Frameworks for Quantum Algorithms

by

Stacey Jeffery

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2014

© Stacey Jeffery 2014

Author's Declaration

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

Due to the difficulty of constructing new quantum algorithms, frameworks that facilitate this construction are of great importance in quantum computing. These frameworks reduce the problem of coming up with a quantum algorithm to that of constructing some combinatorial object that is often much simpler to reason about. The implementation and analysis of an algorithm for the specified problem follow from the properties of this object. A number of such frameworks have been extremely successful in leading to the development of numerous quantum algorithms for a variety of problems. In this thesis, we build on two of these frameworks, the quantum walk search framework, and the span program framework, extending their algorithmic potential.

The quantum walk search framework gives a generic quantum analogue to a specific type of classical algorithm based on random walks. If one can construct a classical algorithm of this form, a corresponding quantum algorithm with better complexity immediately follows. In this framework, a generic algorithm is constructed from several subroutines for which implementations must be given for each application. One of these subroutines, a checking subroutine, is run many times throughout the algorithm. This subroutine may be implemented by any quantum algorithm that satisfies the required functionality, including another quantum walk algorithm. By making a slight modification to the quantum walk framework, we can show how to nest a quantum walk algorithm in the checking subroutine of another quantum walk algorithm in a way that gives better complexity than the naive nesting. This modification allows us to reproduce a number of upper bounds previously obtained in another framework, the learning graph framework, including upper bounds for triangle finding, and more generally, subgraph finding for constant-sized subgraphs. Porting these upper bounds over to the setting of quantum walks is desirable because the algorithms achieved in the quantum walk search framework are much more explicit than those of the learning graph framework, making them easier to work with, modify, and build on, as needed. Our efficient nested checking idea has already been used to come up with new quantum algorithms for finding sub-hypergraphs.

Another subroutine that is called repeatedly by the generic quantum walk search algorithm is the update subroutine. It was not clear how to use a quantum walk algorithm to perform this step, but by making another slight modification to the quantum walk search framework, we are able to show how to nest a quantum walk in the update step of another quantum walk in an efficient way. Our technique for doing this is a special case of a technique that allows the update to be implemented with garbage — i.e., some unwanted data in an auxiliary register, entangled with the desired state. This technique may have other applications.

Using the nested update technique, we are able to improve the best known upper bounds on the time complexity of k-distinctness. Previously the best known upper bound on the time complexity was $n^{k/(k+1)}$, due to Ambainis. Belovs had recently improved the query complexity of k-distinctness to $o(n^{3/4})$ for all k, but since this upper bound was obtained in a framework called span programs, which only gives upper bounds on quantum query complexity, there was no known time efficient implementation of his k-distinctness algorithm. We use ideas from his construction and our nested update technique to get a quantum time upper bound for 3-distinctness of $n^{5/7}$, matching his quantum query upper bound. We can generalize our algorithm get a time upper bound of $n^{(k-1)/k}$ for any $k > 3$, slightly improving on the best previous upper bound.

Another framework, the span program framework, is known to be equivalent to quantum query complexity, in the sense that for any Boolean decision problem, there exists a span program construction that yields a tight upper bound on its quantum query complexity. We explore several variations of this framework. First, we slightly modify the definition of a span program so that we can show that for *any* (not necessarily Boolean) decision problem, there is a span program construction that yields a tight upper bound on its quantum query complexity. Previously this was only known up to logarithmic factors. We also explore several approximate versions of span programs, and show them to be equivalent. Finally, we explore the structure of span program witnesses, and use this structure to present an algorithm for evaluating span programs that is straightforward and intuitive. We also show how to evaluate approximate span programs, opening the possibility for the construction of new upper bounds using approximate span programs.

Acknowledgements

I would like to begin by expressing my gratitude to my supervisor, Michele Mosca, for his mentorship, advice, and unending support and encouragement throughout my undergraduate, masters, and doctoral degrees. I would also like to thank the rest of my committee, Andrew Childs, Ashwin Nayak, Jérémie Roland and John Watrous for taking the time to read my thesis, and for their useful comments and suggestions.

I've had the good fortune to work with a number of collaborators from whom I have learned a great deal over the course of my Ph.D. I would like to begin by thanking the co-authors of the work presented in this thesis: Aleksandrs Belovs, Andrew Childs, Tsuyoshi Ito, Robin Kothari, and Frédéric Magniez. I would also like to thank Robin for his friendship and useful perspectives on life and career; and Frédéric for his outstanding mentorship and support, and his uncanny ability to choose good wines. In addition, I would like to thank my collaborators on works not appearing in this thesis: Gorjan Alagic, Anne Broadbent, Stephen Jordan, Francois Le Gall, and Ronald de Wolf. I learned a great deal from all of them, and would like to thank Anne in particular for her patient mentorship since I was an undergraduate student.

I have been fortunate to learn a lot from visiting and working with several other brilliant researchers not already mentioned. I would like to thank Martin Roetteler for being a great boss and mentor while I worked as an intern at NEC Labs, and Miklos Santha who, in addition to hosting me on several visits to CQT, has given me invaluable perspectives and advice on both career and life.

As a member of the quantum computing community, I have been fortunate to be a part of a strong support network of fellow students. To name just a few, I would like to thank Robin Kothari and Shelby Kimmel for sharing the ups and downs of finishing a Ph.D and transitioning to the next stage of life; Laura Mancinska for being my gym buddy, both at the gym and in life; Alessandro Cossentino for engaging philosophical discussions; and Vinayak Pathak for inspiring me to continue working on my thesis even after I ran out of energy.

Most importantly, I am very lucky to have the strong support of my immediate and extended family. I would like to thank my mother, and grandmothers for giving me three strong role models in whose footsteps to follow; my brother, Thomas Jeffery, whom I've always looked up to, for letting me follow him around everywhere when we were children, and for interesting discussions about Mathematics now that we're adults; and my step-father, aunts, uncles, and godparents, for supporting, encouraging, and believing in me.

Above all, I would like to thank my wonderful husband, Moritz Ernst, for being my partner in everything, and my number one support. Thank you for bringing my life into focus and giving meaning to the work that has gone into this thesis.

Dedication

In memory of Michael Jeffery.

Table of Contents

Chapter 1

Introduction

Due to the difficulty of conceptualizing the behaviour of quantum mechanical systems, inventing new ways for quantum computers to solve tasks faster than any classical computer is extremely difficult. As such, there are only a small number of completely "new" quantum algorithms — two famous examples being Shor's factoring algorithm and Grover's search algorithm [Sho97, Gro96] — most other quantum algorithms are built upon already existing quantum algorithms, and their main contributions often stem from coming up with clever ways to apply known quantum techniques in a novel way.

A quantum algorithmic framework provides a formal way for doing this. Such a framework breaks down the problem of coming up with a quantum algorithm to constructing some combinatorial object, which is typically much easier to conceptualize. The implementation and analysis of the algorithm then follow directly from some properties of this object. A very large number of quantum algorithms have been created in this way, through frameworks such as amplitude amplification, which generalizes Grover search so that a generic squareroot speedup can be obtained for any unstructured search problem.

Some of these frameworks, such as amplitude amplification, also have the advantage that they can be applied with essentially no background in quantum algorithms. The quantum walk search framework [MNRS11, Sze04] can also be applied with little understanding of the underlying quantum algorithm, as algorithms in this framework are analogous to a class of classical algorithms based on random walks. Generally speaking, given a classical algorithm of this particular form, there is an analogous quantum algorithm with smaller complexity. Thus, to apply this framework, one simply needs to come up with a classical algorithm of a particular form, for which one only needs an understanding of the classical theory of random walks.

Quantum Walk Search The MNRS quantum walk search framework [MNRS11] is built on a similar framework due to Szegedy [Sze04], which in turn can be seen as a generalization of an algorithm due to Ambainis [Amb04]. For any random walk on a finite state space, this framework gives a quantum algorithm for finding a marked state that requires access to three subroutines. The first, the *setup subroutine*, must create a superposition over all states in the search space. The second, the *checking subroutine*, must check if a state is marked. Finally, the *update subroutine* must create a superposition over all states that are adjacent to the current state. If the respective costs of implementing these subroutines are S, C, and U, ε is the probability a state is marked, and δ is the spectral gap of the random walk, then the complexity of the algorithm scales like

$$\mathsf{S} + \frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C}\right).$$

This framework makes it quite simple to construct and analyze a quantum search algorithm, by choosing some appropriate random walk, implementing the specified subroutines, and computing their costs.

The framework can be seen as a generalization of an algorithm for element distinctness, where the input is a list of integers, and we want to find two that are the same [Amb04]. More generally, Ref. [Amb04] gives a quantum walk algorithm for k-distinctness, in which we want to find k copies of the same integer in the input, with quantum time complexity $\widetilde{O}(n^{k/(k+1)})$, for any $k \geq 2$. When $k = 2$, Ambainis's upper bound is tight [AS04, Amb05, Kut05]. The best known lower bound for any $k > 2$ is also $O(n^{2/3})$.

Another early algorithm in the quantum walk search framework was an algorithm for triangle finding, the problem of finding a triangle in an input graph, given by its adjacency matrix [MSS07]. Ref. [MSS07] used a quantum walk search algorithm to give an upper bound of $O(n^{13/10})$ on the query complexity of triangle finding.

Learning Graphs A subsequent quantum algorithmic framework, the *learning graph framework* [Bel12b], constructs a quantum algorithm from a combinatorial object called a learning graph. The query complexity of this algorithm is then given by an expression that is simple to compute for structured learning graphs. However, the framework does not give an upper bound on the time complexity. The underlying algorithm consists of alternating two reflections. One of these, although it is known to be implementable with query complexity 0, because it is independent of the input, may take a very general form, and no implementation is specified by the framework, so it is not clear how to implement this reflection in a time-efficient manner. Nonetheless, the framework has been very successful in proving upper bounds on quantum query complexity. The first was an upper bound on triangle finding [Bel12b], improving the previous quantum walk upper bound to $O(n^{35/27}) = O(n^{1.296})$. The combinatorics of this construction are similar to those of the quantum walk algorithm for triangle finding, but with one extra idea, which we call *edge sparsification*. It is possible to use this technique in the quantum walk algorithm, but doing so yields no improvement.

A subsequent learning graph construction further improved the quantum query complexity of triangle finding to $O(n^{9/7}) = O(n^{1.286})$ [LMS13].[1]

Span Programs A final framework of note is the span program framework, first used in the context of quantum query complexity in [RŠ12]. This framework is very general for quantum query complexity: For every Boolean decision problem, there exists an algorithm in this framework with optimal quantum query complexity [Rei09, Rei11]. However, this generality makes the framework somewhat difficult to apply. Furthermore, it suffers from the same drawback as learning graphs, learning graphs being a specialization of span programs: it does not generally give upper bounds on the time complexity, since, as with learning graphs, the underlying algorithm is specified by a pair of reflections with no implementation given. Despite the difficulty of applying the framework, Belovs was able to use it to improve the quantum query complexity of k-distinctness for all $k > 2$, to $o(n^{3/4})$ [Bel12a], improving upon Ambainis's previous upper bound, which had been the best known for many years. However, since this algorithm could not be analyzed for time complexity, Ambainis's upper bound on the time complexity of k-distinctness was still the best known, before the work presented in this thesis.

1.1 Contributions in this Thesis

This thesis presents extensions to two important frameworks in the design of quantum algorithms: the quantum walk search framework, and the span program framework.

Quantum Walks The major work of this thesis is an extension to the MNRS quantum walk search framework, along with a number of applications. The extension to the quantum walk framework is based on two publications. The first publication, [JKM13b], in collaboration with Robin Kothari and Frédéric Magniez, presents a modification to the MNRS quantum walk search framework that allows the implementation of a technique for efficiently nesting one quantum walk search algorithm in the checking step of another. Suppose a quantum walk search algorithm has setup cost S, update cost U, spectral gap δ, and probability of a state being marked ε. It may be possible to implement the checking subroutine using a quantum walk algorithm. If this algorithm has costs $(\mathsf{S}', \mathsf{U}', \mathsf{C}', \delta', \varepsilon')$, then the checking cost is $\mathsf{C} = \mathsf{S}' + \frac{1}{\sqrt{\varepsilon'}}\left(\frac{1}{\sqrt{\delta'}}\mathsf{U}' + \mathsf{C}'\right)$, so the total cost of the algorithm becomes:

$$\mathsf{S} + \frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{S}' + \frac{1}{\sqrt{\varepsilon'}}\left(\frac{1}{\sqrt{\delta'}}\mathsf{U}' + \mathsf{C}'\right)\right),$$

[1]Following the submission of this thesis, Le Gall used the MNRS quantum walk search framework to give an upper bound of $\widetilde{O}(n^{5/4})$ on the quantum query complexity of triangle finding [Le 14].

up to logarithmic factors incurred from amplifying the success probability of the checking subroutine. We show a method for improving this cost to (up to logarithmic factors):

$$\mathsf{S}+\mathsf{S}'+\frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U}+\frac{1}{\sqrt{\varepsilon'}}\left(\frac{1}{\sqrt{\delta'}}\mathsf{U}'+\mathsf{C}'\right)\right).$$

Since there are no longer any multiplicative factors on S', this can decrease the cost. As an application, we show how to implement the edge sparsification idea used by Belovs to obtain the $O(n^{35/27})$ upper bound on triangle finding as a quantum walk search algorithm, reproducing his upper bound up to logarithmic factors. We are also able to use the ideas of [LMS13] to reproduce their triangle finding upper bound of $O(n^{9/7})$ up to logarithmic factors, as well as a more general construction they give for finding any constant-sized subgraph. Although neither of these applications gives a new upper bound for triangle finding, the quantum walk framework leads to an explicit algorithm that is easy to analyze and modify, making it subjectively more desirable to work with. In addition, the simplicity of the nested-checking quantum walk search algorithmic technique will hopefully facilitate the construction of new upper bounds. Our framework has already been used to generalize the work of [LMS13] to obtain new upper bounds on the quantum query complexity of finding sub-hypergraphs [FLT14].

The second publication consisting of work from Part II, Ref. [BCJ+13], is a merger of two manuscripts: [CJKM13], which is joint work with Andrew Childs, Robin Kothari, and Frédéric Magniez; and [Bel13], by Aleksandrs Belovs. This work shows two different means of obtaining a new quantum time upper bound of $n^{5/7}$ for 3-distinctness. In this thesis, we will only be presenting the work in Ref. [CJKM13]. In this work, we make a further modification to the quantum walk framework that allows us to efficiently nest one quantum walk search algorithm in the update step of another, just as we did for checking, to obtain a cost expression (in most cases) like:

$$\mathsf{S}+\mathsf{S}'+\frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}\frac{1}{\sqrt{\varepsilon'}}\left(\frac{1}{\sqrt{\delta'}}\mathsf{U}'+\mathsf{C}'\right)+\mathsf{C}\right),$$

up to logarithmic factors. Accomplishing this type of nesting is significantly more difficult than nested checking. The technique we use to accomplish this is a special case of a technique that allows us to do updates with garbage, which may be of independent interest. As an application of this technique, we prove the aforementioned upper bound on the time complexity of 3-distinctness of $\widetilde{O}(n^{5/7})$, improving upon the previous best time complexity upper bound of $\widetilde{O}(n^{3/4})$, and matching the best known query complexity.

Additionally, in Section 6.2, we show how to generalize the algorithm for 3-distinctness to k-distinctness, obtaining a nested-update quantum walk search algorithm that solves k-distinctness in time $\widetilde{O}(n^{(k-1)/k})$, for any $k>3$, improving upon the best previous upper bound of $\widetilde{O}(n^{k/(k+1)})$. On the one hand, this improvement is only slight, and still does not match the best known upper bound on the quantum query complexity, of $o(n^{3/4})$ for all k. However, this is the first progress on the time complexity of this problem in 10 years, and our algorithm may serve as a starting point for another, better algorithm.

The modifications to the quantum walk framework, as well as their applications, appear in Part II. To begin, in Chapter 4, we present a new quantum walk search framework, generalizing [MNRS11], that includes the extensions of both Ref. [JKM13a] and Ref. [CJKM13]. In Chapter 5, we show how to use this new framework to achieve the algorithmic technique of nested checking, and the applications of this technique to triangle finding and more general subgraph finding. Finally, in Chapter 6, we present the nested update technique from Ref. [CJKM13], and its application to 3-distinctness, and more generally, k-distinctness.

Span Programs In Part III, we explore extensions to the span program framework, based on collaboration with Tsuyoshi Ito. We first make a very slight modification to the definition of a span program that allows span programs to be used to obtain tight upper bounds on the quantum query complexity of any decision problem, not only those over a Boolean alphabet, which we prove in Section 7.1. This is not of huge practical importance, since another formalism, the *dual adversary bound*, is equivalent to span programs, and can already obtain tight upper bounds for problems on non-Boolean alphabets, however, as with other results in this Chapter, the main motivation is theoretical interest.

Another theoretical problem we consider, is that a very natural construction for converting quantum algorithms to span programs, presented in Section 7.2, only works for algorithms with one-sided error, which is theoretically unsatisfying, since it is known that for every Boolean decision problem, there exists a span program for that problem whose complexity matches the *two-sided* error quantum query complexity. This motivates us to consider various notions of *approximate* span programs in Section 7.3. In a very satisfying turn of events, we are able to show that all of these notions are actually equivalent.

In Section 7.4, we investigate the underlying structure of *witnesses* in span programs, uncovering a structure that facilitates a more straightforward algorithm for evaluating span programs, and in Section 7.5, we present a general transformation that allows us to put any span program in a specific form. These results allow us to present a simple and easy to analyze algorithm for evaluating span programs in Section 7.6.1, as compared with previous span program evaluation algorithms. Finally, in Section 7.6.2, we show that our definition of approximate span programs is meaningful by giving an algorithm to evaluate an approximate span program. This opens up the possibility for new upper bounds on quantum query complexity to be proven in our approximate span program framework.

1.2 Other Contributions

During my PhD, I have been involved in several research projects whose results I have opted not to include, in the interest of coherence. Several of these relate to quantum algorithms. The first of these is a quantum algorithm for Boolean matrix multiplication that has optimal quantum query complexity, up to logarithmic factors, $O(n\sqrt{\ell})$, where ℓ is the number of 1s in the output. This algorithm was joint work with Robin Kothari and Frédéric Magniez, and appears in Ref. [JKM13a].

The second quantum algorithmic result is a quantum walk algorithm for subset sum, which is joint work with Dan Bernstein, Tanja Lange, and Alex Meurer [BJLM13]. We speed up a classical heuristic algorithm [HGJ10], to obtain a quantum heuristic algorithm whose average case complexity is exponential, but with the smallest known constant in the exponent. Since such algorithms can be used to attack code-based cryptosystems, the best known achievable complexity is important in the field of cryptography. The speedup is obtained by a straightforward modification of Ambainis's element distinctness algorithm, which is time-efficient when an appropriate quantum data structure is used. Ambainis uses a data structure that is a combination of a hash table and a skip-list. In contrast, we modify a simpler data structure, called a radix tree, to the quantum setting. We will use this data structure for time-efficient algorithms in this thesis. Its description can be found in Section 3.3.4.

The final algorithmic result gives matching upper and lower bounds on the parallel quantum query complexities of element distinctness, and k-sum for all $k > 1$. We show that for any $p \leq n$, the p-parallel quantum query complexity of element distinctness, in which we can make p queries at each time step, is $\Theta((n/p)^{2/3})$, and the p-parallel quantum query complexity of k sum is $\Theta((n/p)^{k/(k+1)})$. Our upper bounds are obtained in the quantum walk search framework, and our lower bounds are obtained by applying the adversary method. This result is joint work with Frédéric Magniez and Ronald de Wolf, and appears in [JMd14].

In addition, I have been involved in two projects related to quantum cryptography. The first project is joint work with Gorjan Alagic and Stephen Jordan, and proposes an obfuscation scheme for both classical and quantum circuits based on braid group representations. This work appears in [AJJ14].

The second quantum cryptography project is a homomorphic encryption scheme for quantum circuits, in which the size of the evaluation circuit scales with the number of non-Clifford gates. This work is in collaboration with Anne Broadbent, and is currently in preparation.

Part I

Preliminaries

Chapter 2

Mathematical Preliminaries

In this section, we give some miscellaneous mathematical preliminaries required for understanding the results of this thesis. We begin by mentioning some standard notation that will be used throughout this thesis for linear algebraic objects. We then give the required background on random walks that will be used in Part II. Finally, in Section 2.3 we present a collection of tail bounds and other facts from probability theory that will be used throughout this thesis.

2.1 Linear Algebra Notation

In this section we outline some common notation that will be used throughout this thesis.

For any vector v in an inner product space $H = \mathrm{span}\{|i\rangle : i \in [n]\}$, we define

$$\|v\| := \sqrt{\sum_{i\in[n]} |\langle i|v\rangle|^2}.$$

For vector spaces H_1 and H_2, we let $\mathcal{L}(H_1, H_2)$ denote the set of linear maps from H_1 to H_2. For any operator $A \in \mathcal{L}(H_1, H_2)$ for inner product spaces H_1 and H_2, we define

$$\|A\| := \max_{|h\rangle\in H_1 : \||h\rangle\|=1} \|A|h\rangle\|.$$

Let V be a subspace of H. Then we will use Π_V to denote the orthogonal projector onto the space V (where H should be clear from context), and $\mathrm{Ref}V = 2\Pi_V - I$, the reflection about V. When $V = \mathrm{span}\{|\psi\rangle\}$ is a one-dimensional space, we will often write Π_V as $\Pi_{|\psi\rangle}$ and $\mathrm{Ref}V = \mathrm{Ref}\{|\psi\rangle\}$.

For subspaces H_1 and H_2 of an inner product space H, we let $H_1 + H_2$ be the span of all vectors in H_1 and H_2. For orthogonal spaces H_1 and H_2, we write this as $H_1 \oplus H_2$.

2.2 Random Walks

For a thorough introduction to the subject of random walks, see [LPW09]. We now outline those properties of random walks and Markov processes that will be used in this thesis.

Random Walks on Graphs For a graph G with vertex set Ω and (possibly weighted) edges E, a random walk is a random process that models a "walker" on some vertex (or distribution of vertices) of the graph. At every step of the process, the walker choose a neighbour of the current vertex at random and moves to that neighbour. Concretely, if u is the current state, and $\Gamma(u) = \{v \in \Omega : \{u, v\} \in E\}$ is the neighbourhood of u, then the probability that the next state will be v is given by 0 if $v \notin \Gamma(u)$, and $\frac{1}{\deg(u)}$ if $v \in \Gamma(u)$, where $\deg(u) = |\Gamma(u)|$. A random walk on a graph is a kind of *Markov process.*

Markov Processes A Markov process is a random process that takes a distribution on some (in this thesis, finite) state space Ω, and maps it to another distribution on Ω, such that the distribution of the next state depends only on the current state. A Markov process on the state space Ω can be specified by a stochastic *transition matrix*, $P \in \mathbb{R}^{\Omega \times \Omega}$, where for any $u, v \in \Omega$, $P(u, v)$ gives the probability of moving to the state v from the state u. A random walk is one type of Markov process, in which $P(u, v)$ is $1/\deg(u)$ if u and v are adjacent, and 0 otherwise.

A Markov process is *irreducible* if for any pair of states $u, v \in \Omega$, there exists an integer t such that starting from u, the probability of reaching v after t steps is non-zero — that is, every state is *reachable* from every other state.

The *period of a state* $u \in \Omega$ with respect to P is the minimum positive integer k such that the probability of being in state u after $t > 0$ steps starting from u is nonzero if and only if k divides t. For example, in a random walk on a bipartite graph, every state has even period $k \geq 2$. The *period of a Markov process* P is the greatest common divisor of the period of every state. For example, the period of a random walk on a bipartite graph is always an even integer $k \geq 2$. A Markov process with period 1 is called *aperiodic.*

A Markov process is *ergodic* if it is irreducible and aperiodic.

Stationary Distribution A *stationary distribution* of P is a probability distribution $\pi \in \mathbb{R}^{\Omega}$ such that $\pi P = \pi$ — that is, π is a left 1-eigenvector of P. This means that if the current state is distributed according to π, the next state will also be distributed according to π.

Fact 2.2.1. *An irreducible Markov process has a unique stationary distribution.*

If π is a stationary distribution for P, then whenever the state is distributed according to π, it will stay that way. For many Markov processes, it is actually the case that for any initial distribution, each step of the Markov process brings us closer to some unique π. This is not always the case: consider a random walk on a bipartite graph with classes A and B. If we start in a particular state $u \in A$, then for odd t, after t steps, all weight is on B, but for even t, after t steps, all weight is on A. Thus, starting from u, we will not converge towards any particular distribution. However, it turns out that this sort of periodicity is the only thing that can prevent us from converging to a unique stationary distribution.

Fact 2.2.2 (Convergence Theorem). *If P is ergodic, then there exists a distribution $\pi \in \mathbb{R}^{\Omega}$ such that for any initial distribution $\rho \in \mathbb{R}^{\Omega}$,*

$$\lim_{t\to\infty} \sum_{u\in\Omega} \left\|\rho P^t - \pi\right\|_{tv} = 0.$$

An irreducible Markov process P such that for every pair of states $u, v \in \Omega$, $\pi(u)P(u,v) = \pi(v)P(v,u)$ is said to be *reversible*. A random walk on an undirected graph is always reversible.

A question of central importance in the study of Markov processes is the question of just how many steps are needed to get within ε of the stationary distribution.

Definition 2.2.3 (Mixing Time). *Let P be ergodic. The* mixing time, *τ_{mix}, of P, is the minimum t such that for any initial distribution $\rho \in \mathbb{R}^{\Omega}$,*

$$\left\|\rho P^t - \pi\right\|_{tv} \le \frac{1}{3}.$$

Here $\|\cdot\|_{tv}$ is the *total variation distance*, defined $\|\rho_1 - \rho_2\|_{tv} = \frac{1}{2}\sum_u |\rho_1(u) - \rho_2(u)|$.

Definition 2.2.4 (Spectral Gap). *Let $\lambda_1, \ldots, \lambda_n$ be the complete set of eigenvalues of P, with multiplicity, and suppose $1 = \lambda_1 \ge |\lambda_2| \ge \cdots \ge |\lambda_n|$. Then the* spectral gap *of P is defined as $\delta := 1 - |\lambda_2|$.*

Fact 2.2.5. *If P is ergodic, then the spectral gap of P is > 0.*

The spectral gap is important because of its relationship with the mixing time.

Fact 2.2.6 (Relaxation Time). *If P is reversible and ergodic with spectral gap δ and stationary distribution π, with $\pi_{\min} := \min_{u\in\Omega} \pi(u)$, then*

$$\left(\frac{1}{\delta} - 1\right)\log\frac{3}{2} \le \tau_{\text{mix}} \le \frac{1}{\delta}\log\frac{3}{\pi_{\min}}.$$

A number of other quantities measure how quickly weight moves between states under P.

Definition 2.2.7 (Hitting Time). *For two states $u, v \in \Omega$, the hitting time from u to v, $H(u,v)$, is defined as the expected number of steps to hit v when starting from u.*

Definition 2.2.8 (Commute Time). *For two states $u, v \in \Omega$, the commute time between u and v is defined as*

$$C(u,v) := H(u,v) + H(v,u).$$

Random Mapping Representation An alternative to representing a Markov process on a state space Ω as a stochastic transition matrix $P \in \mathbb{R}^{\Omega\times\Omega}$ is called a *random mapping representation.* A random mapping is a function $f : \Omega \times \Lambda \to \Omega$ for some finite set Λ, which we call the *coin space* of P, and a random variable R on Λ. Then f represents P if for all $u, v \in \Omega$,

$$\Pr_R[f(u,R) = v] = P(u,v). \tag{2.1}$$

Intuitively, we can think of implementing the Markov process P by tossing a $|\Lambda|$-sided coin, and then choosing the next state v as a function of the current state u and the coin outcome r, according to f. Then whenever the coin lands on side r, we move to the state $f(u,r)$. The condition in (2.1) ensures that the resulting distribution is exactly $P(u,\cdot)$, the row of P labelled by the current state u.

As an example, consider a random walk on state space $\mathbb{Z}/n\mathbb{Z}$, the cyclic group of n elements, defined as

$$P(u,v) = \begin{cases} \frac{1}{2} & \text{if } v \in \{u-1, u+1\} \\ 0 & \text{else.} \end{cases}$$

A much more succinct and intuitive way to describe this random walk is that at each step, with probability 1/2 we add 1 to the current state, and with probability 1/2 we subtract 1 from the current state. In the much more natural random mapping representation, this looks like $f(u,R) = u + R$, where R is the uniform distribution on $\{-1, 1\}$. If we do not restrict Λ to be finite, then every Markov process given by a transition matrix P has a random mapping representation.

Our interest in random mapping representations is due to their potential for providing a succinct encoding of a transition from one state to another: if we consider some $(u,v) \in \overrightarrow{E}(P)$, where $\overrightarrow{E}(P)$ is the set of pairs $(u,v) \in \Omega^2$ such that $P(u,v) > 0$, we can encode this edge by explicitly writing down both states, as (u,v), or we can encode this edge as (u,r) for some r such that $f(u,r) = v$. A function f need not be, strictly speaking, a random mapping in order to accomplish this task. For example, let $\Lambda = \Omega$, and define $f : \Omega \times \Lambda \to \Omega$ by $f(u,v) = v$ and $\Pr[R = v] = P(u,v)$. This is not a random mapping, because the distribution of R depends on u, but it still fulfills our requirements in terms of representing a state. We define a *non-strict random mapping representation* of P to be a function $f : \Omega \times \Lambda \to \Omega$ and random variables $\{R_u : u \in \Omega\}$ on Λ. The transition representation, given by the matrix P, satisfies this definition, with $\Lambda = \Omega$, $f = P$, and R_u distributed according to $P(u,\cdot)$, the u^{th} row of P, but there may be more succinct non-strict random mapping representations for P.

Johnson Graphs For positive integers n and r with $r < n$, the *Johnson graph*, $J(n,r)$, is the graph on vertex set $\Omega = \{S \subseteq [n] : |S| = r\}$, with edges $E = \{\{S, S'\} \subset \Omega : |S \cap S'| = r - 1\}$.

The random walk on $J(n,r)$ has the following transition matrix:

$$P(S,S') = \begin{cases} \frac{1}{r(n-r)} & \text{if } |S \cap S'| = r-1 \\ 0 & \text{else.} \end{cases}$$

The random walk on $J(n,r)$ is more intuitively described by saying that at every step, if S is the current state, we move to a new state by removing an element from S, and adding a new element to S, to obtain S'. We can formalize this as a non-strict random mapping representation, $f : \Omega \times \Lambda \to \Omega$, and random variables $\{R_S\}_{S\in\Omega}$. Define $\Lambda = \{(i,j) \in [n]^2\}$, and for every $S \in \Omega$, let R_S be the distribution on Λ with uniform weight on (i,j) such that $i \in S$ and $j \notin S$. Then we set $f(S,(i,j)) = (S \setminus \{i\}) \cup \{j\}$. In this way, we can succinctly represent the transitions of the random walk on $J(n,r)$.

We can further generalize the graph $J(n,r)$. For positive integers $m < r < n$, let $J(n,r,m)$ denote the graph on vertices $\Omega = \{S \subset [n] : |S| = r\}$ with edges $E = \{\{S,S'\} \subset [n] : |S \cap S'| = r - m\}$. In this case, transitions can be done by removing m elements from the current state S, and then adding m new elements. For any $m < r < n$, the unique stationary distribution of the walk on $J(n,r,m)$ is uniform, because $J(n,r,m)$ is a degree-regular graph. Furthermore, we have the following fact about the spectral gap.

Fact 2.2.9 (See for example [God10]). *For all n and r in $\mathbb{N}$ with $r < n$, the spectral gap of the random walk on $J(n,r)$ is at least $\frac{1}{r}$. For all $n,r,m \in \mathbb{N}$ with $m < r < n$, the spectral gap of the random walk on $J(n,r,m)$ is at least $\frac{m}{r}$.*

2.3 Probability Theory and Tail Bounds

We will make use of several tail bounds in order to lower bound the weight of a distribution around the mean. The following tail bounds apply to random variables distributed according to a hypergeometric distribution.

Fact 2.3.1 (Hypergeometric Tail Bounds[JLR11]). *Let Z be a random variable distributed according to a hypergeometric distribution with m draws, population N, and k possible successes. Let $\mu = \frac{mk}{N}$. Then for every $z \geq 7\mu$:*

$$\Pr[Z \geq z] \leq e^{-z}.$$

Furthermore, for any $t \geq 0$:

$$\Pr[Z \geq \mu + t] \leq e^{-2t^2/m}.$$

Finally, for any $\epsilon > 0$:

$$\Pr[|Z - \mu| \geq \epsilon\mu] \leq 2\exp\{-((1+\epsilon)\log(1+\epsilon) - \epsilon)\mu\}.$$

The next tail bound can be applied to any sum of negatively correlated random variables.

Fact 2.3.2 (Tail Bound for Negative Correlation)**.** *Let $\{Z_i\}_{i\in[n]}$ be a set of random variables on $\{0,1\}$, such that for each i, $\Pr[Z_i = 1] = p$, and for each $i \neq j$, $\Pr[Z_iZ_j = 1] \leq p^2$. Let $Z = \sum_{i=1}^{n} Z_i$. Then for any $\epsilon > 0$:*

$$\Pr[Z < np - \epsilon] \leq e^{-\epsilon^2/(2np)}.$$

We will require the use of a k-wise independent partition, defined below.

Definition 2.3.3. *A random partition of $[n]$ into k disjoint sets $A_1, \ldots, A_k$ is called* uniform k-wise independent *if for every $t \leq k$, and distinct $i_1, \ldots, i_t \in [n]$, and any $j_1, \ldots, j_t \in [k]$*

$$\Pr[i_1 \in A_{j_1}, \ldots, i_t \in A_{j_t}] = \frac{1}{k^t}.$$

It is often useful in randomized or quantum algorithms to randomly partition some set $[n]$. A k-wise independent partition is sufficiently random for many purposes, but as we will now see, it can have the desirable property of being easy to compute. We can construct a k-wise independent partition using a k-wise independent function.

Definition 2.3.4. *A random function, $F : [n] \to [\ell]$, is said to be* k-wise independent *if for any $t \leq k$ and distinct $i_1, \ldots, i_t \in [n]$, and any $(j_1, \ldots, j_t) \in [\ell]^t$,*

$$\Pr[(F(i_1), \ldots, F(i_t)) = (j_1, \ldots, j_t)] = \frac{1}{\ell^t}.$$

When $n = \ell$ is a prime power, a simple example of a k-wise independent function is a uniform random polynomial of degree $k-1$ over the finite field $\mathrm{GF}(n)$ [WC81]. Each such polynomial can be represented using $O(k \log n)$ bits and evaluated using $O(k)$ additions and multiplications over $\mathrm{GF}(n)$. More efficient constructions exist, but the polynomial construction suffices here. It can be extended to any integer n by allowing a small statistical distance between the distribution $(F(i_1), \ldots, F(i_k))$ and the uniform distribution over $[n]^k$.

