates* $\mathbf{F}$ if $|\mathbf{F}(f) - p(f)| \leq \frac{1}{3}$ for all $f \in \{0,1\}^{\tilde{N}}$. We define $\widetilde{\deg}(\mathbf{F})$ to be the minimum degree of any polynomial that approximates $\mathbf{F}$.

A polynomial of the form $p(x_1, \ldots, x_{\tilde{N}}) = \sum_{\mathcal{S} \subset [\tilde{N}]} \Pi_{i \in \mathcal{S}} x_i$ is called *multilinear*. Notice that since $x^k = x$ for any k whenever $x \in \{0,1\}$, we can, without loss of generality, restrict to multilinear p. There is a unique multilinear p such that p represents $\mathbf{F}$ and $\deg(p) = \deg(\mathbf{F})$.

We now examine the relationship between polynomials and quantum query algorithms.

Lemma 7.1.1. *Let* A *be a quantum circuit that makes* T *queries to some input* $f \in \{0,1\}^{\tilde{N}}$*. Then there are complex-valued* $\tilde{N}$*-variate multilinear polynomials* $p_{i,w}$ *for each basis state* $|i,w\rangle$*, each of degree at most* T*, such that the final state of the circuit is* $\sum_{i,w} p_{i,w}(f[1],\ldots,f[\tilde{N}])|i,w\rangle$ *for any input* f*.*

Proof. Let $|\psi_f^t\rangle$ be the state just before the t^{th} query, so $|\psi_f^{t+1}\rangle = U_t\mathcal{O}_f|\psi_f^t\rangle$.

The amplitudes of $|\psi_f^0\rangle$ are necessarily independent of f, since no query to f has been made yet. Thus, they are constant (of degree 0) in the bits of f.

A query $\mathcal{O}_f$ maps $|i,w\rangle \mapsto (-1)^{f[i]}|i,w\rangle = (1-2f[i])|i,w\rangle$ so if the amplitude of $|i,w\rangle$ in $|\psi_f^t\rangle$ is $\alpha(f)$ then the amplitude after application of $\mathcal{O}_f$ is $(1-2f[i])\alpha(f)$, which has degree at most $\deg(\alpha)+1$ in $f[1],\ldots,f[\tilde{N}]$. Thus the maximum degree over all amplitudes increases by at most 1 with each query.

Note that the U_t cannot increase the maximum degree in f, since the amplitudes after applying U_t are just linear combinations of the amplitudes before. Thus after T steps, the maximum degree in f of any amplitude is at most T. □

This proof exploits various features of a quantum algorithm, namely the nature of a query operator, and the linearity of all other operators. We have so far said nothing about the algorithm outputting the correct answer. The following lemma moves us in this direction.

Lemma 7.1.2. *Let* A *be a quantum algorithm that makes* T *queries to* f *and* $\mathcal{B}$ *a set of basis states. Then there is a real-valued multilinear polynomial* $P(f)$ *of degree at most* $2T$ *such that* $P(f)$ *is the probability of observing a state from* $\mathcal{B}$ *upon measuring the final state of* A *on input* f*.*

Proof. By the previous lemma, write the final state as $\sum_{i,w} p_{i,w}(f)|i,w\rangle$. Then

$$\begin{aligned} P(f) &:= \textstyle\sum_{(i,w)\in\mathcal{B}} |p_{i,w}(f)|^2 \\ &= \textstyle\sum_{(i,w)\in\mathcal{B}} |\Re(p_{i,w}(f)) + i\Im(p_{i,w}(f))|^2 \\ &= \textstyle\sum_{(i,w)\in\mathcal{B}} \Re(p_{i,w}(f))^2 + \sum_{(i,w)\in\mathcal{B}} \Im(p_{i,w}(f))^2 \end{aligned}$$

which has degree at most $2T$, since each of $\Im(p_{i,w})(f) = \Im(p_{i,w}(f))$ and $\Re(p_{i,w})(f) = \Re(p_{i,w}(f))$ have degree at most T for each (i,w), and $P(f)$ is real-valued, since $\Re(p_{i,w})$ and $\Im(p_{i,w})$ are real-valued. □

If we let $\mathcal{B}$ be the set of basis states with right-most bit 1 (the *accepting* states) then we can write the acceptance probability of a circuit as a degree $2T$ polynomial $P(f)$. In

the exact model, we require $P(f) = \mathbf{F}(f)$, that is, if $\mathbf{F}(f) = 1$, the algorithm should accept (output 1) with probability 1, so P must represent $\mathbf{F}$, and thus $2T \geq \deg(\mathbf{F})$. However, we are concerned with bounded error algorithms, so we merely require that $P(f) \geq 1 - \epsilon$ if $\mathbf{F}(f) = 1$ and $P(f) \leq \epsilon$ if $\mathbf{F}(f) = 0$. Setting $\epsilon = \frac{1}{3}$, we get $|P(f) - \mathbf{F}(f)| \leq \frac{1}{3}$, or in other words, P approximates $\mathbf{F}$. This directly yields the following:

Theorem 7.1.3. $Q(\mathbf{F}) \geq \frac{\widetilde{\deg}(\mathbf{F})}{2}$

The difficulty is now in lower bounding the approximate degree of $\mathbf{F}$. In the remainder of this section, we will consider techniques for doing so.

Definition 7.1.4. *A function* $\mathbf{F} : \{0,1\}^{\tilde{N}} \to \{0,1\}$ *is* symmetric *if for all* $f, g \in \{0,1\}^{\tilde{N}}$ *with* $|f| = |g|$, $\mathbf{F}(f) = \mathbf{F}(g)$. *That is,* $\mathbf{F}$ *depends only on the Hamming weight of its input.*

An example of a symmetric problem is $\mathbf{OR}_N$. The problems of $\mathbf{COL}_N$ and $\mathbf{ED}_N$ are not symmetric by this definition, however they do exhibit certain symmetries. If we view the inputs as binary strings, then permuting the bits may change the output, however, if we view the inputs as vectors in $[R]^N$, permuting the entries does not change the output. These types of symmetries are exploited in polynomial lower bounds for $\mathbf{ED}_N$ and $\mathbf{COL}_N$, which we will briefly discuss in Section 7.1.2.

Definition 7.1.5. *We define the* symmetrization *of* $p : \mathbb{R}^{\tilde{N}} \to \mathbb{R}$ *as* $p^{sym}(f) := \frac{1}{\tilde{N}!} \sum_{\pi \in S_{\tilde{N}}} p(\pi(f))$ *where* $\pi(f) = (f[\pi(1)], \ldots, f[\pi(\tilde{N})])$.

Note that if p is symmetric, then $p^{sym} = p$. Also note that $\deg(p) \geq \deg(p^{sym})$. Finally, note that $p^{sym}(f)$ is the expectation of $p(\pi(f))$ over a uniform choice of π.

Lemma 7.1.6. *[12] Let* $p : \mathbb{R}^{\tilde{N}} \to \mathbb{R}$ *be a multilinear polynomial. There exists a univariate polynomial* $q : \mathbb{R} \to \mathbb{R}$ *such that* $\deg(q) \leq \deg(p)$ *and* $p^{sym}(f) = q(|f|)$ *for all* $f \in \{0,1\}^{\tilde{N}}$.

Proof. Let V_j be the sum of all possible terms of degree j (terms with j variables). Each V_j has $\binom{\tilde{N}}{j}$ terms.

Since p^{sym} is symmetric, if it has one term from V_j with coefficient c_j, then it must have every term from V_j with coefficient c_j. Thus we can write $p^{sym} = c_0 + \sum_{j=1}^{d} c_j V_j$ for $d = \deg(p^{sym})$.

Evaluating $V_j(f)$ for $f \in \{0,1\}^{\tilde{N}}$ gives the number of terms in which only 1-bits of f appear, corresponding to subsets of $\mathcal{S} \subseteq [\tilde{N}]$ of size j such that $\forall i \in \mathcal{S}, f[i] = 1$. There are

$|f|$ 1-bits in f so $V_j(f) = \binom{|f|}{j}$. We can thus write:

$$p^{sym}(f) = c_0 + \sum_{j=1}^{d} c_j \binom{|f|}{j}$$

whenever $f \in \{0,1\}^{\tilde{N}}$. We then get the desired q by setting:

$$q(|f|) = c_0 + \sum_{j=1}^{d} c_j \binom{|f|}{j}$$

□

Thus, if we can lower bound the degree of the univariate polynomial q, we can lower bound the approximate degree of $\mathbf{F}$. The following theorem is an important tool for bounding the degree of a polynomial which will come up again in Section 7.1.2.

Theorem 7.1.7 (Paturi's Theorem)**.** *[43] Given b_1, b_2, and $a > 0$, let $p : \mathbb{R} \to \mathbb{R}$ be a polynomial such that:*

1. *$b_1 \leq p(i) \leq b_2$ for all $i \in \{0, \ldots, a\}$*
2. *$|p'(x)| \geq O(1)$ for some $x \in [0, a]$*

Then $\deg(p) \geq \Omega\left(\sqrt{\frac{a}{b_2 - b_1}}\right)$.

7.1.1 Relation to Block Sensitivity

Theorem 7.1.8. *[12] If $\mathbf{F}$ is a Boolean function, $Q(\mathbf{F}) \geq \sqrt{\mathsf{bs}(\mathbf{F})/16}$.*

Proof. Let A be a quantum algorithm making $T = Q(\mathbf{F})$ queries to f and computing $\mathbf{F}(f)$ with bounded error $\leq \frac{1}{3}$. Let P be a degree $2T$ polynomial that approximates $\mathbf{F}$.

Let $b = \mathsf{bs}(\mathbf{F}) = \mathsf{bs}_f(\mathbf{F})$ for some f, and $B_0, \ldots, B_{b-1}$ a set of blocks to which $\mathbf{F}$ is sensitive on f. Without loss of generality, assume $\mathbf{F}(f) = 0$.

Fix $v \in \mathbb{R}^b$ and define $u \in \mathbb{R}^{\tilde{N}}$ as:

$$u[i] = \begin{cases} v[j] & \text{if } f[i] = 0 \text{ and } i \in B_j \\ v[\bar{j}] & \text{if } f[i] = 1 \text{ and } i \in B_j \\ f[i] & \text{else} \end{cases}$$

Define $q(v) = P(u)$, a b-variate polynomial of degree $\leq 2T$ with the following properties:

- $q(v) \in [0,1]$ for all $v \in \{0,1\}^b$ since P is a probability function.
- $|q(0) - 0| = |P(f) - \mathbf{F}(f)| \leq \frac{1}{3}$, so $0 \leq q(\vec{0}) \leq \frac{1}{3}$, since $v = 0 \Rightarrow u = f$.
- $|q(v) - 1| = \left|P(f^{B_i}) - \mathbf{F}(f^{B_i})\right| \leq \frac{1}{3}$ if v has $v[i] = 1$ and $v[j] = 0$ for all $j \neq i$, so $\frac{2}{3} \leq q(v) \leq 1$ if $|v| = 1$.

The last observation follows from the fact that if $v = (0, \ldots, 0, 1, 0, \ldots, 0)$ with $v[i] = 1$, then $u = f^{B_i}$. To see this, note that for $j \notin B_i$, $u[j] = f[j]$ if $j \notin \bigcup_{k=0}^{b-1} B_k$, and if $j \in B_k$ for some $k \neq i$, then $u[j] = f[j]$, and if $j \in B_i$, then $u[j] = \overline{f[j]}$.

Let r be the univariate polynomial of degree $2T$ such that $r(|v|) = q^{sym}(v)$ for all $v \in \{0,1\}^b$. We have $0 \leq r(i) \leq 1$ for $i = 1, \ldots, b$. We also have $r(0) \leq \frac{1}{3}$ and $r(1) \geq \frac{2}{3}$ so for some $x \in [0,1]$ we have $r'(x) \geq \frac{1}{3}$. Thus by Theorem 7.1.7 we have:

$$\deg(r) \geq \sqrt{\frac{1}{3}\frac{b}{\frac{1}{3}+1-0}}$$

$$2T \geq \sqrt{\frac{b}{4}}$$

$$T \geq \sqrt{\frac{b}{16}}$$

$$Q(\mathbf{F}) \geq \sqrt{\frac{\mathsf{bs}(\mathbf{F})}{16}}$$

□

The key to the ease of dealing with symmetric polynomials is that there is a single variable, the Hamming weight of the input, which determines the outcome, so we can easily construct a univariate polynomial, which is much easier to deal with by, for instance, applying Paturi's Theorem. The tricks involved in applying polynomial lower bounds to more general problems are often aimed at reducing the number of variables to as few as possible by exploiting symmetries in the problem. We give several examples in the following section.

7.1.2 Applications to Collision Finding and Element Distinctness

In this section we briefly summarize polynomial lower bounds for $\mathbf{ED}_N$ and $\mathbf{COL}_N$ and the techniques used to obtain them. The tight quantum query complexity lower bound of $Q(\mathbf{COL}_N) \in \Omega(N^{1/3})$ (which implies $Q(\mathbf{ED}_N) \in \Omega(N^{2/3})$ by Lemma 3.0.8) was developed in stages. In 2002, Aaronson [1] developed novel techniques to extend the polynomial method and proved a lower bound of $Q(\mathbf{COL}_N) \in \Omega(N^{1/5})$. Shortly after, this proof was extended by Shi [46, 3] using similar techniques as well as novel ones to $Q(\mathbf{COL}_N) \in \Omega(N^{1/4})$ or, if we restrict to inputs with $R \geq \frac{3}{2}N$, a lower bound of $\Omega(N^{1/3})$. In 2003, Ambainis [9] and Kutin [35] simultaneously and independently extended these results to $Q(\mathbf{COL}_N) \in \Omega(N^{1/3})$. Kutin did so by extending the techniques of Aaronson and Shi, in a somewhat simplified and quite elegant way, and Ambainis by proving a general result about lower bounds on functions with large range applying to functions with small range. We will briefly outline the proof of Kutin, which uses techniques of Aaronson and Shi, and then summarize the result of Ambainis.