For simplicity, we now assume that we have at our disposal a (perfect) k-wise independent function $F : [n] \to [n]$. We can use this to construct an efficiently computable k-wise independent partition by defining $i \in A_j$ whenever $F(i) = j \pmod{n}$.

Chapter 3

Survey of Quantum Algorithmic Frameworks

In this chapter, we survey the state of the art in frameworks that facilitate the construction of quantum algorithms. The new work presented in Parts II and III will build on the MNRS quantum walk search framework, presented in Section 3.3.2, and the span program framework presented in Section 3.4. The algorithms we present in Sections 5.2, 5.3, 5.4 and 6.2 are based on algorithmic examples presented in this chapter.

This chapter is organized as follows. In Section 3.1, we begin by describing some preliminary material on quantum algorithmics that will be used throughout this thesis, including the phase estimation technique in Section 3.1.1, and methods for dealing with errors in quantum subroutines in Section 3.1.2. Next, in Section 3.2, we describe the first quantum algorithmic framework, the amplitude amplification framework.

In Section 3.3, we describe the MNRS quantum walk search framework, upon which the work of Part II of this thesis is based. After describing this framework, we present several applications of the framework. The first is an algorithm for k-distinctness, due to Ambainis [Amb04], presented in Section 3.3.3. Our improved k-distinctness algorithm in Section 6.2 will use a variation of this algorithm as a subroutine. In Section 3.3.4, we will present a quantum data structure, first used in [BJLM13], called a quantum radix tree, that will allow us to make Ambainis's k-distinctness algorithm time-efficient (Ambainis did this using a different quantum data structure). The data structure presented in Section 3.3.4 will also be used in our improved k-distinctness algorithm in Section 6.2. The final quantum walk application, in Section 3.3.5, is an algorithm for triangle finding. Subsequent improvements to triangle finding can be seen as extensions to this algorithm.

In Section 3.4, we will briefly describe the span program framework, on which the work of Part III is based. In Section 3.5, we will discuss the learning graph framework, which facilitates the construction of a restricted class of span programs. A new upper bound on the quantum query complexity of triangle finding using the learning graph framework was the inspiration for the nested-checking quantum walk framework. We briefly describe the ideas underlying this learning graph upper bound, as we will apply those same ideas to reproduce this upper bound in Section 5.2.

3.1 Quantum Algorithms Preliminaries

We assume the reader is familiar with the basics of quantum computation. For an introduction, see [KLM06] or [NC00]. In this thesis, we will consider algorithms for search problems, and decision problems, which can be defined as follows.

Definition 3.1.1. *A* search problem*, f, on a finite set Ω, is a collection of sets $\{M_x \subset \Omega\}_{x\in D}$ where D is some finite domain.*

We say that an algorithm $\mathtt{A}$ solves f with bounded error *if, for any $x \in D$, if $M_x \neq \emptyset$,* $\Pr[\mathtt{A}(x) \in M_x] \geq \frac{2}{3}$*, and if $M_x = \emptyset$,* $\Pr[\mathtt{A}(x) = \emptyset] \geq \frac{2}{3}$.

We say that an algorithm $\mathtt{A}$ solves f with one-sided error *if, for any $x \in D$, if $M_x \neq \emptyset$,* $\Pr[\mathtt{A}(x) \in M_x] \geq \frac{2}{3}$*, and if $M_x = \emptyset$,* $\Pr[\mathtt{A}(x) = \emptyset] = 1$.

Definition 3.1.2. *A* decision problem*, $f : D \to \{0, 1\}$, is a function on finite domain D.*

We say that an algorithm $\mathtt{A}$ solves f with bounded error *if, for any $x \in D$,* $\Pr[\mathtt{A}(x) = f(x)] \geq \frac{2}{3}$.

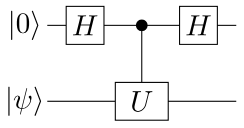
We say that an algorithm $\mathtt{A}$ solves f with one-sided error *if, for any $x \in D$, if $f(x) = 1$,* $\Pr[\mathtt{A}(x) = 1] \geq \frac{2}{3}$*, and if $f(x) = 0$,* $\Pr[\mathtt{A}(x) = 0] = 1$.

In the remainder of this section, we will present some details necessary to understand the work presented in this thesis. In Section 3.1.1, we will describe an important quantum algorithmic primitive called phase estimation, and in Section 3.1.2, we will describe how we deal with subroutines that have some error.

3.1.1 Phase Estimation

An important quantum algorithmic primitive is *phase estimation*, developed by [Kit95] and further explored in Ref. [CEMM98], which is a procedure for estimating the phase of a unitary on a particular state using controlled calls to an implementation of the unitary. For some inuition, fix a unitary U, and a state $|\psi_0\rangle$. Suppose $U|\psi_0\rangle = |\psi_0\rangle$, and for all other eigenstates of U, $|\psi\rangle$, we have $U|\psi\rangle = -|\psi\rangle$, so U is simply $\mathrm{Ref}\{|\psi_0\rangle\}$, the reflection about $|\psi_0\rangle$. Intuitively, since U

has such different behaviour on $|\psi_0\rangle$ as on any orthogonal state, U can allow us to "detect" $|\psi_0\rangle$, distinguishing it from all orthogonal states. The following circuit accomplishes this "detection" of $|\psi_0\rangle$, where H is a Hadamard gate:



This circuit acts as

$$|0\rangle|\psi\rangle \overset{H}{\mapsto} \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|\psi\rangle \overset{U}{\mapsto} \frac{1}{\sqrt{2}}(|0\rangle|\psi\rangle + |1\rangle U|\psi\rangle).$$

If the input state $|\psi\rangle$ is equal to $|\psi_0\rangle$, then U has no effect, and the final Hadamard acts as $(|0\rangle|\psi_0\rangle + |1\rangle|\psi_0\rangle)/\sqrt{2} \mapsto |0\rangle|\psi_0\rangle$. On the other hand, if $|\psi\rangle$ is orthogonal to $|\psi_0\rangle$, then U adds a -1-phase, and so the final Hadamard acts as $(|0\rangle|\psi\rangle - |1\rangle|\psi\rangle)/\sqrt{2} \mapsto |1\rangle|\psi\rangle$. For a more general state $|\phi\rangle = \alpha|\psi_0\rangle + \beta|\psi_0^\perp\rangle$, with $|\psi_0^\perp\rangle$ orthogonal to $|\psi_0\rangle$, the circuit acts as $|0\rangle|\phi\rangle \mapsto \alpha|0\rangle|\psi_0\rangle + \beta|1\rangle|\psi_0^\perp\rangle$. Essentially, the circuit "marks" the part of the state that is orthogonal to $|\psi_0\rangle$.

More generally, suppose U is not necessarily a reflection, but rather has many eigenvalues $e^{i\theta}$ for $\theta \in (-\pi, \pi]$. If $|\psi_0\rangle$ is the *only* 1-eigenstate of U, then U still distinguishes $|\psi_0\rangle$ from orthogonal vectors, but possibly not as well, since if $U|\psi_\theta\rangle = e^{i\theta}|\psi_\theta\rangle$ for some θ very close to 0, then the action of U on $|\psi_\theta\rangle$ is not that different from its action on $|\psi_0\rangle$. However, for any positive integer k, we have $U^k|\psi_\theta\rangle = e^{ik\theta}|\psi_\theta\rangle$. If $k\theta$ is close to π, then the unitary U^k does distinguish $|\psi_0\rangle$ from $|\psi_\theta\rangle$, so we can use U^k in the above circuit to distinguish these states. The smaller θ, the larger we need to set k so that U^k almost reflects $|\psi_\theta\rangle$. We can turn this intuition into an algorithm that distinguishes states based on their phase with respect to some unitary U. The closer together the phases, the more times we will need to use U to accurately distinguish the states. The following Theorem describes such an algorithm.

Theorem 3.1.3 (Phase Estimation [CEMM98]). *Let* $\mathtt{U}$ *be a quantum algorithm that implements some* 2^m*-dimensional unitary* U*. Then for any* $s \geq 1$*, there exists a quantum algorithm* $\mathtt{Phase}^{\mathtt{U}}_{1/2^s}$ *such that*

1. $\mathtt{Phase}^{\mathtt{U}}_{1/2^s}$ *acts on* $m + s$ *qubits;*

2. $\mathtt{Phase}^{\mathtt{U}}_{1/2^s}$ *consists of* $O(s^2)$ *elementary gates, and* 2^{s+1} *controlled calls to* $\mathtt{U}$*;*

3. *if* $U|\psi\rangle = |\psi\rangle$ *then* $\mathtt{Phase}^{\mathtt{U}}_{1/2^s}(|\psi\rangle|0^s\rangle) = |\psi\rangle|0^s\rangle$*;*

4. *and if* $U|\psi\rangle = e^{i\theta}|\psi\rangle$*, for* $\theta \in (-\pi, 0) \cup (0, \pi]$*, then* $\mathtt{Phase}^{\mathtt{U}}_{1/2^s}(|\psi\rangle|0^s\rangle) = |\psi\rangle|\omega\rangle$ *for some state* $|\omega\rangle$ *such that* $|\langle 0^s|\omega\rangle| = \frac{\sin(2^{s-1}\theta)}{2^s \sin\frac{\theta}{2}} \leq \frac{\pi}{|\theta| 2^s}$.

We say that $\texttt{Phase}^{\texttt{U}}_{1/2^s}$ estimates the phase of U with precision $\frac{1}{2^s}$.

Item 3 says that if $|\psi\rangle$ is a 1-eigenstate of U, then the state of the auxiliary register of the output is $|0^s\rangle$, and Item 4 says that if $|\psi\rangle$ is an eigenstate of U with a phase that is sufficiently far from 0, the state of the auxiliary register of the output will be far from $|0^s\rangle$, allowing us to distinguish these two cases. What we mean by "sufficiently far from 0" is determined by the precision, $1/2^s$. By Item 4, if $U|\psi\rangle = e^{i\theta}|\psi\rangle$ for θ such that $|\theta| \geq \frac{\pi}{2}\frac{1}{2^s}$, then $|\langle 0^s|\omega\rangle| \leq \frac{1}{2}$.

This algorithmic primitive will be an important subroutine in virtually all of the algorithms presented in this thesis, as it plays a central role in the algorithms underlying the MNRS quantum walk framework, and the span program framework.

3.1.2 Tolerating Errors

We will sometimes want to implement some specific unitary map U, but will instead be able to implement a quantum algorithm that only approximates the functionality described by U. The following definition quantifies this approximation.

Definition 3.1.4 (Approximating a unitary)**.** *Let U be a unitary on a finite-dimensional inner product space H, and let $\tilde{U}$ be another operator on H. For any subspace $H' \subseteq H$, we say that $\tilde{U}$* approximates U with error ϵ on H' *if for all $|\psi\rangle \in H'$ such that $\||\psi\rangle\|^2 = 1$,*

$$\left\|\tilde{U}|\psi\rangle - U|\psi\rangle\right\| \leq \epsilon.$$

We may have a quantum algorithm in which we approximate many unitaries, in which case, we will still want the error to remain small. The following fact is simply an elementary statement about the operator norm of the difference of two unitaries, and gives an upper bound on the total error of an algorithm consisting of multiple approximate unitaries.

Fact 3.1.5. *Let $U_1, \dots, U_t$ be unitaries acting on a finite-dimensional inner product space H, and let $U = U_t U_{t-1} \dots U_1$. For each $j \in [t]$, let $\tilde{U}_j$ be an operator on H such that for all $|\psi\rangle \in H$ with norm 1, $\left\|\tilde{U}_j|\psi\rangle - U_j|\psi\rangle\right\| \leq \epsilon$. Then if $\tilde{U} := \tilde{U}_t\tilde{U}_{t-1}\dots\tilde{U}_1$, for all unit vectors $|\psi\rangle \in H$,*

$$\left\|\tilde{U}|\psi\rangle - U|\psi\rangle\right\| \leq t\epsilon.$$

In order to keep the overall error low when using t unitaries with error, we will need the error of each individual unitary to be less than $\frac{1}{t}$. Given a unitary U with error ϵ, we can often reduce this error at the cost of multiple uses of U. The following Theorem gives one type of unitary for which this is possible.

Theorem 3.1.6 (Error reduction). *Let $H = \text{span}\{|u\rangle : u \in \Omega\}$ for some finite set Ω be an inner product space, and let $M \subset \Omega$. Let* $\mathtt{A}$ *be a quantum algorithm on $H \otimes \mathbb{C}^2$, and $p < 1$ some constant, such that for all $u \in M$, $\mathtt{A}|u\rangle|0\rangle = |u\rangle|\omega\rangle$ with $|\langle 0|\omega\rangle|^2 \leq p$, and for all $u \in \Omega \setminus M$, $|\langle 0|\omega\rangle|^2 \geq 1 - p$. Then for any $\epsilon < \frac{1}{2}$, there exists an algorithm $\mathtt{A}_\epsilon$ such that:*

- *$\mathtt{A}_\epsilon$ uses $O\left(\log \frac{1}{\epsilon}\right)$ applications of $\mathtt{A}$;*
- *for all $u \in M$, there exists $|\omega\rangle$ such that $\mathtt{A}_\epsilon|u\rangle|0\rangle = |u\rangle|\omega\rangle$ with $\langle 0|\omega\rangle \leq \epsilon$ and;*
- *for all $u \notin M$, there exists $|\omega\rangle$ such that $\mathtt{A}_\epsilon|u\rangle|0\rangle = |u\rangle|\omega\rangle$ with $\langle 0|\omega\rangle \geq 1 - \epsilon$.*

The error reduction procedure uses the standard classical technique of computing $\mathtt{A}|u\rangle|0\rangle$ for many different auxiliary registers $|0\rangle$, and taking the majority. One application of this error reduction technique is to reduce the error in the phase estimation algorithm of Theorem 3.1.3:

Theorem 3.1.7 (Phase Estimation with Small Error). *Let* $\mathtt{U}$ *be a quantum algorithm that implements some 2^m-dimensional unitary U. Then for any $s \geq 1$ and $\varepsilon \in (0, 1/3)$, there exists a quantum algorithm $\mathtt{Phase}^{\mathtt{U}}_{1/2^s,\varepsilon}$ such that*

- *$\mathtt{Phase}^{\mathtt{U}}_{1/2^s,\varepsilon}$ acts on $m + s + O(1/\varepsilon)$ qubits;*
- *$\mathtt{Phase}^{\mathtt{U}}_{1/2^s,\varepsilon}$ consists of $O(s^2 \log \frac{1}{\varepsilon})$ elementary gates and $O(2^s \log \frac{1}{\varepsilon})$ controlled calls to $\mathtt{U}$;*
- *if $U|\psi\rangle = |\psi\rangle$ then $\mathtt{Phase}^{\mathtt{U}}_{1/2^s,\varepsilon}(|\psi\rangle|0\rangle) - |\psi\rangle|0\rangle$;*
- *and if $U|\psi\rangle = e^{i\theta}|\psi\rangle$, for $\theta \in (-\pi, 0) \cup (0, \pi]$, then $\mathtt{Phase}^{\mathtt{U}}_{1/2^s,\varepsilon}(|\psi\rangle|0\rangle) = |\psi\rangle|\omega\rangle$ for some state ω such that $|\langle 0|\omega\rangle|^2 \leq \varepsilon$.*

We say that $\mathtt{Phase}^{\mathtt{U}}_{1/2^s,\varepsilon}$ estimates the phase of U with precision $\frac{1}{2^s}$ and error ε.

It is easy to see that if instead of a computation, we have a reflection with some error, we can use phase estimation on the reflection and apply the same majority trick, which gives us the following.

Theorem 3.1.8 (Error reduction for reflections). *Let $H = \text{span}\{|u\rangle : u \in \Omega\}$ for some finite set Ω be an inner product space, and let V be a subspace of H. Let* $\mathtt{A}$ *be a quantum algorithm on H such that* $\mathtt{A}$ *approximates* $\text{Ref}V$ *with error p for some constant $p < 1/2$. Then for any $\epsilon > 0$, there exists an algorithm $\mathtt{A}_\epsilon$ such that:*

- *$\mathtt{A}_\epsilon$ uses $O\left(\log \frac{1}{\epsilon}\right)$ applications of $\mathtt{A}$;*
- *$\mathtt{A}_\epsilon$ approximates $\text{Ref}V$ with error ϵ.*

3.2 Amplitude Amplification

We will now present the first quantum search framework, amplitude amplification. This was developed in [BBHT98] and [BHMT02] as a generalization of Grover's famous quantum search algorithm [Gro96].

For intuition, consider a very simple classical algorithm that searches for a particular "marked" object by sampling from some distribution of objects, checking if the sampled object is one of the marked objects, and repeating until a marked object is found. More concretely, let M be some set of marked objects, and suppose we have the ability to sample from some distribution π on Ω, where Ω is a finite set containing M. Then each time we sample according to π, we have probability $p_M = \sum_{u\in M}\pi(u)$ of drawing a marked object. If we are promised that $p_M \geq \varepsilon$ whenever $M \neq \emptyset$, then with high probability, we expect to find a marked object after at most $\frac{1}{\varepsilon}$ samples. We can formalize this algorithm as follows:

```
GuessAndCheck($x$)

1. Repeat $\frac{1}{\varepsilon}$ times:
    (a) Sample $u$ from $\Omega$ according to $\pi$
    (b) Check if $u \in M_x$
```

If the cost of the procedure to sample according to π is S, and the cost of the procedure to check if $u \in M_x$ is C, then `GuessAndCheck` has cost $\frac{1}{\varepsilon}(\mathsf{S}+\mathsf{C})$. Here cost can mean time complexity, query complexity, or various other means of counting operations.

The quantum analogue of this very general type of classical algorithm is a framework called *Amplitude Amplification* [BHMT02], based on Grover search [Gro96]. This framework allows one to construct a similarly general type of *quantum* algorithm, whose cost is smaller than the corresponding classical algorithm. This framework is very simple to apply, and has been used to obtain quantum algorithms with improved time or query complexity for a number of problems including collision finding [BHT98], and Boolean matrix multiplication [JKM13a]. The framework is stated in the following theorem.

Theorem 3.2.1 (Amplitude Amplification [BHMT02, Gro96]). *For some set Ω, let $\{M_x\}_{x\in D}$ be a search problem. Let $|\pi_x\rangle \in H_\Omega := \mathrm{span}\{|u\rangle : u \in \Omega\}$ be any state. Let S be the cost of constructing the state $|\pi_x\rangle$; C the cost of checking, for any u, if $u \in M_x$; and $\varepsilon > 0$ a lower bound on $\sum_{u\in M_x} |\langle u|\pi_x\rangle|^2$ for all x such that $M_x \neq \emptyset$. Then there is a quantum algorithm that, on input x, finds an element in M_x or reports that none exists, with bounded error, in cost*

$$O\left(\frac{1}{\sqrt{\varepsilon}}(\mathsf{S}+\mathsf{C})\right).$$

We can summarize the framework described in Theorem 3.2.1 in the following table. We have included a data function d that acts on Ω. The idea is that we will always store some information $d(u)$ with the state u, which may help us to implement the procedure $\mathtt{C}$ that checks if a state is marked.

Amplitude Amplification		
Parameters:	Ω	Finite set of objects
(for each $x \in D$)	$M_x \subset \Omega$	Set of marked objects
	$d_x : \Omega \to S_d$	Function from Ω to some finite set S_d
	$\|\pi_x\rangle \in \mathrm{span}\{\|u, d_x(u)\rangle : u \in \Omega\}$	Initial state
Properties:	$\varepsilon \in \mathbb{R}_+$	Lower bound on $p_{M_x} := \sum_{u \in M_x} \|\langle \pi_x \| u, d_x(u)\rangle\|^2$ for all x s.t. $p_{M_x} \neq 0$
Subroutines:	Setup, $\mathtt{S}(x)$, with cost S	$\|0\rangle \mapsto \|\pi_x\rangle$
	Checking, $\mathtt{C}(x)$, with cost C	$\|u, d_x(u)\rangle \mapsto \begin{cases} -\|u, d_x(u)\rangle & \text{if } u \in M_x \\ \|u, d_x(u)\rangle & \text{else} \end{cases}$
Cost:	$\frac{1}{\sqrt{\varepsilon}}(\mathsf{S} + \mathsf{C})$	

In order to apply the amplitude amplification framework, we must choose parameters Ω, $M \subset \Omega$, d a function on Ω, and $|\pi\rangle$ some superposition over states $|u, d(u)\rangle$. We choose Ω and M based on the search problem we want to solve, and $d(u)$ may contain some information that will help us check if $u \in M$. Next we implement setup and checking subroutines with the specified functionalities. This leads directly to a quantum algorithm, with the specified cost, that uses the setup and checking procedures as subroutines.

The quantum algorithm of Theorem 3.2.1, which implements the amplitude amplification framework, begins with the state $|\pi\rangle$, and alternatively applies $\mathrm{Ref}\{|\pi\rangle\}$ and $-\mathrm{Ref} H_M$, where $H_M = \mathrm{span}\{|u, d(u)\rangle : u \in M\}$, on the order of $\frac{1}{\sqrt{\varepsilon}}$ times. We measure the state, to obtain some u, and check if $u \in M$. If so, we return u, and if not, we return $\emptyset$.

We can implement $-\mathrm{Ref} H_M$ using $\mathtt{C}$ in cost C, and $\mathrm{Ref}\{|\pi\rangle\}$ by $\mathtt{S} \cdot \mathrm{Ref}\{|0\rangle\} \cdot \mathtt{S}^\dagger$, in cost $2\mathsf{S}$, so this algorithm certainly has cost $O\left(\frac{1}{\sqrt{\varepsilon}}(\mathsf{S} + \mathsf{C})\right)$.

To see that this algorithm works, first note that if $M = \emptyset$, then clearly applying $G := -\mathrm{Ref}\{|\pi\rangle\}\mathrm{Ref} H_M$ to $|\pi\rangle$ has no effect, but if $M \neq \emptyset$, then one application of G moves $|\pi\rangle$ closer to $\Pi_{H_M}|\pi\rangle$, its projection onto the marked states. More precisely, if $|\psi_M\rangle := \frac{\Pi_{H_M}|\pi\rangle}{\|\Pi_{H_M}|\pi\rangle\|}$ and $|\psi_{\overline{M}}\rangle := \frac{(I - \Pi_{H_M})|\pi\rangle}{\|(I - \Pi_{H_M})|\pi\rangle\|}$, and $a \in [0, \pi)$ is such that $\sin a = \|\Pi_{H_M}|\pi\rangle\|$, so $\sin^2 a = p_M$, we can write:

$$|\pi\rangle = \sin a |\psi_M\rangle + \cos a |\psi_{\overline{M}}\rangle.$$

Define $|\psi_\theta\rangle := \sin\theta|\psi_M\rangle + \cos\theta|\psi_{\overline{M}}\rangle$. It is a simple exercise to show that for any θ, $G|\psi_\theta\rangle = |\psi_{\theta+2a}\rangle$. Thus, we can see that $G^t|\pi\rangle = G^t|\psi_a\rangle = |\psi_{(2t+1)a}\rangle$. To have constant probability of measuring a marked state, we need to choose t such that $(2t+1)a$ is constant, so $t = \Theta(\frac{1}{a})$. For all $a \in (-\pi, \pi)$, we have $a \leq \frac{\pi}{2}\sin a = \frac{\pi}{2}\sqrt{p_M}$, thus, choosing $t = \Theta\left(\frac{1}{\sqrt{p_M}}\right)$ is sufficient. If ε is a lower bound on p_{M_x} for all x such that $M_x \neq \emptyset$, then even if ε is not known, we can still get an algorithm that uses $O\left(\frac{1}{\sqrt{\varepsilon}}\right)$ applications of G, for a total cost of $O\left(\frac{1}{\sqrt{\varepsilon}}(\mathsf{S}+\mathsf{C})\right)$, using standard techniques, in which we begin by estimating ε as $\frac{1}{2}$, and repeatedly halve our estimate and run the algorithm, until a marked item is found.

3.2.1 Amplitude Amplification with Reflection Errors

It is possible to implement amplitude amplification even when one of the two reflections cannot be implemented perfectly with no additional asymptotic cost. In particular, we will make use of the following theorem when discussing the quantum walk search framework in Chapter 4.

Theorem 3.2.2 ([MNRS11], Lemma 1)**.** *Let H_M be a subspace of some finite-dimensional inner product space H, and let $|\pi\rangle \in H$. Let $\mathtt{R}_M$ be a procedure that implements $\mathrm{Ref}\, H_M$ exactly, with cost R_1. For all $\Theta \in (0,1)$, let $\tilde{\mathtt{R}}_\Theta$ be a circuit that can be implemented in cost $\mathsf{R}_2 \log\frac{1}{\Theta}$ such that:*

- *$\tilde{\mathtt{R}}_\Theta(|\pi\rangle|0^s\rangle) = |\pi\rangle|0^s\rangle$, and*
- *for all $|\psi\rangle$ orthogonal to $|\pi\rangle$, $\left\|(\tilde{\mathtt{R}}_\Theta + I)|\psi\rangle\right\| \leq \Theta$.*

Let $p_M = \||\Pi_{H_M}|\pi\rangle\|^2$. Then there exists a quantum algorithm $\mathtt{A}$ with cost $O\left(\frac{1}{\sqrt{p_M}}(\mathsf{R}_1+\mathsf{R}_2)\right)$ such that $\|\Pi_{H_M}\mathtt{A}|\pi, 0^s\rangle\| \geq \frac{2}{3}$.

Note that the condition $\|\Pi_{H_M}\mathtt{A}|\pi, 0^s\rangle\| \geq \frac{2}{3}$ is exactly what we need to solve a search problem with bounded error.

3.3 Quantum Walk Search

Discrete time quantum walks can be traced back to the work of Meyers on cellular automata, [Mey96a, Mey96b], leading up to the work of Watrous [Wat01]. The potential for applications to search problems was uncovered by Shenvi, Kempe, and Whaley, [SKW03], who showed how to simulate Grover's search algorithm using a discrete time quantum walk.

In 2003, Ambainis was able to extend the ideas behind amplitude amplification to get a quantum walk algorithm for the problem of element distinctness with complexity better than the best possible classical algorithm [Amb04]. Later, Szegedy generalized the ideas behind this algorithm to get a general framework for speeding up similar algorithms based on different reversible Markov processes [Sze04]. This framework was later improved by Ref. [MNRS11] to what we now call the *MNRS quantum walk search framework*. In this section we will first describe Szegedy's framework, and then the improved MNRS quantum walk search framework. Finally, we show several applications of the framework. In Section 3.3.3, we present Ambainis's element distinctness algorithm and its generalization to k-distinctness in the MNRS quantum walk search framework, and analyze its quantum query complexity. In Section 3.3.4, we discuss how careful use of quantum data structures lets us upper bound the time complexity of the k-distinctness algorithm. Finally, in Section 3.3.5, we present a quantum walk algorithm for triangle finding, due to [MSS07].

3.3.1 Szegedy's Quantum Walk Search Framework

Consider a classical algorithm based on a Markov process P on Ω, which begins in a state drawn from the stationary distribution of P, and repeatedly samples a new state according to P, checking each time if a state from some marked set $M_x \subseteq \Omega$, has been reached. We can write this algorithm as follows.

```
RandomWalkSearchI($x$)

1. Sample $u \in \Omega$ according to $\pi$
2. Check if $u \in M_x$
3. Repeat $T$ times:
    (a) Sample $v$ according to $P(u, \cdot)$ and set $u$ to $v$
    (b) Check if $u \in M_x$, and if so, return $u$
4. Return "no marked items"
```

We would like this algorithm to find a marked state, if there is one, with high probability, so we need to make T sufficiently large so that we can be fairly confident that we will find a marked state after T steps beginning from the stationary distribution. This is satisfied by setting T at least the hitting time of M, or the expected number of steps before a random walk reaches a state in M, beginning from the stationary distribution. It turns out that if δ is the spectral gap of P, then the quantity $\frac{1}{\delta\pi(M)}$, where $\pi(M) = \sum_{u\in M}\pi(u)$, is an upper bound on the hitting time of M (see, for example, [Sze04]). Thus, if we know a lower bound ε on $\pi(M)$, we can set T to be a constant multiple of $\frac{1}{\delta\varepsilon}$ and expect to find a marked state within this time, with high probability. Thus, if S is the cost of sampling a state $u \in \Omega$, C is the cost of checking if a state is marked, and

U is the cost of sampling from $P(u,\cdot)$ for any $u \in \Omega$, then we can find a marked state with high probability in asymptotic cost $\mathsf{S} + \frac{1}{\varepsilon\delta}(\mathsf{U} + \mathsf{C})$.

Generalizing the ideas from Ambainis's element distinctness algorithm, Szegedy was able to show a quantum analogue of this very generic search algorithm. He showed that for any symmetric Markov process, P, if S is the cost to construct the superposition $|\pi\rangle := \sum_{u\in\Omega}\sqrt{\pi(u)}|u\rangle$, C is the cost to checking if a state is marked, and U is the cost to construct, for any $u \in \Omega$, the superposition $|P(u,\cdot)\rangle := \sum_{v\in\Omega}|v\rangle$, then we can find a marked state with high probability in asymptotic cost $\mathsf{S} + \frac{1}{\sqrt{\varepsilon\delta}}(\mathsf{U} + \mathsf{C})$.

3.3.2 The MNRS Quantum Walk Search Framework

In [MNRS11], Magniez, Nayak, Roland and Santha generalized and improved the ideas of Szegedy, extending his results to a broader class of Markov processes, and obtaining an often faster generic algorithm that can be applied to a variety of search problems. To illustrate their algorithm, consider the following very generic classical algorithm for solving a search problem $\{M_x\}_{x\in D}$, where $d_x : \Omega \to S_d$ is some arbitrary *data function*, which may be helpful in determining if $u \in M_x$; P is a reversible ergodic Markov process with stationary distribution π; and τ_{mix} is the mixing time of P.

```
RandomWalkSearchII($x$)

1. Sample $u \in \Omega$ according to $\pi$, and compute $d_x(u)$
2. Check if $u \in M_x$ using $d_x(u)$
3. Repeat $\frac{1}{\varepsilon}$ times:
    (a) Repeat $\tau_{\text{mix}}$ times:
        i. Sample $v$ according to $P(u,\cdot)$, compute $d_x(v)$ and set $u$ to $v$
    (b) Check if $u \in M_x$ using $d_x(u)$, and if so, return $u$
4. Return "no marked items"
```

Recall that τ_{mix}, the mixing time of a random walk, is the minimum number of steps after which a walker is close to the stationary distribution of P, no matter the starting state. Thus, step 3a has the effect of sampling a new state approximately according to π. So `RandomWalkSearchII` is essentially acting as the algorithm `GuessAndCheck` from Section 3.2. It repeatedly samples a state u, and checks if u is marked. Then if $\varepsilon \leq \pi(M_x)$, S is the cost of sampling according to π, C is the cost of deciding if a state u is marked from $u, d_x(u)$, and U is the cost of sampling from

$P(u,\cdot)$, and δ is the spectral gap of P, the algorithm finds a marked state with high probability after $\mathsf{S}+\frac{1}{\varepsilon}\left(\tau_{\mathrm{mix}}\mathsf{U}+\mathsf{C}\right)=O\left(\mathsf{S}+\frac{1}{\varepsilon}\left(\frac{1}{\delta}\mathsf{U}+\mathsf{C}\right)\right)$ operations, since $\tau_{\mathrm{mix}}=O(\frac{1}{\delta})$.

The MNRS framework gives a quantum analogue of this very generic classical algorithm. They show the following.

Theorem 3.3.1 (MNRS Framework [MNRS11])**.** *Let P be a reversible, ergodic Markov process with stationary distribution $|\pi\rangle$ and spectral gap δ. Let d be any function on Ω. Let S be the cost of constructing the state $|\pi\rangle=\sum_{u\in\Omega}\sqrt{\pi(u)}|u,d(u)\rangle$, U the cost of constructing, for any $u\in\Omega$, the state $|P(u,\cdot)\rangle=\sum_{v\in\Omega}\sqrt{P(u,v)}|v,d(v)\rangle$ using $|d(u)\rangle$, and C the cost of deciding if a state u is in M_x using $d(u)$. Let ε be a lower bound on $\pi(M_x)$ for all x such that $M_x\neq\emptyset$. Then the cost of solving the search problem $\{M_x\}_x$ with bounded error is $\mathsf{S}+\frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U}+\mathsf{C}\right)$.*

The resulting framework is summarized in the table below. We note that allowing all parameters, even the set Ω, to depend on the input makes sense when we are solving a search problem that is not identical to $\{M_x\}_x$. For example, in the next section, we will present an algorithm for k-distinctness in which the elements of M_x will be sets that *contain* a k-collision, whereas the problem we are actually trying to solve is finding a k-collision itself.

MNRS Quantum Walk Search Framework		
Parameters: (for all $x\in D$)	Ω_x	Finite set of objects
	$M_x\subset\Omega_x$	Set of marked objects
	$d_x:\Omega_x\to S_d$	Function from Ω_x to some finite set S_d
	P_x	A reversible, ergodic Markov Process on Ω_x
Properties:	$\pi_x:\Omega\to\mathbb{R}$	The stationary distribution of P_x
	$\varepsilon\in\mathbb{R}$	A lower bound on $\pi_x(M_x)$ whenever $M_x\neq\emptyset$
	$\delta\in\mathbb{R}$	The spectral gap of P_x
Subroutines:	Setup, $\mathtt{S}$, with cost S	$|0\rangle\mapsto|\pi\rangle:=\sum_{u\in\Omega}\sqrt{\pi(u)}|u,d(u)\rangle$
	Checking, $\mathtt{C}$, with cost C	$|u,d(u)\rangle\mapsto\begin{cases}-|u,d(u)\rangle & \text{if } u\in M\\ |u,d(u)\rangle & \text{else}\end{cases}$
	Update, $\mathtt{U}$, with cost U	$|u,d(u)\rangle|0\rangle\mapsto|u,d(u)\rangle\sum_{v\in\Omega}\sqrt{P_{u,v}}|v,d(v)\rangle$
Cost:	$\mathsf{S}+\dfrac{1}{\sqrt{\varepsilon}}\left(\dfrac{1}{\sqrt{\delta}}\mathsf{U}+\mathsf{C}\right)$	

In order to apply this framework, we must specify parameters (Ω,M,d,P), determine the associated properties, and implement and analyze subroutines with the required functionalities. This immediately gives an algorithm for solving the associated search problem with bounded error, whose cost can be easily analyzed using the cost formula. For the remainder of this section, we will present example applications of the MNRS quantum walk search framework. The applications presented in Chapters 5 and 6 are based on these examples.

3.3.3 Application to k-Distinctness

In this section, we present a quantum walk algorithm that solves 2-distinctness, also known as element distinctness. This algorithm is due to Ambainis [Amb04], and predates either the MNRS or Szegedy quantum walk search framework. However, the Szegedy framework is a generalization of this algorithm, so it is easily presented in the framework. Element distinctness is the following problem.

Element Distinctness, 2-$\mathbf{Dist}_n$
Input: $x \in [q]^n$
Output: $i, j \in [n]$ such that $i \neq j$ and $x_i = x_j$ or $\emptyset$ if no such pair of indices exists

This problem arises naturally as a subproblem of many classical applications, but its query complexity is also important to the general understanding of quantum query complexity. Grover's search algorithm can solve any search problem on a space of size n in $\sqrt{n}$ without exploiting any structure of the problem, but this is only useful when the problem has no structure to exploit. If we try to solve element distinctness using Grover's algorithm, then since we would be searching the space of pairs $(i, j) \in [n]^2$, Grover's search would cost $\sqrt{n^2} = n$. However, since element distinctness does exhibit some structure, this is not much better than the best classical algorithm, which simply queries everything into a database and sorts it in n queries and $n \log n$ time steps.

It was therefore natural to ask if there is a different quantum algorithm that can exploit some of the structure and does get a speedup over the best classical algorithm. This problem was a natural next step after Grover search, having some amount of structure, but almost as little as possible. The first quantum algorithm for this problem used only amplitude amplification, but in a more clever way, to get an upper bound of $\widetilde{O}\left(n^{3/4}\right)$ quantum time complexity by searching for a set of r indices that contains a collision, for some very large r related to n [BDH+05]. Later Ambainis went beyond amplitude amplification, presenting a quantum walk algorithm that proved the following theorem.

Theorem 3.3.2 (Ambainis [Amb04])**.** *The quantum query complexity of* 2-$\mathbf{Dist}_n$ *is at most* $O(n^{2/3})$*, and the quantum time complexity of* 2-$\mathbf{Dist}_n$ *is at most* $\widetilde{O}(n^{2/3})$*.*

Ambainis's upper bound on the quantum query complexity of element distinctness is known to be tight [AS04, Amb05, Kut05].

One natural generalization of element distinctness is the following family of problems.

k-Distinctness, k-**Dist**$_n$

Input: $x \in [q]^n$
Output: distinct $i_1, \dots, i_k \in [n]$ such that $x_{i_1} = \dots = x_{i_k}$ or $\emptyset$ if no such k-tuple of indices exists

Ambainis also showed upper bounds for k-distinctness for $k > 2$.

Theorem 3.3.3 (Ambainis [Amb04]). *For any $k \geq 2$, the quantum query complexity of k-$\mathbf{Dist}_n$ is at most $O(n^{k/(k+1)})$, and the quantum time complexity of k-$\mathbf{Dist}_n$ is at most $\widetilde{O}(n^{k/(k+1)})$.*

These upper bounds are not tight. The best known lower bound is $\Omega(n^{2/3})$, and follows from the element distinctness lower bound. The query complexity of this problem has subsequently been shown to be $o(n^{3/4})$ for any k, but this was the best known upper bound on the time complexity prior to improvements we will present in Section 6.2.

In the remainder of this section, we will construct quantum walk algorithms that prove Theorems 3.3.2 and 3.3.3 in the MNRS quantum walk search framework. To begin, we will present a very straightforward construction from which query complexity upper bounds will follow immediately. We will subsequently discuss the time complexity of this construction, which is much more technical, and involves careful use of quantum data structures. In all discussions of quantum time complexity, we will mean time complexity in a quantum RAM model, in which we can access a quantum memory of size n in cost $O(\log n)$.

Quantum Walk for k-Distinctness

Parameters Let $\Omega = \binom{[n]}{r}$ be the state space of the walk, for some $r < n$ to be specified later. Each set $S \in \Omega$ will correspond to a set of indices into the input. We will define the marked set as

$$M = \{S : \exists \text{ distinct } i_1, \dots, i_k \in S \text{ such that } x_{i_1} = \dots = x_{i_k}\}.$$

To this end, we define the data

$$d(S) := \{(i, x_i) : i \in S\}.$$

The walk will be on a Johnson graph $J(n, r)$, with vertex set Ω, and edges between two vertices S and S' whenever $|S \cap S'| = r - 1$. This walk is governed by the reversible Markov process

$$P(S, S') = \begin{cases} \frac{1}{n(r-r)} & \text{if } |S \cap S'| = r - 1 \\ 0 & \text{else.} \end{cases}$$

Properties The stationary distribution of P is $\pi(S) = \frac{1}{\binom{n}{r}}$ for all $S \in \Omega$, the uniform distribution on Ω. The proportion of marked states, $\pi(M)$, increases as the number of k-collisions in the input increases, with $\pi(M) = 0$ if and only if there are no k-collisions in the input. Thus, if

$\pi(M) > 0$, then $\pi(M)$ is lowest when there is exactly one k-collision in the input, in which case, all indices of this k-collision must be in S for it to be marked. We can thus compute a lower bound on $\pi(M)$ whenever it is nonzero as:

$$\pi(M) = \frac{|M|}{|\Omega|} \geq \frac{\binom{n-k}{r-k}}{\binom{n}{r}} = \frac{r(r-1)\dots(r-k+1)}{n(n-1)\dots(n-k+1)} =: \varepsilon.$$

Finally, the spectral gap of a random walk on $J(n,r)$ is $\delta \geq \frac{1}{r}$, by Fact 2.2.9.

Setup Subroutine We need to set up the state $|\pi\rangle = \sum_{S\in\Omega} \frac{1}{\sqrt{\binom{n}{r}}}|S, d(S)\rangle$, where we can encode S as a list of sorted indices, and $d(S)$ as a list of sorted elements of the form (i, x_i). Since we only care about query complexity, the encoding is not important. Constructing this state costs $\mathsf{S} = O(r)$ queries, since we must query every $i \in S$ to compute $d(S)$.

Checking Subroutine If a state is marked, we can detect this with no additional queries; we need only scan $d(S)$ for k-collisions. Thus, we can implement the checking subroutine with $\mathsf{C} = 0$ queries.

Update Subroutine To implement the update subroutine,

$$|S, d(S)\rangle|0\rangle \mapsto |S, d(S)\rangle \sum_{S'\in\Omega:|S\cap S'|=r-1} \frac{1}{\sqrt{r(n-r)}}|S', d(S')\rangle,$$

we need to construct $d(S')$ from $d(S)$, which we can do by querying the unique index in $S' \setminus S$. All other query values of indices in S' are known, since they are stored in $d(S)$. Thus $\mathsf{U} = O(1)$.

Now we can compute the query complexity of the resulting quantum algorithm using Theorem 3.3.1:

$$\begin{aligned} \mathsf{S} + \frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C}\right) &= O\left(r + \sqrt{\frac{n^k}{r^k}}\left(\sqrt{r} + 0\right)\right) \\ &= O\left(r + \frac{n^{k/2}}{r^{(k-1)/2}}\right). \end{aligned}$$

This is optimized by using $r = n^{k/(k+1)}$, in which case, we get an algorithm with query complexity $O(n^{k/(k+1)})$, as claimed in Theorems 3.3.2 and 3.3.3. To complete the proof, we need to analyze the time complexity of the subroutines, however, we would not immediately get a matching time upper bound. Consider for example the checking subroutine, which simply inspects $d(S)$ to see if there are k colliding indices. If we store the members (i, x_i) of the set $d(S)$ as a list sorted in x_i, then members of a k-collision would appear together, but we would still have to read through a list of $r = n^{k/(k+1)}$ entries, bring the checking cost up from 0 to r, and completely destroying the quantum speedup. In the next section, we describe a data structure in which we can encode the sets S and $d(S)$ so that the time complexity of this algorithm will match the query complexity up to logarithmic factors.

3.3.4 Quantum Radix Trees for Time-Efficient k-Distinctness

In order to make the k-distinctness algorithm presented in the previous section time-efficient, we need a data structure such that we can perform each of the following operations in worst case time logarithmic in n:

- inserting a new (i, x_i) into the data structure;
- removing some (i, x_i) from the data structure, for a specified i;
- constructing a superposition over all i such that (i, x_i) is in the data structure;
- determining if a particular (i, x_i) is in the data structure; and
- determining if there is a collision $(i, x_i), (j, x_j)$ such that $i \neq j$ and $x_i = x_j$, in the data structure.

Furthermore, in order for quantum interference to occur correctly in the quantum walk, we require that for any S, the encoding of $S, d(S)$ by the data structure is unique. For example, it may not depend on the order in which elements are added to S or previously removed elements.

Such a data structure was defined by Ambainis in Ref. [Amb04] in order to prove Theorem 3.3.2, based on a combination of a hash table and a skip-list. We will use a simpler data structure, called a radix tree, first used in the quantum walk setting in Ref. [BJLM13]. The details of the data structure are not as important as the fact that it satisfies the above properties, however, we describe it here for completeness, as we will also make use of this data structure in Section 6.2.2.

Radix Trees A radix tree is binary rooted tree that stores key-value pairs in which the keys are ℓ-bit strings. Each leaf is labelled by a key, and each edge is labelled by a substring, such that the concatenation of all substrings along the path from the root to the leaf yields the key stored at the leaf. A radix tree storing a set of strings S is the unique such tree in which the labels of the children of any non-leaf node start with different bits. An example of a radix tree appears in Figure 3.1.

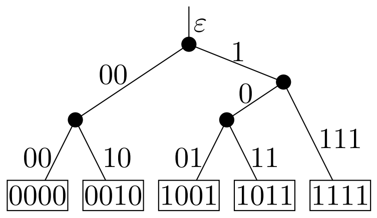


Figure 3.1: The unique radix tree storing $\{0000, 0010, 1001, 1011, 1111\}$.

Let $R(S)$ denote the radix tree storing a set S. We will want to implement reversible operations of the form:

$$\texttt{Insert} : |i\rangle|R(S)\rangle \mapsto |i\rangle|R(S \cup \{i\})\rangle;$$

$$\texttt{Delete} : |i\rangle|R(S)\rangle \mapsto |i\rangle|R(S \setminus \{i\})\rangle;$$

$$\text{and } \texttt{Lookup} : |i\rangle|R(S)\rangle|0\rangle \mapsto |i\rangle|R(S)\rangle|v_i\rangle,$$

where v_i is the value stored with key i if $i \in S$, or 0 if $i \notin S$. The standard radix tree operations can easily be made into reversible operations of this form. We roughly describe each operation on a radix tree R:

Lookup To lookup a string z, we simply start at the root and traverse the tree downwards, each time choosing an edge such that the concatenation of labels along the path taken is a prefix of z. We either find a leaf labelled by z, or we end up at a node with no valid outgoing edge, in which case z is not in R.

Insert(z) Attempt to lookup z in R until we hit a node with no outgoing edge still consistent with z. If it's a leaf, it must already store z (or be a substring of z, which is not permitted when all keys are the same length). Otherwise, there must be an outgoing edge labelled by a substring that shares a non-empty prefix with the remaining substring of z. Split this edge into two edges with a new node in the middle: $e_{old} \mapsto e_1\nu_{new}e_2$. The first edge, e_1 is labelled by the common prefix of z and the old label. The second edge is labelled by the remainder of the old label. The new node's other outgoing edge is labelled by the unmatched part of z.

Delete Run the insertion procedure in reverse.

All of these operations can be done in cost ℓ:

Lemma 3.3.4. *Let R be a radix tree storing r strings of length $\ell = \log n$. We can lookup a string in R in cost at most $O(\log n)$, we can insert a new string into R, or delete a string in R, reversibly, in cost $O(\log n)$.*

Quantum Radix Trees A tree is encoded as a set of nodes, $\{(\nu, p)\}$, in memory, where ν stores the node, and p is some vector of pointers to the memory locations of other nodes, in this case, a parent and two children. In general these nodes can be stored anywhere in memory, but since we want a unique encoding, we cannot allow the decision of where in the memory array we place each node to depend on things like which elements we have added so far, etc. Thus, we will consider the memory array as an array of cells, each large enough to store a single node, and whenever we insert a new element in our data structure, we will choose from all available cells in superposition.

Definition 3.3.5 (Quantum Radix Tree). *Fix positive integers m and ℓ. Let $S \subset \{0,1\}^\ell \times \Sigma$ be a set of key-value pairs of size at most $\frac{m-1}{2}$. Let N be the set of valid tree nodes encoding elements of $\{0,1\}^\ell \times \Sigma$:*

$$N := \{0,1\}^\ell \times \Sigma \times (\{0,1\}^{\leq \ell} \times [m])^3$$

so a node has the form $(z, v, L_1, p_1, L_2, p_2, L_3, p_3)$, with p_1, p_2, p_3 pointers to the parent, left child, and right child, in a memory of m cells, and L_1, L_2, L_3 the labels of the corresponding edges. Define

$$H_N := \mathrm{span}\{|\nu\rangle : \nu \in N \cup \{0\}\}, \quad \textit{and} \quad H := H_N^{\otimes m}.$$

A quantum radix tree *storing S in H is a uniform superposition over all m-tuples $\tau \in (N \cup \{0\})^m$ such that the non-zero cells of τ encode S as a valid radix tree, with the root node in the first entry.*

Augmented Quantum Radix Trees To use this data structure for k-distinctness, we will need to be able to efficiently check if a tree storing items of the form (i, x_i) contains a k-collision, and later we will even want to count the number of k-collisions being stored. In addition, we may even like to exclude k-collisions that fail to satisfy some extra predicate ρ. For example, in Section 6.2, we will partition $[n]$ into $A_1, \dots, A_k$, and only count k-collisions that are in $A_1 \times \cdots \times A_k$. In that case, we will only consider a k-tuple a true k collision if $x_{i_1} = \cdots = x_{i_k}$ and $\rho(i_1, \dots, i_k) = 1$. We augment the data structure to make these operations possible.

In order to be able to count the number of k-collisions being stored, we will augment each node with an extra cell that stores the number of k-collisions in the subtree below it. We will also store, at each node, a counter recording the size of the subtree below it. This will allow us to construct a uniform superposition of all elements stored in the tree. To allow a tree to store two items with the same key, but different value, we will allow a leaf, which necessarily has a unique key, to store a sorted list of up to k values. Figure 3.2 shows an augmented quantum radix tree.

Definition 3.3.6 (k-Augmented Quantum Radix Tree). *Fix positive integers m and ℓ. Let $S \subset \{0,1\}^\ell \times \Sigma$ be a set of key-value pairs of size at most $\frac{m-1}{2}$. Let N be the set of valid tree nodes encoding elements of $\{0,1\}^\ell \times \Sigma$, with up to k distinct values per key:*

$$N := \{0,1\}^\ell \times (\Sigma \cup \{0\})^k \times \{0, \dots, m\} \times (\{0,1\}^{\leq \ell} \times [m])^3.$$

A node has the form $(z, v_1, \ldots, v_k, c, L_1, p_1, L_2, p_2, L_3, p_3)$, where c is a count of the number of leaves in the node's subtree that store k non-zero values; p_1, p_2, p_3 pointers to the parent, left child, and right child, in a memory of m cells; and L_1, L_2, L_3 are the labels of the corresponding edges.

A root node has $L_1 = \emptyset$, root and internal nodes have $v_1 = \cdots = v_k = 0$, and leaves have $p_2 = p_3 = L_2 = L_3 = 0$.

Define

$$H_N := \operatorname{span}\{|\nu\rangle : \nu \in N \cup \{0\}\}, \quad \textit{and} \quad H := H_N^{\otimes m}.$$

An augmented quantum radix tree of size m, storing S *is a uniform superposition over all $(\tau, \mu(\tau))$ such that: $\tau \in (N \cup \{0\})^m$ and the non-zero cells of τ encode S as a valid radix tree, with the root node in the first entry; and $\mu(\tau)$ is a bit map encoding the empty memory cells of τ.*

Lemma 3.3.7. *Let $|R\rangle$ be a k-augmented quantum radix tree of size $m < n$, for some constant k, storing $r < n$ strings of length $\ell = \log n$. Then we can lookup an element in R, insert a new element into R, or delete an element from R in time complexity $O(\log n)$. Furthermore, we can check the number of k-collisions in R in time complexity $O(1)$ and construct a uniform superposition over elements of R or k-collisions in R in time complexity $O(\log n)$.*

Proof. The quantum data structure is just a superposition of classical data structures, so we just need to define classical operations that keep the superposition uniform over all valid orderings of memory. The only other difference is that we need to maintain correct count registers at each node when we insert and delete. Lookup does not change the data structure, so this is clear. For checking the number of k-collisions, it's easy to see that we can simply read the count of the root node in cost $O(1)$, which also does not change the data structure, so this is fine as well.

Constructing a uniform superposition of elements in R can be done by traversing the tree from the root downwards. At any node, we can read the counter of each child to find out how many elements are stored in each sub-tree. If the left-child's counter has value n_0, and the right child's counter has value n_1, then we create the superposition $\sqrt{\frac{n_0}{n_0+n_1}}|0\rangle + \sqrt{\frac{n_1}{n_0+n_1}}|1\rangle$ in an auxiliary register, and use the value in that register to choose the path, so that we take each path with the correct amplitude. Creating a uniform superposition over k-collisions is similar.

Suppose we want to insert a new element (z, v). If there is already a leaf labelled by z, we simply insert v into its list of values, in the appropriate sorted position, in cost $O(\log n)$, since k is constant, $O(\log n)$ is the cost to lookup z (if there are already k values we simply fail to insert (z, v)). If the list previously had $k - 1$ members, and now after adding v has k, then we traverse back up the tree to the root, incrementing the counter of each node we pass, in cost $O(\log n)$.

Suppose there is no leaf labelled by z. Then we proceed as in the standard radix tree insertion, with the following changes:

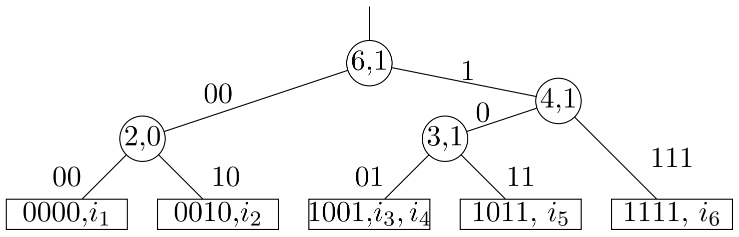


Figure 3.2: An augmented radix tree storing key-value pairs. The key 1001 has two associated values, and represents a 2-collision. Each node has counters recording the number of values stored below, and the number of 2-collisions stored below.

- When we create a new internal node, its counter is set to the counter of its child (it has just one child at first, we add a second one).
- When we create a new leaf node, its counter is set to 0.
- After we have created any new node, we construct a uniform superposition of all addresses of empty memory cells, and then put the new node in the cell specified by this register and update the bitmap keeping track of free memory cells to indicate that this cell is now occupied.

We create at most $O(1)$ new nodes, so the cost of inserting is still $O(\log n)$. Since we always choose from the available memory positions in uniform superposition, we always maintain a uniform superposition over possible memory configurations.

Finally, in order to delete an index, we run the insertion procedure in reverse. □

Time-Efficient k-Distinctness Using a quantum radix tree to store S, with indices being both key and value, and a k-augmented quantum radix tree to store $d(S)$, with i being the value and x_i the key of (i, x_i), Ambainis's algorithm can be made time-efficient. The setup now requires r insert operations to S and r insert operations to $d(S)$, costing $\mathsf{S} = O(r(\log n + \log q))$. The update now requires an insert and a delete in each of S and $d(S)$, to obtain S' and $d(S')$ for S' such that $|S \cap S'| = r - 1$, costing $\mathsf{U} = O(\log n + \log q)$. Finally, the checking procedure can be done in cost $\mathsf{C} = O(1)$. By Theorem 3.3.1, this gives complexity:

$$\begin{aligned} \mathsf{S} + \frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C}\right) &= O\left(\left(r + \sqrt{\frac{n^k}{r^k}}\left(\sqrt{r}+1\right)\left(\log n + \log q\right)\right)\right) \\ &= O\left(\left(r + \frac{n^{k/2}}{r^{(k-1)/2}}\right)\left(\log q + \log n\right)\right). \end{aligned}$$

This is optimized by using $r = n^{k/(k+1)}$, in which case, we get an algorithm with time complexity $O(n^{k/(k+1)}(\log n + \log q))$.

3.3.5 Application to Triangle Finding

Another important problem in quantum query complex is triangle finding, in which we want to find a triangle in an input graph, or decide the graph is triangle-free.

Triangle Finding, **Triangle**$_n$

Input: The adjacency matrix of an undirected graph $G \in \{0,1\}^{n\times n}$
Output: $i, j, k \in [n]$ such that $|\{i, j, k\}| = 3$ and $G_{i,j} = G_{j,k} = G_{i,k} = 1$, or $\emptyset$ if no such triple of vertices exists

Although upper bounds on the query complexity of triangle finding do not yield practical applications, it is of interest in the general study of quantum query complexity as being a problem with slightly more structure than element distinctness. It was therefore a natural candidate for a quantum walk algorithm. Using the quantum walk search framework, Magniez, Santha and Szegedy obtained the following upper bound on the quantum query complexity of triangle finding.

Theorem 3.3.8 ([MSS07]). *The quantum query complexity of* **Triangle**$_n$ *is at most* $\widetilde{O}(n^{1.3})$.

This beats the classical query complexity of triangle finding of $\Theta(n^2)$. The true quantum query complexity of triangle finding is still unknown; although we will present improved upper bounds in subsequent chapters, none matches the best known lower bound of $\Omega(n)$.

In the remainder of this section we will prove Theorem 3.3.8 by constructing a quantum algorithm in the MNRS quantum walk search framework.

Parameters Let $\Omega = \binom{[n]}{r}$, as in the algorithm for k-**Dist**. This time, a set $S \in \Omega$ will correspond to a subset of the vertices of the input graph. We will define the marked set as

$$M := \{S \in \Omega : \exists i, j \in S \text{ such that } G_{i,j} = 1 \text{ and } \exists k \in [n] \text{ such that } G_{i,k} = G_{j,k} = 1\}.$$

In other words, a set S is marked if it contains two vertices that are part of a triangle. With each set S, we will store

$$d(S) := \{(i, j, G_{i,j}) : i < j \text{ and } i, j \in S\}.$$

That is, we will store the full subgraph induced by S, denoted G^S. Once again, P will be defined by the random walk on the Johnson graph $J(n, r)$.

Properties As before, π is the uniform distribution over Ω, and $\delta = \Omega(1/r)$. The marked set is non-empty if and only if the input graph G contains a triangle. In that case, let $\{i, j, k\}$ be a triangle in G. Then the probability that S is in M is at least the probability that $i, j \in M$, so

$$\pi(M) = \frac{|M|}{|\Omega|} \geq \frac{\binom{n-2}{r-2}}{\binom{n}{r}} = \frac{r(r-1)}{n(n-1)}.$$

Thus $\varepsilon := \frac{r(r-1)}{n(n-1)}$ is a lower bound on $\pi(M)$ for all non-empty M.

Setup Subroutine To construct the state $|\pi\rangle = \sum_{S\in\Omega:|S|=r} \binom{n}{r}^{-1/2}|S\rangle|d(S)\rangle$, we must query $G_{i,j}$ for every pair of $i, j \in S$. This costs $\mathsf{S} = \binom{r}{2}$ queries.

Checking Subroutine In order to check if a set S contains two vertices i and j that are part of a triangle, we must search for a third vertex k, such that i, j, k is a triangle in the input graph. To check if a state S contains two vertices that form a triangle with some particular k, we will use a quantum algorithm, which we will call `GraphCollision`, after the problem this algorithm was originally designed to solve. The following theorem describes this algorithm.

Theorem 3.3.9 (Graph Collision Algorithm)**.** *Let G be a graph with vertex set $[n]$. Let K be a known subgraph of G with r vertices, and $k \in [n]$. Then there exists an algorithm, $\texttt{GraphCollision}_{K,k}$, that decides if there exists an edge $\{i, j\}$ in K such that $\{i, j, k\}$ is a triangle in G, with bounded error, using $O\left(r^{2/3}\right)$ quantum queries.*

Proof. We will accomplish this task using a quantum walk almost identical to the one for element distinctness presented in Section 3.3.3. We will have P a walk on $J(r, m)$ for $m < r$, $\Omega = \{T \subset V(K) : |T| = m\}$, as in the element distinctness walk, but the marked set will be defined:

$$M := \{T \in \Omega : \exists i, j \in T : K_{i,j} = G_{i,k} = G_{j,k} = 1\},$$

and the data will be:

$$d(T) := \{(i, G_{i,k}) : i \in T\}.$$

To construct $d(T)$ initially costs $\mathsf{S} = m$, and to update $d(T)$ after replacing one index costs $\mathsf{U} = 2$. To check if $d(T)$ is marked costs 0 queries, since for all edges $\{i, j\}$ of K such that $i, j \in R$, $G_{i,k}$ and $G_{j,k}$ are part of $d(T)$, and $K_{i,j}$ is known. Finally, just as in the element distinctness walk, we have $\delta = \frac{1}{m}$, and $\varepsilon \geq \frac{m^2}{r^2}$, so we have query complexity:

$$\mathsf{S} + \frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C}\right) = m + \frac{r}{m}\left(\sqrt{m} + 0\right) = m + \frac{r}{\sqrt{m}}.$$

Plugging in the optimal value of $m = r^{2/3}$ gives query complexity $O\left(r^{2/3}\right)$. $\square$

We can check if S contains two endpoints of a triangle by searching for $k \in [n]$ such that $\texttt{GraphCollision}_{G^S,k}$ returns true, using $\sqrt{n}$ calls to $\texttt{GraphCollision}_{G^S,k}$, each of which costs $O\left(r^{2/3}\right)$ queries, since G^S is known. However, since `GraphCollision` is only correct with bounded error, and we will call it a number of times polynomial in n, we must reduce the error to inverse polynomial to keep the overall error small, using Theorem 3.1.6, at the expense of a multiplicative factor of $O(\log n)$. The total cost of the checking subroutine becomes $\widetilde{O}\left(\sqrt{n}r^{2/3}\right)$.

Update Subroutine The update step requires us to implement the map

$$|S, d(S)\rangle \mapsto |S, d(S)\rangle \sum_{S' \in \Omega} \sqrt{P(S,S')}|S', d(S')\rangle$$

for any $S \in \Omega$. Since $P(S,S')$ is only nonzero when $|S \cap S'| = r-1$, there is exactly one vertex in each of $S' \setminus S = \{i'\}$ and $S \setminus S' = \{i\}$. To compute $d(S')$ from $d(S)$, we must query $G_{i',j}$ for all $j \in S$, and unquery $G_{i,j}$ for all $j \in S$. This costs $\mathsf{U} = O(r)$ queries to G.

Final Analysis Putting all of these costs together, and applying Theorem 3.3.1, we have the following upper bound on the quantum query complexity of triangle finding:

$$\begin{aligned} \mathsf{S} + \frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C}\right) &= r^2 + \frac{n}{r}\left(\sqrt{r}r + \widetilde{O}(\sqrt{n}r^{2/3})\right) \\ &= \widetilde{O}\left(r^2 + n\sqrt{r} + \frac{n^{3/2}}{r^{1/3}}\right). \end{aligned}$$

Plugging in the optimal value of $r = n^{3/5}$ gives an upper bound of $\widetilde{O}(n^{1.3})$, as claimed in Theorem 3.3.8.

3.4 Span Programs

The span program framework does not easily yield upper bounds in quantum time complexity, however in the area of quantum query complexity, it is of incredible theoretical importance, since Reichardt proved that there is an optimal span program for every decision problem over a Boolean alphabet [Rei11, Rei09], and used this result to show that the quantum adversary lower bound, a method of lower bounding the quantum query complexity, can always give an optimal lower bound. The span program framework is outlined in the following table.

Span Program Framework		
Parameters:	$\{H_{j,b}\}_{j\in[n],b\in\{0,1\}}, H_{\text{true}}, H_{\text{false}}$	Finite-dimensional inner product spaces
	V	A vector space
	$\tau \in V$	The target vector
	$A \in \mathcal{L}(\bigoplus_{j\in[n],b\in\{0,1\}} H_{j,b} \oplus H_{\text{true}} \oplus H_{\text{false}}, V)$	A linear operator
Properties:	$H(x) = \bigoplus_{j\in[n]} H_{j,x_j} \oplus H_{\text{true}}$	The space of x
	$\text{wsize}_+(x) = \min_{\|w\rangle\in H(x):A\|w\rangle=\tau} \\|\|w\rangle\\|^2$	Positive witness size of x
	$\text{wsize}_-(x) = \min_{\omega\in\mathcal{L}(V,\mathbb{C}):\omega(\tau)=1,\omega A\Pi_{H(x)}=0} \\|\omega A\\|^2$	Negative witness size of x
	$W_+ = \max_{x:\text{wsize}_+(x)\neq\infty} \text{wsize}_+(x)$	Positive complexity
	$W_- = \max_{x:\text{wsize}_-(x)\neq\infty} \text{wsize}_-(x)$	Negative complexity
Output:	$\begin{cases} 1 & \text{if } \exists\|w\rangle \in H(x) \text{ s.t. } A\|w\rangle = \tau \\ 0 & \text{else} \end{cases}$	
Query Complexity:	$\sqrt{W_+W_-}$	

We will go into greater detail on this framework in Chapter 7. In brief, a span program is said to "accept" an input x if the pre-image of τ under A intersects $H(x)$, and it is said to "reject" x otherwise. If a span program accepts x, then there is some vector $|w\rangle \in H(x)$ such that $A|w\rangle = \tau$, and such a vector is called a positive witness. Otherwise, there must be some linear function $\omega : V \to \mathbb{C}$ such that $\omega\tau = 1$, and $\omega A \Pi_{H(x)} = 0$.

The algorithm that implements the span program framework involves two reflections, which we describe in more detail in Section 7.6. One of these reflections depends only on the span program, and not the input, so it can be implemented with query complexity 0, however, since no implementation for this reflection is actually given by the framework, the framework does not yield time upper bounds. In some cases, the reflections that result from the span program have enough structure that they can be implemented time-efficiently, for example [BR12], however, in other cases, analyzing these reflections is not feasible. One example is an upper bound on the quantum query complexity of k-distinctness proven by Belovs [Bel12a]. He improves Ambainis's query upper bound of $O\left(n^{k/(k+1)}\right)$, outlined in Section 3.3.3, to $o(n^{3/4})$ for any k, using span programs. However, there is no known way to implement the reflections in the resulting algorithm, so the best known time complexity was still Ambainis's $\widetilde{O}\left(n^{k/(k+1)}\right)$ prior to the improvements presented in Section 6.2.

3.5 The Learning Graph Framework

Span programs are sufficiently general that it is very difficult to construct an optimal span program. With this motivation, in 2011, Belovs developed a restricted class of span programs

[Bel12b], which can be elegantly described by objects called learning graphs.

A learning graph is a weighted graph on vertex set $2^{[n]}$, and edge set $E = \{\{S, S'\} : S \subset S', |S' \setminus S| = 1\}$. In some sense, a learning graph algorithm can be viewed as a quantum analogue of the most general kind of random walk query algorithm, in which at every step, we randomly query a new index or unquery an already queried value, according to some specified distribution, until we find a 1-certificate for the function being decided.

We want to be sure that we find a 1-certificate for x, if one exists, so we must choose t sufficiently large. Let $M = \{S \subseteq [n] : S \text{ contains a certificate for } x\}$. The expected time before we reach a state in M, if M is non-empty, is captured by the *hitting time* from $\emptyset$ (our natural starting state) to M, $H(\emptyset, M)$. Thus, for any choice of edge weights, by making $t = \Theta(H(\emptyset, M))$ steps on the graph (which depends on the choice of weights), we get a classical algorithm that decides f with one-sided bounded error in $\Theta(H(\emptyset, M))$ queries.

We now define the learning graph framework. In order to get an algorithm from a learning graph, we construct a span program (or a dual adversary solution), which we can then "evaluate", in the manner outlined in Section 7.6. The result of this conversion is that the final algorithm is generally not easy to analyze in any model other than the query model, as is the case with span programs.

Learning Graph Framework		
Parameters:	$f : [q]^n \to \{0,1\}$	A decision problem
(for all $x \in [q]^n$)	$M_x \subseteq 2^{[n]}$	A set of 1-certificates for x w.r.t. f
	$w_x : \overrightarrow{E} \to \mathbb{R}_{\geq 0}$	A weight function on the edge set, $\overrightarrow{E} := \{(S, S') : S \subset S' \subset [n], \lvert S' \setminus S\rvert = 1\}$
Properties:	$W_+ := \max_{x \in F_1} \min_p \sum_{e \in \overrightarrow{E}} \frac{p(e)^2}{w_x(e)}$	The *positive complexity*, where p ranges over all unit flows from $\emptyset$ to M_x
	$W_- := \max_{x \in F_0} \sum_{e \in \overrightarrow{E}} w_x(e)$	The *negative complexity*
Output:	$f(x)$	
Query Complexity:	$\sqrt{W_+ W_-}$	

The query complexity upper bound yielded by a learning graph may appear strange, but it can be understood to be quite natural by Theorem 3.5.1, which comes from the beautiful theory of electrical networks, and can be found, for example, in [DS84]. Before stating the theorem, recall that for vertices s and t, $C(s, t)$ denotes the *commute time from s to t*; the expected time to hit t, when starting from s, and return to t.

Theorem 3.5.1. *Let $G = (V, E)$ be a weighted graph with weight function $w : E \to \mathbb{R}_{\geq 0}$. Then*

$$C(s,t) = \min_p \sum_{e \in \overrightarrow{E}} \frac{p(e)^2}{w(e)} \sum_{e \in \overrightarrow{E}} w(e),$$

where p ranges over all unit flows with source s and sink t.

Thus, the learning graph framework gives us an upper bound that is the squareroot of the commute time from the starting state to the marked set.

3.5.1 Application to Triangle Finding

Belovs demonstrated the power of the learning graph framework by improving upon the best known upper bound on the query complexity of triangle finding, proving the following theorem.

Theorem 3.5.2 ([Bel12b])**.** *The quantum query complexity of triangle finding is at most $O(n^{35/27})$.*

His construction had a similar combinatorial structure to the quantum walk algorithm for triangle finding presented in Section 3.3.5, in the sense that his learning graph had paths from $\emptyset$ to sets representing sets S of r vertices. However, he had an additional technique, of edge sparsification. He did not query every pair of vertices in a set S, but rather some subset of m of them.

To illustrate this idea, rather than presenting Belovs' construction, let us try to apply it to the quantum walk algorithm of [MSS07], presented in Section 3.3.5. Recall that this quantum walk search algorithm walks on sets $S \subset [n]$ of size r, representing subsets of the vertices of G, and keeps data $d(S) = \{(i, j, G_{i,j}) : i, j \in S\}$, which encodes the subgraph of G induced by S. The idea of edge sparsification is to use some sparsification parameter $s \in (0, 1]$, and only query sr^2 of the edges.