Let A be a quantum query algorithm that computes $\mathbf{COL}_N$ with bounded error in T queries.

Define $N \times R$ Boolean variables as follows. For all $(x, y) \in [N] \times [R]$, $\delta_{x,y}(f) = 1$ if and only if $f(x) = y$. By a similar argument as Lemma 7.1.1, the amplitudes of each basis state of $|\psi_f^T\rangle$ are multilinear polynomials in the $\delta_{x,y}(f)$ of degree at most T. Similarly, the acceptance probability is a multilinear polynomial in the $\delta_{x,y}$ of degree at most $2T$.

Since P is a multilinear polynomial in the $\delta_{x,y}(f)$, each term is the product of some subset of the $\delta_{x,y}$, and some coefficient. For any subset $\mathcal{S} \subseteq [N] \times [R]$ we can write: $I_{\mathcal{S}}(f) = \Pi_{(x,y)\in\mathcal{S}}\delta_{x,y}(f)$. Since each x maps to exactly one image y, any term with both δ_{x,y_1} and δ_{x,y_2} for $y_1 \neq y_2$ will be 0 (since $f(x) = y_1$ and $f(x) = y_2$ cannot both be true). Thus, we can restrict to subsets $\mathcal{S}$ in which each $x \in [N]$ appears in at most one pair. The subset $\mathcal{S}$ then looks like $\bigcup_{i=1}^{t} \mathcal{S}_i \times \{y_i\}$ for some disjoint $\mathcal{S}_i \subseteq [N]$ and $y_i \in [R]$. We can then write $I_{\mathcal{S}}$ as $I_{\mathcal{S}} = \Pi_{i=1}^{t}\Pi_{x\in\mathcal{S}_i}\delta_{x,y_i}(f)$, which is a monomial in the $\delta_{x,y}$ of degree at most $\sum_{i=1}^{t} |\mathcal{S}_i|$. Then we have $P(f) = \sum_{\mathcal{S}\subseteq[N]\times[R]} c_{\mathcal{S}} I_{\mathcal{S}}(f)$ for some coefficients $c_{\mathcal{S}}$.

Let $\mathcal{F}(N, R)$ be the set of functions $f : [N] \to [R]$, (isomorphic to $[R]^N$). To look at symmetries among these functions, Kutin turns to $S_N \times S_R$. For any $(\sigma, \tau) \in S_N \times S_R$, define $\Gamma_{\tau,\sigma} : \mathcal{F}(N, R) \to \mathcal{F}(N, R)$ by $\Gamma_{\tau,\sigma}(f) = \tau \circ f \circ \sigma$. Note that $\Gamma_{\tau,\sigma}(f)$ is 1-to-1 if and only if f is 1-to-1. Thus, $\mathbf{COL}_N$ is invariant under application of $\Gamma_{\tau,\sigma}$.

For $m \in [N]$, $k, \ell > 0$, such that $k|m$ and $\ell|N - m$ (call such a triple (m, k, ℓ) *valid*), let $f_{m,k,\ell}$ denote a function in $\mathcal{F}(N, R)$ such that $f_{m,k,\ell}|_{[m]}$ is a k-to-1 function with image in $[m]$, and $f_{m,k,\ell}|_{[m+1,N]}$ is ℓ-to-1, with image in $[m+1, R]$. Consider $\Gamma_{\tau,\sigma}(f_{m,k,\ell}) = \tau \circ f_{m,k,\ell} \circ \sigma$. We

have that $\Gamma_{\tau,\sigma}(f_{m,k,\ell})|_{\sigma([m])}$ is a k-to-1 function on $\tau([m])$, since if $f_{m,k,\ell}$ maps $\{x_1, \ldots, x_k\}$ to y, then $\Gamma_{\tau,\sigma}(f_{m,k,\ell})$ maps $\sigma(x_1), \ldots, \sigma(x_k)$ to $\tau(y)$ (and τ and σ are permutations, and so cannot introduce new collisions). Similarly, $\Gamma_{\tau,\sigma}(f_{m,k,\ell})|_{\sigma([m+1,N])}$ is ℓ-to-1 with image in $\tau([m+1, R])$.

Lemma 7.1.9. *Define a polynomial $q(m, k, \ell)$ by $q(m, k, \ell) = E_{\sigma,\tau}[P(\Gamma_{\tau,\sigma}(f_{m,k,\ell}))]$. Then q has degree deg(P) in m, k, and ℓ.*

The expectation is simply a kind of symmetrization.

It remains only to show that $\deg(q) \in \Omega(N^{1/3})$. To apply Paturi's Theorem, we just need to show that q is within some constant sized interval $[b_1, b_2]$ for all integral values of the variables in some range of size c, and that q has some constant increase or decrease, over some constant interval within that range. Then we get $\deg(q) \in \Omega(\sqrt{c})$. A modified version of Paturi's Theorem is used for the second part:

Theorem 7.1.10. *[46, 35] Given integers $a < b$ and a real number $\alpha \in [a, b]$, let $p : \mathbb{R} \to \mathbb{R}$ be a polynomial such that:*

1. *$p(x) \leq 1$ for all integers $x \in [a, b]$*
2. *$|p(\alpha - 1) - p(\alpha)| \geq c$ for some constant $c > 0$*

Then deg$(p) \in \Omega(\sqrt{(\alpha - a + 1)(b - \alpha + 1)})$.

Since q is a convex combination of $P(f)$ for various functions $f \in \mathcal{F}(N, R)$, we have $q(m, k, \ell) \in [0, 1]$ for all valid triples of integers (m, k, ℓ). Consider $q(m, 1, 1)$ for any m:

$$q(m, 1, 1) = \sum_{\sigma,\tau} p(\sigma, \tau) P(\Gamma_{\tau,\sigma}(f_{m,1,1}))$$

where p is a probability function. Since $f_{m,1,1}$ is a 1-to-1 function, $\Gamma_{\tau,\sigma}(f_{m,1,1})$ is a 1-to-1 function, so $P(\Gamma_{\tau,\sigma}(f_{m,1,1})) \leq \frac{1}{3}$. Thus we can write:

$$q(m, 1, 1) \leq \sum_{\sigma,\tau} p(\sigma, \tau) \frac{1}{3} = \frac{1}{3}$$

Thus $q(m, 1, 1) \in [0, \frac{1}{3}]$. On the other hand, consider $q(m, 2, 2)$:

$$q(m, 2, 2) = \sum_{\sigma,\tau} p(\sigma, \tau) P(\Gamma_{\tau,\sigma}(f_{m,2,2})) \geq \sum_{\sigma,\tau} p(\sigma, \tau) \frac{2}{3} = \frac{2}{3}$$

since $\Gamma_{\tau,\sigma}(f_{m,2,2})$ is a 2-to-1 function, so A must accept with probability at least $\frac{2}{3}$. Thus $q(m,2,2) \in [\frac{2}{3}, 1]$.

Suppose $q(\frac{N}{2}, 1, 2) \geq \frac{1}{2}$. Let c be the least integer for which $\left|q(\frac{N}{2}, 1, c)\right| \geq 2$ (so $c \not| \frac{N}{2}$, or $(\frac{N}{2}, 1, c)$ would be a valid triple). Then of course we have $q \in [-2, 2]$ on the range $(m, k, \ell) \in \{\frac{N}{2}\} \times \{1\} \times \{0, \ldots, c\}$. Further, we have $|q(m,1,1) - q(m,1,2)| \geq \frac{1}{2} - \frac{1}{3} = \frac{1}{6}$, so we can apply Paturi's Theorem to get $\deg(q) \in \Omega(\sqrt{\frac{c}{4}}) = \Omega(\sqrt{c})$.

We can also consider elements in the range $(m, k, \ell) \in \{0, c, 2c, \ldots, m\lfloor\frac{N}{m}\rfloor\} \times \{1\} \times \{c\}$. In this range, we have $q(m, k, \ell) \in [0, 1]$ but $q(\frac{N}{2}, 1, c) \geq 2$, so the value of q changes by at least 1 on an interval of size less than 1. Furthermore, this interval occurs at distance $\frac{N}{2c}$ points from the edge of the range. Thus, we can apply the modified version of Paturi's Theorem to get $\deg(q) \in \Omega\left(\sqrt{\frac{N}{2c}\frac{N}{2c}}\right) \in \Omega\left(\frac{N}{c}\right)$. Thus, we have $\deg(q) \in \Omega\left(\sqrt{c} + \frac{N}{c}\right) \in \Omega(N^{1/3})$.

In the case where $q(m,1,2) < \frac{1}{2}$, the argument is similar.

To summarize, this method succeeded by finding symmetries in the problem so that a new polynomial with fewer variables could be constructed, and applying Paturi's Theorem to the new polynomial to lower bound its degree, and thus the degree of P.

Ambainis shows the following, which implies tight lower bounds from the work of Aaronson and Shi:

Theorem 7.1.11. *Let $\mathbf{F} : \mathcal{F}(N,R) \to \{0,1\}$ be a Boolean problem such that for all $\sigma \in S_N$ and $\tau \in S_R$, $\mathbf{F}(f) = \mathbf{F}(\tau \circ f \circ \sigma)$. Let $\mathbf{F}'$ be the restriction of $\mathbf{F}$ to $\mathcal{F}(N,N)$. Then $\widetilde{deg}(\mathbf{F}) = \widetilde{deg}(\mathbf{F}')$ where degrees are in $\delta_{x,y}$.*

Call a function $\mathbf{F}$ such that $\mathbf{F}(f) = \mathbf{F}(\tau \circ f \circ \sigma)$ for all $f \in \mathcal{F}(N,R)$, $\sigma \in S_N$ and $\tau \in S_R$, *symmetric in $S_N \times S_R$*.

To prove this result, Ambainis introduces new variables: $z_y = |f^{-1}(y)|$ for each $y \in [R]$. He shows that for all $\mathbf{F}$ symmetric in $S_N \times S_R$, there exists a polynomial $q(z_1, \ldots, z_R)$ of degree d that approximates $\mathbf{F}$ if and only if there exists a polynomial $p(\delta_{1,1}, \ldots, \delta_{N,R})$ of degree d that approximates $\mathbf{F}$. This is another example of finding symmetries in a problem in order to reduce the number of variables.

Certainly if $q(z_1, \ldots, z_R)$ approximates $\mathbf{F}$, then $q(z_1, \ldots, z_N, 0, \ldots, 0)$ approximates $\mathbf{F}'$ and this only makes the degree smaller, so $\widetilde{\deg}(\mathbf{F}) \geq \widetilde{\deg}(\mathbf{F}')$.

On the other hand, suppose we have a polynomial $q'(z_1, \ldots, z_N)$ that approximates $\mathbf{F}'$. Let q'_{sym} be the symmetrization of q', which has the same degree (and is just q' if q' is

symmetric). Then we can write $q'_{sym} = \sum_E c_E q'_E$ where E is taken over all ordered subsets of $\mathbb{Z}^+$ in which the members add up to at most d, and $q'_E = \sum_{\mathcal{S}\subseteq[N]:|\mathcal{S}|=|E|} \Pi_{s_i\in\mathcal{S}} z_{s_i}^{E_i}$.

Consider $q = \sum_E c_E q_E$ where $q_E = \sum_{\mathcal{S}\subseteq[R]:|\mathcal{S}|=|E|} \Pi_{s_i\in\mathcal{S}} z_{s_i}^{E_i}$. Then q actually approximates $\mathbf{F}$. This relies on the fact that at most N members of $[R]$ have a preimage in any f, so we certainly don't need to consider q_E with $|E| > N$. We have $q(z_1, \ldots, z_N, 0, \ldots, 0) = q'(z_1, \ldots, z_N)$, so q approximates $\mathbf{F}$ for any function $f : [N] \to [N]$. Given any $f' : [N] \to [R]$, we can construct $f : [N] \to [N]$ by applying some appropriate permutations to get $f' = \tau \circ f \circ \sigma$. Since q is symmetric, we have $q(z_1(f'), \ldots, z_R(f')) = q(z_1(f), \ldots, z_R(f))$, and so, since $\mathbf{F}$ is symmetric in $S_N \times S_R$, q approximates $\mathbf{F}$ on any input.

7.2 The Quantum Adversary Method

One sense in which the polynomial method is not very quantum is that it does not use the fact that intermediate operations must be unitary: the argument in Lemma 7.1.1 would work for any linear operators U_t. In this section we outline a class of lower bound methods that take more advantage of the structure of a quantum query algorithm.

In the classical adversary method, deterministic lower bounds are proven by showing that a certain number of queries are required to distinguish between 0 and 1 inputs, where some adversary can adaptively choose the input as it is queried. The adversary is assumed to choose the query responses in such a way as to avoid committing to a particular output for as long as possible. For example, in the case of $\mathbf{OR}_N$, the adversary will return 0 to every query made, up until the last element is queried. In this way, it is not known until *all* elements have been queried whether or not the input has a marked element. In other words, the adversary avoids providing a certificate for as long as possible.

In the quantum case, since queries are made in superposition, this method is not directly applicable. However, in 2000, Ambainis [6] published the first quantum analogue to the classical adversary method. This method works by choosing a particular set of inputs that are difficult to distinguish and considering running an algorithm on a superposition of these inputs. This is similar to the classical adversary method, in that the adversary attempts to remain consistent with some 0 instance as well as some 1 instance for as many queries as possible. The argument is that any algorithm that computes $\mathbf{F}$ must, in particular, be able to distinguish between any two inputs f, g with $\mathbf{F}(f) \neq \mathbf{F}(g)$. In order to distinguish between these inputs, the state of the algorithm running on the superposition of inputs must have a sufficient amount of entanglement between the input space $\mathcal{H}_I$ and

the computation space $\mathcal{H}_C$ by the time it terminates. By upper bounding the amount by which entanglement can increase with each query, a lower bound on the required number of queries can be computed.