Note that this reduces the setup cost to $\mathsf{S} = sr^2$, from r^2, and the update cost to $\mathsf{U} = sr$, from r, (under some assumptions about average costs) since when we replace a vertex in S, it is incident to sr edges on average that must be requeried. However, a state is only marked if it contains a triangle edge (we cannot detect a state that has two triangle vertices but does not encode the corresponding edge), and this probability has decreased by a factor of s, so we now have $\varepsilon = s\frac{r^2}{n^2}$ compared with $\frac{r^2}{n^2}$ previously. Since we still have $\delta \geq 1/r$ and $\mathsf{C} = \sqrt{n}r^{2/3}$, we can compute the total cost as:

$$\mathsf{S} + \frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{C}\right) = sr^2 + \frac{n}{r\sqrt{s}}\sqrt{r}sr + \frac{n}{r\sqrt{s}}\sqrt{n}r^{2/3} = sr^2 + n\sqrt{rs} + \frac{n^{3/2}}{r^{1/3}\sqrt{s}}.$$

This is optimized by setting $s = 1$ and $r = n^{3/5}$, yielding no improvement.

It would appear then that the learning graph framework is somehow more powerful than the quantum walk search framework, however, in Chapter 4 we give a very small modification of the framework that allows us to gain a speedup using the edge sparsification technique in a quantum walk algorithm, which we present in Section 5.2.

A subsequent improvement to the triangle finding upper bound also used learning graphs:

Theorem 3.5.3 ([LMS13])**.** *The quantum query complexity of triangle finding is at most $O(n^{9/7})$.*

Their learning graph construction does not use the edge sparsification idea, but rather, looks for an edge between two vertex sets of unbalanced sizes. We also reproduce this upper bound as a quantum walk, in Section 5.3.

At the time that this thesis was submitted, this was the best known upper bound for triangle finding, however, Le Gall has since improved this to $\widetilde{O}(n^{5/4})$ using the MNRS quantum walk search framework [Le 14].

Part II

Nested Quantum Walks

Chapter 4

Extension of the Quantum Walk Search Framework

The main task in designing a quantum walk algorithm in the MNRS quantum walk search framework is the implementation and analysis of the setup, update and checking operations. The functionality that must be implemented for each task depends on the chosen parameters of the walk, $\mathcal{W} = (\Omega, M, P, d)$. The prescribed functionality may be trivial to implement, as in the checking step of the algorithm presented in Section 3.3.3 for element distinctness, in which we need only examine the data of a state to determine if it is marked. On the other hand, the prescribed functionalities of these operations may require nontrivial quantum algorithms to implement, as in the checking step for the quantum walk algorithm for triangle finding, presented in Section 3.3.5, in which a quantum walk search algorithm based on graph collision is used to implement the checking step.

In this chapter, we will give a slightly more general version of the quantum walk search framework, that will allow us to implement several useful algorithmic techniques, including *nested checking*, in which we efficiently use a quantum walk search algorithm as the subroutine in the checking step of another, described in Section 5.1; and *nested updates*, in which we efficiently use a quantum walk search algorithm to implement the update step of another, described in Section 6.1. The nested update technique relies on a special case of a technique we can implement in our new framework — updates with garbage. Although efficient nested updates are the only application of updates with garbage we have discovered thus far, it looks like it could be a useful technique in general.

This chapter, Chapter 5, and Chapter 6 are based on two publications. In the first publication, Ref. [JKM13a], we show how one small modification of the MNRS quantum walk search framework — allowing the data function to encode a quantum state — allows us to implement nested checking. This might not even be called a change to the framework, strictly speaking, as it was not precluded by the MNRS quantum walk search framework, however, it was not explicitly allowed, and it had never been done before. As applications of this technique, we are able to give quantum walk algorithms that reproduce all known learning graph upper bounds up to logarithmic factors. In particular, we present quantum walk algorithms matching the best known upper bounds for triangle finding and subgraph finding, up to logarithmic factors. The nested checking technique and its applications are presented in Chapter 5.

In the second publication, Ref. [CJKM13], we make a further modification to the quantum walk search framework. We now modify the data function by allowing it to depend, not only on a state $u \in \Omega$, but on an edge (u, v). More specifically, for a Markov process P on Ω, define

$$\overrightarrow{E}(P) := \{(u, v) \in \Omega \times \Omega : P(u, v) \neq 0\}\,.$$

Whereas before we had a data function $d : \Omega \to S_d$ for some finite set S_d, we now have a data function $d : \overrightarrow{E} \cup (\Omega \times 0) \to H_d$ for some finite-dimensional inner product space H_d. Since we often call the register encoding v the *coin register*, since v tells us the next step of the walk, we refer to data that depends on not only u, but also v, as *coin-dependent data*. This modification allows us to implement nested updates, and in [CJKM13], we also show how to apply this technique to get new upper bounds on the quantum time complexity of k-$\mathbf{Dist}_n$ for all $k \geq 3$. We present the nested updates technique and its applications in Chapter 6.

In this chapter, we present a new version of the quantum walk framework that includes both the modification from [JKM13a], and the modifications from [CJKM13].

4.1 The New Framework

In this section, we will present our new quantum walk search framework, and present and analyze the algorithm that implements it. Our analysis follows that of the MNRS quantum walk search framework [MNRS11], which in turn is based on [Sze04]. The only parts of our analysis that differ from [MNRS11] are those pertaining to the data, since our data may be a quantum state that depends on the coin register.

A quantum walk search specification specifies a problem, as well as an algorithm for solving this problem, given access to certain subroutines. Given a family $\{\mathcal{W}_x = (\Omega_x, M_x, P^x, d_x)\}_{x \in D}$, for some finite domain D, we consider the problem as having input x and the output should be some element of the *marked set* M_x, or "$M_x = \emptyset$" if there are no marked states. The subroutines that need to be implemented in order to solve this problem will all take x as input, and the functionality they must implement will depend on the parameters of $\mathcal{W}_x$. We will often not make the dependence on x explicit, and instead write $\mathcal{W} = (\Omega, M, P, d)$.

We will first discuss the parameters that specify the quantum walk search algorithm, $\mathcal{W} = (\Omega, M, P, d)$, and the inner product space on which the algorithm acts. Next, we will define the functionalities of subroutines that our final algorithm will make use of. We will then construct the full algorithm from these. The complexity of this algorithm will depend on the complexity of the various subroutines it calls. Because these subroutines dominate the complexity of the algorithm, we are fairly free in choosing whatever notion of complexity we are interested in, be it quantum time complexity, quantum query complexity, or some other measure — we must simply analyze the subroutines with respect to this measure of complexity. However, in the interest of rigor, we will only make claims for quantum time complexity, and quantum query complexity. We will refer to the *cost* of a subroutine, by which we mean, the reader's choice of quantum time complexity, or quantum query complexity. Our statements will hold for many other measures of complexity of quantum algorithms, and should be simple to verify, for any particular measure.

The algorithm is very similar to amplitude amplification. We first construct an initial state, $|\pi\rangle$ by some *setup procedure* $\texttt{S}$. In amplitude amplification, we would then alternate two reflections: a checking reflection, $\texttt{C}$, and a reflection about $|\pi\rangle$, which is implemented by reversing the setup to get $|\pi\rangle \mapsto |0\rangle$, reflecting about $|0\rangle$, and then performing the setup, $|0\rangle \mapsto |\pi\rangle$. We will also have a checking reflection, however, we will simulate a reflection about $|\pi\rangle$ in a different manner. We will show how to implement a *walk operator* $U(\mathcal{W})$, using an assumed procedure $\texttt{U}$. This walk operator will have $|\pi\rangle$ as a unique 1-eigenvector, and its other eigenvalues, $e^{i\theta}$, will have θ not too small, so we will be able to distinguish $|\pi\rangle$ from orthogonal vectors in order to reflect about $|\pi\rangle$ by applying phase estimation on $U(\mathcal{W})$.

4.1.1 The Parameters and Workspace

Parameters A quantum walk search problem, $\mathcal{W}$, is specified by:

- a *state space*, Ω, which is a finite set of objects;
- a *marked set*, $M \subseteq \Omega$, which specifies which objects we are searching for;
- a reversible, ergodic Markov process, P, acting on Ω; and
- a data function $d : \overrightarrow{E}(P) \cup (\Omega \times \{0\}) \to H_d$ for some finite-dimensional inner product space H_d.

Each of these objects may depend on some input x ranging over a finite domain D, so the full specification for a quantum walk search problem is given by some family $\{\mathcal{W}_x = (\Omega_x, M_x, P_x, d_x)\}_{x\in D}$. However, it will often be convenient to omit the explicit x subscripts, and simply keep in mind that these objects depend on the input.

We require that P be ergodic so that it has a unique stationary distribution, π. We further require that the discriminant of P have non-zero spectral gap, δ (see Section 2.2). Reversibility is a sufficient condition for this to hold, although it is not a necessary condition.

For any quantum walk search specification $\{\mathcal{W}_x\}_x$, the quantum walk search framework specifies a number of subroutines that must be implemented. Given implementations for these, the quantum walk search framework gives an explicit algorithm for finding a marked element or determining that $M_x = \emptyset$, built from the implemented subroutines.

Workspace The algorithm will work on a space $H_E \otimes H_d$, where $H_E := \mathrm{span}\{|u,v\rangle : (u,v) \in \overrightarrow{E} \cup (\Omega \times \{0\})\}$. It will be convenient to write $H_E = H_\Omega \otimes H_C$, where $H_\Omega := \mathrm{span}\{|u\rangle : u \in \Omega\}$ encodes the current state, and H_C encodes the next state, or coin space (see Section 2.2). We can always set $H_C = H_\Omega$, to encode an edge $|u,v\rangle$ as $|u\rangle|v\rangle$, but this might not always be necessary, or desirable. To encode an edge $|u,v\rangle$ as $|u\rangle_\Omega|r\rangle_C$, it need not be the case that r fully specify v, it need only be the case that (u,r) specifies (u,v) — in other words, r should specify v *given* u, so there must be some fixed function f such that $f(u,r) = v$. For example, if there is some fixed random mapping representation of P, $f : \Omega \times \Lambda \to \Omega$, for a finite set Λ, such that $f(u,r) = v$, then r specifies v, given u. In order to avoid explicitly introducing a random mapping or similar function (in the interest of remaining consistent with historical notation) we will write the coin register as $|v \shortmid u\rangle_C$ when we want to be explicit that the coin register may not fully specify the state, v.

It will be useful to consider a part of the data, d_1, that is *coin-independent*; it depends only on a state, and not on an edge. Explicitly, suppose we can decompose the data function as, for any $(u,v) \in \overrightarrow{E}$:

$$|d(u,v)\rangle = |d_1(u)\rangle|d_2(u,v)\rangle$$

for some functions $d_1 : \Omega \to H_{d_1}$ and $d_2 : \overrightarrow{E} \cup (\Omega \times \{0\}) \to H_{d_2}$. We can always do this in a trivial way by taking $d_2 = d$ and $d_1 = 1$, but it may be possible, and desirable, to do this in a nontrivial way. We can think of the data d_1 as being the "checking data", that will facilitate deciding if a state is marked, and d_2 may have some other purpose. The data d_2 introduces some extra costs, and the data d_1 facilitates the checking procedure, so both of these factors will determine the optimal decomposition of d.

We will often permute the order of the tensor product spaces for convenience of notation. When there is ambiguity, we will specify the spaces with subscripts; for examples: $|u\rangle_\Omega|v \shortmid u\rangle_C$.

We will work in a subspace of $H_\Omega \otimes H_C \otimes H_{d_1} \otimes H_{d_2}$, defined:

$$\overline{H} := \bigoplus_{u\in\Omega} H_u, \qquad \text{where} \qquad H_u := \mathrm{span}\{|u\rangle|d_1(u)\rangle : u \in \Omega\} \otimes H_C \otimes H_{d_2}.$$

4.1.2 The Subroutines

The Setup The initial state in the algorithm will be

$$|\pi\rangle := \sum_{u\in\Omega}\sqrt{\pi(u)}\sum_{v\in\Omega}\sqrt{P(u,v)}|u,v\rangle|d(u,v)\rangle.$$

However, a slightly simpler state

$$|\pi\rangle^0 := \sum_{u\in\Omega}\sqrt{\pi(u)}|u\rangle|d_1(u)\rangle|0\rangle_C|d_2(u,0)\rangle,$$

can be mapped to $|\pi\rangle$ by one application of the update operation $\mathtt{U}$, defined below. Thus, we can consider the setup operation as merely constructing $|\pi\rangle^0$. Thus, our algorithm requires access to a setup subroutine, that acts as:

$$\mathtt{S} : |0\rangle \mapsto |\pi\rangle^0.$$

The *setup cost* of the walk specified by $\mathcal{W}$, $\mathsf{S} = \mathsf{S}(\mathcal{W})$, is the cost of the subroutine $\mathtt{S}$.

The Checking We assume access to a subroutine that reflects about the marked states. The coin-independent part of the data may be used to facilitate this checking. Specifically, the subroutine acts on $\mathrm{span}\{|u,d_1(u)\rangle : u \in \Omega\}$ as:

$$\mathtt{C} : |u,d_1(u)\rangle \mapsto \begin{cases} -|u,d_1(u)\rangle & \text{if } u \in M \\ |u,d_1(u)\rangle & \text{if } u \in \Omega\setminus M. \end{cases}$$

The *checking cost* of the walk specified by $\mathcal{W}$, $\mathsf{C} = \mathsf{C}(\mathcal{W})$, is the cost of the subroutine $\mathtt{C}$.

The Update: Local Diffusion The update operation, $\mathtt{U}$, is analogous to taking a step of the random walk P. Classically, we can think of doing this by drawing the next state from the distribution $P(u,\cdot)$, where u is the current state. The quantum analogue of this distribution is the *coin state of* u, defined as:

$$|P(u,\cdot)\rangle := \sum_{v\in\Omega}\sqrt{P(u,v)}|v \shortmid u\rangle.$$

If we want to include some coin-dependent data, d_2, we write:

$$|P(u,\cdot)_{d_2}\rangle := \sum_{v\in\Omega}\sqrt{P(u,v)}|v \shortmid u\rangle|d_2(u,v)\rangle.$$

For every $u \in \Omega$, let U_u be any operator on the space $H_C \otimes H_{d_2}$ that acts as

$$U_u : |0\rangle_C|d_2(u,0)\rangle_{d_2} \mapsto |P(u,\cdot)_{d_2}\rangle := \sum_{v\in\Omega}\sqrt{P(u,v)}|v \shortmid u\rangle|d_2(u,v)\rangle.$$

We assume access to a subroutine U that acts on $\overline{H}$ as:

$$\sum_{u\in\Omega} |u, d_1(u)\rangle\langle u, d_1(u)| \otimes U_u.$$

In other words, U implements the mapping

$$|u,0\rangle|d_1(u)\rangle|d_2(u,0)\rangle \mapsto \sum_{v\in\Omega} \sqrt{P(u,v)}|u,v\rangle|d_1(u)\rangle|d_2(u,v)\rangle$$

in a way that is *controlled* on $|u, d_1(u)\rangle$; it does not alter this part of the state. This requirement ensures that we remain in $\overline{H}$.

The Update: Data Swap We will also assume access to a subroutine SWAP that acts on the space span $\left\{|u,v\rangle|d(u,v)\rangle : (u,v) \in \overrightarrow{E}\right\} \subseteq \overline{H}$ as:

$$\texttt{SWAP} : |u,v\rangle|d(u,v)\rangle = |u\rangle_\Omega|v \shortmid u\rangle_C|d(u,v)\rangle_d \mapsto |v\rangle_\Omega|u \shortmid v\rangle_C|d(v,u)\rangle_d = |v,u\rangle|d(v,u)\rangle.$$

The update consists of the two subroutines U and SWAP. The *update cost* of the walk specified by $\mathcal{W}$, $\mathsf{U} = \mathsf{U}(\mathcal{W})$, is the maximum of the cost of U and the cost of SWAP.

We note that it will usually be trivial to map: $|u,v\rangle \mapsto |v,u\rangle$ for an edge (u,v) under a reasonable encoding of the edges. The two tasks required to implement the data swap that are non-trivial are the maps

$$|d_2(u,v)\rangle \mapsto |d_2(v,u)\rangle \quad \text{and} \quad |d_1(u)\rangle \mapsto |d_1(v)\rangle$$

for an edge (u,v).

The $(\Omega, 0)$-Phase Flip The final subroutine that we will need access to is specific to our new framework, and is the only substantial difference between the implementation of a quantum walk search algorithm specified in our framework versus one specified in the MNRS framework. We require access to a subroutine Φ that acts on $H_\Omega \otimes H_{d_2}$ as

$$|u\rangle|d_2(u,0)\rangle \mapsto -|u\rangle|d_2(u,0)\rangle$$

and for any $|\psi\rangle \in H_{d_2}$ such that $\langle\psi|d_2(u,0)\rangle = 0$,

$$|u\rangle|\psi\rangle \mapsto |u\rangle|\psi\rangle.$$

This operation essentially checks that the data is correct. We only need to check this for d_2, since d_1 is always correct, as long as we stay in the space $\overline{H}$.

The *phase flip cost* of the walk specified by $\mathcal{W}$, $\mathsf{\Phi} = \mathsf{\Phi}(\mathcal{W})$, is the cost of the subroutine Φ.

4.1.3 The Quantum Walk Algorithm

The Walk Operator We are now ready to define and implement the walk operator. Define two subspaces of $\overline{H}$:

$$A := \text{span}\left\{|\psi_u\rangle := \sum_{v\in\Omega}\sqrt{P(u,v)}|u,v\rangle|d(u,v)\rangle : u \in \Omega\right\},$$

$$\text{and}\quad B := \text{span}\left\{|\phi_v\rangle := \sum_{u\in\Omega}\sqrt{P(v,u)}|u,v\rangle|d(u,v)\rangle : v \in \Omega\right\}.$$

Note that $|\psi_u\rangle = |u\rangle|P(u,\cdot)_d\rangle = |u\rangle|d_1(u)\rangle|P(u,\cdot)_{d_2}\rangle$. To see that these are indeed subspaces of $\overline{H}$, we note that

$$|u,v\rangle|d(u,v)\rangle = |u\rangle|v \shortmid u\rangle|d_1(u)\rangle|d_2(u,v)\rangle.$$

We can now define the *walk operator*:

$$U(\mathcal{W}) := \text{Ref}B \cdot \text{Ref}A.$$

We will show that $U(\mathcal{W})$ can be used to approximate $\text{Ref}\{|\pi\rangle\}$, but first, we consider its implementation.

Theorem 4.1.1. *Let* $\mathtt{R}$ *be an operator on* $\overline{H}$ *that acts as the* $(\Omega, 0)$*-phase flip,* Φ*, on* $H_\Omega \otimes H_{d_2}$*, controlled on the state of* H_C *being 0. Then* $\mathtt{URU}^\dagger$ *acts as* $\text{Ref}A$ *on* $\overline{H}$*.*

Proof. First we note

$$\begin{aligned}(\mathtt{URU}^\dagger)|\psi_u\rangle &= \mathtt{URU}^\dagger(|u\rangle|d_1(u)\rangle|P(u,\cdot)_{d_2}\rangle) = \mathtt{UR}(|u\rangle|d_1(u)\rangle\mathtt{U}_u^\dagger|P(u,\cdot)_{d_2}\rangle)\\ &= \mathtt{UR}(|u\rangle|d_1(u)\rangle|0\rangle_C|d_2(u,0)\rangle) = \mathtt{U}(-|u\rangle|d_1(u)\rangle|0\rangle_C|d_2(u,0)\rangle)\\ &= -|\psi_u\rangle.\end{aligned}$$

Thus, for any $|\psi\rangle \in A$, $(\mathtt{URU}^\dagger)|\psi\rangle = -|\psi\rangle$.

Suppose on the other hand that $|\psi^\perp\rangle = \sum_{u\in\Omega}|u, d_1(u)\rangle|\alpha_u\rangle$, and $|\psi^\perp\rangle \in A^\perp$, so $\langle\psi^\perp|\psi_u\rangle = 0$ for all $u \in \Omega$. We have

$$\begin{aligned}0 = \langle\psi^\perp|\psi_u\rangle = \langle\psi^\perp|(|u\rangle|d_1(u)\rangle|P(u,\cdot)_{d_2}\rangle) &= \langle\alpha_u|P(u,\cdot)_{d_2}\rangle\\ &= \langle\alpha_u|U_uU_u^\dagger|P(u,\cdot)_{d_2}\rangle = \langle\alpha_u|U_u(|0\rangle_C|d_2(u,0)\rangle),\end{aligned}$$

and so $\mathtt{R}|u\rangle U_u^\dagger|\alpha_u\rangle = |u\rangle U_u^\dagger|\alpha_u\rangle$. Thus, we have:

$$\begin{aligned}\mathtt{URU}^\dagger|\psi^\perp\rangle &= \sum_{u\in\Omega}\mathtt{URU}^\dagger(|u\rangle|d_1(u)\rangle|\alpha_u\rangle) = \sum_{u\in\Omega}\mathtt{UR}(|u\rangle|d_1(u)\rangle U_u^\dagger|\alpha_u\rangle)\\ &= \sum_{u\in\Omega}\mathtt{U}(|u\rangle|d_1(u)\rangle U_u^\dagger|\alpha_u\rangle) = \sum_{u\in\Omega}|u\rangle|d_1(u)\rangle|\alpha_u\rangle = |\psi^\perp\rangle.\end{aligned}$$

So for all $|\psi^\perp\rangle \in \overline{H}\cap A^\perp$, $\mathtt{URU}^\dagger|\psi^\perp\rangle = |\psi^\perp\rangle$, completing the proof. □

We can now implement the walk operator $U(\mathcal{W})$, by noticing that A and B are related by the swap operation:

$$\begin{aligned}\mathtt{SWAP}|\psi_u\rangle &= \mathtt{SWAP}\sum_{v\in\Omega}\sqrt{P(u,v)}|u,v\rangle|d(u,v)\rangle\\ &= \sum_{v\in\Omega}\sqrt{P(u,v)}|v,u\rangle|d(v,u)\rangle = |\phi_u\rangle.\end{aligned}$$

We can therefore implement Ref$B$ as $\mathtt{SWAP}\cdot\mathrm{Ref}A\cdot\mathtt{SWAP}$, so we can implement $U(\mathcal{W})$ by the procedure:

$$\mathtt{Walk}_{\mathcal{W}} := \mathtt{U}\cdot\mathtt{R}\cdot\mathtt{U}^\dagger\cdot\mathtt{SWAP}\cdot\mathtt{U}\cdot\mathtt{R}\cdot\mathtt{U}^\dagger\cdot\mathtt{SWAP}.$$

The only small issue is that $\mathtt{U}\cdot\mathtt{R}\cdot\mathtt{U}^\dagger$ is only equal to Ref$A$ on the subspace $\overline{H}$. This is sufficient for our purposes, because we will begin with initial state $|\pi\rangle \in \overline{H}$, and no operation we do will take us out of $\overline{H}$. Thus, we have the following corollary about the cost of implementing $U(\mathcal{W})$.

Corollary 4.1.2. *We can implement $U(\mathcal{W})$ on $\overline{H}$ in cost $O(\mathsf{U}(\mathcal{W})+\mathsf{\Phi}(\mathcal{W}))$.*

Proof. We need only observe that the operator R, can be implemented by a conditional application of $\mathtt{\Phi}$. Thus, the cost of $\mathtt{R}$ is $\Theta(\mathsf{\Phi})$. The result follows. □

Using $U(\mathcal{W})$ to Approximate $\mathtt{Ref}\{|\pi\rangle\}$ We now describe how to use the walk operator, $U(\mathcal{W})$, to approximate the reflection about $|\pi\rangle$. Recall that

$$|\pi\rangle = \sum_{u\in\Omega}\sqrt{\pi(u)}\sum_{v\in\Omega}\sqrt{P(u,v)}|u,v\rangle|d(u,v)\rangle.$$

Since $|\pi\rangle \in A\cap B$, it is a 1-eigenvector of $U(\mathcal{W})$. If we could show that any vector orthogonal to $|\pi\rangle$ is a -1-eigenvector of $U(\mathcal{W})$, or in other words, that $U(\mathcal{W})$ only has ± 1 eigenvalues, this would show that $U(\mathcal{W})$ implements Ref$\{|\pi\rangle\}$ exactly, however, this is generally far from true. Generally, $U(\mathcal{W})$ has eigenvalues $e^{i\theta}$ for many different values θ, some of them close to 0. Thus, it will take multiple applications of $U(\mathcal{W})$ to distinguish $|\pi\rangle$ from its orthogonal vectors. Intuitively, the closer to 0 the next smallest phase, the more difficult it will be to distinguish $|\pi\rangle$ from orthogonal vectors, since some of them are affected by $U(\mathcal{W})$ in a very similar way. We formalize this with the notion of the phase gap.

Definition 4.1.3. *Let U be a unitary operator with a nontrivial 1-eigenspace. Let $\{e^{i\theta_1} = 1, e^{i\theta_2}, \dots, e^{i\theta_r}\}$ be an enumeration of the eigenvalues of U, ordered such that $0 = |\theta_1| \leq |\theta_2| \leq \dots \leq |\theta_r| \leq \pi$. Then the* phase gap *of U is defined $\Delta(U) := |\theta_2|$.*

In order to estimate the number of iterations of $U(\mathcal{W})$ required to distinguish $|\pi\rangle$ from any orthogonal states, we must show that $|\pi\rangle$ is the only 1-eigenvector of $U(\mathcal{W})$, and lower bound the phase gap of $U(\mathcal{W})$.

To facilitate the spectral analysis of $U(\mathcal{W})$, we use the following theorem, derived from Jordan's lemma [Jor75] (or see [Bha93]), first used in this context by Szegedy [Sze04]. The version below is taken from [Chi13].

Theorem 4.1.4. *Let $U := (2\Pi_A - I)(2\Pi_B - I)$ be a unitary on some finite-dimensional inner product space H, where $\Pi_A = \sum_{j=1}^{a} |\psi_j\rangle\langle\psi_j|$ is the orthogonal projector onto some subspace A of H, and $\Pi_B = \sum_{j=1}^{b} |\phi_j\rangle\langle\phi_j|$ is the orthogonal projector onto some subspace B of H. Define the* discriminant *of U, $D \in \mathbb{C}^{a\times b}$ by $D(j,k) = \langle\psi_j|\phi_k\rangle$. Define isometries corresponding to these two projectors:*

$$\Lambda_A = \sum_{j=1}^{a} |\psi_j\rangle\langle j| \in \mathcal{L}(\mathbb{C}^a, A), \quad \text{and } \Lambda_B = \sum_{j=1}^{b} |\phi_j\rangle\langle j| \in \mathcal{L}(\mathbb{C}^b, B).$$

Then we have $D = \Lambda_A^\dagger \Lambda_B$. Let $D = \sum_j \cos\theta_j |\alpha_j\rangle\langle\beta_j|$ for $\theta_j \in [0, \frac{\pi}{2}]$, $|\alpha_j\rangle \in \mathbb{C}^a$, $|\beta_j\rangle \in \mathbb{C}^b$ be the singular value decomposition of D. Then the spectrum of U is completely specified by the following:

1. *For every $\theta_j \in (0, \frac{\pi}{2})$, $\mathrm{span}\{\Lambda_A|\alpha_j\rangle, \Lambda_B|\beta_j\rangle\}$ is invariant under U.*
2. *The eigenvalues of U on $\mathrm{span}\{\Lambda_A|\alpha_j\rangle, \Lambda_B|\beta_j\rangle\}$ are $\{e^{2i\theta_j}, e^{-2i\theta_j}\}$ with eigenvectors $\Lambda_A|\alpha_j\rangle - e^{-i\theta_j}\Lambda_B|\beta_j\rangle$ and $\Lambda_A|\alpha_j\rangle - e^{i\theta_j}\Lambda_B|\beta_j\rangle$ respectively.*
3. *In addition, U has -1-eigenspaces $A \cap B^\perp$ and $A^\perp \cap B$; and 1-eigenspaces $A \cap B$ and $A^\perp \cap B^\perp$. The space $A \cap B$ is spanned by the left and right 1-singular vectors of D.*

We can use Theorem 4.1.4 to analyze the spectrum of $U(\mathcal{W})$. In our case, we have

$$\Lambda_A := \sum_{u\in\Omega} |\psi_u\rangle\langle u|, \quad \text{and} \quad \Lambda_B = \sum_{u\in\Omega} |\phi_u\rangle\langle u|.$$

The discriminant of $U(\mathcal{W})$ is

$$D(P) := \sum_{u,v\in\Omega} \langle \psi_u|\phi_v\rangle |u\rangle\langle v|.$$

We can compute $\langle\psi_u|\phi_v\rangle = \sqrt{P(u,v)P(v,u)} = \sqrt{\frac{\pi(v)}{\pi(u)}}P(v,u)$, by the reversibility of P. Thus,

$$D(P) = \sum_{u,v\in\Omega} \sqrt{\frac{\pi(u)}{\pi(v)}} P(u,v)|v\rangle\langle u| = \sum_{v\in\Omega} \frac{1}{\sqrt{\pi(v)}}|v\rangle\langle v|P \sum_{u\in\Omega} \sqrt{\pi(u)}|u\rangle\langle u|,$$

so P and $D(P)$ are similar and therefore have the same spectrum.

It is enough to analyze the spectrum of $U(\mathcal{W})$ on the space $A+B$, because we start in this space, and although we may leave this space during our implementation of $U(\mathcal{W})$, at every point where we apply phase estimation of $U(\mathcal{W})$, we will be in a state that lies in $A+B$.

Lemma 4.1.5. *The state $|\pi\rangle$ is the unique 1-eigenstate of $U(\mathcal{W})$ in $A+B$.*

Proof. Since $|\pi\rangle$ is clearly in $A\cap B$, and so a 1-eigenvector of $U(\mathcal{W})$, we need only show that it is the only such vector in $A+B$.

By the Perron-Frobenius theorem, since P is a positive real-valued square matrix, its largest eigenvalue has multiplicity 1. Since P is stochastic, all of its eigenvalues have absolute value at most 1, and since it has a 1-left-eigenvector, π, 1 is necessarily its largest eigenvalue.

Since $D(P)$ is similar to P, it follows that $D(P)$ has 1 as an eigenvalue with multiplicity 1. By Theorem 4.1.4, the only 1-eigenvectors of $U(\mathcal{W})$ in $A+B$ are in $A\cap B$, and this space is spanned by the right and left 1-eigenvectors of $D(P)$. It is not difficult to verify that $|\hat{\pi}\rangle := \sum_{u\in\Omega}\sqrt{\pi(u)}|u\rangle$ is a left and right singular vector of $D(P)$. It follows that $A\cap B$ is spanned by $\Lambda_A|\hat{\pi}\rangle = |\pi\rangle$ and $\Lambda_B|\hat{\pi}\rangle = |\pi\rangle$. Thus $|\pi\rangle$ is the unique 1-eigenvector of $U(\mathcal{W})$ in $A+B$. □

This shows that the only 0-phase eigenvector of $U(\mathcal{W})$ on $A+B$ is $|\pi\rangle$, so we can use phase estimation to distinguish $|\pi\rangle$ from its orthogonal vectors (for any state in $A+B$), however, in order to upper bound the number of calls to $U(\mathcal{W})$ needed for the phase estimation to succeed, we need to show a lower bound on the absolute value of the phase closest to 0, by lower bounding the phase gap of $U(\mathcal{W})$.

Lemma 4.1.6. *Let Δ be the phase gap of $U(\mathcal{W})$. Then $\Delta \geq \sqrt{\delta}$, where δ is the spectral gap of P.*

Proof. Suppose the phases in the statement of Theorem 4.1.4 are ordered so that $\theta_1 \leq \theta_2 \leq \cdots \leq \theta_\ell$. Then the phase gap of $U(\mathcal{W})$ is $2\theta_1$. Furthermore, since $\theta_1,\dots,\theta_\ell \in (0,\pi/2)$, we have

$$\cos\theta_1 \geq \cdots \geq \cos\theta_\ell,$$

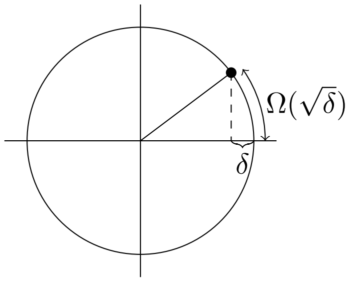


Figure 4.1: The phase gap scales like the square root of the spectral gap.

so $\delta = 1 - \cos\theta_1$ is the spectral gap of P, since P has the same spectrum as $D(P)$. Thus, using the inequality $\cos\theta \geq 1 - \frac{\theta^2}{2}$, we have

$$\delta = 1 - \cos\theta_1 \leq \frac{\theta_1^2}{2},$$

so $U(\mathcal{W})$ has phase gap $\Delta = 2\theta_1 \geq 2\sqrt{2\delta}$, giving the desired bound. $\square$

We can therefore use phase estimation with precision like $\sqrt{\delta}$ (see Section 3.1.1) to identify and reflect states orthogonal to $|\pi\rangle$. In particular, the following Theorem shows how we can approximate $\mathrm{Ref}\{|\pi\rangle\}$ to any error using phase estimation.

Theorem 4.1.7. *Assume* $\mathsf{U} + \Phi \geq 1$. *For any* $p \in (0,1)$, *there exists a unitary* $\tilde{R}$ *that approximates* $\mathrm{Ref}\{|\pi\rangle\}$ *on* $A + B$ *with error* p, *and can be implemented in cost* $O\left(\left(\log\frac{1}{p}\right)\frac{1}{\sqrt{\delta}}(\mathsf{U} + \Phi)\right)$.

Proof. We will use phase estimation to estimate the phase of the input state $|\psi\rangle \in A + B$, under the action of $U(\mathcal{W})$, with precision $\Theta = \frac{1}{8\pi}\sqrt{\delta}$. Let H_a be the auxiliary space, and suppose we begin by initializing it to $|0\rangle_a$. After applying phase estimation, we will apply a -1 phase, conditioned on the state of H_a being nonzero. We will then uncompute the phase estimation.

Suppose $|\psi\rangle \in A + B$ is an eigenstate of $U(\mathcal{W})$ — the result will follow by linearity. If $|\psi\rangle = |\pi\rangle$, since $U(\mathcal{W})|\pi\rangle = e^{i0}|\pi\rangle$, the phase estimation will put a 0 in the auxiliary register, acting as $|\pi\rangle|0\rangle_a \mapsto |\pi\rangle|0\rangle_a$. The conditional reflection will do nothing, since the auxiliary register is in the state 0. Applying the phase estimation in reverse will similarly do nothing. Thus we will leave the state unchanged.