In order for the computation to distinguish between two inputs, we need the computation system to have sufficient information about the input system, that is, we need the *mutual information* between the two systems to be sufficiently high. We can define the mutual information between two quantum systems as:

$$I(\rho_{IC}) := S(\rho_I) + S(\rho_C) - S(\rho_{IC})$$

Since ρ_{IC} is a pure state, it has $S(\rho_{IC}) = 0$, so we want the systems ρ_I and ρ_C to have high entropy, or in other words, to be mixed. The more entangled two systems, the more mixed the resulting state when one system is traced out.

Consider a quantum query circuit as in Section 2.2. The non-query operations $U_0, \ldots, U_T$ act locally on $\mathcal{H}_C$, and so cannot increase the entanglement between the two systems. Only the query operator $\mathcal{O}$ can increase entanglement between $\mathcal{H}_I$ and $\mathcal{H}_C$. Before any query has been made, the initial state on $\mathcal{H}_C$ is independent of the input, and so the two systems are completely unentangled.

Consider the initial state of $\mathcal{H}_I \otimes \mathcal{H}_C$ when we run the circuit on a superposition of inputs from some set $\mathcal{S}$:

$$|\Psi^0\rangle := \sum_{f \in S} \alpha_f |f\rangle \otimes U_0 |0 \ldots 0\rangle$$

In general, we will denote the state on $\mathcal{H}_I \otimes \mathcal{H}_C$ after t steps as $|\Psi^t\rangle$, so the final state looks like:

$$|\Psi^T\rangle := \sum_{f \in S} \alpha_f |f\rangle |\psi_f^T\rangle$$

If this state has no entanglement between the two systems, then $|\psi_f^T\rangle$ does not depend on f at all. In this case, the output cannot be correct, because it is independent of the input, and there are both 0 and 1 inputs in $\mathcal{S}$. On the other hand, if every element of f has been queried and stored, then $|\psi_f^T\rangle$ actually contains f, that is, $|\psi_f^T\rangle = |f\rangle|\phi\rangle$ for some $|\phi\rangle$, so the system is actually maximally entangled. Intuitively, we need $|\psi_f^T\rangle$ to contain *enough* information about f that it can distinguish it from the other inputs, so there must be a high degree of entanglement between the two systems.

Let $\rho^{(T)} = |\Psi^T\rangle\langle\Psi^T|$ and $\rho_I^{(T)} = \mathrm{Tr}_C(\rho)$, the final system on the input space. We have:

$$\rho_I^{(T)} = \sum_{f\in\mathcal{S}}\sum_{g\in\mathcal{S}} \alpha_f\alpha_g^* |f\rangle\langle g|\langle\psi_g^T|\psi_f^T\rangle$$

so:

$$\rho_I^{(T)}[f,g] = \langle\psi_g^T|\psi_f^T\rangle\alpha_f\alpha_g^*$$

Suppose $\mathbf{F}(f) = 0$ and $\mathbf{F}(g) = 1$ and suppose A computes $\mathbf{F}$ with bounded error ϵ. We can write the final states of A on inputs f and g as:

$$|\psi_f^T\rangle = \sum_z c_z^{(0)}|z\rangle|0\rangle + \sum_z c_z^{(1)}|z\rangle|1\rangle$$

$$|\psi_g^T\rangle = \sum_z d_z^{(0)}|z\rangle|0\rangle + \sum_z d_z^{(1)}|z\rangle|1\rangle$$

where $|z\rangle$ ranges over all possible states of the non-answer part of the computation register. Then if $\epsilon_1 = \sum_z \left|c_z^{(1)}\right|^2$ and $\epsilon_2 = \sum_z \left|d_z^{(0)}\right|^2$, we have $\epsilon_1 \leq \epsilon$ and $\epsilon_2 \leq \epsilon$, since ϵ_1 is the probability of measuring 1 on input f and ϵ_2 is the probability of measuring 0 on input g.

We then have:

$$\begin{aligned}
\left|\langle\psi_g^T|\psi_f^T\rangle\right| &= \left|\sum_z c_z^{(0)*}d_z^{(0)} + \sum_z c_z^{(1)*}d_z^{(1)}\right| \\
&\leq \sum_z \left|c_z^{(0)}\right|\left|d_z^{(0)}\right| + \sum_z \left|c_z^{(1)}\right|\left|d_z^{(1)}\right| \\
&\leq \sqrt{\sum_z \left|c_z^{(0)}\right|}\sqrt{\sum_z \left|d_z^{(0)}\right|} + \sqrt{\sum_z \left|c_z^{(1)}\right|}\sqrt{\sum_z \left|d_z^{(1)}\right|} \\
&= \sqrt{\epsilon_1(1-\epsilon_2)} + \sqrt{\epsilon_2(1-\epsilon_1)}
\end{aligned}$$

This expression is maximized by taking ϵ_1 and ϵ_2 as close as possible to $\frac{1}{2}$, so:

$$\left|\langle\psi_g^T|\psi_f^T\rangle\right| \leq 2\sqrt{\epsilon(1-\epsilon)}$$

This gives the following:

Theorem 7.2.1. *[6] Let A be a quantum algorithm for computing $\mathbf{F}$ as described above with bounded error ϵ. Then for all f, g such that $\mathbf{F}(f) \neq \mathbf{F}(g)$, we have:*

$$\left|\rho_I^{(T)}[f,g]\right| \leq 2\sqrt{\epsilon(1-\epsilon)}\left|\alpha_f^{(T)}\right|\left|\alpha_g^{(T)}\right|$$

The only thing that remains is to show that a single query cannot increase the entanglement by very much. Exactly how much should, of course, depend on the problem in question. For example, consider the trivial problem $\mathbf{F} : \{0,1\}^N \to \{0,1\}$ given by $\mathbf{F}(f) = f[1]$, that is, $\mathbf{F}(f)$ is just the first bit of f. Then obviously a single query is sufficient to compute $\mathbf{F}$. Let $f, g : [N] \to \{0,1\}$ be such that $f[1] = 0$ and $g[1] = 1$. Let $\mathcal{S} = \{f, g\}$. Then after querying the first bit, we have $\alpha_f|f\rangle|0\rangle + \alpha_g|g\rangle|1\rangle$. If $\alpha_f = \alpha_g = \frac{1}{\sqrt{2}}$ then the two systems are maximally entangled after a single query. In general, however, this will not be the case.

Note that the above also depends on $\mathcal{S}$ and the amplitudes α_f of $f \in \mathcal{S}$. Choosing these optimally will result in the best lower bound obtainable by this method. The first result on how the choice of $\mathcal{S}$ and the superposition of inputs leads to a particular lower bound is from Ambainis' original paper on the quantum adversary method [6]. If we choose some difficult to distinguish set $\mathcal{S}$, then, in particular, for any set of pairs $(f, g) \in \mathcal{S} \times \mathcal{S}$ such that $\mathbf{F}(f) \neq \mathbf{F}(g)$ we want the algorithm to be able to distinguish each pair. We represent this as a symmetric $|\mathcal{S}| \times |\mathcal{S}|$ $\{0,1\}$-matrix Γ with $\Gamma[f, g] = 0$ when $\mathbf{F}(f) = \mathbf{F}(g)$. When $\Gamma[f, g] = 1$, we consider f, g as a pair we want the algorithm to distinguish. We can also picture Γ as a bipartite graph with vertex classes $\mathcal{S}_0 := \{f \in \mathcal{S} : \mathbf{F}(f) = 0\}$ and $\mathcal{S}_1 := \{g \in \mathcal{S} : \mathbf{F}(g) = 1\}$.

We are interested in how a single query may help distinguish between inputs f and g with $\Gamma[f, g] = 1$, and so we define, for each $x \in [N]$, Γ_x by: $\Gamma_x[f, g] = 1$ if and only if $\Gamma[f, g] = 1$ and $f[x] \neq g[x]$. Of course, we can also consider Γ_x as a subgraph of Γ.

Define $r(A)$ as the set of rows of a matrix A and $c(A)$ as the set of columns of A. For matrices A and B of the same dimensions, let $\langle A, B\rangle = \mathrm{Tr}(A^\dagger B) = \sum_{i,j} A[i, j]^* B[i, j]$ denote the Hilbert-Schmidt inner product.

Theorem 7.2.2. *[6] Let $\mathcal{S}$ be a set of valid inputs to some problem $\mathbf{F}$. Let Γ be a symmetric $|\mathcal{S}| \times |\mathcal{S}|$ $\{0,1\}$-matrix such that if $\mathbf{F}(f) = \mathbf{F}(g)$ for $f, g \in \mathcal{S}$, then $\Gamma[f, g] = 0$. Then any quantum circuit computing $\mathbf{F}$ with bounded error requires at least*

$$\Omega\left(\frac{\min\{\|r\|_2 : r \in r(\Gamma)\}\min\{\|c\|_2 : c \in c(\Gamma)\}}{\max_x \max\{\|r\|_2 : r \in r(\Gamma_x)\}\max_x \max\{\|c\|_2 : c \in c(\Gamma_x)\}}\right)$$

queries. Considering Γ as the adjacency matrix of a graph, this quantity can be stated as:

$$\sqrt{\frac{\min\{\deg_\Gamma(f) : f \in \mathcal{S}_0\}\min\{\deg_\Gamma(g) : g \in \mathcal{S}_1\}}{\max_x \max\{\deg_{\Gamma_x}(f) : f \in \mathcal{S}_0\}\max_x \max\{\deg_{\Gamma_x}(g) : g \in \mathcal{S}_1\}}}$$

Proof. Let A be any quantum circuit that computes $\mathbf{F}$ with bounded error in T queries. We will consider running A on the following superposition of inputs:

$$\frac{1}{\sqrt{2}} \sum_{f \in \mathcal{S}_0} \frac{1}{\sqrt{|\mathcal{S}_0|}} |f\rangle + \frac{1}{\sqrt{2}} \sum_{g \in \mathcal{S}_1} \frac{1}{\sqrt{|\mathcal{S}_1|}} |g\rangle$$

We will define a progress function to measure the entanglement between the input and computation space at each step t:

$$W(t) := \sum_{\{f,g\} \in E(\Gamma)} \left| \rho_I^{(t)}[f,g] \right| = \langle \rho_I^{(t)}, \Gamma \rangle$$

Notice that we have:

$$W(0) = \sum_{\{f,g\} \in E(\Gamma)} \frac{1}{2\sqrt{|\mathcal{S}_0|\,|\mathcal{S}_1|}} = \frac{|E(\Gamma)|}{2\sqrt{|\mathcal{S}_0|\,|\mathcal{S}_1|}}$$

And by Theorem 7.2.1, we need:

$$W(T) \leq \sum_{\{f,g\} \in E(\Gamma)} \frac{1}{\sqrt{|\mathcal{S}_0|\,|\mathcal{S}_1|}} \sqrt{\epsilon(1-\epsilon)} = |E(\Gamma)| \sqrt{\frac{\epsilon(1-\epsilon)}{|\mathcal{S}_0|\,|\mathcal{S}_1|}}$$

So we have:

$$W(0) - W(T) \in \Omega\left(\frac{|E(\Gamma)|}{\sqrt{|\mathcal{S}_0|\,|\mathcal{S}_1|}} \right)$$

Thus we need only bound the change in weight at each step as:

$$W(t) - W(t+1) \in O\left(\frac{|E(\Gamma)|}{\sqrt{|\mathcal{S}_0|\,|\mathcal{S}_1|}} \sqrt{\frac{\max_x \max\{\deg_{\Gamma_x}(f) : f \in \mathcal{S}_0\} \max_x \max\{\deg_{\Gamma_x}(g) : g \in \mathcal{S}_1\}}{\min\{\deg_\Gamma(f) : f \in \mathcal{S}_0\} \min\{\deg_\Gamma(g) : g \in \mathcal{S}_1\}}} \right)$$

Since $|E(\Gamma)| \geq \frac{1}{2} |\mathcal{S}_0| \min\{\deg_\Gamma(f) : f \in \mathcal{S}_0\} |\mathcal{S}_1| \min\{\deg_\Gamma(g) : g \in \mathcal{S}_1\}$, it suffices to prove:

$$W(t) - W(t+1) \in O\left(\sqrt{\max_x \max\{\deg_{\Gamma_x}(f) : f \in \mathcal{S}_0\} \max_x \max\{\deg_{\Gamma_x}(g) : g \in \mathcal{S}_1\}} \right)$$

We have:

$$W(t) - W(t+1) = \langle \rho_I^{(t)} - \rho_I^{(t+1)}, \Gamma \rangle = \sum_{\{f,g\} \in E(\Gamma)} (\rho_I^{(t)}[f,g] - \rho_I^{(t+1)})$$

$$\leq \sum_{\{f,g\}\in E(\Gamma)} \frac{1}{2\sqrt{|\mathcal{S}_0|\,|\mathcal{S}_1|}}[\langle\psi_f^t|\psi_g^t\rangle - \langle\psi_f^{t+1}|\psi_g^{t+1}\rangle]$$

Let $|\psi_f^t\rangle = \sum_{x,w} a_{x,w}^f |x,w\rangle$ where the first register is the query register and the second register is the rest of the computation space. Then after applying the query operator we get: $|\widehat{\psi}_f^t\rangle = \sum_{w,x:f(x)=0} a_{x,w}^f |x,w\rangle - \sum_{w,x:f(x)=1} a_{x,w}^f |x,w\rangle$.

Then:

$$\langle\psi_f^t|\psi_g^t\rangle - \langle\psi_f^{t+1}|\psi_g^{t+1}\rangle = \langle\psi_f^t|\psi_g^t\rangle - \langle\widehat{\psi}_f^t|U_{t+1}^\dagger U_{t+1}|\widehat{\psi}_g^t\rangle = \langle\psi_f^t|\psi_g^t\rangle - \langle\widehat{\psi}_f^t|\widehat{\psi}_g^t\rangle$$

$$= \sum_{x,w} a_{x,w}^{f*} a_{x,w}^g - \sum_{w,x:f(x)=g(x)} a_{x,w}^{f*} a_{x,w}^g + \sum_{w,x:f(x)\neq g(x)} a_{x,w}^{f*} a_{x,w}^g$$

$$= 2 \sum_{x,w:f(x)\neq g(x)} a_{x,w}^{f*} a_{x,w}^g$$

Plugging in, we get:

$$|W(t) - W(t+1)| \leq \frac{1}{\sqrt{|\mathcal{S}_0|\,|\mathcal{S}_1|}} \sum_{x,w:f(x)\neq g(x)} \left|a_{x,w}^f\right| \left|a_{x,w}^g\right|$$

$$\leq \frac{1}{\sqrt{|\mathcal{S}_0|\,|\mathcal{S}_1|}} \sum_{x,w} \sum_{\{f,g\}\in E(\Gamma_x)} \left|a_{x,w}^f\right| \left|a_{x,w}^g\right|$$

$$\leq \frac{1}{\sqrt{|\mathcal{S}_0|\,|\mathcal{S}_1|}} \sum_{x,w} \sqrt{\sum_{f\in\mathcal{S}_0} \deg_{\Gamma_x}(f) \left|a_{x,w}^f\right|^2 \sum_{g\in\mathcal{S}_1} \deg_{\Gamma_x}(g) \left|a_{x,w}^g\right|^2}$$

by Cauchy-Schwartz. Let $d_x^i := \max\{\deg_{\Gamma_x}(f) : f \in \mathcal{S}_i\}$ for $i = 0, 1$, and $d^i := \max_x d_x^i$. Then:

$$|W(t) - W(t+1)| \leq \frac{1}{\sqrt{|\mathcal{S}_0|\,|\mathcal{S}_1|}} \sum_{x,w} \sqrt{\left(d_x^0 \sum_{f\in\mathcal{S}_0} \left|a_{x,w}^f\right|^2\right)\left(d_x^1 \sum_{g\in\mathcal{S}_1} \left|a_{x,w}^g\right|^2\right)}$$

$$\leq \sqrt{d^0 d^1} \sum_{x,w} \sqrt{\frac{1}{|\mathcal{S}_0|} \sum_{f\in\mathcal{S}_0} \left|a_{x,w}^f\right|^2 \frac{1}{|\mathcal{S}_1|} \sum_{g\in\mathcal{S}_1} \left|a_{x,w}^g\right|^2}$$

$$\leq \sqrt{d^0 d^1} \sum_{x,w} \frac{1}{2}\left[\frac{1}{|\mathcal{S}_0|} \sum_{f\in\mathcal{S}_0} \left|a_{x,w}^f\right|^2 + \frac{1}{|\mathcal{S}_1|} \sum_{g\in\mathcal{S}_1} \left|a_{x,w}^g\right|^2\right]$$

by the arithmetic mean geometric mean inequality. Moving the sum over x, w into the other two sums gives:

$$|W(t) - W(t+1)| \leq \frac{\sqrt{d^0 d^1}}{2} \left[\frac{1}{|\mathcal{S}_0|} \sum_{f \in \mathcal{S}_0} 1 + \frac{1}{|\mathcal{S}_1|} \sum_{g \in \mathcal{S}_1} 1 \right] = \frac{\sqrt{d^0 d^1}}{2}$$

By definition of d^0 and d^1, this gives exactly:

$$|W(t) - W(t+1)| \in O\left(\sqrt{\max_x \max\{\deg_{\Gamma_x}(f) : f \in \mathcal{S}_0\} \max_x \max\{\deg_{\Gamma_x}(g) : g \in \mathcal{S}_1\}} \right)$$

□

Though clearly not the most general construction possible, it is fairly simple to relate this to the block sensitivity (and thus certificate complexity). Let f be an input that is sensitive to a maximum number of blocks $B_1, \ldots, B_t$, with $t = \mathsf{bs}(F)$. Then we can set (without loss of generality) $\mathcal{S} = \{f, f^{B_1}, \ldots, f^{B_t}\}$, so that $\mathcal{S}_0 = \{f\}$ and $\mathcal{S}_1 = \{f^{B_1}, \ldots, f^{B_t}\}$ to prove the following:

Theorem 7.2.3. *[6] Let* $\mathbf{F}$ *be a Boolean function and* A *a quantum algorithm that computes* $\mathbf{F}$ *with bounded error using* T *queries. Then* $T \in \Omega(\sqrt{\mathsf{bs}(\mathbf{F})})$.

Proof. In this case, Γ is simply the star graph on $t+1$ vertices, and Γ_x for $x \in B_i$ (it can be in at most one since the blocks are disjoint) consists of a single edge $\{f, f^{B_i}\}$. We thus have:

$$\min\{\deg_\Gamma(g) : g \in \mathcal{S}_0\} = t = \mathsf{bs}(\mathbf{F})$$

$$\min\{\deg_\Gamma(g) : g \in \mathcal{S}_1\} = 1$$

$$\max_x \max\{\deg_{\Gamma_x}(g) : g \in \mathcal{S}_0\} = \max_x \max\{\deg_{\Gamma_x}(g) : g \in \mathcal{S}_1\} = 1$$

So $Q(\mathbf{F}) \in \Omega(\sqrt{\mathsf{bs}(\mathbf{F})})$. □

This is not necessarily the best construction for Γ, and in the original adversary method paper, Ambainis gives examples of several unweighted adversary lower bounds that are higher than $\sqrt{\mathsf{bs}(\mathbf{F})}$.

The adversary method can be generalized and strengthened by assigning weights to each pair in $\mathcal{S}$. There are a number of equivalent [52] ways of formulating this generalization, and depending on the context, it may be helpful to use one or the other. Equivalent formulations

include: spectral adversary [11], weighted adversary [7], strong weighted adversary [56], and Kolmogorov adversary [36]. We call this the *weighted adversary method.*

We assign weights to each pair of inputs by defining an $|\mathcal{S}| \times |\mathcal{S}|$ non-negative real symmetric matrix Γ with diagonal entries all 0. More generally, if $\mathbf{F}(f) = \mathbf{F}(g)$, then $\Gamma[f, g] = 0$. The weight of a pair of inputs is meant to measure how difficult to distinguish those two inputs are. The weighted version of Γ is certainly more general than the $\{0, 1\}$ version. The better our choice of Γ, the better the lower bound that this method yields.

If we allow negative weights, we get the more powerful *negative weights* adversary method, which we discuss in Section 7.2.1. For now we assume all weights are non-negative.

We are still concerned with the difficulty of distinguishing some set of inputs, but we now consider running the algorithm on a superposition of inputs defined in the following way.

Let δ be a normalized eigenvector of Γ corresponding to its largest eigenvalue. Then we define the initial state as $|\Psi_0\rangle = \sum_{f \in \mathcal{S}} \delta[f]|f\rangle \otimes |0 \dots 0\rangle$. As in the unweighted case, we will define some notion of progress and show a lower bound on the progress required to correctly distinguish between different inputs, as well as an upper bound on the increase in progress given by a single query.

Define Γ_x as the $|\mathcal{S}| \times |\mathcal{S}|$ $\{0, 1\}$-matrix such that $\Gamma_x[f, g] = \Gamma[f, g]$ if and only if $f[x] \neq g[x]$ and $\Gamma_x[f, g] = 0$ otherwise.

Definition 7.2.4. *Define the* spectral norm *of a matrix A as $\|A\|_2 = \sigma_{max}(A)$, where σ_{max} is the largest singular value. If A is Hermitean, $\|A\|_2$ is equal to the largest eigenvalue of A, up to sign. We can also write $\|A\|_2 = \max_{v \neq 0} \frac{\|Av\|_2}{\|v\|_2}$, the induced operator norm of the Euclidean norm.*

Definition 7.2.5. *Define the* weighted adversary bound *of a problem $\mathbf{F}$ as:*

$$ADV(\mathbf{F}) = \max_{\Gamma} \frac{\|\Gamma\|_2}{\max_x \|\Gamma_x\|_2}$$

where Γ is taken over all non-negative real valued symmetric matrices with $\Gamma[f, g] = 0$ if $\mathbf{F}(f) = \mathbf{F}(g)$.

We now show that $ADV(\mathbf{F})$ is truly a lower bound on the quantum query complexity of $\mathbf{F}$.

Theorem 7.2.6. *[28, 11] Let $\mathbf{F} : [R]^N \to \{0, 1\}$ be a Boolean function. Then $ADV(\mathbf{F}) \leq Q(\mathbf{F})$.*

Proof. Let Γ be a weighted adversary matrix with δ as defined above. Suppose we begin in the state $|\Psi_0\rangle = \sum_f \delta[f]|f\rangle \otimes U_0|0\dots0\rangle$. Denote the state right before the $t+1^{\text{th}}$ oracle call as $|\Psi_t\rangle = \sum_f \delta[f]|f\rangle|\psi_f^t\rangle$. Let $\rho_I^{(t)}$ be the state of the input space after t steps, that is, $\rho_I^{(t)} = Tr_C(|\Psi_t\rangle\langle\Psi_t|)$. Note that $\rho_I^{(t)}[f,g] = \delta[f]^*\delta[g]\langle\psi_f|\psi_g\rangle$.

Define $W(t) := \langle\rho_I^{(t)}, \Gamma\rangle$, the progress after t steps. We will prove the following three things:

1. $W(0) = \|\Gamma\|_2$
2. $W(t) - W(t+1) \leq 2\max_x \|\Gamma_x\|_2$
3. $W(T) \leq 2\sqrt{\epsilon(1-\epsilon)}\,\|\Gamma\|_2$

The first fact follows from the definition of W:

$$\langle\psi_f^0|\psi_g^0\rangle = 1$$

for all f and g, since no query has been made yet. Thus $\rho_I^{(0)}[f,g] = \delta[f]^*\delta[g]$ so $\rho_I^{(0)} = \delta\delta^\dagger$, so:

$$W(0) = \langle\delta\delta^\dagger, \Gamma\rangle = \mathrm{Tr}((\delta\delta^\dagger)^\dagger\Gamma) = \mathrm{Tr}(\delta^\dagger\Gamma\delta) = \mathrm{Tr}(\delta^\dagger \|\Gamma\|_2 \delta) = \|\Gamma\|_2 \mathrm{Tr}(\delta^\dagger\delta) = \|\Gamma\|_2$$

The third fact follows from Theorem 7.2.1: $\langle\psi_f^T|\psi_g^T\rangle \leq 2\sqrt{\epsilon(1-\epsilon)}$ for every f and g such that $\mathbf{F}(f) \neq \mathbf{F}(g)$:

$$\begin{aligned} W(T) = \langle\rho_I^{(T)}, \Gamma\rangle &= \sum_{f,g} \Gamma[f,g]\delta[f]^*\delta[g]\langle\psi_f^T|\psi_g^T\rangle \leq 2\sqrt{\epsilon(1-\epsilon)}\sum_{f,g}\Gamma[f,g]\delta[f]^*\delta[g] \\ &= 2\sqrt{\epsilon(1-\epsilon)}\langle\delta\delta^\dagger,\Gamma\rangle = 2\sqrt{\epsilon(1-\epsilon)}\,\|\Gamma\|_2 \end{aligned}$$

It remains only to bound the change in weight given by a single query. We have (using the notation of Theorem 7.2.2):

$$|W(t) - W(t+1)| \leq \sum_{f,g}\Gamma[f,g]\delta[f]\delta[g]\left|\langle\psi_f^t|\psi_g^t\rangle - \langle\psi_f^{t+1}|\psi_g^{t+1}\rangle\right|$$

$$\leq \sum_{f,g}\Gamma[f,g]\delta[f]\delta[g]2\sum_{x,w:f(x)\neq g(x)}\left|a_{x,w}^f\right|\left|a_{x,w}^g\right|$$

$$\leq 2 \sum_{x,w} \sum_{f,g} \Gamma_x[f,g]\delta[f]\delta[g] \left|a^f_{x,w}\right| \left|a^g_{x,w}\right|$$

Define $v_{x,w} \in \mathbb{R}^{|\mathcal{S}|}$ by $v_{x,w}[f] = \delta[f] \left|a^f_{x,w}\right|$. Note that $\|v_{x,w}\|_2^2 = \sum_f \delta[f]^2 \left|a^f_{x,w}\right|^2 \leq \sum_f \left|a^f_{x,w}\right|^2$. This gives:

$$|W(t) - W(t+1)| \leq 2 \sum_{x,w} \sum_f v_{x,w}[f] \sum_g \Gamma_x[f,g] v_{x,w}[g]$$

$$= 2 \sum_{x,w} \sum_f v_{x,w}[f] (\Gamma_x v_{x,w})[f]$$

$$= 2 \sum_{x,w} v^\dagger_{x,w} \Gamma_x v_{x,w}$$

$$\leq \|\Gamma_x\|_2 \|v_{x,w}\|_2^2$$

$$\leq 2 \max_x \|\Gamma_x\|_2 \sum_{x,w,f} \left|a^f_{x,w}\right|$$

$$= 2 \max_x \|\Gamma_x\|_2$$

The result follows. □

The weighted adversary method faces two strict limitations, known as the *certificate complexity barrier* and the *property testing barrier*. These barriers put strict limits on the lower bounds obtainable by the weighted adversary method. They are stated below:

Theorem 7.2.7 (Certificate Complexity Barrier)**.** *[52] If* $\mathbf{F} : [R]^N \to \{0,1\}$ *is a total Boolean function, then* $ADV(\mathbf{F}) \leq \sqrt{\mathsf{C}^{(0)}(\mathbf{F})\mathsf{C}^{(1)}(\mathbf{F})}$*, and if* $\mathbf{F}$ *is a partial Boolean function,* $ADV(\mathbf{F}) \leq 2\sqrt{\min\{\mathsf{C}^{(0)}(\mathbf{F}), \mathsf{C}^{(1)}(\mathbf{F})\}N}$.