On the other hand, suppose $|\psi\rangle$ is an eigenstate of $U(\mathcal{W})$ orthogonal to $|\pi\rangle$. Suppose $U(\mathcal{W})|\psi\rangle = e^{i\theta}|\psi\rangle$ for θ such that $|\theta| \in (0, \pi]$. Then by Theorem 3.1.3, applying phase estimation acts as

$$|\psi\rangle|0\rangle_a \mapsto |\psi\rangle|\omega\rangle_a,$$

for some state $|\omega\rangle$ such that $|\langle 0|\omega\rangle| \leq \Theta\frac{\pi}{|\theta|} = \frac{\sqrt{\delta}}{8|\theta|}$. The conditional reflection acts as $2I_H \otimes |0\rangle\langle 0| - I$, so we get:

$$|\psi\rangle|\omega\rangle_a \mapsto |\psi\rangle\left(2\langle 0|\omega\rangle|0\rangle_a - |\omega\rangle_a\right).$$

Finally, let Φ be the unitary that is implemented by phase estimation. Applying the phase estimation algorithm in reverse gives:

$$2\langle 0|\omega\rangle|\psi\rangle|0\rangle_a - |\psi\rangle|\omega\rangle_a \mapsto 2\langle 0|\omega\rangle\Phi^\dagger|\psi\rangle|0\rangle_a - |\psi\rangle|0\rangle_a.$$

Thus, if $\tilde{R}$ denotes the implemented unitary, and R denotes $\mathrm{Ref}\{|\pi\rangle\}$:

$$\left\|\tilde{R}|\psi\rangle - R|\psi\rangle\right\| = \left\|2\langle 0|\omega\rangle\Phi^\dagger|0\rangle\right\| = 2|\langle 0|\omega\rangle| \leq \frac{\sqrt{\delta}}{4|\theta|} \leq \frac{1}{4}$$

where the last inequality follows from Lemmas 4.1.5 and 4.1.6, since $|\psi\rangle$ is in $A+B$ and orthogonal to $|\pi\rangle$, so it must have phase $|\theta| \geq \sqrt{\delta}$. Thus, we have error at most $\frac{1}{4}$. Using Theorem 3.1.8, we can decrease this error to p with a multiplicative factor of $\log\frac{1}{p}$ in the asymptotic cost.

To complete the proof, we simply note that implementing phase estimation to precision Θ uses $O\left(\left(\log\frac{1}{\Theta}\right)^2\right) = O\left(\left(\log\frac{8\pi}{\sqrt{\delta}}\right)^2\right)$ elementary gates, and $\frac{2}{\Theta} = \frac{16\pi}{\sqrt{\delta}}$ controlled calls to $U(\mathcal{W})$, each of which costs $O(\mathsf{U}+\mathsf{\Phi})$. The total cost becomes

$$O\left(\log\frac{1}{p}\left(\frac{1}{\sqrt{\delta}}(\mathsf{U}+\mathsf{\Phi}) + \left(\log\frac{1}{\sqrt{\delta}}\right)^2\right)\right).$$

Assuming $\mathsf{U}+\mathsf{\Phi} \geq 1$, the second term is subsumed by the first, giving the desired complexity. □

The Full Algorithm We now have all necessary ingredients to present the full algorithm. A very simple algorithm would be to construct $|\pi\rangle$, and then repeatedly alternate $\mathrm{Ref}\{|\pi\rangle\}$ and the checking procedure, which reflects about the marked states, about $\frac{1}{\sqrt{\varepsilon}}$ times, just as in amplitude amplification. However, unlike in amplitude amplification, we cannot implement $\mathrm{Ref}\{|\pi\rangle\}$ exactly, so we would need to reduce the error of our approximation of $\mathrm{Ref}\{|\pi\rangle\}$ to $O(\sqrt{\varepsilon})$ in order to keep the overall error of the algorithm below $\frac{1}{3}$, at a cost of a multiplicative factor of $\log\frac{1}{\varepsilon}$. We can avoid this additional cost using the search-with-errors procedure, presented in Section 3.2.1, Theorem 3.2.2. This procedure takes an approximate implementation of $\mathrm{Ref}\{|\pi\rangle\}$, and an exact implementation of a reflection about marked states, and maps the state $|\pi\rangle$ to its projection onto the marked states. We use this procedure to prove the following theorem.

Theorem 4.1.8. *Let P be a reversible Markov process on state space Ω, with stationary distri-*

bution π. Let $M \subseteq \Omega$. Then we can approximate the mapping:

$$W : |\pi\rangle \mapsto \begin{cases} |M\rangle := \sum_{u\in M}\sqrt{\frac{\pi(u)}{\pi(M)}}\sum_{v\in\Omega}\sqrt{P(u,v)}|u,v\rangle|d(u,v)\rangle & \textit{if } M \neq \emptyset \\ |\pi\rangle & \textit{else,} \end{cases}$$

with error $1/3$ *in cost*

$$O\left(\frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}(\mathsf{U}+\Phi)+\mathsf{C}\right)\right).$$

Proof. Define a subspace of $\overline{H}$, H_M, by:

$$H_M := \text{span}\{|u, d_1(u)\rangle : u \in M\} \otimes H_C \otimes H_{d_2}$$

and let Π_M be the orthogonal projector onto H_M.

By Theorem 3.2.2, if we can implement $-\text{Ref}\, H_M$ in cost R_1, and we can approximate the reflection about $|\pi\rangle$ with error q in cost $\mathsf{R}_2 \log\frac{1}{q}$ for any $q \in (0, 1/2)$, then we can approximate W with constant error in cost $O\left(\frac{1}{\sqrt{p_M}}(\mathsf{R}_1 + \mathsf{R}_2)\right)$, where p_M is a lower bound on $\|\Pi_M|\pi\rangle\|^2$ whenever this quantity is nonzero. We have

$$\|\Pi_M|\pi\rangle\|^2 = \left\|\sum_{u\in M}\sum_{v\in\Omega}\sqrt{\pi(u)P(u,v)}|u,v\rangle|d(u,v)\rangle\right\|^2 = \sum_{u\in M}\pi(u)\sum_{v\in\Omega}P(u,v) = \pi(M),$$

so we can set $p_M = \varepsilon$.

Next, notice that the checking operation implements $\text{Ref}\, H_M$. By assumption, we can implement the checking operation in cost $\mathsf{R}_1 = \mathsf{C}$, and by Theorem 4.1.7, we can approximate $\text{Ref}\{|\pi\rangle\}$ with error q in cost $\mathsf{R}_2 \log\frac{1}{q} = \frac{1}{\sqrt{\delta}}(\mathsf{U}+\Phi)\log\frac{1}{q}$. Thus, we can implement W with error $1/3$ in cost

$$O\left(\frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}(\mathsf{U}+\Phi)+\mathsf{C}\right)\right).$$

□

Letting $\texttt{ApproxAmpAmp}^{\texttt{U},\Phi,\texttt{C}}$ denote the procedure from Theorem 4.1.8, which uses $\texttt{U}$, Φ and $\texttt{C}$ as subroutines, our algorithm is simply:

```
QuantumWalkSearch$^{\mathtt{S},\mathtt{U},\mathtt{\Phi},\mathtt{C}}_{\Omega,M,P,d}$

1. Repeat 2 times:
    (a) Construct $|\pi\rangle^0$ using S
    (b) Map $|\pi\rangle^0$ to $|\pi\rangle$ using U
    (c) Run ApproxAmpAmp$^{\mathtt{U},\mathtt{\Phi},\mathtt{C}}$ on $|\pi\rangle$ to get a state $|\widetilde{M}\rangle$
    (d) Measure the state register of $|\widetilde{M}\rangle$ to get some state $u$
    (e) Check if $u$ is marked, and if so, output $u$
2. Output "$M = \emptyset$"
```

Corollary 4.1.9. *Let $\{\mathcal{W}_x = (\Omega_x, M_x, P_x, d_x)\}_{x\in[q]^n}$ specify a quantum walk search problem. Let π_x denote the stationary distribution of P_x. Let $\varepsilon \in \mathbb{R}_+$ be some (possibly unknown) parameter such that for all x such that $M_x \neq \emptyset$, $\pi_x(M_x) \geq \varepsilon$. Let δ be a lower bound on the spectral gap of every P_x. Then the query complexity of finding an element of M_x or deciding M_x is empty with bounded error is*

$$O\left(\mathsf{S} + \frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}(\mathsf{U}+\Phi)+\mathsf{C}\right)\right).$$

Proof. The complexity of the algorithm `QuantumWalkSearch` is clear from Theorem 4.1.8, since we use constant p. To see that the algorithm solves the problem with bounded error, first note that if $M = \emptyset$, we will not find a marked state, so we will output the correct answer with probability 1.

For the case $M \neq \emptyset$, we need only notice that since we run `ApproxAmpAmp`, the error of $|\widetilde{M}\rangle$ is at most $\frac{1}{3}$, so $\left\||M\rangle - |\widetilde{M}\rangle\right\| \leq \frac{1}{3}$. Let $|\mathrm{err}\rangle := |M\rangle - |\widetilde{M}\rangle$. We can compute

$$\left|\langle M|\widetilde{M}\rangle\right| = |1 - \langle M|\mathrm{err}\rangle| \geq 1 - \||\mathrm{err}\rangle\| \geq 1 - \frac{1}{3} = \frac{2}{3}.$$

The probability that we measure a marked state in any given iteration is given by

$$\left\|\Pi_M|\widetilde{M}\rangle\right\|^2 \geq \left|\langle M|\widetilde{M}\rangle\right|^2 \geq \left(\frac{2}{3}\right)^2 > \frac{4}{9}.$$

Thus the probability that in both iterations we do not output a marked state is $(1-4/9)^2 = 25/81 < 1/3$. Thus, the probability that we find a marked state in one of the two rounds is at least $2/3$. □

We summarize this new framework below. We have made the separation between coin-independent and coin-dependent data explicit, and will now write $\mathcal{W} = (\Omega, M, P, d_1, d_2)$ when we would like to explicitly define this decomposition.

New Quantum Walk Search Framework		
Parameters: (for each $x \in D$)	Ω_x	Finite set of objects
	$M_x \subseteq \Omega_x$	Set of marked objects
	P_x	A reversible, ergodic Markov Process on Ω_x
	$d_{x,1} : \Omega_x \to H_{d_1}$	Coin-independent data
	$d_{x,2} : \overrightarrow{E}(P_x) \cup (\Omega_x \times \{0\}) \to H_{d_2}$	Coin-dependent data
Properties:	$\pi_x : \Omega_x \to \mathbb{R}$	The stationary distribution of P_x
	$\varepsilon \in \mathbb{R}_+$	A lower bound on $\pi_x(M_x)$ for all x such that $M_x \neq \emptyset$
	$\delta \in \mathbb{R}_+$	A lower boun on the spectral gap of P_x for all x
Subroutines:	Setup, $\mathtt{S}(x)$ with cost S	$\lvert 0\rangle \mapsto \lvert\pi_x\rangle^0 := \sum_{u\in\Omega_x} \sqrt{\pi_x(u)}\lvert u,0\rangle\lvert d_{x,1}(u)\rangle\lvert d_{x,2}(u,0)\rangle$
	Checking, $\mathtt{C}(x)$ with cost C	$\lvert u, d_{x,1}(u)\rangle \mapsto \begin{cases} -\lvert u, d_{x,1}(u)\rangle & \text{if } u \in M \\ \lvert u, d_{x,1}(u)\rangle & \text{else} \end{cases}$
	Local Diffusion, $\mathtt{U}(x)$ with cost U	$\lvert u,0\rangle\lvert d_2(u,0)\rangle \mapsto \sum_{v\in\Omega_x} \sqrt{P_x(u,v)}\lvert u,v\rangle\lvert d_2(u,v)\rangle$
	Data Swap, $\mathtt{SWAP}(x)$ with cost U	$\lvert u,v\rangle\lvert d_{x,1}(u)\rangle\lvert d_{x,2}(u,v)\rangle \mapsto \lvert v,u\rangle\lvert d_{x,1}(v)\rangle\lvert d_{x,2}(v,u)\rangle,\ \ \forall (u,v) \in \overrightarrow{E}(P_x)$
	Phase Flip, $\mathtt{\Phi}(x)$ with cost $\mathsf{\Phi}$	$\lvert u\rangle\lvert d_{x,2}(u,0)\rangle \mapsto -\lvert u\rangle\lvert d_{x,2}(u,0)\rangle,\ \ \forall u \in \Omega_x$
Cost:	$\mathsf{S} + \frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}(\mathsf{U}+\mathsf{\Phi}) + \mathsf{C}\right)$	

If the data is completely coin-independent, as in the previous framework, then the cost of the phase flip is negligible.

Chapter 5

Nested Checking

Consider implementing the checking subroutine of a quantum walk specified by $\mathcal{W}$ by using another quantum walk, $\mathcal{W}'$. One example of this is the implementation of the checking subroutine in the quantum walk algorithm for triangle finding, due to [MSS07], and outlined in Section 3.3.5. In this quantum walk algorithm, the checking step is implemented by a quantum algorithm, which consists of a quantum search, and a quantum walk, although for convenience, we can view the full checking algorithm as a quantum walk search algorithm. This gives rise to a checking subroutine with cost $\mathsf{C} = O\left(\mathsf{S}' + \frac{1}{\sqrt{\varepsilon'}}\left(\frac{1}{\sqrt{\delta'}}\mathsf{U}' + \mathsf{C}'\right)\right)$, where $(\delta', \varepsilon', \mathsf{S}', \mathsf{U}', \mathsf{C}')$ are the parameters and costs associated with the implementation of $\mathcal{W}'$. If the remaining parameters and costs of the implementation of $\mathcal{W}$ are $(\delta, \varepsilon, \mathsf{S}, \mathsf{U})$, then the full algorithm implementing $\mathcal{W}$ costs:

$$\widetilde{O}\left(\mathsf{S} + \frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U} + \mathsf{S}' + \frac{1}{\sqrt{\varepsilon'}}\left(\frac{1}{\sqrt{\delta'}}\mathsf{U}' + \mathsf{C}'\right)\right)\right),$$

where the suppressed logarithmic factors occur because to naively use a bounded-error subroutine for checking, we must apply error reduction to get inverse polynomial error on the checking subroutine.

In this Section, we will show how to use our new quantum walk search framework to implement this type of nesting more efficiently. We will show how to achieve nesting with a cost like:

$$\widetilde{O}\left(\mathsf{S} + \mathsf{S}' + \frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U} + \frac{1}{\sqrt{\varepsilon'}}\left(\frac{1}{\sqrt{\delta'}}\mathsf{U}' + \mathsf{C}'\right)\right)\right).$$[1]

Since we now have no multiplicative factors in front of S', this cost may be cheaper.

The idea of our efficient nested checking is simple: since the setup step simply constructs the initial state, we would like to do this once at the beginning of the algorithm, rather than each time

[1] The cost U could change when we bring S' to the front of the expression, but in our applications, it does not.

the checking subroutine is called. In order to implement this within the quantum walk search framework, we simply include the checking subroutine's initial state, $|\pi'\rangle$ in the data structure of the outer state — this data can then be used to perform the inner checking step. We present the details of this technique as a "framework" in Section 5.1.

The first application, and also the original motivation, for the framework is an algorithm for triangle finding, presented in Section 5.2. This algorithm is based on the learning graph upper bound of Ref. [Bel12b], and has matching quantum query complexity, up to logarithmic factors.

Although the quantum walk algorithm for triangle finding gives a nice illustration of the power of the framework, it is not the best known algorithm. A later learning graph upper bound, from [LMS13], gives an even better upper bound on the quantum query complexity of triangle finding, which, at the time of this writing, is the best known. We are also able to very simply reproduce this upper bound in our framework using nested checking, and this is the subject of Section 5.3. Finally, in Section 5.4, we give a generalized nested quantum walk algorithm for subgraph finding problems, based on the similar learning graph upper bound of [LMS13].

5.1 Nested Checking via Quantum Data Structures

The technique of efficient nested checking is simply one particular way in which we can apply the new quantum walk search framework. However, to illustrate how to use this technique, we write it explicitly as its own framework below. We suppose that we are given a quantum walk specification, $\mathcal{W} = (\Omega, M, P, d)$, that we want to implement, and along with $\mathcal{W}$, we are given, for each $u \in \Omega$, a specification for an *inner* quantum walk, $\mathcal{W}^u = (\Omega^u, M^u, P^u, d^u)$, whose purpose is to help check if u is marked. In order to use $\mathcal{W}^u$ to accomplish this task, we require that M^u be non-empty exactly when $u \in M$. We can then use the inner walk $\mathcal{W}^u$ to check if M^u is non-empty, which exactly checks if u is marked.

Since we will need to run both the outer walk $\mathcal{W}$ and the inner walks $\mathcal{W}^u$, our full workspace will look like:

$$H = H_\Omega \otimes H_C \otimes H_d \otimes H_{\Omega'} \otimes H_{C'} \otimes H_{d'},$$

where $H_\Omega \otimes H_C \otimes H_d$ is the workspace of $\mathcal{W}$, $H_{\Omega'}$ must be able to encode any state in Ω^u for any $u \in \Omega$, $H_{C'}$ must be able to encode any coin state $|P^u(s, \cdot)\rangle$ for $u \in \Omega, s \in \Omega^u$, and all inner data functions, d^u, must have domain $H_{d'}$.

In order to implement the full algorithm, we will need some subroutines for both the outer and inner walks. For the outer walk, we do not need a checking subroutine, since checking will be accomplished by the nested quantum walks. The outer setup and update are nearly identical to those of the standard framework, with the change that the initial state of the inner walk $\mathcal{W}^u$, $|\pi^u\rangle$, will now be include as part of the data of each u. Finally, we will need update and checking subroutines for the inner walks. The setup step of the inner walks are taken care of by the outer setup subroutine S. Its cost, S, will include the cost of two tasks. It will include the cost of constructing

$$|\pi_d\rangle^0 := \sum_{u\in\Omega} \sqrt{\pi(u)}|u,0\rangle|d(u)\rangle,$$

which is the initial state of $\mathcal{W}$ without nesting — call the cost of constructing this S_d. In addition, it will include the cost of constructing (maximized over u) the initial state for $\mathcal{W}^u$, $|\pi^u\rangle^0 = \sum_{s\in\Omega^u} \sqrt{\pi^u(s)}|s,0\rangle|d^u(s)\rangle$, which we can call S'. The full setup cost ends up being $\mathsf{S} = \mathsf{S}_d + \mathsf{S}'$.

We have chosen to give the nested checking framework in terms of coin-independent data, for simplicity, since this is all that is required by our applications. Thus, we suppose that the given walk specifications all include only coin-independent data $d : \Omega \to H_d$ for some finite-dimensional inner product space H_d. Of course, the idea is easily extended to walks that use coin-dependent data (with the additional associated costs).

Nested-Checking Quantum Walk		
Parameters:	$\mathcal{W}_x = (\Omega_x, M_x, P_x, d_x)$	An *outer quantum walk*, with coin-independent data
(for each $x \in D$)	$\{\mathcal{W}_x^u = (\Omega_x^u, M_x^u, P_x^u, d_x^u)\}_{u\in\Omega}$	A family of *inner quantum walks* with coin-independent data, such that for all $u \in \Omega_x$, $M_x^u \neq \emptyset$ if and only if $u \in M_x$
Properties:	$(\{\pi_x\}_x, \varepsilon, \delta)$	The inner walk properties
	$\{\pi_x^u : \Omega_x^u \to \mathbb{R}\}_{u\in\Omega}$	The stationary distribution of P_x^u
	$\varepsilon' \in \mathbb{R}$	A lower bound on $\pi_x^u(M_x^u)$ for all $u \in \Omega$ such that $M_x^u \neq \emptyset$
	$\delta' \in \mathbb{R}$	The minimum spectral gap of P_x^u over all $u \in \Omega$, $x \in D$
Subroutines:	Setup, $\mathtt{S}(x)$ with cost S	$\|0\rangle \mapsto \sum_{u\in\Omega_x} \sqrt{\pi_x(u)}\|u,0\rangle\|d_x(u)\rangle \sum_{s\in\Omega_x^u} \sqrt{\pi_x^u(s)}\|s,0\rangle\|d_x^u(s)\rangle$
	Local Diffusion, $\mathtt{U}(x)$ with cost U	$\|u,0\rangle \mapsto \sum_{v\in\Omega_x} \sqrt{P_x(u,v)}\|u,v\rangle, \ \forall u \in \Omega_x$
	Data Swap, $\mathtt{SWAP}(x)$ with cost U	$\|u,v\rangle\|d_x(u)\rangle \sum_{s\in\Omega_x^u} \sqrt{\pi_x^u(s)}\|s,0\rangle\|d_x^u(s)\rangle \mapsto \|v,u\rangle\|d_x(v)\rangle \sum_{s\in\Omega_x^v} \sqrt{\pi_x^v(s)}\|s,0\rangle\|d_x^v(s)\rangle, \ \forall (u,v) \in \overrightarrow{E}(P_x)$
	Inner Local Diffusion, $\mathtt{U}^u(x)$ with cost U'	$\|s,0\rangle \mapsto \sum_{t\in\Omega_x'} \sqrt{P_x^u(s,t)}\|s,t\rangle, \ \forall s \in \Omega_x^u$
	Inner Data Swap, $\mathtt{SWAP}^u(x)$ with cost U'	$\|s,t\rangle\|d_x^u(s)\rangle \mapsto \|t,s\rangle\|d_x^u(t)\rangle, \ \forall (s,t) \in \overrightarrow{E}(P_x^u)$
	Inner Checking, $\mathtt{C}^u(x)$ with cost C'	$\|s,d_x^u(s)\rangle \mapsto \begin{cases} -\|s,d_x^u(s)\rangle & \text{if } s \in M_x^u \\ \|s,d_x^u(s)\rangle & \text{else} \end{cases}$
Cost:	$\widetilde{O}\left(\mathsf{S} + \frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U} + \frac{1}{\sqrt{\varepsilon'}}\left(\frac{1}{\sqrt{\delta'}}\mathsf{U}' + \mathsf{C}'\right)\right)\right)$	

Theorem 5.1.1. *Let $\mathcal{N} = \{(\mathcal{W}_x = (\Omega_x, M_x, P_x, d_x), \{\mathcal{W}_x^u = (\Omega_x^u, M_x^u, P_x^u, d_x^u)\}_{u\in\Omega})\}_{x\in D}$ specify a nested-checking quantum walk. Let $(\delta, \varepsilon, \delta', \varepsilon')$ be the associated parameters. Then the cost of finding an element of M_x or deciding $M_x = \emptyset$ with bounded error is*

$$\widetilde{O}\left(\mathsf{S}(\mathcal{N}) + \frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U}(\mathcal{N}) + \frac{1}{\sqrt{\varepsilon'}}\left(\frac{1}{\sqrt{\delta'}}\mathsf{U}'(\mathcal{N}) + \mathsf{C}'(\mathcal{N})\right)\right)\right).$$

Proof. We will prove the statement by recasting the nested walk as a standard quantum walk search. Define a new coin-independent data function by

$$|\hat{d}(u)\rangle = |d(u)\rangle|\pi^u\rangle^0, \ \forall u \in \Omega \cup \{0\}$$

where $|\pi^u\rangle^0 = \sum_{s\in\Omega'} \sqrt{\pi^u(s)}|s,0\rangle|d^u(s)\rangle$. Then we will describe how to implement the subroutines for the quantum walk $\widehat{\mathcal{W}} = (\Omega, M, P, \hat{d})$.

Setup Subroutine We must prepare the state

$$|\pi_{\hat{d}}\rangle^0 := \sum_{u\in\Omega} \sqrt{\pi(u)}|u\rangle|\hat{d}(u)\rangle = \sum_{u\in\Omega} \sqrt{\pi(u)}|u\rangle|d(u)\rangle|\pi^u\rangle^0,$$

which is exactly, what S accomplishes in the nested checking framework, so the setup cost is $\mathsf{S}(\widehat{\mathcal{W}}) = \mathsf{S}(\mathcal{N})$.

Update Subroutines The local diffusion part of the update for $\widehat{\mathcal{W}}$ must achieve:

$$|u,0\rangle|\hat{d}(u)\rangle = |u,0\rangle|d(u)\rangle|\pi^u\rangle^0 \mapsto \sum_{v\in\Omega} \sqrt{P(u,v)}|u,v\rangle|\hat{d}(u)\rangle = \sum_{v\in\Omega} \sqrt{P(u,v)}|u,v\rangle|d(u)\rangle|\pi^u\rangle^0,$$

which is exactly what the local diffusion of $\mathcal{N}$ accomplishes. Furthermore, the data swap for $\widehat{\mathcal{W}}$ must achieve

$$|u,v\rangle|\hat{d}(u)\rangle = |v,u\rangle|d(u)\rangle|\pi^u\rangle^0 \mapsto |v,u\rangle|\hat{d}(v)\rangle = |v,u\rangle|d(v)\rangle|\pi^v\rangle^0,$$

which is exactly the data swap of $\mathcal{N}$. Thus $\mathsf{U}(\widehat{\mathcal{W}}) = \mathsf{U}(\mathcal{N})$.

Checking Subroutine We will implement the checking operation using a nested quantum walk algorithm. Let W_u be the mapping

$$W_u : |\pi^u\rangle^0 \mapsto \begin{cases} |M^u\rangle := \sum_{s\in\Omega^u} \sqrt{\frac{\pi^u(s)}{\pi^u(M^u)}}|s,0\rangle|d(s)\rangle & \text{if } M^u \neq \emptyset \\ |\pi^u\rangle^0 & \text{else,} \end{cases}$$

We first show how to use this mapping to implement the outer checking operation. We first apply W_u:

$$|u,\hat{d}(u)\rangle = |u\rangle|d(u)\rangle|\pi^u\rangle^0 \mapsto \begin{cases} |u\rangle|d(u)\rangle|M^u\rangle & \text{if } M^u \neq \emptyset \\ |u\rangle|d(u)\rangle|\pi^u\rangle^0 & \text{else.} \end{cases}$$

Next, using the inner checking, C^u, we perform

$$\begin{cases} |u\rangle|d(u)\rangle|M^u\rangle & \mapsto & -|u\rangle|d(u)\rangle|M^u\rangle \\ |u\rangle|d(u)\rangle|\pi^u\rangle^0 & \mapsto & |u\rangle|d(u)\rangle|\pi^u\rangle^0 \qquad \text{when } M^u = \emptyset. \end{cases}$$

Finally, applying $W_u^\dagger$ gives:

$$\begin{cases} -|u\rangle|d(u)\rangle|M^u\rangle & \mapsto & -|u\rangle|d(u)\rangle|\pi^u\rangle^0 = -|u\rangle|\hat{d}(u)\rangle \\ |u\rangle|d(u)\rangle|\pi^u\rangle^0 & \mapsto & |u\rangle|d(u)\rangle|\pi^u\rangle^0 = |u\rangle|\hat{d}(u)\rangle. \end{cases}$$

By Theorem 4.1.8, we can approximate W_u with error p in cost $\left(\log\frac{1}{p}\right)\frac{1}{\sqrt{\varepsilon'}}\left(\frac{1}{\sqrt{\delta'}}\mathsf{U}'(\mathcal{N}) + \mathsf{C}'(\mathcal{N})\right)$. By Fact 3.1.5, if we call our approximation of this map t times, the overall error in the final state will be tp. Since we will call the checking map on the order of $\frac{1}{\sqrt{\varepsilon}}$ times, in order to keep the overall error below $\frac{1}{3}$, we will need $\frac{1}{p} \in O\left(\frac{1}{\sqrt{\varepsilon}}\right)$. Each call to the approximation of W_u with error p then costs

$$\mathsf{C}(\widehat{\mathcal{W}}) = O\left(\left(\log\frac{1}{\varepsilon}\right)\frac{1}{\sqrt{\varepsilon'}}\left(\frac{1}{\sqrt{\delta'}}\mathsf{U}'(\mathcal{N}) + \mathsf{C}'(\mathcal{N})\right)\right).$$

Finally, we notice that, since $\widehat{\mathcal{W}}$ only differs from $\mathcal{W}$ in its data, it has the same spectral gap, δ, and lower bound on the proportion of marked states, ε. Thus, by Corollary 4.1.9, we can find an element of M or decide that M is empty in cost (up to constants):

$$\mathsf{S}(\widehat{\mathcal{W}}) + \frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U}(\widehat{\mathcal{W}}) + \mathsf{C}(\widehat{\mathcal{W}})\right) = \mathsf{S}(\mathcal{N}) + \frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U}(\mathcal{N}) + \frac{1}{\sqrt{\varepsilon'}}\left(\frac{1}{\sqrt{\delta'}}\mathsf{U}'(\mathcal{N}) + \mathsf{C}'(\mathcal{N})\right)\log\frac{1}{\varepsilon}\right).$$

□

5.1.1 Nesting to Arbitrary Depth

The simple two-level nesting from Theorem 5.1.1 can be generalized in a straightforward way to k levels of nesting.

Let x be the input, from some domain D, and $u_1, \dots, u_i$ be a list of states from the i outermost walks. Then we will use a quantum walk parametrized by $x, u_1, \dots, u_i$ to check if u_i is marked in the walk parametrized by $x, u_1, \dots, u_{i-1}$. To make this precise, we recursively define the following sets:

$$\Omega^{(0)} := D, \text{ and } \forall i > 0,\ \Omega^{(i)} := \{(x, u_1, \dots, u_i) : u^{(i-1)} := (x, u_1, \dots, u_{i-1}) \in \Omega^{(i-1)}, u_i \in \Omega_i^{u^{(i-1)}}\}.$$

Then there will be one i-depth walk defined for each element of $\Omega^{(i-1)}$.

Define the outermost walk family, parametrized by the input x, by:

$$\{\mathcal{W}_0^x = (\Omega_0^x, M_0^x, P_0^x, d_0^x)\}_{x\in D}$$

Then for $i = 1, \dots, k-1$, let the ith level of nested walk be specified by:

$$\left\{\mathcal{W}_i^{u^{(i)}} = (\Omega_i^{u^{(i)}}, M_i^{u^{(i)}}, P_i^{u^{(i)}}, d_i^{u^{(i)}})\right\}_{u^{(i)}\in\Omega^{(i)}}$$

with the condition that for all $i \in [0, k-2]$, $u^{(i)} \in \Omega^{(i)}$,

$$M_i^{u^{(i)}} = \{u_{i+1} \in \Omega_i^{u^{(i)}} : M_{i+1}^{(u^{(i)},u_{i+1})} \neq \emptyset\}.$$

The situation simplifies immensely when all Markov processes at the ith level are the same Markov process, P_i. Even when we allow them to vary slightly, there must, of course, be some structure for the analysis to be feasible.

Let δ_i be a lower bound on the spectral gap of $P_i^{u^{(i)}}$ over all $u^{(i)} \in \Omega^{(i)}$, and ε_i be a lower bound for non-empty $\pi_i^{u^{(i)}}(M_i^{u^{(i)}})$ for all $u^{(i)} \in \Omega^{(i)}$, where $\pi_i^{u^{(i)}}$ is the stationary distribution of $P_i^{u^{(i)}}$.

We recursively define, for each $u^{(k-1)} \in \Omega^{(k-1)}$,

$$|\pi_{k-1}^{u^{(k-1)}}\rangle := \sum_{u_{k-1}\in\Omega_{k-1}^{u^{(k-1)}}} \sqrt{\pi_{k-1}^{u^{(k-1)}}(u_{k-1})}|u_{k-1}\rangle|d_{k-1}^{u^{(k-1)}}(u_{k-1})\rangle$$

and for $i = 0, \dots, k-2$, and each $u^{(i)} \in \Omega^{(i)}$:

$$|\pi_i^{u^{(i)}}\rangle := \sum_{u_{i+1}\in\Omega_i^{u^{(i)}}} \sqrt{\pi_i^{u^{(i)}}(u_{i+1})}|u_{i+1}\rangle|d_i^{u^{(i)}}(u_{i+1})\rangle|\pi_{i+1}^{(u^{(i)},u_{i+1})}\rangle.$$

Let S denote the cost of constructing $|\pi_0^x\rangle$. For $i = 0, \dots, k-1$, let U_i be the maximum update cost of the i^{th} walk, that is, for each $u^{(i)} \in \Omega^{(i)}$, $u_{i+1} \in \Omega_i^{u^{(i)}}$, the cost of implementing:

$$|u_{i+1}\rangle|d^{u^{(i)}}(u_{i+1})\rangle|0\rangle \mapsto \sum_{u'_{i+1}\in\Omega_i^{u^{(i)}}} \sqrt{P_i^{u^{(i)}}(u_{i+1}, u'_{i+1})}|u_{i+1}, u'_{i+1}\rangle|d^{u^{(i)}}(u_{i+1}), d^{u^{(i)}}(u'_{i+1})\rangle$$

and for each $(u_{i+1}, u'_{i+1}) \in \overrightarrow{E}(P_i^{u^{(i)}})$,

$$|u_{i+1}, u'_{i+1}\rangle|d_i^{u^{(i)}}(u_{i+1}), d_i^{u^{(i)}}(u'_{i+1})\rangle \mapsto |u'_{i+1}, u_{i+1}\rangle|d_i^{u^{(i)}}(u'_{i+1}), d_i^{u^{(i)}}(u_{i+1})\rangle.$$

Let C be the maximum cost of implementing, for $u^{(k-1)} \in \Omega^{(k-1)}$, and $u_k \in \Omega_k^{u^{(k-1)}}$,

$$|u_k\rangle|d_{k-1}^{u^{(k-1)}}(u_k)\rangle \mapsto \begin{cases} -|u_k\rangle|d_{k-1}^{u^{(k-1)}}(u_k)\rangle & \text{if } u_k \in M_{k-1}^{u^{(k-1)}} \\ |u_k\rangle|d_{k-1}^{u^{(k-1)}}(u_k)\rangle & \text{else.} \end{cases}$$

Then we have the following.

Corollary 5.1.2. *We can decide whether M_0^x is non-empty with bounded error in*

$$\widetilde{O}\left(\mathsf{S} + \sum_{i=0}^{k-1}\left(\prod_{j=0}^{i}\frac{1}{\sqrt{\varepsilon_j}}\right)\frac{1}{\sqrt{\delta_i}}\mathsf{U}_i + \left(\prod_{j=0}^{k-1}\frac{1}{\sqrt{\varepsilon_j}}\right)\mathsf{C}\right) \tag{5.1}$$

queries.

This deeper nesting is applied in Section 5.4 to subgraph finding.

5.2 First Application to Triangle Finding

A relatively new framework for quantum query upper bounds is the learning graph framework of Belovs [Bel12b]. Belovs demonstrated the power of this new framework by demonstrating a new upper bound on the quantum query complexity of triangle finding of $O(n^{35/27})$ compared with the previous $\widetilde{O}(n^{1.3})$ quantum walk upper bound [MSS07], presented in Section 3.3.5. On the surface, there are a number of similarities between these two upper bounds, but Belovs' upper bound employed a new ingredient, edge sparsification, which, although easily applicable in the learning graph setting, did not seem to work in the setting of the quantum walk search framework.