The problems of interest to us have constant sized 1-certificates, and thus, the weighted adversary method can yield a lower bound no higher than $\sqrt{N}$. It is known that the actual lower bound on $\mathbf{ED}_N$ is greater than this ($\Omega(N^{2/3})$), and this was proven using the polynomial method (see Section 7.1.2).

Theorem 7.2.8 (Property Testing Barrier)**.** *[27] Let* $\mathbf{F} : [R]^N \to \{0,1\}$ *be a partial Boolean function with the promise that if* $\mathbf{F}(f) \neq \mathbf{F}(g)$ *then* $\Delta(f,g) \geq N\epsilon$ *where* Δ *is the Hamming distance. Then* $ADV(\mathbf{F}) \leq \frac{1}{\epsilon}$.

For $\widetilde{\mathbf{COL}}_N$, all almost strictly many-to-1 functions have Hamming distance at least $N/2 - c \log N$ from 1-to-1 functions, so the weighted adversary method cannot prove a lower bound higher than 2 for this problem. All known lower bounds for collision finding are thus proven using the polynomial method.

The limitations of the weighted adversary method stem partially from the fact that it does not take into account that the information learned from some query may be more or less than that learned from another query. The amount of information currently known by the algorithm is not captured, only the maximum increase per step. This does not take into account the possibility of discarding information, as is done in the space-bounded setting. Interestingly, however, an early application of an adversary-like argument (pre-dating the actual formalization of the adversary method) by Zalka [55] does manage to take this into account. This result is of particular interest to us, because it includes a lower bound for $\mathbf{OR}_N$ in a *parallel-query* model. We discuss this more in Section 7.2.2. First, we discuss one final generalization of the adversary method.

7.2.1 The Negative Weights Adversary Method

This further generalization of the adversary method, which is strictly stronger than the weighted adversary method, is the *negative weights adversary method* [27] obtained by allowing Γ to take negative values. It is subject neither to the certificate complexity nor the property testing barrier. It is simple to generalize to this method, but the proof of its validity is nontrivial. Whereas the weighted adversary method lower bounds the number of queries required to *distinguish* various inputs, the negative weights adversary method also uses the fact that the algorithm must not only distinguish inputs, but actually *compute* the function on any input.

It is surprising that such a simple change in the method could allow for such a strong difference. Intuitively, it is:

> good to give negative weight to entries with large Hamming distance, entries which are easier to distinguish by queries. Consider an entry (f, g) where f and g have large Hamming distance. This entry appears in several Γ_x matrices but only appears in the Γ matrix once. Thus by giving this entry negative weight we can simultaneously decrease $\|\Gamma_x\|_2$ for several x's, while doing relatively little damage to the large Γ matrix [27].

Definition 7.2.9. *Define the* negative weights adversary bound *of a problem* $\mathbf{F}$ *as:*

$$ADV^{\pm}(\mathbf{F}) = \max_{\Gamma} \frac{\|\Gamma\|_2}{\max_x \|\Gamma_x\|_2}$$

where Γ *is taken over all real valued symmetric matrices with* $\Gamma[f,g] = 0$ *if* $\mathbf{F}(f) = \mathbf{F}(g)$.

In the weighted adversary method, we prove that the progress function must have dropped below a certain threshold after T queries using the fact that the algorithm must distinguish different inputs. In the negative weights adversary method, we use the stronger fact that there must exist a measurement that would give the right answer after T queries.

Theorem 7.2.10. *[27] Let* $\mathbf{F} : [R]^N \to \{0,1\}$ *be a Boolean function. Then* $Q(\mathbf{F}) \in \Omega(ADV^{\pm}(\mathbf{F}))$.

To prove this theorem, we will use the following lemma:

Lemma 7.2.11. *Let A, B, and C be square matrices of the same dimension. Then:*

$$\langle AB^\dagger, C\rangle \leq \sqrt{\langle A,A\rangle\langle B,B\rangle}\,\|C\|_2$$

Proof. We have:

$$\langle AB^\dagger, C\rangle \leq \frac{\left|\langle AB^\dagger, C\rangle\right|}{\|C\|_2}\|C\|_2 \leq \max M \frac{\left|\langle AB^\dagger, M\rangle\right|}{\|M\|_2}\|C\|_2$$

We have that $\|\cdot\|_2$ is the Schatten ∞-norm: $\|M\|_2 = \|M\|_{(\infty)} = \lim_{p\to\infty}\left[\mathrm{Tr}((M^\dagger M)^{p/2})\right]^{1/p}$. For Schatten norms $\|\cdot\|_{(p)}$ and $\|\cdot\|_{(q)}$ such that $\frac{1}{p}+\frac{1}{q}=1$ we have:

$$\|X\|_{(p)} = \max M \frac{|\langle X, M\rangle|}{\|M\|_{(q)}}$$

so we have:

$$\langle AB^\dagger, C\rangle \leq \left\|AB^\dagger\right\|_{(1)}\|C\|_2 = \mathrm{Tr}(\sqrt{(AB^\dagger)^\dagger AB^\dagger})\,\|C\|_2$$

$$= \mathrm{Tr}(A^\dagger B)\,\|C\|_2 \leq \|A\|_{(2)}\,\|B\|_{(2)}\,\|C\|_2 \leq \sqrt{\langle A,A\rangle\langle B,B\rangle}\,\|C\|_2$$

where the second last inequality is by Hölder's inequality. □

We're now ready to prove the theorem:

Proof. Define $W(t)$ as in Theorem 7.2.6. Just as in the proof of Theorem 7.2.6, we will prove the following:

1. $W(0) = \|\Gamma\|_2$
2. $W(t) - W(t+1) \leq 2 \max_x \|\Gamma_x\|_2$
3. $W(T) \leq 2\sqrt{\epsilon(1-\epsilon)}\,\|\Gamma\|_2$

The proof of the first item is identical to that of Theorem 7.2.6 and the second is similar enough that we omit it. However, if we try to apply the same method for the third item, we will run into a problem, since Γ may now have negative entries, so we cannot substitute the upper bound on $\langle\psi_f^T|\psi_g^T\rangle$ into the formula $W(T) = \sum_{f,g}\Gamma[f,g]\delta[f]\delta[g]\langle\psi_f^T|\psi_g^T\rangle$ to get an upper bound. Instead, they make use of the fact that there exists a complete set of orthogonal projectors $\{M_0, M_1\}$ (or more generally, for non-Boolean functions, $\{M_y\}$, however we will assume $\mathbf{F}$ is Boolean) such that for all valid inputs f, $\left\|M_{\mathbf{F}(f)}|\psi_f^T\rangle\right\|_2^2 \geq 1-\epsilon$.

Define $D \in \{0,1\}^{|\mathcal{S}|\times|\mathcal{S}|}$ such that $D[f,g] = 1$ if and only if $\mathbf{F}(f) \neq \mathbf{F}(g)$. Then we have $\Gamma = \Gamma \circ D$, where $\circ$ denotes the Hadamard product. If we define Y_0 and Y_1 such that $\rho_I^{(T)} \circ D = Y_0 Y_1^\dagger$, we can write:

$$\langle\rho_I^{(T)}, \Gamma\rangle = \langle D \circ \rho_I^{(T)}, \Gamma\rangle \leq \sqrt{\langle Y_0, Y_0\rangle\langle Y_1, Y_1\rangle}\,\|\Gamma\|_2$$

by the previous Lemma.

Define X_0 and X_1 in $\mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ as follows:

$$X_b[f,:] := M_{\mathbf{F}(f)+b}|\psi_f^T\rangle$$

So X_0 is the projection onto correct outcomes and X_1 is the projection onto incorrect outcomes. We then have:

$$(X_0X_1^\dagger + X_1X_0^\dagger)[f,g] = \delta_f\delta_g^*[\langle\psi_g^T|M_{\overline{\mathbf{F}(g)}}M_{\mathbf{F}(f)}|\psi_f^T\rangle + \langle\psi_g^T|M_{\mathbf{F}(g)}M_{\mathbf{F}(\bar{f})}|\psi_f^T\rangle]$$

If $f[x] \neq g[x]$:

$$\begin{aligned}
&= \delta_f\delta_g^*[\langle\psi_g^T|M_0^2|\psi_f^T\rangle + \langle\psi_g^T|M_1^2|\psi_f^T\rangle] \\
&= \delta_f\delta_g^*\langle\psi_g^T|[M_0+M_1]|\psi_f^T\rangle \\
&= \delta_f\delta_g^*\langle\psi_g^T|\psi_f^T\rangle
\end{aligned}$$

since M_0, M_1 form a complete measurement. On the other hand, if $f[x] = g[x] = b$:

$$= \delta_f \delta_g^* [\langle \psi_g^T | M_{\bar{b}} M_b | \psi_f^T \rangle + \langle \psi_g^T | M_b M_{\bar{b}} | \psi_f^T \rangle] = 0$$

Since M_0 and M_1 are orthogonal projectors. Thus, $D \circ \rho_I^{(T)} = X_0 X_1^\dagger + X_1 X_0^\dagger$. This gives:

$$\langle D \circ \rho_I^{(T)}, \Gamma \rangle \leq \langle X_0 X_1^\dagger, \Gamma \rangle + \langle X_1 X_0^\dagger, \Gamma \rangle$$

$$\leq \sqrt{\langle X_0, X_0 \rangle \langle X_1, X_1 \rangle} \, \|\Gamma\|_2 + \sqrt{\langle X_1, X_1 \rangle \langle X_0, X_0 \rangle} \, \|\Gamma\|_2 = 2\sqrt{\langle X_0, X_0 \rangle \langle X_1, X_1 \rangle} \, \|\Gamma\|_2$$

$$= 2\sqrt{\sum_{f \in \mathcal{S}} |\delta_f|^2 \left\| M_{\mathbf{F}(f)} |\psi_f^T\rangle \right\|_2^2 \sum_{f \in \mathcal{S}} |\delta_f|^2 \left\| M_{\overline{\mathbf{F}(f)}} |\psi_f^T\rangle \right\|_2^2} \, \|\Gamma\|_2 \leq 2\sqrt{(1-\epsilon)\epsilon} \, \|\Gamma\|_2$$

The result is immediate. □

The lower bound obtained by $ADV^{\pm}$ is tight for any Boolean function [45]. This isn't so surprising given the previous proof: the fact that there must exist measurements that output the correct answer with sufficient probability is exploited, and this is exactly what it means to compute something with bounded error.

7.2.2 An Adversary-Like Bound on Search with Parallel Queries

Before Ambainis first defined the formal adversary method, Zalka [55] used a similar technique to prove a tight lower bound on $\mathbf{OR}_N$. Of particular interest to us, he also proved a lower bound on the number of *parallel queries* needed to solve $\mathbf{OR}_N$.

Suppose we have a black-box for f such that if we input a vector $(x_1, \ldots, x_k) \in [N]^k$, we get $(f(x_1), \ldots, f(x_k))$. We call one such query a *k-parallel query*. Since we require $\Omega(\sqrt{N})$ quantum queries to solve $\mathbf{OR}_N$, we require at least $\Omega(\frac{\sqrt{N}}{k})$ k-parallel queries. For any problem $\mathbf{F}$, let $Q^{(k)}(\mathbf{F})$ be the number of k-parallel queries required to solve $\mathbf{F}$. Trivially, we have $Q^{(k)}(\mathbf{F}) \in \Omega(\frac{Q(\mathbf{F})}{k})$. Zalka showed an even tighter bound for search: $Q^{(k)}(\mathbf{OR}_N) \in \Omega(\frac{\sqrt{N}}{\sqrt{k}})$.

Zalka's bound implies that we cannot simply divide the $\Theta(\sqrt{N})$ queries required for $\mathbf{OR}_N$ among M processors. In general, if we can prove a parallel query lower bound, this implies a lower bound in the grid model. We have:

Lemma 7.2.12.

$$Q_M^{\boxplus}(\mathbf{F}) \geq Q^{(M)}(\mathbf{F}) + \sqrt{M}$$

For this reason, parallel query results are of great interest. We therefore now outline Zalka' proof. It does not use the technique of defining a Γ matrix, however it is based on the idea that an algorithm that computes $\mathbf{OR}_N$ must be able to distinguish between any inputs with $\mathbf{OR}_N(f) \neq \mathbf{OR}_N(g)$.

Let $\mathcal{S}$ be the set of inputs to $\mathbf{OR}_N$ containing all f with Hamming weight 0 or 1 (where f are viewed as binary strings of length N). That is, $\mathcal{S}$ contains $f = 0$ and g_x defined by $g_x(y) = 1$ if and only if $y = x$, for $x = 1, \ldots, N$. We are concerned with the algorithm's ability to distinguish any (f, g_x) pair of inputs. We thus consider the average *distance* between final states $|\psi_f^T\rangle$ and $|\psi_{g_x}^T\rangle$. This gives an upper bound on the success probability $(1 - \epsilon)$.

Lemma 7.2.13. *[55] Let* A *be a quantum algorithm that computes* $\mathbf{OR}_N$ *with bounded error that has final state* $|\psi_g^T\rangle$ *on any input* g*. Then:*

$$\frac{1}{N}\sum_{x=1}^{N} \left\| |\psi_f^T\rangle - |\psi_{g_x}\rangle \right\|_2^2 \in \Omega(1)$$

The constant in the above term is determined by the allowed error probability, ϵ (which we assume to be constant). The main result is to upper bound the average distance as follows:

Theorem 7.2.14. *[55] Suppose we have an algorithm* A *that computes* $\mathbf{OR}_N$ *with bounded error using* T *k-queries. Then:*

$$\frac{1}{N}\sum_{x=1}^{N} \left\| |\psi_{g_x}^T\rangle - |\psi_f^T\rangle \right\|_2^2 \in O\left(\frac{kT^2}{N}\right)$$

This result, combined with the previous lemma, imply that $T \in \Omega\left(\sqrt{\frac{N}{k}}\right)$.

Proof. Write $U_f^t := U_t\mathcal{O}_f$, $U_x^t = U_t\mathcal{O}_{g_x}$ and $D_x^t = U_f^t - U_x^t$. Then we can write:

$$|\psi_f^T\rangle = (U_x^T + D_x^T)|\psi_f^{T-1}\rangle = |\psi_x^T\rangle + \sum_{t=0}^{T-1} U_x^T \ldots U_x^{T-t+1} D_x^{T-t} |\psi_f^{T-t-1}\rangle$$

Thus we can write:

$$\left\| |\psi_{g_x}^T\rangle - |\psi_f^T\rangle \right\|_2 = \left\| \sum_{t=0}^{T-1} U_x^T \ldots U_x^{T-t+1} D_x^{T-t} |\psi_f^{T-t-1}\rangle \right\|_2$$

$$\leq \sum_{t=0}^{T-1} \left\| U_x^T \dots U_x^{T-t+1} D_x^{T-t} |\psi_f^{T-t-1}\rangle \right\|_2$$

$$= \sum_{t=0}^{T-1} \left\| D_x^{T-t} |\psi_f^{T-t-1}\rangle \right\|_2$$

since the norm is invariant under unitary transformations. The states $U_f^{T-t}|\psi_f^{T-t-1}\rangle$ and $U_x^{T-t}|\psi_f^{T-t-1}\rangle$ only differ in the amplitudes of basis states that look like $|x_1, \dots, x_k, w\rangle$ where $x_i = x$ for at least one i, and w is any state of the workspace. Intuitively, this means that only a small number of amplitudes notice the difference between the two types of queries. This also means that $U_f^{T-t}|\psi_f^{T-t-1}\rangle - U_x^{T-t}|\psi_f^{T-t-1}\rangle = U_f^{T-t}P_x|\psi_f^{T-t-1}\rangle - U_x^{T-t}P_x|\psi_f^{T-t-1}\rangle$, where P_x is the projection onto basis states containing x in one of the query registers. This gives:

$$\left\| |\psi_{g_x}^T\rangle - |\psi_f^T\rangle \right\|_2 \leq \sum_{t=0}^{T-1} \left\| U_f^{T-t}P_x|\psi_f^{T-t-1}\rangle - U_x^{T-t}P_x|\psi_f^{T-t-1}\rangle \right\|_2$$

$$\leq \sum_{t=0}^{T-1} \left[\left\| U_f^{T-t}P_x|\psi_f^{T-t-1}\rangle \right\|_2 + \left\| U_x^{T-t}P_x|\psi_f^{T-t-1}\rangle \right\|_2 \right]$$

$$\leq \sum_{t=0}^{T-1} 2 \left\| P_x|\psi_f^{T-t-1}\rangle \right\|_2$$

Therefore:

$$\left\| |\psi_{g_x}^T\rangle - |\psi_f^T\rangle \right\|_2^2 \leq 4 \left(\sum_{t=0}^{T-1} \left\| P_x|\psi_f^{T-t-1}\rangle \right\|_2 \right)^2$$

$$\leq 4T \sum_{t=0}^{T-1} \left\| P_x|\psi_f^{T-t-1}\rangle \right\|_2^2$$

Now let P_x^j be the projection onto basis states $|x_1, \dots, x_k, w\rangle$ such that $x_j = x$. This gives:

$$\frac{1}{N} \sum_{x=1}^{N} \left\| |\psi_{g_x}^T\rangle - |\psi_f^T\rangle \right\|_2 \leq \frac{4T}{N} \sum_{x=1}^{N} \sum_{t=0}^{T-1} \left\| P_x|\psi_f^{T-t-1}\rangle \right\|_2^2$$

$$\leq \frac{4T}{N} \sum_{x=1}^{N} \sum_{t=0}^{T-1} \sum_{j=1}^{k} \left\| P_x^j|\psi_f^{T-t-1}\rangle \right\|_2^2$$

$$= \frac{4T}{N} \sum_{t=0}^{T-1} \sum_{j=1}^{k} \sum_{x=1}^{N} \left\| P_x^j |\psi_f^{T-t-1}\rangle \right\|_2^2$$

$$= \frac{4T}{N} Tk$$

where the last equality follows from the fact that $\sum_{x=1}^{N} \left\| P_x^j |\psi_f^t\rangle \right\|_2^2 = 1$ for each j. This gives $\frac{1}{N} \sum_{x=1}^{N} \left\| |\psi_{g_x}^T\rangle - |\psi_f^T\rangle \right\|_2^2 \in O\left(\frac{kT^2}{N}\right)$, as desired. □

Chapter 8

Lower Bounds in the Grid Model

To the end of finding lower bounds in the grid and locality-sensitive models, there are several possible directions to take. As previously mentioned, query lower bounds lead trivially to lower bounds in the locality-sensitive and grid models (see Table 2.1) and we derive trivial lower bounds for $\mathbf{ED}_N$ and $\mathbf{COL}_N$ in the grid model in this manner in the following section. However, in order to prove tight lower bounds, we will need something more. One possible direction is that of parallel query lower bounds. The parallel query lower bound for $\mathbf{OR}_N$ ([55] and see Section 7.2.2) leads to a tight quantum grid lower bound (see Table 8.1). Similar future results for $\mathbf{ED}_N$ and $\mathbf{COL}_N$ may give tight grid lower bounds.

Another possible direction of interest is space-time tradeoffs. These are of interest on their own, however they would have implications in the locality-sensitive or grid models, since in the locality-sensitive single processor setting, access to space is expensive, so using less of it is desirable, and in the grid model, each processor is space bounded.

After summarizing the trivial bounds in Section 8.1, we will outline the prospects for future work towards lower bounds in Sections 8.2, 8.3 and 8.4.

8.1 Some Trivial Lower Bounds

The following table summarizes the known bounds in the grid model. For classical upper bounds on $\mathbf{COL}_N$ and $\mathbf{ED}_N$, we include both the rigorous upper bounds, and heuristic upper bounds (denoted $\hat{O}$).

Distributed Grid Complexity Bounds				
	Classical ($R^{\boxplus}$)		Quantum ($Q^{\boxplus}$)	
	Lower Bound	Upper Bound	Lower Bound	Upper Bound
$\mathbf{OR}_N$	$\Theta(N^{1/3})$	$\Theta(N^{1/3})$	$\Theta(N^{1/4})$	$\Theta(N^{1/4})$
$\mathbf{ED}_N$	$\Omega(N^{1/3})$	$O(N^{1/2})$ / $\hat{O}(N^{1/3})$	$\Omega(N^{2/9})$	$O(N^{1/2})$
$\mathbf{COL}_N$	$\Omega(N^{1/6})$	$O(N^{1/4})$ / $\hat{O}(N^{1/6})$	$\Omega(N^{1/9})$	$O(N^{1/4})$

Table 8.1: Grid Lower Bounds

The non-heuristic upper bounds are taken from Table 4.2, and the heuristic upper bounds are based on the rho method, described in Chapter 5. Following is a short explanation of each lower bound.

Quantum Search Lower Bound This lower bound is based on Zalka's result that $Q^{(k)}(\mathbf{OR}_N) \in \Omega\left(\sqrt{\frac{N}{k}}\right)$ ([55] or see Section 7.2.2). From this we get

$$Q_M^{\boxplus}(\mathbf{OR}_N) \in \Omega(Q^{(M)}(\mathbf{OR}_N) + \sqrt{M}) \in \Omega(\sqrt{\frac{N}{M}} + \sqrt{M}) \in \Omega(N^{1/4})$$

The remaining lower bounds are obtained directly from the query lower bounds using: $R_M^{\boxplus}(\mathbf{F}) \geq \frac{R(\mathbf{F})}{M} + \sqrt{M}$ and $Q_M^{\boxplus}(\mathbf{F}) \geq \frac{Q(\mathbf{F})}{M} + \sqrt{M}$.

Quantum Element Distinctness Lower Bound Any quantum algorithm that solves $\mathbf{ED}_N$ must make $Q(\mathbf{ED}_N) \in \Theta(N^{2/3})$ queries. Therefore, a grid algorithm with M processors must have at least one processor that performs $\Omega(\frac{N^{2/3}}{M})$ queries, so it has grid complexity $Q_M^{\boxplus}(\mathbf{ED}_N) \in \Omega(\frac{N^{2/3}}{M} + \sqrt{M}) \in \Omega(N^{2/9})$.

Quantum Collision Finding Lower Bound Similarly, we have $Q(\mathbf{COL}_N) \in \Theta(N^{1/3})$ and so $Q_M^{\boxplus}(\mathbf{COL}_N) \in \Omega(\frac{N^{1/3}}{M} + \sqrt{M}) \in \Omega(N^{1/9})$.

Classical Search Lower Bound We have $R(\mathbf{OR}_N) \in \Omega(N)$, so $R_M^{\boxplus}(\mathbf{OR}_N) \in \Omega(\frac{N}{M} + \sqrt{M}) \in \Omega(N^{1/3})$.

Classical Element Distinctness Lower Bound We have $R(\mathbf{ED}_N) \in \Omega(N)$, so $R_M^{\boxplus}(\mathbf{ED}_N) \in \Omega(\frac{N}{M} + \sqrt{M}) \in \Omega(N^{1/3})$.

Classical Collision Finding Lower Bound We have $R(\mathbf{COL}_N) \in \Omega(\sqrt{N})$, so $R_M^{\boxplus}(\mathbf{COL}_N) \in \Omega(\frac{\sqrt{N}}{M} + \sqrt{M}) \in \Omega(N^{1/6})$.

8.2 Prospects for Applying the Polynomial Method

The polynomial method does not take very many computational restrictions into account in its lower bounds (essentially only linearity of the operations) and it certainly does not account for space bounds. However, it could potentially provide parallel query lower bounds in the following way.

Given a function $f : [N] \to [R]$, we can consider the function $f^{(k)} : [N]^k \to [R]^k$ given by $f^{(k)}(x_1, \ldots, x_k) = (f(x_1), \ldots, f(x_k))$. We can consider the problem k-$\mathbf{COL}_N$, which takes a black-box input $f^{(k)}$ and looks for some $(x_1, \ldots, x_k)$ such that there are some $i \neq j$ such that $x_i \neq x_j$ and $f^k(x_1, \ldots, x_k)[i] = f^{(k)}(x_1, \ldots, x_k)[j]$, that is, $f(x_i) = f(x_j)$. We wish to lower bound the number of queries required to $f^{(k)}$, which is, of course, a lower bound on the k-query complexity of $\mathbf{ED}_N$ or $\mathbf{COL}_N$, depending on which inputs we allow. This, in turn, would imply a grid lower bound. It may be possible to exploit symmetries in this problem to find a polynomial lower bound on its query complexity. Though finding such symmetries is not an easy task, this is still a direction that merits further study.

8.3 Prospects for Applying the Adversary Method

The negative weights adversary method gives tight lower bounds on the minimum number of queries required overall, but does not account for any notion of either space or parallelization. However, Zalka's proof shows that adversary-like arguments can be used to prove parallel query lower bounds, which in turn imply grid lower bounds. It would be interesting to succeed in applying a similar argument to $\mathbf{COL}_N$.

Of perhaps greater interest would be to somehow modify the negative weights adversary method to take space bounds into account. To prove such a lower bound, such as a lower bound for a log-space single quantum processor, we would need to take into account that the algorithm "forgets" query results, since it can only store a small amount of information.

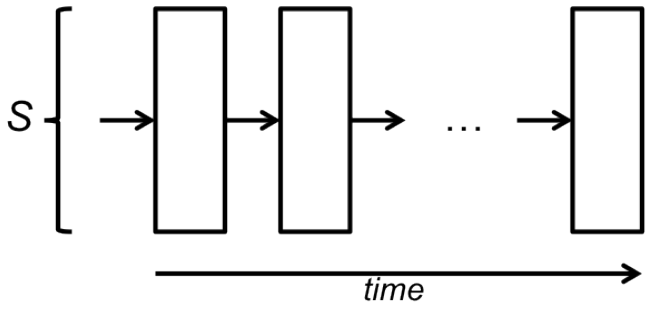

Figure 8.1: A space-bounded algorithm broken into slices. The input to the first slice is the inital state of size S. Each slice sends a quantum state of size S to the next slice.

Thus, at each step we increase the difference between $|\psi_f\rangle$ and $|\psi_g\rangle$ by querying, but this query necessarily requires the uncomputing of some other information (to make room for the query result) which should intuitively make the total increase smaller. Taking this into account would require modifying the step where we upper bound the change in the weight function to account for the space bound. A general adversary method applicable to space-bounded settings could potentially have wide-ranging applications.

8.4 Relation to Communication Complexity

There have been several quantum space-time tradeoffs proven in the past (for example [10, 33]), but all that we know of have been for non-Boolean functions with multi-part outputs. A standard approach goes as follows. Consider running some algorithm on space S for time T, and break the algorithm into "time slices" (as in Figure 8.1), each of which outputs some number k of the output parts. We can consider one slice as performing some computation, and then passing its state to the next slice, as a kind of S-qubit message. The next slice then performs its portion of the computation, starting from the state it received from the previous slice. If we suppose it succeeds in outputting its k outputs with probability $\frac{2}{3}$, then if we consider instead running the slice with a completely mixed starting state, it must output correctly with probability at least $\frac{2}{3}2^{-S}$, because the completely mixed state has overlap at least 2^{-S} with the proper starting state, whatever it is. Using some kind of direct product theorem, which upper bounds the success probability, in terms of time, of computing k instances of some problem (which we have somehow embedded in the outputs) we get a tradeoff between S and k, and thus a tradeoff between S and the time required.