There are several reasons why it might be desirable to implement the idea as a quantum walk. Unlike with learning graphs, whose evaluation is done by first converting the learning graph to a span program, which is then converted to a quantum algorithm, quantum walk algorithms have a very explicit structure that is easy to work with. This is useful for understanding the resulting algorithms, but it also facilitates combining and modifying algorithms to construct new algorithms.

Our new quantum walk search framework allows us to implement Belovs' edge sparsification idea to reproduce Belovs upper bound by a quantum walk algorithm that solves triangle finding in $\widetilde{O}(n^{35/27})$ quantum queries, proving the following theorem.

Theorem 5.2.1. *There exists a quantum walk search algorithm that solves* **Triangle**$_n$ *with bounded error using $\widetilde{O}(n^{35/27})$ quantum queries.*

In the remainder of this section, we will present this algorithm by specifying a quantum walk with nested checking, $(\mathcal{W} = (\Omega, M, P, d), \{\mathcal{W}^u = (\Omega^u, M^u, P^u, d^u)\}_{u\in\Omega})$, and implementing and analyzing its subroutines.

Parameters As usual, we will work with a Johnson graph, so our states will be subsets of $[n]$ of some size r to be specified later:

$$\Omega := \{S \subset [n] : |S| = r\}.$$

The states S correspond to subsets of the vertex set of G. We will say that a state is marked if it contains two triangle vertices, that is

$$M = \{S \in \Omega : \exists i, j \in S, k \in [n] \text{ such that } G_{i,j} = G_{j,k} = G_{i,k} = 1\}.$$

The walk will be the Johnson graph walk:

$$P(S, S') = \begin{cases} \frac{1}{r(n-r)} & \text{if } |S \cap S'| = r - 1 \\ 0 & \text{else.} \end{cases}$$

The outer walk will not have any data of its own (aside from $|\pi^S\rangle$, the initial state of an inner walk, $\mathcal{W}^S$, which is implicitly part of the data of S), so we set $d = 1$.

For each $S \in \Omega$, the inner walk $\mathcal{W}^S$ will have as its Markov process, P^S, the random walk on the Johnson graph $J(r^2, m)$, for some m such that $r < m < r^2$, to be specified later. We will have

$$\Omega^S := \{T \subset S^2 : |T| = m\}.$$

Each element $(i, j) \in T$ will correspond to a potential edge in the subgraph of G induced by S. We can define s so that $m = sr^2$. Then s is analogous to Belovs' sparsification parameter.

Let $G_S(T)$ denote the subgraph of G with vertex set S and edge set $T \cap E(G)$. Then we will define

$$d^S(T) := G_S(T);$$

that is, $d^S(T)$ will encode $\{(i, j, G_{i,j}) : (i, j) \in T\}$.

We will say that a state T is marked if T contains an edge that is part of a triangle; that is:

$$M^S := \{T \in \Omega^S : \exists (i, j) \in T, k \in [n] \setminus \{i, j\} \text{ s.t. } G_{i,j} = G_{i,k} = G_{j,k} = 1\}.$$

If $S \in M$, then S contains a pair of triangle vertices, i and j, so any $T \in \Omega^S$ containing (i, j) will be in M^S, so $M^S \neq \emptyset$. On the other hand, if $S \notin M$, then there is no pair of vertices i and j in S such that $\{i, j\}$ is a triangle edge, so no $T \in \Omega^S$ can contain such an (i, j). Thus, we can use the family $\{\mathcal{W}^S = (\Omega^S, M^S, P^S, d^S)\}_{S \in \Omega}$ to implement the checking subroutine of $\mathcal{W}$.

Properties As usual, the walk on the Johnson graph has the uniform distribution on Ω as its stationary distribution: $\pi(S) = \binom{n}{r}^{-1}$. Furthermore, P has $\delta \geq 1/r$. Finally, M is non-empty whenever the input graph contains a triangle, in which case, it has at least 2 triangle vertices, so:

$$\pi(M) = \frac{|M|}{|\Omega|} \geq \frac{\binom{n-2}{r-2}}{\binom{n}{r}} = \frac{(n-2)!r!}{n!(r-2)!} = \frac{r(r-1)}{n(n-1)}.$$

Thus, $\varepsilon := \frac{r(r-1)}{n(n-1)} \in \Omega\left(\frac{r^2}{n^2}\right)$ is a lower bound on $\pi(M)$ whenever $M \neq \emptyset$.

Similarly for the inner walk, P^S has spectral gap $\delta' \geq \frac{1}{m}$, and π^S the uniform distribution on Ω^S. Suppose M^S is non-empty. Then let $i, j \in S$ be two vertices of some triangle. Then any $T \in \Omega^S$ containing (i,j) is in M, so we have:

$$\pi^S(M^S) = \frac{|M^S|}{|\Omega^S|} \geq \frac{\binom{r^2-1}{m-1}}{\binom{r^2}{m}} = \frac{m}{r^2}.$$

Thus, $\varepsilon' = \frac{m}{r^2}$ is a lower bound on $\pi^S(M^S)$ whenever $M^S \neq \emptyset$.

Workspace The algorithm will work on the space

$$H_\Omega \otimes H_C \otimes H_d \otimes H_{\Omega'} \otimes H_{C'} \otimes H_{d'},$$

where each of these spaces will be defined presently. First, we have:

$$H_\Omega = \text{span}\{|S\rangle : S \subset [n], |S| = r\} \quad \text{and} \quad H_{\Omega'} = \text{span}\{|T\rangle : T \subset [n]^2, |T| = m\}.$$

Since we are only concerned with query complexity, we need not worry about how we encode each set S or T. A naive encoding of S or T as a sorted list will not affect the query complexity of any operations.

The coin space, H_C, should be such that $H_\Omega \otimes H_C$ can encode directed edges of $J(n,r)$. Since any directed edge (S, S') of $J(n,r)$ has $|S \cap S'| = r-1$, we can uniquely specify (S, S') by (S, i, i'), where $S \setminus S' = \{i\}$, and $S' \setminus S = \{i'\}$. We will thus use coin spaces:

$$H_C := \text{span}\{|i, i'\rangle : i, i' \in [n]\}, \quad \text{and} \quad H_{C'} := \text{span}\{|(i,j),(i',j')\rangle : i, i', j', j' \in [n]\}.$$

Then for any $S \in \Omega$, the respective coin state looks like

$$|P(S, \cdot)\rangle = \sum_{i \in S, i' \in [n]\setminus S} \frac{1}{\sqrt{r(n-r)}}|i, i'\rangle,$$

and for any $T \in \Omega^S$, the coin state looks like

$$|P^S(T,\cdot)\rangle = \sum_{(i,j)\in T,(i',j')\in S^2\setminus T} \frac{1}{\sqrt{m(r^2-m)}}|(i,j),(i',j')\rangle.$$

Finally, since the outer walk has no data, we have $H_d = 1$, and since the data functions of the inner walk are defined $d^S(T) = G_S(T)$, the space $H_{d'}$ can simply encode lists of m objects of the form $(i,j,G_{i,j})$, that is:

$$H_{d'} := \text{span}\{|E\rangle : E \subset \{(i,j,b) : i,j \in [n], b \in \{0,1\}\} : |E| = m\}.$$

Setup Subroutine We must implement the construction of the state

$$|\pi\rangle^0 = \sum_{S\in\Omega} \frac{1}{\sqrt{\binom{n}{r}}}|S\rangle|d(S)\rangle = \sum_{S\in\Omega} \frac{1}{\sqrt{\binom{n}{r}}}|S\rangle|\pi^S\rangle = \sum_{S\in\Omega} \frac{1}{\sqrt{\binom{n}{r}}}|S\rangle \sum_{T\in\Omega^S} \frac{1}{\sqrt{\binom{r^2}{m}}}|T\rangle|d^S(T)\rangle.$$

To do this, we must query all members (i,j) of T, which has cost $\mathsf{S} = m$ quantum queries.

Update Subroutines To implement the update, we must implement the local diffusion for $\mathcal{W}$:

$$|S,0\rangle \mapsto \sum_{S'\in\Omega} \sqrt{P(S,S')}|S,S'\rangle;$$

and the data swap for $\mathcal{W}$:

$$|S,S'\rangle \sum_{T\in\Omega^S} \sqrt{\pi^S(T)}|T,0\rangle|d^S(T)\rangle \mapsto |S',S\rangle \sum_{T'\in\Omega^{S'}} \sqrt{\pi^{S'}(T')}|T,0\rangle|d^{S'}(T')\rangle,$$

for any $S,S' \in \Omega$ such that $|S \cap S'| = m-1$.

To implement the local diffusion, we need only be able to construct, for any $S \in \Omega$, the coin state $|P(S,\cdot)\rangle$. This can be constructed in 0 queries, since it is input independent.

Fix some edge (S,S'). To implement the data swap, since $\pi^S(T) = \pi^{S'}(T') = \binom{r^2}{m}^{-1}$ for any $T \in \Omega^S$, and $T' \in \Omega^{S'}$, we need only show a bijection $\varphi : \Omega^S \to \Omega^{S'}$, and show how to implement a map D with the action

$$\sum_{T\in\Omega^S} \binom{r^2}{m}^{-1/2} |T\rangle|d^S(T)\rangle \mapsto \sum_{T\in\Omega^S} \binom{r^2}{m}^{-1/2} |\varphi(T)\rangle|d^{S'}(\varphi(T))\rangle.$$

Let $S \setminus S' = \{i\}$ and $S' \setminus S = \{i'\}$. Define $\gamma : S \to S'$ by $\gamma(i) = i'$, and for all $j \in S \cap S'$, $\gamma(j) = j$. Then define

$$\varphi(T) := \{(\gamma(j), \gamma(k)) : (i, j) \in T\}.$$

It is easy to verify that φ is a bijection from Ω^S to $\Omega^{S'}$. Furthermore, we can map $|T\rangle$ to $|\varphi(T)\rangle$ in 0 queries to the input, since mapping depends only on i and i', which are encoded by (S, S').

Let D_T be a unitary map with the action:

$$|d^S(T)\rangle = |\{(j, k, G_{j,k}) : (j, k) \in T\}\rangle \mapsto |d^{S'}(\varphi(T))\rangle = |\{(\gamma(j), \gamma(k), G_{\gamma(j),\gamma(k)}) : (j, k) \in T\}\rangle,$$

and note that $D = \sum_{T \in \Omega^S} |T\rangle\langle T| \otimes D_T$ has the desired action to implement the data swap for (S, S'). In order to implement D_T, we must query $G_{\gamma(j),\gamma(k)}$ and unquery $G_{j,k}$ for every $(j, k) \in T$ such that $(\gamma(j), \gamma(k)) \neq (j, k)$. In other words, we must perform 2 queries for every edge $(j, k) \in T$ incident to vertex i. The number of such pairs is exactly the degree of i in the graph with vertex set S and edge set T, which we denote K_T. Let $q_T = 2\deg_{K_T}(i)$, where $\deg_{K_T}(i)$ is the degree of i in the graph K_T, so that q_T is the query complexity of implementing D_T. For a uniform random T, $\deg_{K_T}(i)$ is a random variable with a hypergeometric distribution and mean $\frac{2m}{r}$, so for a random T, we expect the cost of D_T to be about $\frac{4m}{r}$. However, in the worst case, i may have degree r in K_T, in which case, the cost of D_T is r. We would certainly rather pay the average cost than the worst case. We can accomplish this in the following manner.

Let $q := \frac{28m}{r} + c \ln n$ for come constant $c \geq 1$. We will implement D approximately, using just q queries, by instead implementing

$$\tilde{D} := \sum_{T \in \Omega^S : q_T \leq q} |T\rangle\langle T| \otimes D_T + \sum_{T \in \Omega^S : q_T > q} |T\rangle\langle T| \otimes I.$$

In other words, we only implement the correct map when it is not too expensive, otherwise we simply don't bother. It is clear that $\tilde{D}$ has query complexity q, so we now consider its error on $|\psi\rangle = \sum_T \binom{r^s}{m}^{-1/2} |T\rangle |d^S(T)\rangle$, the only state to which it will be applied. We have:

$$\left\| (\tilde{D} - D)|\psi\rangle \right\|^2 = \left\| \sum_{T \in \Omega^S : q_T > q} \binom{r^2}{m}^{-1/2} |T\rangle (I - D_T) |d^S(T)\rangle \right\|^2 \leq 2 \sum_{T \in \Omega^S : q_T > q} \binom{r^2}{m}^{-1},$$

which is just twice the probability that $q_T > q$ when T is uniformly chosen. Since q_T is distributed according to a hypergeometric distribution with mean $\mu = \frac{4m}{r}$, by Fact 2.3.1, since $q \geq 7\mu$, we can bound this as

$$\left\| (\tilde{D} - D)|\psi\rangle \right\|^2 \leq 2e^{-\frac{28m}{r} - c \ln n} \leq 2e^{-c \ln n} = \frac{2}{n^c}.$$

Since this operation will be called $o(n^2)$ times, this is certainly sufficiently small error to keep the overall error at most $\frac{1}{3}$ (see Fact 3.1.5).

Thus, we can implement the update procedure with total query complexity:

$$\mathsf{U} = q = \frac{28m}{r} + c\log n = O\left(\frac{m}{r}\right),$$

since m and r will be set such that $\frac{m}{r} = n^k$ for some $k > 0$.

Inner Checking In order to check if a state T is marked, we will search $[n]$ for a vertex i such that there exists $j, k \in T$ such that $\{i, j, k\}$ is a triangle in G. For any $i \in [n]$, we can check if this condition holds in $r^{2/3}$ queries to G, using the algorithm `GraphCollision`, from Theorem 3.3.9, on the graph $G_S(T)$, since this graph is completely known, and stored in $d^S(T)$. Then the total cost to search for such a vertex i is

$$\mathsf{C}' = \sqrt{n}r^{2/3}.$$

Inner Update To implement the inner update, we must implement the local diffusion for $\mathcal{W}^u$:

$$|T, 0\rangle \mapsto \sum_{T' \in \Omega^S} \sqrt{P^S(T, T')}|T, T'\rangle;$$

and the data swap of $\mathcal{W}^u$:

$$|T, T'\rangle|d^S(T)\rangle \mapsto |T', T\rangle|d^S(T')\rangle,$$

for any $T, T' \in \Omega^S$ such that $|T \cap T'| = m - 1$.

To implement the local diffusion, we need only construct the state

$$|P(T, \cdot)\rangle = \sum_{(i,j)\in T} \frac{1}{\sqrt{m}}|(i, j)\rangle \sum_{(i',j')\notin T} \frac{1}{\sqrt{r^2 - m}}|(i', j')\rangle,$$

which has query complexity 0.

To implement the data swap, it suffices to implement, for each $(T, T') \in \overrightarrow{E}(P^S)$, the map:

$$|d^S(T)\rangle = |\{(i, j, G_{i,j}) : (i, j) \in T\}\rangle \mapsto |d^S(T')\rangle = |\{(i, j, G_{i,j}) : (i, j) \in T'\}\rangle.$$

To implement this, we must simply query the unique $(i', j') \in T' \setminus T$ and unquery the unique $(i, j) \in T \setminus T'$, so the full update cost for any inner update is $\mathsf{U}' = 2$.

Thus, by Theorem 5.1.1, we can find a marked state or decide there is no marked state in query complexity (neglecting logarithmic factors):

$$\begin{aligned}
\mathsf{S} + \frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U} + \frac{1}{\sqrt{\varepsilon'}}\left(\frac{1}{\sqrt{\delta'}}\mathsf{U}' + \mathsf{C}'\right)\right) &= m + \sqrt{\frac{n^2}{r^2}}\left(\sqrt{r}\frac{m}{r} + \sqrt{\frac{r^2}{m}}\left(\sqrt{m}2 + \sqrt{n}r^{2/3}\right)\right) \\
&= m + \frac{nm}{r^{3/2}} + n + \frac{n^{3/2}r^{2/3}}{\sqrt{m}}
\end{aligned}$$

Setting m and r to the optimal values of $r = n^{2/3}$ and $m = n^{35/27}$ gives total complexity $\widetilde{O}\left(n^{35/27}\right)$, proving Theorem 5.2.1.

5.3 Second Application to Triangle Finding

The algorithm presented in the previous section is based on the learning graph upper bound for triangle finding that inspired this framework, however, it is not the best known upper bound. Subsequently, the following theorem was proven by Lee, Magniez and Santha using the learning graph framework.

Theorem 5.3.1 ([LMS13]). *The bounded error quantum query complexity of* **Triangle**$_n$ *is in* $O\left(n^{9/7}\right) = O\left(n^{1.286}\right)$.

This new upper bound does not use the edge sparsification idea, but rather, first searches for a set S_1 containing a triangle vertex, and then another set S_2 containing another vertex from the same triangle, and detects such a pair of sets by doing a version of the graph collision subroutine on the bipartite subgraph of G consisting of edges from S_1 to S_2.

We can also reproduce this improved result, up to logarithmic factors, in our framework, using nested checking. In this section we will prove the following theorem.

Theorem 5.3.2. *There exists a quantum walk search algorithm that solves* **Triangle**$_n$ *with bounded error using* $\widetilde{O}\left(n^{9/7}\right)$ *quantum queries.*

In the remainder of this section, we will specify a nested-checking quantum walk, $(\mathcal{W} = (\Omega, M, P, d), \{\mathcal{W}^u = (\Omega^u, M^u, P^u, d^u)\}_{u\in\Omega})$, and describe and analyze implementations of its subroutines. Our construction is based on the learning graph construction used by [LMS13].

Parameters The walk will, once again, take place on a Johnson graph, so we will have:

$$\Omega := \{S_1 \subseteq [n] : |S_1| = r_1\}$$

for some r_1 to be chosen later, with P being the random walk on the Johnson graph $J(n, r_1)$. The states represent subsets of the vertices of G. We will say a set is marked if it contains a vertex that is part of a triangle:

$$M := \{S_1 \in \Omega : \exists i \in S_1, j, k \in [n] \text{ s.t. } G_{i,k} = G_{j,k} = G_{i,k} = 1\}.$$

We will not set any data for the outer walk, so we set $d = 1$.

Next, for each $S_1 \in \Omega$, we define an inner walk $\mathcal{W}^{S_1} = (\Omega', M^{S_1}, P', d^{S_1})$ (in this case, only the marked set and data depend on S_1). These inner walks P' will also be on a Johnson graph $J(n, r_2)$, for some r_2 to be specified, so we set

$$\Omega' := \{S_2 \subseteq [n] : |S_2| = r_2\}.$$

Like the outer states, each inner state, S_2, represents a subset of the vertex set. We define a state $S_2 \in \Omega'$ to be marked (with respect to S_1) if there is a vertex in S_2 that is part of a triangle, for which S_1 also contains a vertex; that is:

$$M^{S_1} = \left\{S_2 \in \Omega' : \exists i \in S_2, j \in S_1, k \in [n] \text{ s.t. } G_{i,k} = G_{j,k} = G_{i,k} = 1\right\}.$$

We can easily verify that $M^{S_1} \neq \emptyset$ if and only if $S_1 \in M$, so we can use the family $\{\mathcal{W}^{S_1}\}_{S_1}$ to implement the checking subroutine of $\mathcal{W}$.

Finally, we define the inner data function:

$$d^{S_1}(S_2) := \{(i, j, G_{i,j}) : i \in S_1, j \in S_2\}.$$

In words, d^{S_1} encodes all the edges of G with one endpoint in S_1 and the other in S_2, so it encodes the induced bipartite subgraph of G, $G(S_1, S_2)$.

Properties The spectral gaps of $J(n, r_1)$ and $J(n, r_2)$ are

$$\delta = \Omega\left(\frac{1}{r_1}\right) \quad \text{and} \quad \delta' = \Omega\left(\frac{1}{r_2}\right)$$

respectively.

The stationary distribution π is the uniform distribution on $\binom{[n]}{r_1}$, while the stationary distribution of P', π' is the uniform distribution on $\binom{[n]}{r_2}$. If there is a triangle $\{i, j, k\}$ in the input graph, then M is non-empty, and in particular, every set S_1 containing i or j or k is in M. Thus, we can compute a lower bound on $\pi(M)$ assuming $M \neq \emptyset$; in that case:

$$\pi(M) = \frac{|M|}{|\Omega|} \geq \frac{\binom{n-1}{r_1-1}}{\binom{n}{r_1}} = \frac{r_1}{n}.$$

Thus, $\varepsilon = \frac{r_1}{n}$ is a lower bound on $\pi(M)$ whenever $M \neq \emptyset$. Finally, M^{S_1} is non-empty exactly when $S_1 \in M$. In that case, let $\{i, j, k\}$ be a triangle with $i \in S_1$. Then a state S_2 is certainly marked if $j \in S_2$, so

$$\pi^{S_1}(M^{S_1}) = \frac{|M^{S_1}|}{|\Omega'|} \geq \frac{\binom{n-1}{r_2-1}}{\binom{n}{r_2}} = \frac{r_2}{n},$$

so $\varepsilon' = \frac{r_2}{n}$ is a lower bound on $\pi^{S_1}(M^{S_1})$ whenever $M^{S_1} \neq \emptyset$.

Workspace The algorithm will work on the space

$$H_\Omega \otimes H_C \otimes H_d \otimes H_{\Omega'} \otimes H_{C'} \otimes H_{d'},$$

where each of these spaces will be defined presently. First, we have:

$$H_\Omega = \mathrm{span}\{|S_1\rangle : S_1 \subseteq [n], |S_1| = r_1\} \quad \text{and} \quad H_{\Omega'} = \mathrm{span}\{|S_2\rangle : S_2 \subseteq [n], |S_2| = r_2\}.$$

Since we are only concerned with query complexity, we can simply use a naive encoding of each set as a sorted list.

From any state S_a, $a \in \{1,2\}$, we can specify the next state by some $i \in S_a$ to be removed, and some $i' \notin S_a$ to be added, so the coin spaces will be defined as

$$H_C := H_{C'} := \mathrm{span}\{|i,i'\rangle : i,i' \in [n]\}.$$

The coin states for any S_1 or S_2 will therefore look like:

$$|P(S_1,\cdot)\rangle = \sum_{i\in S_1, i'\in[n]\setminus S_1} \frac{1}{\sqrt{r_1(n-r_1)}}|i,i'\rangle \quad \text{and} \quad |P(S_2,\cdot)\rangle = \sum_{i\in S_2, i'\in[n]\setminus S_2} \frac{1}{\sqrt{r_2(n-r_2)}}|i,i'\rangle.$$

Finally, since the outer walk has no data, we have $H_d = 1$, and since the inner data consists of sets of queried edges, the space $H_{d'}$ can simply encode lists of m objects of the form $(i,j,G_{i,j})$, that is:

$$H_{d'} := \mathrm{span}\{E \subseteq \{(i,j,b) : i,j \in [n], b \in \{0,1\}\} : |E| = m\}.$$

Setup Subroutine Constructing the state

$$|\pi\rangle^0 = \sum_{S_1\in\Omega} \sqrt{\pi(S_1)}|S_1,0\rangle \sum_{S_2\in\Omega'} \sqrt{\pi'(S_2)}|S_2,0\rangle|d^{S_1}(S_2)\rangle$$

costs $\mathsf{S} = r_1 r_2$ queries, since we must query $G_{i,j}$ for every $(i,j) \in S_1 \times S_2$ to create $d^{S_1}(S_2)$.

Update Subroutines To implement the update, we must implement the local diffusion for $\mathcal{W}$:

$$|S_1,0\rangle \mapsto \sum_{S_1'\in\Omega} \sqrt{P(S_1,S_1')}|S_1,S_1'\rangle;$$

and the data swap for $\mathcal{W}$:

$$|S_1,S_1'\rangle \sum_{S_2\in\Omega'} \sqrt{\pi'(S_2)}|S_2,0\rangle|d^{S_1}(S_2)\rangle \mapsto |S_1',S_1\rangle \sum_{S_2\in\Omega'} \sqrt{\pi'(S_2)}|S_2,0\rangle|d^{S_1'}(S_2)\rangle,$$

for any $S_1, S_1' \in \Omega$ such that $|S_1 \cap S_1'| = r_1 - 1$.

To implement the local diffusion, we need only be able to construct, for any $S_1 \in \Omega$, the coin state $|P(S_1, \cdot)\rangle$, which we can do in 0 queries.

To implement the data swap, we need only implement the map $|d^{S_1}(S_2)\rangle \mapsto |d^{S_1'}(S_2)\rangle$ for any edges $(S_1, S_1') \in \overrightarrow{E}(P)$ and $S_2 \in \Omega'$. Fix an edge (S_1, S_1') and let i be the unique element in $S_1 \setminus S_1'$, and i' the unique element in $S_1' \setminus S_1$. Then since $d^{S_1}(S_2)$ encodes a bipartite subgraph of G, with vertex classes S_1 and S_2, when we remove the vertex i from S_1, and add the vertex i', to get the subgraph of G with vertex classes S_1' and S_2, we must unquery all edge values in $G(S_1, S_2)$ incident to i, and query all edge values in $G(S_1', S_2)$ incident to i'. That is, we must unquery the set $\{(i, j) : j \in S_2\}$, which has size r_2, and query the set $\{(i', j) : j \in S_2\}$, which also has size r_2. This costs $\mathsf{U} = 2r_2$ queries.

Inner Update Subroutines The inner update step is almost identical to the outer update. Once again, the local diffusion can be constructed with 0 queries, since the coin state $|P'(S_2, \cdot)\rangle$ is input-independent. The data swap is nearly identical as well: We must implement, for $S_2, S_2' \in \Omega$ such that $|S_2 \cap S_2'| = r_2 - 1$, the map

$$|S_2, S_2'\rangle |d^{S_1}(S_2)\rangle \mapsto |S_2', S_2\rangle |d^{S_1}(S_2')\rangle.$$

For an edge (S_2, S_2'), let i be the unique member of $S_2 \setminus S_2'$, and i' the unique member of $S_2' \setminus S_2$. Mapping $|d^{S_1}(S_2)\rangle = G(S_2, S_2')$ to $|d^{S_1}(S_2')\rangle = G(S_1, S_2')$ requires unquerying the set $\{(j, i) : j \in S_1\}$, which has size r_1, and querying the set $\{(j, i') : j \in S_1\}$, which has size r_1 as well, so the full inner update cost is $\mathsf{U}' = 2r_1$ queries.

Inner Checking Subroutine The data $d^{S_1}(S_2)$ exactly encodes the induced bipartite subgraph $G(S_1, S_2)$. A state S_2 is marked if and only if this graph contains a triangle edge. We know how to check if a graph contains a triangle edge in $O(\sqrt{n} r^{2/3})$ queries, where $r = r_1 + r_2$ is the number of vertices in $G(S_1, S_1)$, by using the graph collision algorithm from Theorem 3.3.9 and Grover search. However, we can optimize the graph collision algorithm on a bipartite graph, which gives us the following generalization of Theorem 3.3.9.

Lemma 5.3.3. *Let G be a graph on vertex set $[n]$. Let K be a known bipartite subgraph of G with vertex classes S_1 and S_2, such that $|S_1| = r_1$ and $|S_2| = r_2$. Then there exists an algorithm that, for any $k \in [n]$, decides if there exists an edge $\{i, j\}$ in K such that $\{i, j, k\}$ is a triangle in G, with bounded error, using $O\left(r_1^{1/3} r_2^{1/3}\right)$ queries to G.*

Proof. Let G be a bipartite graph on classes S_1 and S_2 with $|S_1| = r_1$ and $|S_2| = r_2$. We proceed as in the standard graph collision algorithm: let $m \leq \min\{r_1, r_2\}$ be a parameter to be optimized later. We walk on the product graph $J(r_1, m) \times J(r_2, m)$, where a state T_1, T_2 corresponds to a pair of subsets; T_1 is an m-sized subset of S_1 and T_2 is an m-sized subset of S_2. Two states (T_1, T_2) and (T_1', T_2') are adjacent in $J(r_1, m) \times J(r_2, m)$ if T_1 is adjacent to T_1' in $J(r_1, m)$ and T_2 is adjacent to T_2' in $J(r_2, m)$.

The data associated with a state T_1, T_2 is defined $d(T_1, T_2) = \{(i, G_{i,k}) : i \in T_1 \cup T_2\}$. Thus, the setup cost is $\mathsf{S} = O(m)$, and the update cost is $\mathsf{U} = O(1)$. A state is marked if there is a pair $i \in T_1$ and $j \in T_2$ such that $K_{i,j} = G_{i,k} = G_{j,k} = 1$. Since the data contains $G_{i,k}$ and $G_{j,k}$, and $K_{i,j}$ is known, checking costs $\mathsf{C} = 0$ queries. The proportion of marked states is at least $\varepsilon = \Omega\left(\frac{m^2}{r_1 r_2}\right)$. The spectral gap of $J(r_1, m) \times J(r_2, m)$ is $\delta = \Omega\left(\frac{1}{m}\right)$. Thus, the query complexity is

$$O\left(m + \frac{\sqrt{r_1 r_2}}{m}\sqrt{m}\right),$$

which is optimized by setting $m = (r_1 r_2)^{1/3}$ (which is smaller than $r_1 + r_2$ when $\max\{r_1, r_2\} \leq \min\{r_1, r_2\}^2$), yielding quantum query complexity $O\left((r_1 r_2)^{1/3}\right)$. □

We can check if a state $S_2 \in \Omega^{S_1}$ is marked with respect to $S_1 \in \Omega$ by searching for an index $i \in [n]$ such that there exists an edge $(j, k) \in S_1 \times S_2$ such that $\{i, j, k\}$ is a triangle in G. We thus have inner checking cost $\mathsf{C}' = O\left(\sqrt{n}(r_1 r_2)^{1/3}\right)$.

Plugging in all the costs and parameters, by Theorem 5.1.1, we have:

$$\begin{aligned} \mathsf{S} + \frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}\mathsf{U} + \frac{1}{\sqrt{\varepsilon'}}\left(\frac{1}{\sqrt{\delta'}}\mathsf{U}' + \mathsf{C}'\right)\right) &= r_1 r_2 + \sqrt{\frac{n}{r_1}}\left(\sqrt{r_1} r_2 + \sqrt{\frac{n}{r_2}}\left(\sqrt{r_2} r_1 + (r_1 r_2)^{1/3}\sqrt{n}\right)\right) \\ &= r_1 r_2 + \sqrt{n r_2} + n\sqrt{r_1} + \frac{n^{3/2}}{(r_1 r_2)^{1/6}}. \end{aligned}$$

Plugging the optimal values of $r_1 = n^{4/7}$ and $r_2 = n^{5/7}$ gives total cost $\widetilde{O}\left(n^{9/7}\right)$, proving Theorem 5.3.2.

5.4 Application to Subgraph Finding

Ref. [LMS13] gives a general upper bound for subgraph finding, for any constant-sized subgraph, of which their triangle finding upper bound is a special case. As with triangle finding in the previous section, we can also recast their subgraph finding upper bound in the nested-checking quantum walk search framework.

We now give an explicit construction of nested quantum walks corresponding to the learning graphs of [LMS13], matching their complexity up to logarithmic factors, and even potentially improving their result for some cases. For simplicity consider the case of undirected graphs with no self-loops. Let K be a graph on vertices $\{1, 2, \dots, k\}$. Then a graph G contains a copy of K if G induces a graph that is isomorphic to K on some subset of k vertices.

The Learning Graph Construction In [LMS13], a meta-language is introduced for designing a learning graph that detects a copy of K in G. Instructions are $\texttt{Setup}$, $\texttt{LoadVertex(}i\texttt{)}$ for $i \in [k]$ and $\texttt{LoadEdge(}i,j\texttt{)}$ for $\{i,j\} \in E(K)$. A 'program' is an ordering of $\{\texttt{Setup}\} \cup \{\texttt{LoadVertex(}i\texttt{)}\}_{i\in[k]} \cup \{\texttt{LoadEdge(}i,j\texttt{)}\}_{\{i,j\}\in E(K)}$ such that $\texttt{Setup}$ appears first and for each edge $\{i,j\}$ of K, instruction $\texttt{LoadEdge(}i,j\texttt{)}$ occurs in the program after instructions $\texttt{LoadVertex(}i\texttt{)}$ and $\texttt{LoadVertex(}j\texttt{)}$.

Parameters are vertex-set sizes $r_i \geq 1$ for each vertex i of K, and average degrees $d_{i,j} \geq 1$ for each edge $\{i,j\}$ of K. These parameters are not related to K but parameterize the resulting learning graph. Assuming some natural conditions on these parameters, we can derive an algorithm whose complexity is a function of the parameters, which can be optimized via a linear program (see [LMS13] for a link to the program). We assume that $r_i \leq n$, $d_{i,j} \leq \max\{r_i, r_j\}$, and that:

$$\textit{For all } i \textit{ there exists } j \textit{ such that } \{i,j\} \in E(K) \textit{ and } d_{ij}\frac{2r_j+1}{2r_i+1} \geq 1. \tag{5.2}$$

Each instruction has a cost that propagates to later instructions, and its total effective cost. The first one is called *global cost* and the second one *local cost*. In [LMS13], a table is given, nearly identical to Table 5.1, providing global and local costs for each of the three types of instructions. Given a program of $\tau = k + |E(K)| + 1$ instructions, if ℓ_t denotes the local cost of instruction t, and g_t the global cost, then the resulting upper bound for deciding if K is in G is

$$\sum_{t=1}^{\tau}\left(\prod_{t'<t} g_{t'}\right)\ell_t. \tag{5.3}$$

The Corresponding Nested Walk Given a program, we can define a sequence of nested walks as follows. The $\texttt{Setup}$ instruction corresponds to our quantum walk setup. Each non-setup instruction corresponds to a quantum walk, with the walk corresponding to the t^{th} instruction being nested at depth t; that is, the t^{th} walk is the checking procedure for the $(t-1)^{\text{th}}$ walk. Each non-setup instruction corresponds to an edge or vertex in K, and so each of our walks will as well.

All walks will be on Johnson graphs. The state of the walk corresponding to a vertex i is described by a subset of vertices R_i of size r_i, and the state of the walk corresponding to an edge

$\{i,j\}$ is described by a subset $T_{i,j}$ of $d_{i,j}\min(r_i,r_j)$ potential edges between S_i and S_j. For each edge $\{i,j\}$ of K, our data function maintains the edges $G(T_{i,j})$ of G in $T_{i,j}$.

A state S_i is marked if there exists some copy of K in G such that: for each S_j in a walk at depth *less than* the depth of S_i, S_j contains the j^{th} vertex in the subgraph, and for each $T_{u,v}$ at depth less than the depth of S_i, $T_{u,v}$ contains edge $\{u,v\}$ in the subgraph, and finally, S_i contains the i^{th} vertex in the subgraph. A state $T_{i,j}$ has a similar condition for being marked, except $T_{i,j}$ must contain edge $\{i,j\}$ in the subgraph.