The problems we're interested in do not have this multi-output structure, but the concept of considering slices of an algorithm communicating an S-qubit state to a future slice is an interesting one. It opens up the possibility of applying results from communication complexity. In the grid model, there is also the issue of lateral communication. Thus the setting of communication complexity, specifically space-bounded communication complexity is similar to the grid setting, in which processors wish to compute some joint function using as little communication as possible.

The idea that communication complexity results could be applied to parallel computation is not a new one. In [53], Yao points out that a VLSI chip of size $m \times m$ computing a function with communication complexity C would require time at least $\frac{C}{m}$ to compute the function. Our setting is somewhat different in that each processor has access to the same input. However, although in theory a single processor could compute the function with no communication by doing all necessary queries itself, we suppose that in any optimal algorithm, the work is somehow divided, and two different processors have queried different portions of the input. Thus, communication complexity techniques may be applicable.

Combining the notions of spatial and temporal communication ideas gives rise to an interesting way of considering the information propagation (through both time and space) in the grid. At time step t, a processor may potentially have access to a neighbour's state from time $t - i$ provided that neighbour is at most distance i away. That is, we can consider each processor's backward light cone (Figure 8.2). This gives a picture of a processor performing computation on decreasing space: at step T it has just $\Omega(\log N)$ memory, but at step $T - i$, it had $\Omega(i^2 \log N)$ memory.

Looking at all M processor, we can consider the entire computation as a 3-dimensional object (1 dimension in time and 2 in space) with communication constraints. We can view this object as a directed graph with vertex set $P^t_{i,j}$ where i and j range over $[\sqrt{M}]$ and $t \in [T]$, and edge set $(P^t_{i,j}, P^{t+1}_{k,\ell}) \in E$ if $k \in \{i, i+1, i-1\}$ and $\ell \in \{j, j+1, j-1\}$. Each vertex represents a processor, $P_{i,j}$ at time t. At time t, $P_{i,j}$ may make a single query based on the input states he receives from incoming edges. For the purpose of considering a lower bound, we may allow the processor unlimited auxiliary computation at step t, at the end of which he must send forward communication along all of his outgoing edges.

Using this communication-based paradigm, it may be possible to use one or more known results or techniques to find grid lower bounds for $\mathbf{COL}_N$, and perhaps even other problems.

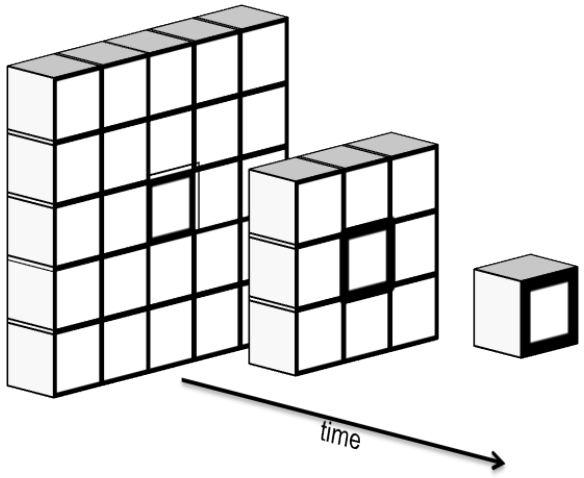

Figure 8.2: The backward light cone of a single processor.

8.5 Final Remarks

We have outlined two new related models of computation: the locality-sensitive model and the grid model, and surveyed algorithms and lower bounds for $\mathbf{COL}_N$, $\mathbf{ED}_N$ and $\mathbf{OR}_N$ in the classical and quantum versions of these models that follow from results in the query model. However, there are many open problems in this model, even for these particular problems. There are gaps in the upper and lower bounds for $\mathbf{ED}_N$ and $\mathbf{COL}_N$ in both the classical (non-heuristic) and quantum grid settings. Of particular interest is the fact that the quantum upper bounds for $\mathbf{ED}_N$ and $\mathbf{COL}_N$ in the grid are no better than their classical counterparts, and are in fact *worse* than the best classical heuristics. Closing these gaps may be a major undertaking, but is certainly of great interest as future work.

References

[1] S. Aaronson. Quantum lower bound for the collision problem. In *Proceedings of the thiry-fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pages 635–642, New York, NY, USA, 2002. ACM. `arXiv:quant-ph/0111102`. 82

[2] S. Aaronson and A. Ambainis. Quantum search of spatial regions. In *Proceedings of the forty-fourth Annual IEEE Symposium on Foundations of Computer Science*, FOCS '03, pages 200–209, Washington, DC, USA, 2003. IEEE Computer Society. 2, 9

[3] S. Aaronson and Y. Shi. Quantum lower bounds for the collision and element distinctness problems. *Journal of the ACM*, 51:595–605, 2004. 82

[4] H. Abelson and P. Andreae. Information transfer and area-time tradeoffs for VLSI multiplication. *Communications of the ACM*, 23:20–23, January 1980. 9

[5] N. Alon, O. Goldreich, and Y. Mansour. Almost k-wise independence versus k-wise independence. *Information Processing Letters*, 88:107–110, 2003. iv, 64, 65

[6] A. Ambainis. Quantum lower bounds by quantum arguments. In *Proceedings of the thirty-second Annual ACM Symposium on Theory of Computing*, STOC '00, pages 636–643, New York, NY, USA, 2000. ACM. `arXiv:quant-ph/0002066v1`. 74, 85, 87, 88, 91

[7] A. Ambainis. Polynomial degree vs. quantum query complexity. In *Proceedings of the forty-fourth Annual IEEE Symposium on Foundations of Computer Science*, pages 230–239, 2003. `arXiv:quant-ph/0305179`. 92

[8] A. Ambainis. Quantum walk algorithm for element distinctness. In *Proceedings of the forty-fifth Annual IEEE Symposium on Foundations of Computer Science*, pages 22–31, 2004. 34, 37

[9] A. Ambainis. Polynomial lower bounds in quantum complexity: collision and element distinctness with small range. *Theory of Computing*, 1:37–46, 2005. `arXiv:quant-ph/0305179v3`. 31, 40, 82

[10] A. Ambainis, R. Špalek, and R. de Wolf. A new quantum lower bound method with applications to direct product theorems and time-space tradeoffs. In *Proceedings of the thrity-eighth Annual ACM Symposium on Theory of Computing*, pages 618–633, 2006. 105

[11] H. Barnum, M. Saks, and M. Szegedy. Quantum query complexity and semi-definite programming. In *Proceedings of the eighteenth IEEE Annual Conference on Computational Complexity*, pages 179–193, 2003. 92

[12] R. Beals, H. Buhrman, R. Cleve, M. Mosca, and R. de Wolf. Quantum lower bounds by polynomials. In *Proceedings of the thirty-ninth Annual IEEE Symposium on Foundations of Computer Science*, pages 352–361, Los Alamitos, California, 1998. IEEE Computer Society Press. `arXiv:quant-ph/9802049v3`. 74, 77, 79, 80

[13] R. Beigel. The polynomial method in circuit complexity. In *Proceedings of the eighth IEEE Structure in Complexity Theory Conference*, pages 82–95, 1993. 77

[14] P. Benioff. Space searches with a quantum robot. *AMS Contemporary Mathematics Series*, 305, 2002. `arXiv:quant-ph/0003006v2`. 9

[15] C. H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani. Strengths and weaknesses of quantum computing. *SIAM Journal on Computing (special issue on quantum computing)*, 26:1510–1523, 1997. `arXiv:quant-ph/9701001v1`. 25

[16] D. J. Bernstein. Cost analysis of hash collisions: Will quantum computers make SHARCS obsolete?, 2009. Workshop Record of SHARCS'09: Special-purpose Hardware for Attacking Cryptographic Systems. 1, 9, 10

[17] R. Bousso. The holographic principle. *Reviews of Modern Physics*, 74:825–874, 2002. `arXiv:hep-th/0203101v2`. 9

[18] M. Boyer, G. Brassard, P. Høyer, and A. Tapp. Tight bounds on quantum searching. *Fortschritte der Physik*, 56:493–505, 1998. `arXiv:quant-ph/9605034v1`. 23, 25

[19] G. Brassard, P. Høyer, M. Mosca, and A. Tapp. Quantum amplitude amplification and estimation. In S. J. Lomonaca and H. E. Brandt, editors, *Quantum Computation and Quantum Information Science*, volume 305 of *AMS Contemporary Mathematics*

Series Millennium Volume, pages 53–74. AMS, 2002. arXiv:quant-ph/0005055v1. 23, 25

[20] G. Brassard, P. Høyer, and A. Tapp. Quantum algorithm for the collision problem. *ACM SIGACT News*, 28:14–19, 1997. arXiv:quant-ph/9705002. 30

[21] J.-P. Delescaille and J.-J. Quisquater. How easy is collision search? applications to DES. In *Lecture Notes in Computer Science 434: Advances in Cryptology*, Eurocrypt '89, pages 429–434. Springer-Verlag, 1989. 43

[22] L. Fortnow. Birthday paradox variance. http://blog.computationalcomplexity.org/2009/11/birthday-paradox-variance.html, 2009. 115

[23] S. Fortune and J. Wyllie. Parallelism in random access machines. In *Proceedings of the tenth Annual ACM Symposium on Theory of Computing*, STOC '78, pages 114–118, New York, NY, USA, 1978. ACM. 12

[24] 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, New York, NY, USA, 1996. ACM. arXiv:quant-ph/9605043. 23, 25

[25] J. Horwitz. *Applications of Cayley Graphs, Bilinearity, and Higher-Order Residues to Cryptology*. PhD thesis, Stanford University, 2004. 43, 54, 55, 56, 58

[26] J. Horwitz and R. Venkatesan. Random Cayley digraphs and the discrete logarithm. In *Proceedings of the fifth International Algorithmic Number Theory Symposium*, pages 100–114, 2002. 42, 43, 54, 55, 56, 57

[27] P. Høyer, T. Lee, and R. Špalek. Negative weights make adversaries stronger. In *Proceedings of the thirty-ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 526–535, New York, NY, USA, 2007. ACM. arXiv:quant-ph/0611054v2. 94, 95, 96

[28] P. Høyer and R. Špalek. Lower bounds on quantum query complexity, 2005. arXiv:quant-ph/0509153v1. 92

[29] P. Kaye, R. Laflamme, and M. Mosca. *An Introduction to Quantum Computing*. Oxford University Press, 2006. 6

[30] J. Kempe. Quantum random walks - an introductory overview, 2003. arXiv:quant-ph/0303081v1. 37

[31] J. H. Kim, R. Montenegro, Y. Peres, and P. Tetali. A birthday paradox for Markov chains, with an optimal bound for collision in the Pollard rho algorithm for discrete logarithm. In *Proceedings of the eighth International Symposium on Algorithmic Number Theory*. Springer-Verlag, 2008. 67

[32] J. H. Kim, R. Montenegro, and P. Tetali. Near optimal bounds for collision in Pollard rho for discrete log. In *Proceedings of the forty-eighth Annual IEEE Symposium on Foundations of Computer Science*, pages 215–223, Washington, DC, USA, 2007. IEEE Computer Society. 67

[33] H. Klauck, R. Špalek, and R. de Wolf. Quantum and classical strong direct product theorems and optimal time-space tradeoffs. *SIAM Journal of Computing*, 36:1472 – 1493, 2007. 105

[34] H. T. Kung and C. D. Thompson. Sorting on a mesh-connected parallel computer. *Communications of the ACM*, 20:263–271, April 1977. 32, 47

[35] S. Kutin. Quantum lower bound for the collision problem with small range. *Theory of Computing*, 1:29–36, 2005. 31, 40, 82, 83

[36] S. Laplante and F. Magniez. Lower bounds for randomized and quantum query complexity using Kolmogorov arguments. In *Proceedings of the nineteenth IEEE Annual Conference on Computational Complexity*, pages 294–304, Washington, DC, USA, 2004. IEEE Computer Society. `arXiv:quant-ph/0311189v1`. 92

[37] C. Leopold. *Parallel and Distributed Computing: A Survey of Models, Paradigms and Approaches*. Wiley, 2000. 13

[38] D. A. Levin, Y. Peres, and E. L. Wilmer. *Markov Chains and Mixing Times*. American Mathematical Society, 2009. 34, 35, 62

[39] F. Magniez, A. Nayak, J. Roland, and M. Santha. Search via quantum walk. In *Proceedings of the thirty-ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 575–584, New York, NY, USA, 2007. ACM. `arXiv:quant-ph/0608026v4`. 37

[40] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001. 4

[41] S. D. Miller and R. Venkatesan. Non-degeneracy of Pollard rho collisions. *International Mathematics Research Notices*, 2009:1–10, 2008. 67

[42] N. Nisan. CREW PRAMs and decision trees. *SIAM Journal of Computing*, 20:999–1007, 1991. 12, 77

[43] R. Paturi. On the degree of polynomials that approximate symmetric Boolean functions (preliminary version). In *Proceedings of the twenty-fourth Annual ACM Symposium on Theory of Computing*, STOC '92, pages 468–474, New York, NY, USA, 1992. ACM. 80

[44] J. M. Pollard. A monte carlo method for factorization. *BIT Numerical Mathematics*, 15:331–334, 1975. 42

[45] B. Reichardt. Reflections for quantum query algorithms. In *Proceedings of the twenty-second ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 560–569, 2011. `arXiv:quant-ph/1005.1601v1`. 98

[46] Y. Shi. Quantum lower bounds for the collision and the element distinctness problems, 2001. `arXiv:quant-ph/0112086v1`. 82, 83

[47] M. Szegedy. Quantum speed-up of Markov chain based algorithms. In *Proceedings of the fory-fifth Annual IEEE Symposium on Foundations of Computer Science*, pages 32–41. IEEE Computer Society Press, 2004. 37

[48] E. Teske. Speeding up Pollard's rho method for computing discrete logarithms. In *Proceedings of the third International Symposium on Algorithmic Number Theory*, pages 541–554, London, UK, 1998. Springer-Verlag. 67

[49] E. Teske. On random walks for Pollard's rho method. *Mathematics of Computation*, 70:809–825, 2000. 67

[50] C. D. Thompson. Area-time complexity for VLSI. In *Proceedings of the eleventh Annual ACM Symposium on Theory of Computing*, STOC '79, pages 81–88, New York, NY, USA, 1979. ACM. 9

[51] P. C. van Oorschot and M. J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12:1–28, 1996. 42

[52] R. Špalek and M. Szegedy. All quantum adversary methods are equivalent. *Theory of Computing*, 2:1–18, 2006. `arXiv:quant-ph/0409116v3`. 91, 94

[53] A. C. Yao. The entropic limitations on VLSI computations. In *Proceedings of the thirteenth Annual ACM Symposium on Theory of Computing*, STOC '81, pages 308–311, New York, NY, USA, 1981. ACM. 106

[54] C. Zalka. Could Grover's quantum algorithm help in searching an actual database?, 1999. arXiv:quant-ph/9901068v1. 10

[55] C. Zalka. Grover's quantum searching algorithm is optimal. *Phys. Rev. A*, 60(4):2746–2751, Oct 1999. arXiv:quant-ph/9711070v2. 25, 26, 95, 98, 99, 102, 103

[56] S. Zhang. On the power of Ambainis' lower bounds. *Theoretical Computer Science*, 339:241–256, 2005. arXiv:quant-ph/0311060. 92

Appendix A

Birthday Paradox Arguments

The following lemma allows us to sample from a set using a limited independent function without worrying about the sample size being affected by the limited independence.

Lemma A.0.1. *Let $f : [N] \to [N]$ be a uniform random function and $h : [N] \to [N]$ be a uniform pairwise independent function. Define random variables $F_k = |\{f(1), \ldots, f(k)\}|$ and $H_k = |\{h(1), \ldots, h(k)\}|$. Then H_k and F_k are identically distributed.*

Proof. The random variable F_k can be seen as the distribution on the k^{th} step of the well-known coupon collector problem, if we start with 0 coupons. In this Markov process, there are N possible coupons, and at each step, a new one is chosen uniformly at random. The state of the Markov chain is (usually) the number of coupons seen so far, so as more coupons are collected, the chances of seeing a new one decrease.

As for H_k, the process of choosing a new element by h is not independent of previous choices, however, we can show that if we only consider the number of distinct elements of $[N]$ we've seen so far, H_k, then the transitions are actually only dependent on the current state.

We have for the first step $Pr[F_1 = 1] = 1$, since after we've chosen exactly one coupon, we have exactly one distinct coupon. For $k > 1$ and $s \leq k$ we have:

$$Pr[F_k = s] = Pr[F_{k-1} = s]Pr[F_k = s|F_{k-1} = s] + Pr[F_{k-1} = s-1]Pr[F_k = s|F_{k-1} = s-1]$$

$$= Pr[F_{k-1} = s]\frac{s}{N} + Pr[F_{k-1} = s-1]\frac{N-s+1}{N}$$

and $Pr[F_k = s] = 0$ when $s > k$.

We can similarly write $Pr[H_1 = 1] = 1$ and for $k > 1$ and $s \leq k$:

$$Pr[H_k = s] = Pr[H_{k-1} = s]Pr[H_k = s|H_{k-1} = s] + Pr[H_{k-1} = s-1]Pr[H_k = s|H_{k-1} = s-1]$$

Since h is pairwise independent, we have

$$Pr[h(i) = h(j)] = \sum_{a \in [N]} Pr[h(i) = a \wedge h(j) = a] = \sum_{a \in [N]} \frac{1}{N^2} = \frac{1}{N}$$

whenever $i \neq j$. We can compute:

$$Pr[H_k = s|H_{k-1} = s] = Pr[h(k) = h(1) \vee \cdots \vee h(k) = h(k-1)]$$

$$= Pr[\bigvee_{\text{unique } h(i)} h(k) = h(i)] = \sum_{\text{unique } h(i)} Pr[h(k) = h(i)] = s\frac{1}{N}$$

and

$$Pr[H_k = s|H_{k-1} = s-1] = Pr[h(k) \neq h(1) \wedge \cdots \wedge h(k-1)]$$

$$= 1 - Pr[\bigvee_{\text{unique } h(i)} h(k) = h(i)] = 1 - \frac{s-1}{N} = \frac{N-s+1}{N}$$

We can thus write:

$$Pr[H_k = s] = Pr[H_{k-1} = s]\frac{s}{N} + Pr[H_{k-1} = s-1]\frac{N-s+1}{N}$$

By induction we can see that $Pr[H_k = s] = Pr[F_k = s]$ for all s and k, so H_k and S_k are identically distributed. □

The following lemma is a generalization of the birthday paradox proof using variance from [22].

Lemma A.0.2. *Let $Y_1, \ldots, Y_m$ be a sequence of variables on $[r]$ such that for $(i,j) \neq (k,\ell)$, the events $Y_i = Y_j$ and $Y_k = Y_\ell$ are independent and identically distributed. Let $\mu = \Pr[Y_i = Y_j]$ for $i \neq j$. Then if $m \geq c\sqrt{\mu^{-1}}$ for some sufficiently large constant c, with at least constant probability, there are $i \neq j$ such that $Y_i = Y_j$.*

Proof. For $i < j$, let C_{ij} indicate the event $Y_i = Y_j$. Then $C = \sum_{i<j} C_{ij}$ is the number of collisions in $Y_1, \ldots, Y_m$. We want an upper bound on $\Pr[C = 0]$. By Chebyshev's inequality, we have:

$$\Pr[C = 0] \leq \Pr[|C - E[C]| \geq E[C]] \leq \frac{Var[C]}{E[C]^2}$$

Since the C_{ij} are pairwise independent, we have:

$$Var[C] = E[C^2] - E[C]^2 = E[\sum_{i<j} \sum_{k<\ell} C_{ij} C_{k\ell}] - E[\sum_{i<j} C_{ij}]^2$$

$$= \sum_{i<j} \sum_{k<\ell} E[C_{ij} C_{k\ell}] - (\sum_{i<j} E[C_{ij}])^2$$

$$= \sum_{i<j} E[C_{ij}^2] + \sum_{i<j} \sum_{k<\ell:(k,\ell)\neq(i,j)} E[C_{ij}]E[C_{k\ell}] - \sum_{i<j} \sum_{k<\ell} E[C_{ij}]E[C_{k\ell}]$$

$$= \sum_{i<j} E[C_{ij}] - \sum_{i<j} E[C_{ij}]^2 = \sum_{i<j} Var[C_{ij}]$$

We have $E[C] = \sum_{i<j} E[C_{ij}] = \binom{m}{2}\mu$ and $Var[C] = \sum_{i<j} E[C_{ij}] - \sum_{i<j} E[C_{ij}]^2 = \binom{m}{2}\mu - \binom{m}{2}\mu^2$. This gives:

$$\Pr[C = 0] \leq \frac{\binom{m}{2}\mu + \binom{m}{2}\mu^2}{\binom{m}{2}^2\mu^2} \leq \frac{1}{\binom{m}{2}\mu} \leq \frac{2}{m^2\mu} \leq \frac{2}{c^2}$$

□

Corollary A.0.3. *Let $Y_1, \ldots, Y_m$ be a sequence of random variables on $[r]$ such that $\Pr[Y_i = Y_j] = \frac{1}{N}$ for $i \neq j$, and the events $Y_i = Y_j$ and $Y_k = Y_\ell$ are independent for $(i, j) \neq (k, \ell)$. Then if $m \geq c\sqrt{N}$ for some sufficiently large constant c, with at least constant probability, there are $i \neq j$ such that $Y_i = Y_j$.*

In the following variation of the birthday paradox, we require only that the sequence be 4-wise independent, and that we have at least some overlap between the distribution of each random variable. If the distribution of the random variables were orthogonal, for instance, $Y_i = i$ with probability 1, of course we will not get a collision.

Lemma A.0.4. *Let $Y_1, \ldots, Y_m$ be a sequence of (not necessarily uniform, nor identically distributed) 4-wise independent random variables on $[r]$ with the property that if p_i is the*

distribution of some Y_i and p_j is the distribution of some P_j, then $\langle p_i, p_j \rangle \in O\left(\frac{1}{N}\right)$. Then if $m \geq c\sqrt{r}$ for some sufficiently large constant c, with at least constant probability, there are $i \neq j$ such that $Y_i = Y_j$.

Proof.

Notice that:

$$E[C_{ij}] = \Pr[Y_i = Y_j] = \sum_{\alpha \in [r]} p_i(\alpha) p_j(\alpha) = \langle p_i, p_j \rangle$$

so:

$$E[C] = \sum_{i<j} \langle p_i, p_j \rangle$$

As in the previous lemma, we have:

$$\Pr[C = 0] \leq \frac{Var[C]}{E[C]^2}$$

However, in this case, since the C_{ij} are not necessarily independent, we need to work harder to compute the variance:

$$Var[C] = \sum_{i<j} \sum_{k<\ell} E[C_{ij} C_{k\ell}] - \sum_{i<j} \sum_{k<\ell} E[C_{ij}] E[C_{k\ell}]$$

$$= \sum_{i<j} \sum_{k<\ell} \Pr[Y_i = Y_j \wedge Y_k = Y_\ell] - \sum_{i<j} \sum_{k<\ell} E[C_{ij}] E[C_{k\ell}]$$

$$= \sum_{i<j} \Pr[Y_i = Y_j] + \sum_{i<j<k} \Pr[Y_i = Y_j = Y_k] + \sum_{i<j,k<\ell, i\neq j\neq k\neq \ell} \Pr[Y_i = Y_j \wedge Y_k = Y_\ell] - \sum_{i<j} \sum_{k<\ell} E[C_{ij}] E[C_{k\ell}]$$

So:

$$\Pr[C = 0] \leq \frac{\sum_{i<j} \langle p_i, p_j \rangle}{\left(\sum_{i<j} \langle p_i, p_j \rangle\right)^2} + \frac{\sum_{i<j<k} \sum_{\alpha \in [r]} p_i(\alpha) p_j(\alpha) p_k(\alpha)}{\left(\sum_{i<j} \langle p_i, p_j \rangle\right)^2}$$
$$+ \frac{\sum_{i\neq j\neq k\neq \ell} \sum_{\alpha,\beta \in [r]} p_i(\alpha) p_j(\alpha) p_k(\beta) p_\ell(\beta)}{\left(\sum_{i<j} \langle p_i, p_j \rangle\right)^2} - 1$$

We will bound each term separately:

$$\frac{\sum_{i<j} \langle p_i, p_j \rangle}{\left(\sum_{i<j} \langle p_i, p_j \rangle\right)^2} = \frac{1}{\sum_{i<j} \langle p_i, p_j \rangle} \leq \frac{1}{\binom{m}{2} \frac{a}{N}}$$

for some constant a, by assumption. So:

$$\frac{\sum_{i<j} \langle p_i, p_j \rangle}{\left(\sum_{i<j} \langle p_i, p_j \rangle\right)^2} \leq \frac{2N}{am^2} \leq \frac{2N}{ac^2 N} = \frac{2}{ac^2}$$

Next we bound:

$$\frac{\sum_{i<j<k} \sum_{\alpha \in [r]} p_i(\alpha) p_j(\alpha) p_k(\alpha)}{\left(\sum_{i<j} \langle p_i, p_j \rangle\right)^2}$$

$$\leq \frac{\sum_{i<j<k} \left(\sum_{\alpha \in [r]} \sqrt{p_i(\alpha) p_j(\alpha)}\right) \left(\sum_{\alpha} \sqrt{p_i(\alpha) p_k(\alpha)}\right) \left(\sum_{\alpha} \sqrt{p_j(\alpha) p_k(\alpha)}\right)}{\left(\sum_{i<j} \langle p_i, p_j \rangle\right)^2}$$

$$\leq \frac{\sum_{i<j<k} \sqrt{\langle p_i, p_j \rangle \langle p_i, p_k \rangle \langle p_j, p_k \rangle}}{\left(\sum_{i<j} \langle p_i, p_j \rangle\right)^2}$$

The above is maximized when the $\langle p_i, p_j \rangle$ are minimized, giving:

$$\frac{\sum_{i<j<k} \sum_{\alpha \in [r]} p_i(\alpha) p_j(\alpha) p_k(\alpha)}{\left(\sum_{i<j} \langle p_i, p_j \rangle\right)^2} \leq \frac{\binom{m}{3} \frac{a^{3/2}}{N^{3/2}}}{\binom{m}{2}^2 \frac{a^2}{N^2}} \leq \frac{2(m-2)\sqrt{N}}{6m(m-1)\sqrt{a}} \leq \frac{\sqrt{N}}{3m\sqrt{a}} \leq \frac{\sqrt{N}}{3c\sqrt{N}\sqrt{a}} = \frac{1}{3c\sqrt{a}}$$

Finally, we bound:

$$\frac{\sum_{i \neq j \neq k \neq \ell} \sum_{\alpha, \beta \in [r]} p_i(\alpha) p_j(\alpha) p_k(\beta) p_\ell(\beta)}{\left(\sum_{i<j} \langle p_i, p_j \rangle\right)^2} = \frac{\sum_{i \neq j \neq k \neq \ell} \langle p_i, p_j \rangle \langle p_k, p_\ell \rangle}{\sum_{i<j, k<\ell} \langle p_i, p_j \rangle \langle p_k, p_\ell \rangle} \leq 1$$

since the sum on the bottom includes strictly more terms than the sum on the top.

Putting these together gives:

$$\Pr[C = 0] \leq \frac{2}{ac^2} + \frac{1}{3c\sqrt{a}}$$

which is bounded above by $\frac{1}{2}$ when c is large enough with respect to a (for instance, $c \geq 2\sqrt{2/a}$). □