Setup To setup, we must create the uniform superposition $|\pi\rangle^0$ of subgraphs $G(T_{i,j})$, one for each edge $\{i,j\}\in E(K)$. The total cost is

$$\mathsf{S} = \sum_{\{i,j\}\in K} \min\{r_i,r_j\}d_{i,j}.$$

Vertex Walks To update a walk on vertex set S_i we remove a vertex and add a new one. To update the data, we must update the graphs $G(T_{i,j})$, for each j adjacent to i in K. By applying the average cost technique used in the update step of the algorithm presented in Section 5.2, in which we only perform updates that have cost at most a constant times the average, we can see that this complexity is related to the average degree in S_i, which we can calculate as $d_{i,j}\min\left\{1,\frac{r_j}{r_i}\right\}$. The total cost for this type of update is then

$$\mathsf{U} = \sum_{j:\{i,j\}\in K} d_{i,j}\min\left\{1,\frac{r_j}{r_i}\right\}.$$

Observe that this quantity is always $\Omega(1)$, by the imposed condition in (5.2).

We can easily calculate that the proportion of marked states in this type of walk is $\varepsilon = r_i/n$, and the spectral gap is $\delta = 1/r_i$.

Edge Walks Lastly, to update a walk on edge set $T_{i,j}$ we simply replace one edge in $T_{i,j}$ and update $G(T_{i,j})$. Thus the cost of this type of update is simply $\mathsf{U}=1$. For this type of walk we have $\varepsilon = \frac{d_{i,j}}{\max(r_i,r_j)}$ (since ε is the probability that a particular edge of $S_i\times S_j$ is in $T_{i,j}$) and $\delta = \frac{1}{\min\{r_i,r_j\}}$.

Checking A state of the deepest walk is marked if the data encodes an entire copy of K in G. Since we can detect this from the data with no further queries to G, the final checking cost is $\mathsf{C}=0$.

Instruction	Global Cost	Local Cost
`Setup`	1	$\mathsf{S} = \sum_{\{i,j\} \in K} \min\{r_i, r_j\} d_{i,j}$
`LoadVertex`(i)	$\frac{1}{\sqrt{\varepsilon}} = \sqrt{\frac{n}{r_i}}$	$\frac{1}{\sqrt{\varepsilon\delta}}\mathsf{U} = \sqrt{n} \times \sum_{j:\{i,j\} \in K} d_{i,j} \min\left\{1, \frac{r_j}{r_i}\right\}$
`LoadEdge`(i,j)	$\frac{1}{\sqrt{\varepsilon}} = \sqrt{\frac{\max\{r_i,r_j\}}{d_{i,j}}}$	$\frac{1}{\sqrt{\varepsilon\delta}}\mathsf{U} = \sqrt{r_i r_j}$

Table 5.1: Global and local costs for quantum walks corresponding to each type of instruction.

Global and Local Costs We can rephrase our costs for each walk into global and local costs. The global cost is simply the part of the cost that propagates forward to deeper walk costs. For the setup (which we can consider as a walk on a complete graph in which every state is marked), this is simply 1, whereas for other walks, the global cost is $\frac{1}{\sqrt{\varepsilon}}$ (see (5.1)). Similarly, we can consider local costs. For the setup this is defined as S, whereas for each other walk, it is defined as $\frac{1}{\sqrt{\varepsilon\delta}}\mathsf{U}$. We list all global and local costs in Table 5.1.

Comparing (5.1) and (5.3), we see that the upper bounds we achieve for subgraph detection are the same as those of [LMS13] (up to logarithmic factors), as a function of the global and local costs. Furthermore, our expressions for global and local costs are almost identical to those of [LMS13]. The only exception is that, depending on some sparsity condition, namely whether $d_{i,j}\min\{r_i, r_j\} < \max\{r_i, r_j\}$, the local cost of `LoadEdge`$(i,j)$ has two different expressions in [LMS13]: either $\sqrt{r_i r_j}$ (when the condition holds) or $\max\{r_i, r_j\}$ (otherwise). Therefore, ignoring logarithmic factors, our total complexity is the same as that of [LMS13] or potentially better in some cases, because of our potentially better edge-local cost, though we know of no particular graph for which we have asymptotically better complexity. We remark that we believe that this improvement in edge-local cost could also be made in the learning graph construction of [LMS13], however, it is not immediately clear how to do so in that setting, whereas in the quantum walk setting it is natural and immediate.

Chapter 6

Nested Updates

Having extended the quantum walk framework to allow for nested checking in [JKM13b], a natural question was whether a similar result would hold for nesting a quantum walk subroutine in the update step of a quantum walk. This turned out to be less straightforward than nested checking, and required the introduction of a new concept in quantum walks; that of *coin-dependent data*. Our main application of coin-dependent data was to allow for efficient nested updates, but it may be of interest on its own. We were able to use nested updates to reproduce the learning graph upper bound on the quantum query complexity of 3-distinctness up to log factors, and also to show that in fact, this upper bound of $\widetilde{O}(n^{5/7})$ also applies to time complexity.

As a motivation, consider the following quantum walk specification, with the goal of finding a 3-collision in an input x. The walk will be on a Johnson graph $J(n_2, r)$, where n_2 is the number of 2-collisions in x. The state space will consists of sets of r 2-collisions with respect to x: pairs (i, j) such that $i < j$ and $x_i = x_j$. It's not difficult to see how this could help us find a 3-collision: consider a state marked if it contains a 2-collision that is part of a 3-collision. We can check this using quantum search for a third index that collides with one of the pairs in the set.

In order to move classically from one state to another, we need to find a new 2-collision to add to the set of 2-collisions. It therefore seems very natural to use Ambainis's element distinctness algorithm to help us perform this task. This algorithm generates a uniform superposition of sets of m indices such that at least two of them are in collision. We can measure the output state of Ambainis's algorithm, and discard the non-collision parts, to get a uniform random 2-collision. However, at this point, our classical intuition breaks down, because in fact, what we need for the *quantum* update is a uniform *superposition* over 2-collisions. We cannot construct this from a uniform superposition of sets containing 2-collisions, because this unwanted *garbage*, or extra information, in each branch of superposition, is not independent of the 2-collision — for example, the unwanted elements are disjoint from the 2-collision — so there is entanglement between the

2-collisions and the garbage. We therefore cannot discard the garbage without destroying the state.

It is easy to imagine this garbage issue arising in other attempts to use a non-trivial quantum algorithm for the update step. In order to update a state u, we need to generate the coin state of u, $|P(u,\cdot)\rangle$, which is some superposition over the coin space of P. This is a kind of state generation problem, and requiring that we accomplish this state generation problem *coherently*; that is, we generate the superposition with no garbage, may make the problem much more difficult.

In Section 6.1, we will show how to overcome this difficulty in our new quantum quantum framework, using the coin-dependent data. We will show how this coin-dependent data can be used to perform nested updates, even when the coin state cannot be generated coherently. Our strategy will essentially be to make whatever garbage we happen to end up with when we try to generate the coin a part of the data. This garbage can be a function of the state we are trying to update, and also of the coin, since we have explicitly allowed this in the data, and it can therefore be entangled with the coin. This strategy can be used to deal with garbage in any quantum update subroutine.

As an application of this nested update idea, we are able to get a quantum walk algorithm for 3-distinctness with time complexity $\widetilde{O}(n^{5/7})$. This nearly matches a recent upper bound on the quantum query complexity of 3-distinctness of $O(n^{5/7})$ [Bel12a]. This query upper bound was proven in the span program framework, and thus did not yield an upper bound on the time complexity. Before this work, the best known time complexity for 3-distinctness was $\widetilde{O}(n^{3/4})$, using the algorithm presented in Section 3.3.3, due to Ambainis [Amb04]. Our nested quantum walk construction for 3-distinctness uses ideas from the span program construction of [Bel12a], but since they are implemented using quantum walks, the time complexity is easily analyzed. Furthermore, we are able to generalize our 3-distinctness algorithm to get an upper bound of $\widetilde{O}(n^{(k-1)/k})$ on the time complexity of k-distinctness for any $k > 3$. This is only slightly better than the previous upper bound of $\widetilde{O}(n^{k/(k+1)})$, and is worse than the best known query complexity of k-distinctness, which is $o(n^{3/4})$ for any k, also proven in [Bel12a]. As a warm-up, we will first show an upper bound of $O(n^{5/7})$ on the quantum query complexity of 3-distinctness, in Section 6.2.1. The full time-efficient algorithm for k-distinctness, presented in Section 6.2.2 is similar, but significantly more technical.

Prior to the work presented in this chapter, the only nontrivial quantum walk update step of which we are aware was a Grover search [CK11]. The ideas of this chapter open the door to using a much broader class of quantum algorithms to implement the update subroutines.

6.1 Nested Updates via Coin-Dependent Data

As with nested checking, we will illustrate the nested update technique by presenting it as an explicit framework. As with nested checking, we suppose that we are given a quantum walk specification, $\mathcal{W} = (\Omega, M, P, d)$, that we want to implement, and along with $\mathcal{W}$, for each $u \in \Omega$, we suppose we have some inner walk specification $\mathcal{W}^u = (\Omega^u, M^u, P^u, d^u)$, whose purpose is to help us to (non-coherently) generate the coin state of u. As in the nested checking section, we will assume for simplicity that the data functions d and d^u are coin-independent (although we will still use coin-dependent data to accomplish the nested update). It is also possible to consider coin-dependent data, but this adds the extra associated cost of the $(\Omega, 0)$-phase flip. Therefore, since we do not require coin-dependent data for our applications, we omit it for the sake of simplicity.

Since we will need to run both the outer walk $\mathcal{W}$ and the inner walks $\mathcal{W}^u$, our full workspace will look like:

$$H = H_\Omega \otimes H_C \otimes H_d \otimes H_{\Omega'} \otimes H_{C'} \otimes H_{d'},$$

where $H_\Omega \otimes H_C \otimes H_d$ is the workspace of $\mathcal{W}$, $H_{\Omega'}$ must be able to encode any state in Ω^u for any $u \in \Omega$, $H_{C'}$ must be able to encode any coin state $|P^u(s, \cdot)\rangle$ for $u \in \Omega, s \in \Omega^u$, and all inner data functions, d^u, must have domain $H_{d'}$.

We will need several new subroutines related to the update step.

Extraction Subroutine In order for the inner walk $\mathcal{W}^u$ to be useful in generating the coin state of u, even with garbage (which we will show how to deal with momentarily), we will need some procedure to *extract* the coin state, non-coherently, from the final state of the walk specified by $\mathcal{W}^u$, $|M^u\rangle = \sum_{s \in M^u} \sqrt{\frac{\pi^u(s)}{\pi^u(M^u)}}|s, 0\rangle|d^u(s)\rangle$. In other words, we will need access to an *extraction subroutine*, $\mathtt{E}$, that acts on $H_\Omega \otimes H_C \otimes H_{\Omega'} \otimes H_{C'} \otimes H_{d'}$ as

$$|u, 0\rangle|M^u\rangle = |u, 0\rangle \sum_{s \in M^u} \sqrt{\frac{\pi^u(s)}{\pi^u(M^u)}}|s, 0\rangle|d^u(s)\rangle \mapsto \sum_{v \in \Omega} \sqrt{P(u, v)}|u, v\rangle|\Gamma(u, v)\rangle,$$

for each $u \in \Omega$, for some states $\{|\Gamma(u, v)\rangle\}_{(u,v) \in \overrightarrow{E}(P)}$, which we call the *garbage states.* The *extraction cost*, E is the cost of the subroutine $\mathtt{E}$.

Dealing with Garbage In order to show that the presence of these garbage states are not, in fact, problematic in the new quantum walk search framework, we simply define coin-dependent data, d_2 so that $d_2(u, 0) = |\pi^u\rangle^0$ and $d_2(u, v) = |\Gamma(u, v)\rangle$. In that case, we have actually generated the coin state with respect to the data d_2, $|P(u, \cdot)_{d_2}\rangle = \sum_{v \in \Omega} \sqrt{P(u, v)}|u, v\rangle|d_2(u, v)\rangle$, which is exactly what is required by the new quantum walk framework. The only remaining requirement is a procedure for swapping the garbage, mapping $|\Gamma(u,v)\rangle$ to $|\Gamma(v,u)\rangle$. This is required to implement the data swap, which requires mapping $|d_2(u,v)\rangle$ to $|d_2(v,u)\rangle$. The *garbage swap cost*, G, is the cost of implementing this operation. In case $\Gamma(u,v) = \Gamma(v,u)$ for all $(u,v) \in \overrightarrow{E}$, this procedure is not required, and $\mathsf{G} = 0$.

These garbage states $\{|\Gamma(u,v)\rangle\}_{(u,v)\in\overrightarrow{E}}$ become a part of the quantum walk search algorithm specification in the sense that the extraction and garbage swap subroutines depend on them, but the correct way to think of them is that they are a byproduct of the extraction subroutine. They should be defined after the extraction procedure is specified, to be whatever remains entangled to the coin register when $|P(u,\cdot)\rangle$ is non-coherently generated. The garbage swap procedure is then implemented based on this definition of Γ.

These simple ideas give rise to the strategy for nested update quantum walks made explicit in the proof of Theorem 6.1.1. The result of the strategy is the following "framework".

Nested-Update Quantum Walk Framework		
Parameters:	$\mathcal{W}_x = (\Omega_x, M_x, P_x, d_x)$	An *outer quantum walk*
(for each $x \in D$)	$\{\mathcal{W}_x^u = (\Omega_x^u, M_x^u, P_x^u, d_x^u)\}_{u\in\Omega}$	A family of *inner quantum walks*
Properties:	$(\{\pi_x\}_x, \varepsilon, \delta)$	The outer walk properties
	$\{\pi_x^u : \Omega_X^u \to \mathbb{R}\}_{u\in\Omega_x}$	The stationary distribution of P_x^u
	$\varepsilon' \in \mathbb{R}_+$	A lower bound on $\pi_x^u(M_x^u)$ for all $u \in \Omega_x$ such that $M_x^u \neq \emptyset$
	$\delta' \in \mathbb{R}_+$	The minimum spectral gap of P_x^u over all $u \in \Omega_x$, $x \in D$
Subroutines:	Setup, $\mathtt{S}(x)$ with cost S	$\lvert 0\rangle \mapsto \sum_{u\in\Omega_x} \sqrt{\pi_x(u)}\lvert u,0\rangle\lvert d_x(u)\rangle \sum_{s\in\Omega_x^u} \sqrt{\pi_x^u(s)}\lvert s,0\rangle\lvert d_x^u(s)\rangle$
	Checking, $\mathtt{C}(x)$ with cost C	$\lvert u, d_x(u)\rangle \mapsto \begin{cases} -\lvert u, d_x(u)\rangle & \text{if } u \in M_x \\ \lvert u, d_x(u)\rangle & \text{else} \end{cases}$
	Inner Checking, $\mathtt{C}^u(x)$ with cost C'	$\lvert s, d_x^u(s)\rangle \mapsto \begin{cases} -\lvert s, d_x^u(s)\rangle & \text{if } s \in M_x^u \\ \lvert s, d_x^u(s)\rangle & \text{else} \end{cases}$
	Inner Local Diffusion, $\mathtt{U}^u(x)$ with cost U'	$\lvert s,0\rangle \mapsto \sum_{t\in\Omega_x^u} \sqrt{P_x^u(s,t)}\lvert s,t\rangle,\ \ \forall s \in \Omega_x^u$
	Inner Data Swap, $\mathtt{SWAP}^u(x)$ with cost U'	$\lvert s,t\rangle\lvert d_x^u(s)\rangle \mapsto \lvert s,t\rangle\lvert d_x^u(t)\rangle,\ \ \forall (s,t) \in \overrightarrow{E}(P_x^u)$
	Extraction, $\mathtt{E}(x)$ with cost E	$\lvert u,0\rangle\lvert M_x^u\rangle \mapsto \sum_{v\in\Omega_x} \sqrt{P_x(u,v)}\lvert u,v\rangle\lvert \Gamma_x(u,v)\rangle,\ \ \forall u \in \Omega_x$
	Garbage Swap, $\mathtt{G}(x)$ with cost G	$\lvert u,v\rangle\lvert d_x(u)\rangle\lvert \Gamma_x(u,v)\rangle \mapsto \lvert v,u\rangle\lvert d_x(v)\rangle\lvert \Gamma_x(v,u)\rangle,\ \ \forall (u,v) \in \overrightarrow{E}(P_x)$
Cost:	$\widetilde{O}\left(\mathsf{S} + \frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}\left(\frac{1}{\sqrt{\varepsilon'}}\left(\frac{1}{\sqrt{\delta'}}\mathsf{U}' + \mathsf{C}'\right) + \mathsf{E} + \mathsf{G}\right) + \mathsf{C}\right)\right)$	

We now make the discussion of this section precise, by proving that this framework gives rise to a quantum algorithm with the specified cost.

Theorem 6.1.1. *Let* $\mathcal{N} = \{(\Omega_x, M_x, P_x, d_x), \{(\Omega_x^u, M_x^u, P_x^u, d_x^u)\}_{u\in\Omega}\}_{x\in D}$ *specify a family of quantum walk search algorithms with nested updates.*

Then there is a quantum algorithm that finds an element of M, if $M \neq \emptyset$, with bounded error *and with cost*

$$\widetilde{O}\left(\mathsf{S}+\tfrac{1}{\sqrt{\varepsilon}}\left(\tfrac{1}{\sqrt{\delta}}\left(\tfrac{1}{\sqrt{\varepsilon'}}\left(\tfrac{1}{\sqrt{\delta'}}\mathsf{U}'+\mathsf{C}'\right)+\mathsf{E}+\mathsf{G}\right)+\mathsf{C}\right)\right).$$

Proof. Let $\mathtt{S}, \mathtt{C}, \{\mathtt{C}^u, \mathtt{U}^u, \mathtt{SWAP}^u, \mathtt{E}, \mathtt{G}\}_{u\in\Omega}$ be subroutines implementing the walk specified by $\mathcal{N}$, and in particular, let $\{|\Gamma(u,v)\rangle\}_{(u,v)\in\overrightarrow{E}(P)}$ be the garbage generated by $\mathtt{E}$. We will show how to implement this nested update quantum walk via a quantum walk search algorithm in our new framework using the given subroutines.

Define a new coin-dependent data function $\hat{d}_2 : \overrightarrow{E}(P)\cup(\Omega\times\{0\})\to H_{\hat{d}_2}$ by

$$\hat{d}_2(u,v)=|\Gamma(u,v)\rangle,\ \forall(u,v)\in\overrightarrow{E}(P),\quad\text{and}\quad \hat{d}_2(u,0)=|\pi^u\rangle^0=\sum_{s\in\Omega^u}\sqrt{\pi^u(u)}|u,0\rangle|d^u(s)\rangle,$$

where π^u is the stationary distribution of P^u. We will describe how to implement the subroutines for the quantum walk $\widehat{\mathcal{W}}:=(\Omega,M,P,d,\hat{d}_2)$.

Setup subroutine The setup subroutine of $\widehat{\mathcal{W}}$ must construct:

$$|\pi\rangle^0=\sum_{u\in\Omega}\sqrt{\pi(u)}|u,0\rangle|d(u)\rangle|\hat{d}_2(u,0)\rangle=\sum_{u\in\Omega}\sqrt{\pi(u)}|u,0\rangle|d(u)\rangle|\pi^u\rangle^0,$$

which is exactly accomplished by the setup subroutine of $\mathcal{N}$, so we have $\mathsf{S}(\widehat{\mathcal{W}})=\mathsf{S}(\mathcal{N})$.

Checking Subroutine We can implement the checking subroutine of $\widehat{\mathcal{W}}$ using the checking subroutine of $\mathcal{N}$, since they are identical. Thus $\mathsf{C}(\widehat{W})=\mathsf{C}(\mathcal{N})$.

Update Subroutines The local diffusion part of the update for $\widehat{\mathcal{W}}$ must achieve, for all $u\in\Omega$:

$$|u,0\rangle|\hat{d}_2(u,0)\rangle=|u,0\rangle|\pi^u\rangle^0\mapsto\sum_{v\in\Omega}\sqrt{P(u,v)}|u,v\rangle|\hat{d}_2(u,v)\rangle=\sum_{v\in\Omega}\sqrt{P(u,v)}|u,v\rangle|\Gamma(u,v)\rangle.$$

Let W^u be a unitary that acts as $W^u|\pi^u\rangle=|M^u\rangle$. We can use W^u and the extraction procedure, $\mathtt{E}$, to achieve the local diffusion for $\widehat{\mathcal{W}}$, since:

$$|u,0\rangle|\pi^u\rangle\overset{W^u}{\mapsto}|u,0\rangle|M^u\rangle\overset{\mathtt{E}}{\mapsto}\sum_{v\in\Omega}\sqrt{P(u,v)}|u,v\rangle|\Gamma(u,v)\rangle.$$

By Theorem 4.1.8, we can approximate the map W^u with error $1/3$ in cost on the order of $\frac{1}{\sqrt{\varepsilon'}}\left(\frac{1}{\sqrt{\delta'}}\mathsf{U}(\mathcal{W}^u)+\mathsf{C}(\mathcal{W}^u)\right)$, and so we can perform the local diffusion for $\widehat{\mathcal{W}}$ with error $1/3$ in cost

$$\frac{1}{\sqrt{\varepsilon'}}\left(\frac{1}{\sqrt{\delta'}}\mathsf{U}(\mathcal{W}^u)+\mathsf{C}(\mathcal{W}^u)\right)+\mathsf{E}(\mathcal{N})=\left(\frac{1}{\sqrt{\delta'}}\mathsf{U}'(\mathcal{N})+\mathsf{C}'(\mathcal{N})\right)+\mathsf{E}(\mathcal{N}).$$

The data swap for $\widehat{\mathcal{W}}$ must achieve, for any edge $(u,v)\in\overrightarrow{E}(P)$:

$$|u,v\rangle|d(u)\rangle|\hat{d}_2(u,v)\rangle=|u,v\rangle|d(u)\rangle|\Gamma(u,v)\rangle\mapsto|v,u\rangle|d(v)\rangle|\hat{d}_2(v,u)\rangle=|v,u\rangle|d(v)\rangle|\Gamma(v,u)\rangle.$$

This is exactly the action of the garbage swap, $\mathtt{G}$, so we can accomplish this in cost $\mathsf{G}(\mathcal{N})$.

Thus, if $\widetilde{\mathsf{U}}(\widehat{\mathcal{W}})$ is the cost to approximate the local diffusion and data swap with bounded error, we have

$$\widetilde{\mathsf{U}}(\widehat{\mathcal{W}})=\widetilde{O}\left(\frac{1}{\sqrt{\varepsilon'}}\left(\frac{1}{\sqrt{\delta'}}\mathsf{U}'(\mathcal{N})+\mathsf{C}'(\mathcal{N})\right)\right)+\mathsf{E}(\mathcal{N})+\mathsf{G}(\mathcal{N}).$$

$(\Omega,0)$**-Phase Flip Subroutine** The phase flip subroutine needs to implement the reflection about states of the form $|u\rangle|\hat{d}_2(u,0)\rangle=|u\rangle|\pi^u\rangle^0$. By Theorem 4.1.7, we can approximate this mapping with precision $1/3$ in cost $\frac{1}{\sqrt{\delta'}}\mathsf{U}(\mathcal{W}^u)$. Thus, if $\widetilde{\Phi}(\widehat{\mathcal{W}})$ is the cost to approximate the phase flip with bounded error, we have

$$\widetilde{\Phi}(\widehat{\mathcal{W}})=O\left(\frac{1}{\sqrt{\delta'}}\mathsf{U}(\mathcal{W}^u)\right)=O\left(\frac{1}{\sqrt{\delta'}}\mathsf{U}'(\mathcal{N})\right).$$

Recall that the local diffusion and phase flip are used to implement the walk operator. Using implementations with constant error, we get a walk operator that has constant error. Since we call the walk operator order of $\frac{1}{\sqrt{\varepsilon\delta}}$ times, we need to reduce the error to $O(\sqrt{\varepsilon\delta})$ using $\log\frac{1}{\varepsilon\delta}$ repetitions, by Theorem 3.1.8, so implementing the walk operator to the required precision has total cost $\left(\widetilde{\mathsf{U}}(\widehat{\mathcal{W}})+\widetilde{\Phi}(\widehat{\mathcal{W}})\right)\log\frac{1}{\varepsilon\delta}$.

Finally, we note that since $\widehat{\mathcal{W}}$ only differs from $\mathcal{W}$ in its data, it has the same spectral gap, δ, and proportion of marked states, ε. We can thus, by Corollary 4.1.9, implement $\widehat{W}$ in asymptotic cost:

$$\begin{aligned}
&\mathsf{S}(\widehat{\mathcal{W}}) + \frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}\left(\widetilde{\mathsf{U}}(\widehat{\mathcal{W}}) + \widetilde{\Phi}(\widehat{\mathcal{W}})\right)\log\frac{1}{\varepsilon\delta} + \mathsf{C}(\widehat{\mathcal{W}})\right)\\
=\;& \widetilde{O}\Bigg(\mathsf{S}(\mathcal{N}) + \frac{1}{\sqrt{\varepsilon}}\Bigg(\frac{1}{\sqrt{\delta}}\Bigg(\frac{1}{\sqrt{\varepsilon'}}\Bigg(\frac{1}{\sqrt{\delta'}}\mathsf{U}'(\mathcal{N}) + \mathsf{C}'(\mathcal{N})\Bigg) + \mathsf{E}(\mathcal{N}) + \mathsf{G}(\mathcal{N}) + \frac{1}{\sqrt{\delta'}}\mathsf{U}'(\mathcal{N})\Bigg) + \mathsf{C}(\mathcal{N})\Bigg)\Bigg)\\
=\;& \widetilde{O}\left(\mathsf{S}(\mathcal{N}) + \frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}\left(\frac{1}{\sqrt{\varepsilon'}}\left(\frac{1}{\sqrt{\delta'}}\mathsf{U}'(\mathcal{N}) + \mathsf{C}'(\mathcal{N})\right) + \mathsf{E}(\mathcal{N}) + \mathsf{G}(\mathcal{N})\right) + \mathsf{C}(\mathcal{N})\right)\right),
\end{aligned}$$

since $\varepsilon' \leq 1$, so $\frac{1}{\sqrt{\varepsilon'\delta'}}\mathsf{U}' \geq \frac{1}{\sqrt{\delta'}}\mathsf{U}'$. □

As in the case of quantum walks with nested checking, we can consider the setup cost as having two parts: the setup of the initial state of $\mathcal{W}$, $|\pi_d\rangle$, with cost S_d, and the setup of the initial state of $\mathcal{W}^u$, with cost S'. Then the total setup cost for a quantum walk with nested updates is simply the sum of these costs: $\mathsf{S} = \mathsf{S}_d + \mathsf{S}'$. Furthermore, if the inner walks do all the work of the update, so that the cost of the extraction is 0, and if the garbage is well-behaved, in the sense that it is either symmetric, or at least, the cost of implementing $|\Gamma(u,v)\rangle \mapsto |\Gamma(v,u)\rangle$ is negligible, then we get cost (neglecting poly-logarithmic factors):

$$\mathsf{S}_d + \mathsf{S}' + \frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}\left(\frac{1}{\sqrt{\varepsilon'}}\left(\frac{1}{\sqrt{\delta'}}\mathsf{U}' + \mathsf{C}'\right)\right) + \mathsf{C}\right),$$

analogous to the nested checking case.

As in the case of nested checking, it is also possible to nest quantum walks in the update step to arbitrary depth (but again, each depth of nested results in additional terms in the complexity), by applying this strategy inductively. However, as of yet, we know of no application for deeper nesting. Even our k-distinctness algorithms use depth 2 nesting for every k.

6.2 Application to k-Distinctness

We will now show how to apply nested-update quantum walks to get an improved upper bound on the time complexity of k-distinctness for all $k > 2$. Specifically, we will prove the following theorem.

Theorem 6.2.1 (Time Complexity of k-Distinctness)**.** *The bounded error quantum time complexity of* 3-$\mathbf{Dist}_n$ *is at most* $\widetilde{O}(n^{5/7})$. *Furthermore, for any* $k > 3$, *the bounded error quantum time complexity of* k-$\mathbf{Dist}_n$ *is at most* $\widetilde{O}(n^{(k-1)/k})$.

We will prove Theorem 6.2.1 in Section 6.2.2. First, as a warm-up, we will present a nested-update quantum walk in Section 6.2.1 that solves 3-distinctness in query complexity $\widetilde{O}(n^{5/7})$.

This upper bound on the query complexity was already known, but using the span program framework [Bel12a]. The proof of Theorem 6.2.1 is a generalization of the algorithm in Section 6.2.1, which we also analyze for time complexity, making the proof much more technical.

6.2.1 Query Complexity of 3-Distinctness

In this section, we will demonstrate how to use the nested-update technique outlined in Section 6.1 to construct a quantum walk for 3-distinctness with query complexity $\widetilde{O}(n^{5/7})$. Specifically, we will prove the following:

Theorem 6.2.2. *There exists a quantum walk algorithm solving 3-distinctness with bounded error in $\widetilde{O}(n^{5/7})$ quantum queries.*

This matches a previous result [Bel12a] up to logarithmic factors.

Main Idea We begin by giving a high-level overview of the algorithm. Our algorithm is conceptually simple. We walk on a Johnson graph, where the states correspond to sets of r_2 2-collisions, looking for a set that contains a 2-collision that is part of a 3-collision. We can check if a set has this desired property by doing a Grover search for an index that forms a 3-collision with one of the 2-collisions in the set.

To update to a neighbouring set, we will use a quantum walk subroutine for 2-distinctness. We will use a variation of Ambainis's quantum walk algorithm for 2-distinctness that finds m collisions for some parameter m. At each step of the outer walk we will use the inner 2-distinctness walk to update the outer walk state by replacing m of its members. That is, we will actually be walking on a generalized Johnson graph $J(n, r_2, m)$.

Assumptions on the Input and Notation We make the simplifying assumptions that there is at most one 3-collision and that the number of 2-collisions is in $\Theta(n)$. The first assumption is justified in [Amb04, Section 5]. To justify the second assumption, note that given an input $x \in [q]^n$, we can construct $x' \in [q+n]^{3n}$ with the same 3-collisions as x, and $\Omega(n)$ 2-collisions, by defining $x'_i = x_i$ for $i \in [n]$ and $x'_i = x'_{i+n} = q + i$ for $i \in \{n+1, \dots, 2n\}$. Note that any two 2-collisions not both part of the 3-collision are disjoint.

A common simplifying technique is to randomly partition the space $[n]$ and assume that the solution respects the partition in some sense. Here we partition the space into three disjoint sets of equal size, A_1, A_2 and A_3, and assume that if there is a 3-collision $\{a_1, a_2, a_3\}$, then we have $a_1 \in A_1$, $a_2 \in A_2$ and $a_3 \in A_3$. This assumption holds with constant probability, so we need only repeat the algorithm $O(1)$ times with independent choices of the tripartition to find any 3-collision with high probability. Thus, we assume we have such a partition.

Since we are assuming that $(a_1, a_2, a_3) \in A_1 \times A_2 \times A_3$, we will restrict our 2-collision search to the set

$$\mathcal{C} := \{(i,j) \in A_1 \times A_2 : x_i = x_j\}.$$

We will only consider 2-collisions in $\mathcal{C}$ to be "real" collisions. Then by the assumption that the only 3-collision has one part in A_2, we can assume that all 2-collisions in $\mathcal{C}$ are disjoint. We define $n_2 := |\mathcal{C}|$. We can assume that $n_2 = \Theta(n)$.

We will be working with sets $S_2 \subset \mathcal{C}$, but it will be useful to consider the set of indices that are part of the 2-collisions in S_2. We define

$$\hat{S_2} := \bigcup_{(i,j)\in S_2} \{i,j\}.$$

For any set $S_1 \subset [n]$ of indices, we let

$$\mathcal{C}(S_1) := \{(i,j) \in \mathcal{C} : i,j \in S_1\}$$

denote the set of 2-collisions contained in S_1. Since we make the assumption that the unique 3-collision has one part in A_3, all 2-collisions in $\mathcal{C}$ are disjoint. This gives us the following very useful fact: for any set $T_1 \subset [n]$ and $T_2 \subset \mathcal{C}$ such that $\hat{T_2}$ is disjoint from T_1, $|\mathcal{C}(T_1 \cup \hat{T_2})| = |\mathcal{C}(T_1)| + |T_2|$. We can see this inductively, by noting that for any $T_1 \subset [n]$, if $(i,j) \in \mathcal{C}$ and $i,j \in T_1$, then if we remove (i,j), it decreases $|\mathcal{C}(T_1)|$ by exactly 1. This is true precisely because i and j cannot be part of any other collision in T_1.

Finally, for any set $S_1 \subset [n]$, we define:

$$x(S_1) := \{(i, x_i) : i \in [n]\},$$

and for any set $S_2 \subset \mathcal{C}$, we define:

$$x(S_2) = \{(i,j,x_i) : (i,j) \in \mathcal{C}\}.$$

Parameters We will begin with the parameters for the outer walk $\mathcal{W} = (\Omega, M, P, d)$. Define

$$\Omega := \{S_2 \subseteq \mathcal{C} : |S_2| = r_2\}.$$

We say that a state is marked if it contains a 2-collision that is part of a 3-collision:

$$M := \{S_2 \in \Omega : \exists a_3 \in A_3, (a_1, a_2) \in S_2 \text{ s.t. } x_{a_1} = x_{a_3}\}.$$

The underlying random walk P will be on a generalized Johnson graph, $J(n_2, r_2, m)$ for some $0 < r_2 < n_2$ and $0 < m < r_2$ to be specified later. Recall that two vertices S_2 and S_2' of $J(n_2, r_2, m)$ are adjacent whenever $|S \cap S'| = r_2 - m$. Thus

$$P(S_2, S_2') = \begin{cases} \frac{1}{\binom{r_2}{m}\binom{n_2-r_2}{m}} & \text{if } |S_2 \cap S_2'| = r_2 - m \\ 0 & \text{else.} \end{cases}$$

Finally, the data for the outer walk will be:

$$d(S_2) := x(S_2) = \{(i, j, x_i) : (i, j) \in S_2\}.$$

Next, we specify parameters for the inner walks $\mathcal{W}^{S_2} = (\Omega^{S_2}, M^{S_2}, P^{S_2}, d')$ for each $S_2 \in \Omega$. The inner walk will be a variation of Ambainis's element distinctness algorithm. When updating a state S_2, we will look for m collisions disjoint from those already in S_2, so we have:

$$\Omega^{S_2} := \{S_1 \subset (A_1 \cup A_2) \setminus \hat{S}_2 : |S_1| = r_1\},$$

with the inner walk P^{S_2} on $J(2n/3 - 2r_2, r_1)$ for some $r_1 \in [n]$. We have inner marked set

$$M^{S_2} := \{S_1 \in \Omega^{S_2} : |\mathcal{C}(S_1)| \geq m\}.$$

In the outer random walk, we replace m elements of the state S_2 at each step. The reason this is desirable is that it turns out that for the optimal r_1, when n_2 is large, as we assume, then most sets S_1 of size r_1 have more than one 2-collision. This means that we need not take any steps of the inner walk to find them, and furthermore, it would be a waste to only use one of the 2-collisions in S_2 and ignore the rest, since the outer walk mixes much faster if we replace more elements in S_2 in each step. We will therefore set m to be half the expected number of 2-collisions in a set of size r_1.

Finally, we will define the inner data:

$$d'(S_1) := x(S_1) = \{(i, x_i) : i \in S_1\}.$$

Properties All random walks on Johnson graphs have uniform stationary distribution, so for all $S_2 \in \Omega$, $S_1 \in \Omega^{S_2}$, we have:

$$\pi(S_2) = \binom{n_2}{r_2}^{-1}, \quad \text{and} \quad \pi^{S_2}(S_1) = \binom{2n/3 - 2r_2}{r_1}^{-1}.$$

We also know the spectral gaps for the walks on the $J(n_2, r_2, m)$ and $J(n-2r_2, r_1)$ from Fact 2.2.9:

$$\delta \geq \frac{m}{r_2}, \quad \delta' \geq \frac{1}{r_1}.$$

Finally, we estimate a lower bound on the proportion of marked states. For the outer walk, assuming that there is a 3-collision, $(a_1, a_2, a_3) \in A_1 \times A_2 \times A_3$, then a state S_2 is marked whenever $(a_1, a_2) \in S_2$. Then:

$$\pi(M) = \frac{|M|}{|\Omega|} \geq \frac{\binom{n_2-1}{r_2-1}}{\binom{n_2}{r_2}} = \frac{r_2}{n_2},$$

so $\varepsilon := \frac{r_2}{n_2}$ is a lower bound on $\pi(M)$. For the inner walk, a state is marked if $|\mathcal{C}(S_1)| \geq m$. Each $(i,j) \in \mathcal{C}$ is in $\mathcal{C}(S_1)$ with probability $\frac{r_1(r_1-1)}{n'(n'-1)}$, where $n' = |A_1 \cup A_2| - 2r_2 = \Theta(n)$. Since $|\mathcal{C}| = n_2$, the expected size of $|\mathcal{C}(S_1)|$ is $\mu = n_2\frac{r_1(r_1-1)}{n'(n'-1)}$. Then we will set $m = \frac{\mu}{2} = \Theta\left(\frac{r_1^2}{n}\right)$, since n_2 and n' are both of order $\Theta(n)$. Since $m = \frac{\mu}{2}$, using Fact 2.3.2, we have

$$\Pr[|\mathcal{C}(S_1)| \geq m] \geq 1 - e^{-\mu/8}.$$

We will make sure to choose r_1 so that $n_2\frac{r_1(r_1-1)}{n'(n'-1)} \geq 1$, so that $\varepsilon' := 1 - e^{-1/8} \geq \frac{1}{10}$ is a lower bound on $\pi^{S_2}(M^{S_2})$.

It may seem somewhat odd that we have $\varepsilon' = \Omega(1)$ for all inner walks, since this implies that an inner marked state S_1 is somehow easy to find. Going back to our classical intuition of a random walk algorithm, we can imagine trying to find m 2-collisions by sampling a set S_1 and then taking steps of the walk until we have found m 2-collisions, however, since we will already have m 2-collisions in the initially sampled state, we do not actually need to do the walk at all. So in some sense, the nested quantum walk we are presenting here does not actually contain a nested walk: we simply make use of the fact that it is sometimes easier to approximately reflect about a state $|\pi\rangle$ than it is to create it, so after the first time we "sample a state S_1", subsequent "samples" are cheaper. More formally, suppose we choose m so that it is within logarithmic factors of the expected number of 2-collisions, so that $\varepsilon' = 1 - \frac{1}{\mathrm{poly}(n)}$. Then we can apply the extraction subroutine directly on the state $|\pi^{S_2}\rangle \approx |M^{S_2}\rangle$, making the update cost (ignoring extraction and garbage swap costs, which we will show to be negligible): $\frac{1}{\sqrt{\delta'}}\mathsf{U}'$. This cost comes from the $(\Omega, 0)$-phase flip, which reflects about $|\pi^{S_2}\rangle$ — the classical analogue of sampling a new S_1. So although the quantum walk does need the coin-dependent data of the new framework, in some sense, it does not use the full machinery of the nested-update quantum walk framework.

Workspace The algorithm will work on the space

$$H_\Omega \otimes H_C \otimes H_d \otimes H_{\Omega'} \otimes H_{C'} \otimes H_{d'},$$

where each of these spaces will be defined presently. First we have:

$$H_\Omega = \mathrm{span}\{|S_2\rangle : S_2 \subset [n]^2, |S_2| = r_2\}, \quad \text{and} \quad H_{\Omega'} := \mathrm{span}\{|S_1\rangle : S_1 \subset [n], |S_1| = r_1\}.$$

Since we are only concerned with query complexity, we can assume all sets are stored as sorted lists. Since $d(S_2)$ is a set of elements of the form (i, j, x_i), and $d'(S_1)$ is a set of elements of the form (i, x_i), we will have:

$$H_d := \mathrm{span}\{|S\rangle : S \subset [n] \times [n] \times [q], |S| = r_2\} \quad \text{and} \quad H_{d'} := \mathrm{span}\{|S\rangle : S \subset [n] \times [q], |S| = r_1\}.$$

Finally, the coin spaces will be defined:

$$H_C := \mathrm{span}\{|I, J\rangle : I, J \subset [n]^2, |I| = |J| = m\} \quad \text{and} \quad H_{C'} := \mathrm{span}\{|i, j\rangle : i, j \in [n]\}.$$

For the outer walk, $|S_2\rangle|I, J\rangle_C$ represents the edge (S_2, S_2'), where $S_2' = (S_2 \setminus I) \cup J$.

Setup Subroutine To set up, we must construct the state:

$$|\pi\rangle^0 = \sum_{S_2 \in \Omega} \frac{1}{\sqrt{\binom{n_2}{r_2}}} |S_2\rangle|d(S_2)\rangle \sum_{S_1 \in \Omega^{S_2}} \frac{1}{\sqrt{\binom{n-2r_2}{r_1}}} |S_1\rangle|d'(S_1)\rangle.$$

We have $\Omega = \{S_2 \subset \mathcal{C} : |S_2| = r_2\}$, where $\mathcal{C} \subset A_1 \times A_2$ for some tripartition of $[n]$, $A_1 \times A_2 \times A_3$. It turns out that we cannot construct the state $|\pi\rangle^0$ with respect to a specified A_1, A_2, A_3. However, we can construct the state for some tripartition, over which we do not have full control, but which is sufficiently random.

Theorem 6.2.3. *Let $\mathcal{C}(A_1, A_2) = \{(i, j) \in A_1 \times A_2 : x_i = x_j\}$. We can construct the state*

$$\sum_{S_2 \subset \mathcal{C}(A_1, A_2) : |S_2| = r_2} \frac{1}{\sqrt{\binom{n_2}{r_2}}} |S_2\rangle|d(S_2)\rangle$$

for random variables A_1 and A_2 such that A_1 and A_2 are disjoint, and if x has a unique 3-collision $\{a_1, a_2, a_3\}$, and $A_3 := [n] \setminus A_1 \cup A_2$, then $\Pr[(a_1, a_2, a_3) \in A_1 \times A_2 \times A_3] = \Omega(1)$, *in $O(r_1 + r_2\sqrt{n/r_1})$ queries.*

Proof. We sketch a proof here, but note that the proof of Theorem 6.2.5 for the case of k-distinctness contains significantly more detail.

We start with a random 3-partition of $[n]$, B_1, B_2, B_3, and let $\mathcal{C} := \{(i, j) \in B_1 \times B_2 : x_i = x_j\}$. Let $n_2 = |\mathcal{C}|$.

To construct the state, we first construct

$$|\psi_1\rangle = \sum_{S_1 \subset B_1 : |S_1| = r_1} \binom{n/3}{r_1}^{-1/2} |S_1\rangle|x(S_1)\rangle,$$

which we can do in r_1 queries, since we must query all of S_1 to get $x(S_1)$. Next we will search for elements of B_2 that collide with S_1. For $S_1 \subset B_1$, define

$$Z(S_1) := \{i \in B_2 : \exists j \in S_1, (j, i) \in \mathcal{C}\}.$$

We will repeatedly Grover search for elements in $Z(S_1)$ until we find r_2 of them. We have to query an index i to see if it is in $Z(S_1)$, so the query complexity of this procedure is $\sqrt{n/|Z(S_1)|}r_2$, assuming $|Z(S_1)| \geq r_2$. For a uniform S_1, the size of $Z(S_1)$ is roughly $\frac{n_2 r_1}{n} = \Omega(r_1)$ in expectation. For most choices of B_1 and B_2, we have $\Pr[|Z(S_1)| \geq \frac{r_1}{4}] \geq 1 - o(1)$. We can thus ignore the part of the state $|\psi_1\rangle$ where $|Z(S_1)| < \epsilon r_1$, for some suitably chosen constant ϵ. Thus, if we will apply Grover search to find r_2 elements of $Z(S_1)$ in cost $O\left(r_2\sqrt{\frac{n}{r_1}}\right)$, the resulting state is close to:

$$|\psi_2\rangle = \sum_{S_1 \subset B_1 : |S_1| = r_1, |Z(S_1)| \geq \epsilon r_1} \binom{n/3}{r_1}^{-1/2} |S_1\rangle|x(S_1)\rangle \sum_{J \subset Z(S_2) : |J| = r_2} \binom{|Z(S_1)|}{r_2}^{-1/2} |J\rangle|x(J)\rangle.$$

For a particular J and S_1, define

$$K^{S_1,J} := \{i \in S_1 : \nexists j \in J \text{ s.t. } (i,j) \in \mathcal{C}\},$$

and

$$S_2^{S_1,J} := \{(i,j) \in \mathcal{C} \text{ s.t. } i \in S_1, j \in J\}.$$

Next, in 0 queries, we perform the reversible map:

$$|S_1\rangle|x(S_1)\rangle|J\rangle|x(J)\rangle \mapsto |K^{S_1,J}\rangle|x(K^{S_1,J})\rangle|S_2^{S_1,J}\rangle|x(S_2^{S_1,J})\rangle.$$

Applying this to $|\psi_2\rangle$ leaves us with a state close to:

$$\binom{n/3 + r_1 - r_2}{s_1}^{-1/2} \sum_{K \in \binom{B_1}{r_1 - r_2} : |Z(K)| \geq \epsilon r_1 - r_2} |K, x(K)\rangle \binom{|Z(K)| + r_2}{r_2}^{-1/2} \sum_{S_2 \in \mathcal{C}(B_1 \setminus K, B_2)} |S_2, x(S_2)\rangle.$$

Note that this state is not uniform in K, but is uniform in S_2 when we restrict to a particular K. Thus we measure the first register to get some K with non-uniform probability that depends only on $|Z(K)|$. The remaining state is the uniform superposition (omitting uniform amplitudes):

$$\sum_{S_2 \subset \mathcal{C}(B_1 \setminus K, B_2) : |S_2| = r_2} |S_2\rangle|d(S_2)\rangle,$$

since $d(S_2) = x(S_2)$. In other words, we have constructed the correct state, but for the wrong partition.

Now let $A_1 = B_1 \setminus K$, $A_2 = B_2$ and $A_3 = B_3 \cup K$. Then we have $\mathcal{C} = \mathcal{C}(B_1 \setminus K, B_2) = \mathcal{C}(A_1, A_2)$, so we have constructed the correct state for the tripartition A_1, A_2, A_3. Clearly, if $\{a_1, a_2, a_3\}$ is the unique 3-collision, then $a_1 \in B_1$, $a_2 \in B_2$ and $a_3 \in B_3$ with constant probability. It remains to consider whether $a_1 \in K$. Although the distribution of K is non-uniform, the distribution restricted to those K with $Z(K) = z$ is uniform for any fixed z, and it is easy to see that $\Pr(a_1 \in K | Z(K) = z)$ is $o(1)$ for any z.

For more details, refer to the proof of Theorem 6.2.5, which also proves an analogous statement for time complexity. $\square$

To complete the setup, we simply construct, for every $|S_2\rangle$, the state $\sum_{S_1 \in \Omega^{S_2}} |S_1\rangle |d'(S_1)\rangle$, which can be done in $r_1 = |S_1|$ queries. The full setup cost is then

$$\mathsf{S} = r_1 + r_2\sqrt{\frac{n}{r_1}}.$$

Checking Subroutine In order to check if a state S_2 is marked, we will search A_3 for an index a_3 such that $x_{a_3} = x_{a_1}$ for some $(a_1, a_2) \in S_2$. We can accomplish this task using Grover search over A_3. Since all values x_{a_1}, x_{a_2} such that $(a_1, a_2) \in S_2$ can be found in $d(S_2)$, we need only query x_{a_3} to know if a_3 is the index we are looking for. Thus, searching for such a value in A_3 costs $\mathsf{C} = O(\sqrt{|A_3|}) = O(\sqrt{n})$ queries.

Inner Checking Subroutine In order to check if a state S_1 is marked, we need only inspect $d'(S_1)$ to see if there are m 2-collisions, $i, j \in S_1$ such that $(i, j) \in A_1 \times A_2$ and $x_i = x_j$. Since $d'(S_1)$ contains x_i and x_j, this can be done with $\mathsf{C}' = 0$ queries.

Inner Update Subroutines To implement the inner update, we must implement the local diffusion for each $\mathcal{W}^{S_2}$:

$$|S_1, 0\rangle \mapsto \sum_{S_1' \in \Omega^{S_2}} \sqrt{P^{S_2}(S_1, S_1')} |S_1, S_1'\rangle;$$

and the data swap for each $\mathcal{W}^{S_2}$:

$$|S_1, S_1'\rangle |d'(S_1)\rangle \mapsto |S_1', S_1\rangle |d'(S_1')\rangle,$$

for every $S_1, S_1' \in \Omega^{S_2}$ such that $|S_1 \cap S_1'| = r_1 - 1$. We can perform the local diffusion in 0 queries, since it is independent of the input. To implement the data swap, since $d'(S_1) = \{(i, x_i) : i \in S_1\}$ and $d'(S_1') = \{(i, x_i) : i \in S_1'\}$, we need only unquery the unique $i \in S_1 \setminus S_1'$ and query the unique $i' \in S_1' \setminus S_1$. Thus the inner update can be implemented in $\mathsf{U}' = 2$ queries.

Extraction Subroutine To implement the extraction, we must implement the map

$$|S_2, 0\rangle \sum_{S_1\in M^{S_2}} \frac{1}{\sqrt{|M^{S_2}|}}|S_1, 0\rangle|d'(S_1)\rangle \mapsto \sum_{S_2'\in\Omega} \sqrt{P(S_2, S_2')}|S_2, S_2'\rangle|\Gamma(S_2, S_2')\rangle = |S_2\rangle|P(S_2, \cdot)_\Gamma\rangle$$

for any states $\{|\Gamma(S_2, S_2')\rangle\}_{(S_2,S_2')\in\overrightarrow{E}}$ in $H_{d'}$. Recall that the coin state of S_2, without data, is

$$|P(S_2, \cdot)\rangle = \sum_{I\subseteq S_2:|I|=m} \frac{1}{\sqrt{\binom{r_2}{m}}}|I\rangle \sum_{J\subseteq \mathcal{C}\setminus S_2:|J|=m} \frac{1}{\sqrt{\binom{n_2-r_2}{m}}}|J\rangle.$$

The first part of this is simple to construct, in 0 queries, with no garbage, from $|S_2\rangle$. To construct the second part, recall that M^{S_2} consists of subsets of $[n]$ that contain at least m pairs of colliding indices, not already in S_2. Roughly speaking, in order to construct the coin state with garbage, we will simply extract m collision pairs from each S_1 in superposition, and the garbage will be whatever is leftover. To start, we implement the map

$$|0\rangle|S_1, d'(S_1)\rangle \mapsto \sum_{J\subseteq\mathcal{C}(S_1):|J|=m} \frac{1}{\sqrt{\binom{|\mathcal{C}(S_1)|}{m}}}|J\rangle|S_1, d'(S_1)\rangle,$$

extracting every possible m-set of collision pairs from S_1 in superposition. This can be done in 0 queries, and the resulting state is (omitting uniform amplitudes):

$$\sum_{S_1\in M^{S_2}} \sum_{\substack{J\subseteq\mathcal{C}(S_1):\\ |J|=m}} \frac{1}{\sqrt{\binom{|\mathcal{C}(S_1)|}{m}}}|J\rangle|S_1\rangle|d'(S_1)\rangle = \sum_{\substack{J\subseteq\mathcal{C}\setminus S_2:\\ |J|=m}} |J\rangle \sum_{S_1\in M^{S_2}:\hat{J}\subset S_1} \frac{1}{\sqrt{\binom{|\mathcal{C}(S_1)|}{m}}}|S_1\rangle|d'(S_1)\rangle.$$

Note that it is important here that the amplitudes not depend on J. This is only true because we assume that the unique 3-collision has one part in A_3, and so all 2-collisions across $A_1 \times A_2$ are disjoint. We can define the garbage as what we happen to have leftover. For any S_2' such that $S_2' \setminus S_2 = J$, we define the garbage state:

$$|\Gamma(S_2, S_2')\rangle := \sqrt{\frac{|M^{S_2}|}{\binom{n_2-r_2}{m}}} \sum_{S_1\in M^{S_2}:\hat{J}\subset S_1} \frac{1}{\sqrt{\binom{|\mathcal{C}(S_1)|}{m}}}|S_1\rangle|d'(S_1)\rangle.$$

With these garbage states, we can implement the extraction in $\mathsf{E} = 0$ queries.

Garbage Swap Having defined the garbage states, we can now describe how to implement the garbage swap. This must act as

$$|S_2, S_2'\rangle|d(S_2)\rangle|\Gamma(S_2, S_2')\rangle \mapsto |S_2', S_2\rangle|d(S_2')\rangle|\Gamma(S_2', S_2)\rangle.$$

Let $I = S_2 \setminus S_2'$ and $J = S_2' \setminus S_2$. To implement the mapping

$$|d(S_2)\rangle = |\{(i,j,x_i) : (i,j) \in S_2\}\rangle \mapsto |d(S_2')\rangle = |\{(i,j,x_i) : (i,j) \in S_2'\}\rangle,$$

we must remove $x(I)$ and add $x(J)$. In fact, we do not need any queries to learn $x(J)$, because Γ consists of superpositions of sets S_1 containing all indices in J, and $d'(S_1)$ contains their query values. In fact, we will see that if we remove all of $J, x(J)$ from $S_1, d'(S_1)$, and replace it with $I, x(I)$, we will have implemented the map $|\Gamma(S_2, S_2')\rangle \mapsto |\Gamma(S_2', S_2)\rangle$ as well, with 0 queries.

We will perform the mapping, controlled on $|S_2, S_2'\rangle = |S_2, I, J\rangle$:

$$|x(S_2)\rangle|S_1\rangle|x(S_1)\rangle \mapsto |x((S_2 \setminus I) \cup J)\rangle|(S_1 \setminus \hat{J}) \cup \hat{I}\rangle|x((S_1 \setminus \hat{J}) \cup \hat{I})\rangle.$$

This costs 0 queries, since it just involves moving around already stored values. Applying this map to $|d(S_2)\rangle|\Gamma(S_2, S_2')\rangle$ gives:

$$|d(S_2')\rangle \sum_{S_1 \in M^{S_2} : \hat{J} \subset S_1} \frac{1}{\sqrt{\binom{|\mathcal{C}(S_1)|}{m}}} |(S_1 \setminus \hat{J}) \cup \hat{I}\rangle|d'((S_1 \setminus \hat{J}) \cup \hat{I})\rangle. \tag{6.1}$$

Next, it is easy to see that the map $\varphi(S_1) = (S_1 \setminus \hat{J}) \cup \hat{I}$ is a bijection from $\{S_1 \in M^{S_2} : \hat{J} \subset S_1\}$ to $\{S_1 \in M^{S_2'} : \hat{I} \subset S_1\}$. Furthermore, since all collisions in $\mathcal{C}$ are disjoint, we have for any S_1,

$$|\mathcal{C}(S_1)| = |\mathcal{C}(S_1 \setminus J)| + |J| = |\mathcal{C}(S_1 \setminus J)| + |I| = |\mathcal{C}((S_1 \setminus J) \cup I)| = |\mathcal{C}(\varphi(S_1))|.$$

We can therefore rewrite (6.1) as

$$|d(S_2')\rangle \sum_{S_1 \in M^{S_2'} : \hat{I} \subset S_1} \frac{1}{\sqrt{\binom{|\mathcal{C}(S_1)|}{m}}} |S_1\rangle|d'(S_1)\rangle.$$

Since $I = S_2 \setminus S_2'$ and $J = S_2' \setminus S_2$, this is exactly $|d(S_2')\rangle|\Gamma(S_2', S_2)\rangle$, as desired. We can complete the garbage swap by mapping $|S_2, S_2'\rangle$ to $|S_2', S_2\rangle$, also in 0 queries. Each of the required operations has 0 queries, so we can implement the garbage swap with query complexity $\mathsf{G} = 0$.

The Full Complexity Applying Theorem 6.1.1, we can compute the query complexity of our nested-update quantum walk algorithm, (neglecting logarithmic factors):

$$\begin{aligned}
& \mathsf{S} + \frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}\left(\frac{1}{\sqrt{\varepsilon'}}\left(\frac{1}{\sqrt{\delta'}}\mathsf{U}' + \mathsf{C}'\right) + \mathsf{E} + \mathsf{G}\right) + \mathsf{C}\right) \\
= \;& r_1 + r_2\sqrt{\frac{n}{r_1}} + r_1 + \sqrt{\frac{n_2}{r_2}}\left(\sqrt{\frac{r_2}{m}}\left(\frac{\sqrt{nm}}{\sqrt{r_1}} + 0 + 0\right) + \sqrt{n}\right) \\
= \;& r_1 + r_2\sqrt{\frac{n}{r_1}} + \sqrt{\frac{n_2 n}{r_1}} + \sqrt{\frac{n_2 n}{r_2}} = \widetilde{O}\left(r_1 + r_2\sqrt{\frac{n}{r_1}} + \frac{n}{\sqrt{r_1}} + \frac{n}{\sqrt{r_2}}\right).
\end{aligned}$$

Setting r_1 and r_2 to the optimal values of $r_1 = n^{5/7}$ and $r_2 = n^{4/7}$ gives query complexity $\widetilde{O}(n^{5/7})$, proving Theorem 6.2.2.

6.2.2 Time Complexity of k-Distinctness for $k \geq 3$

In this section, we will present a generalization of the quantum walk search algorithm presented in Section 6.2 that will solve k-distinctness for any $k \geq 3$. We will analyze this algorithm in terms of not only query complexity, but also time complexity, to prove Theorem 6.2.1.

Main Idea We begin by giving a high-level overview of the algorithm. As in the case of 3-distinctness, our main idea is very simple. Whereas the algorithm from Section 6.2.1 walks on a Johnson graph of sets of 2-collisions, our more general version will walk on sets of r_{k-1} $(k-1)$-collisions. To update the state, we will use a variation of the $(k-1)$-**Dist** algorithm of Ambainis, in which we search for a set containing m $(k-1)$-collisions. Thus, at each step of the outer walk, we replace m elements of the set. We remark that it may be possible to improve this algorithm by finding a way to use the improved $(k-1)$-**Dist** algorithm presented here rather than that of Ambainis's algorithm, thus using $k-1$ levels of nesting, however, it is not immediately clear how to do this.

Assumptions on the Input, and Notation As in the previous section, we will assume that the input has at most 1 k-collision, and $n_{k-1} = \Theta(n)$ $(k-1)$-collisions.

As in the previous section, we can partition $[n]$ into equal parts $A_1, \dots, A_k$, and only consider k-collisions $(a_1, \dots, a_k) \in A_1 \times \cdots \times A_k$. Define

$$\mathcal{C}_\ell(A_1, \dots, A_\ell) := \{(a_1, \dots, a_\ell) \in A_1 \times \cdots \times A_\ell : x_{a_1} = \cdots = x_{a_\ell}\}.$$

Note that $\mathcal{C}_1(A_1) = A_1$. When $A_1, \dots, A_k$ is fixed, and there is no ambiguity, we write

$$\mathcal{C}_{k-1} := \{(i_1, \dots, i_{k-1}) \in A_1 \times \cdots \times A_{k-1} : x_{i_1} = \cdots = x_{i_{k-1}}\}.$$

Let $n_{k-1} := |\mathcal{C}_{k-1}|$. For $S_{k-1} \subset \mathcal{C}_{k-1}$, define:

$$\hat{S}_{k-1} := \bigcup_{(i_1,\dots,i_{k-1}) \in S_{k-1}} \{i_1, \dots, i_{k-1}\}.$$

For any $S_1 \subset [n]$, define:

$$\mathcal{C}_{k-1}(S_1) := \{(i_1, \dots, i_{k-1}) \in \mathcal{C}_{k-1} : i_1, \dots, i_{k-1} \in S_1\}.$$

As in the case of 3-collision, since we assume a unique k-collision, and assume part of it is in A_k, the $(k-1)$-collisions in $\mathcal{C}_{k-1}$ are all disjoint.

Finally, for any $I \subseteq \mathcal{C}_\ell(A_1, \dots, A_\ell)$, define $x(I) = \{(i_1, \dots, i_\ell, x_{i_1}) : (i_1, \dots, i_\ell) \in I\}$.

Parameters The outer walk P will be on a Johnson graph $J(n_{k-1}, r_{k-1}, m)$ for some $r_{k-1} \in [n]$ and $m \in [r_{k-1}]$ to be defined later, with states representing sets of $(k-1)$-collisions. Thus, we define:

$$\Omega := \{S_{k-1} \subseteq \mathcal{C}_{k-1} : |S_{k-1}| = r_{k-1}\}.$$

We will consider a state S_1 marked if it has m distinct $(k-1)$-collisions, so

$$M := \{S_{k-1} \in \Omega : \exists a_k \in A_k, (a_1, \ldots, a_{k-1}) \in S_{k-1} \text{ s.t. } x_{a_k} = x_{a_1}\}.$$

Finally, the data for the outer walk will be:

$$d(S_{k-1}) := x(S_{k-1}) = \{(i_1, \ldots, i_{k-1}, x_{i_1}) : (i_1, \ldots, i_{k-1}) \in S_{k-1}\}.$$

The inner walks will be used to search for $(k-1)$-collisions in $A_1 \cup \cdots \cup A_{k-1}$ using the algorithm of Ambainis. When updating S_{k-1}, we will look for collisions disjoint from those already in S_{k-1}. Thus the inner walk will be on $J(n - |A_k| - (k-1)r_{k-1}, r_1)$ for some parameter r_1 to be specified later, with state space:

$$\Omega^{S_{k-1}} := \{S_1 \subseteq (A_1 \cup \cdots \cup A_{k-1}) \setminus \hat{S}_{k-1} : |S_1| = r_1\}.$$

We will consider a state S_1 *marked* if it contains m $(k-1)$-collisions, for some m that will be specified shortly, so we define

$$M^{S_{k-1}} := \left\{S_1 \in \Omega^{S_{k-1}} : |\mathcal{C}_{k-1}(S_1)| \geq m\right\}.$$

Finally, we define the inner data function:

$$d^{S_{k-1}}(S_1) = d'(S_1) := \{(i, x_i) : i \in S_1\}.$$

Properties Random walks on Johnson graphs have uniform stationary distribution, so the stationary distribution of P, π, is the uniform distribution on Ω, and the stationary distribution of $P^{S_{k-1}}$, $\pi^{S_{k-1}}$, is the uniform distribution on $\Omega^{S_{k-1}}$. We also know the spectral gap for any walk on a Johnson graph, so in particular:

$$\delta \geq \frac{m}{r_{k-1}}, \quad \text{and} \quad \delta' \geq \frac{1}{r_1}.$$

Next, we estimate a lower bound on the proportion of marked states. For the outer walk, assuming that there is at least one $(k-1)$-collision, $(a_1, \ldots, a_k)$, then a state S_{k-1} is marked whenever $(a_1, \ldots, a_{k-1}) \in S_{k-1}$. Then:

$$\pi(M) = \frac{|M|}{|\Omega|} \geq \frac{\binom{n_{k-1}-1}{r_{k-1}-1}}{\binom{n_{k-1}}{r_{k-1}}} = \frac{r_{k-1}}{n_{k-1}},$$

so $\varepsilon := \frac{r_{k-1}}{n_{k-1}}$ is a lower bound on the proportion of states that are marked.

Finally, we compute a lower bound ε' on the proportion of marked states in an inner walk. By assumption that $n_{k-1} = \Theta(n)$, there are *always* marked states in the inner walk, since $m < r_1 = o(n)$. As in the case of 3-distinctness, we will actually end up setting parameters so that the expected number of $(k-1)$-collisions in a state S_1 is more than 1. Thus, just as in the case of 3-distinctness, we will choose m to be $\frac{\mu}{2}$, where μ is the expected size of $\mathcal{C}_{k-1}(S_1)$, and then we will have ε' constant. We formalize this in the following lemma.

Lemma 6.2.4. *Suppose* $\frac{r_1^{k-1}}{n^{k-2}} \geq 1$. *Then there exist* $m \in O\left(\frac{r_1^{k-1}}{n^{k-2}}\right)$ *and* $\varepsilon' \in \Omega(1)$ *such that if* $M^{S_{k-1}} = \{S_1 \in \Omega^{S_{k-1}} : |\mathcal{C}_{k-1}(S_1)| \geq m\}$, *then for all* $S_{k-1} \in \Omega$, $\pi^{S_{k-1}}(M^{S_{k-1}}) \geq \varepsilon'$.

Proof. Let $Z = |\mathcal{C}_{k-1}(S_1)|$ for uniform random $S_1 \in \Omega^{S_{k-1}}$. For any $\vec{a} \in \mathcal{C}_{k-1}$, let $Z_{\vec{a}}$ be the indicator variable for the event that all of its coordinates occur in S_1, so we have $Z = \sum_{\vec{a}} Z_{\vec{a}}$. We have

$$\mu := E[Z] = \sum_{\vec{a} \in \mathcal{C}_{k-1}} \Pr[Z_{\vec{a}} = 1] = n_{k-1} \frac{\binom{n-(k-1)r_{k-1}-(k-1)}{r_1-(k-1)}}{\binom{n-(k-1)r_{k-1}}{r_1}} = \Theta\left(\frac{r_1^{k-1}}{n^{k-2}}\right),$$

since $n_{k-1} = \Theta(n)$. We will define $m := \frac{\mu}{2}$.

For any $\vec{a}, \vec{a}' \in \mathcal{C}_{k-1}$, $\Pr[Z_{\vec{a}} Z_{\vec{a}'} = 1] \leq \Pr[Z_{\vec{a}} = 1] \Pr[Z_{\vec{a}'} = 1]$, that is, the random variables are negatively correlated, so we can apply tail bounds (see Fact 2.3.2) to get $\Pr[Z < \mu - \epsilon] < e^{-\epsilon^2/(2\mu)}$ for any $\epsilon > 0$, where $p = \Pr[Z_{\vec{a}} = 1]$. Setting $\epsilon = \frac{\mu}{2}$, we get:

$$\Pr[Z \geq m] \geq 1 - e^{-\mu/8} \geq 1 - e^{-1/8} \geq \frac{1}{10}.$$

Setting $\varepsilon' = 1/10$ completes the proof. □

Workspace and Encoding Since we will be concerned with time complexity, we will need to carefully consider how we encode sets $|S\rangle$ and their data, so that elementary operations like inserting a new element can be done efficiently. As usual, the full workspace will have the tensor product structure:

$$H = H_\Omega \otimes H_C \otimes H_d \otimes H_{\Omega'} \otimes H_{C'} \otimes H_{d'}.$$

Each of these spaces will be spanned by states that encode sets — either of indices, $(k-1)$-tuples of indices, or index-query tuples of the form (i, x_i) or $(i_1, \ldots, i_{k-1}, x_{i_1})$. We first consider encoding sets of indices or $(k-1)$-tuples of indices. We will use the quantum radix tree structure, defined in Section 3.3.4, using the index or index tuple as both the key and value. Thus, H_Ω will store quantum radix trees of size r_{k-1}, containing strings of length $(k-1)\log n$, and $H_{\Omega'}$ will store quantum radix trees of size r_1, containing strings of length $\log n$, (see Definition 3.3.5). Recall that quantum radix trees storing strings of length $O(\log n)$ allows us to insert, delete, lookup, and construct a uniform superposition of the elements stored, all in time $O(\log n)$. For the remainder of this section, when we write $|S\rangle$, it will be implicit that S is encoded as a quantum radix tree.

The coin state of a state $S_{k-1} \in \Omega$ can be defined

$$|P(S_{k-1}, \cdot)\rangle := \frac{1}{\sqrt{\binom{r_{k-1}}{m}\binom{n_{k-1}-r_{k-1}}{m}}} \sum_{I \subset S_{k-1}:|I|=m} |I\rangle \sum_{J \subset \mathcal{C}_{k-1} \setminus S_{k-1}:|J|=m} |J\rangle,$$

so the outer coin also encodes sets. We will store these as quantum radix trees as well.

The inner coin is just $\sum_{i\in S_1} |i\rangle \sum_{j\in[n]\setminus S_1} |j\rangle$, so we can just let $H_{C'} = \mathrm{span}\{|i,j\rangle : i,j \in [n]\}$.

To store sets of items of the form (i, x_i), we use a $(k-1)$-augmented quantum radix tree structure (Definition 3.3.6). The x_i part will act as the key, and the i part the value. Recall that, in addition to being able to insert, remove and lookup in cost $O(\log n)$, we can also check the number of $(k-1)$-collisions stored in the tree in cost $O(1)$, as well as create a uniform superposition over all $(k-1)$-collisions stored in the tree in cost $O(\log n)$. This is even possible when we restrict to only those $(k-1)$-collisions in $A_1 \times \cdots \times A_{k-1}$. Thus, since $d'(S_1)$ is a set of elements of this form, $H_{d'}$ will store $(k-1)$-augmented quantum radix trees of size r_1.

Finally, we will store $|d(S_{k-1})\rangle$, which is a set of items of the form $(i_1, \ldots, i_{k-1}, x_{i_1})$, in a quantum radix tree, with the x_i part acting as the key, and the $(i_1, \ldots, i_{k-1})$ part acting as the value. So H_d will store quantum radix trees of size r_{k-1}.

Setup Subroutine Since we are now concerned with time complexity, we will need some efficient way to store and compute the partition $A_1, \ldots, A_k$. To this end, we will use a k-wise independent function to construct a k-wise independent partition of $[n]$ (see Definition 2.3.3). Such a partition is sufficiently random for our purposes, but still allows us to efficiently compute to which A_j an index i belongs, and encode the partition in a reasonable sized space.

Roughly, the setup subroutine will work as follows. Let $r_1 > r_2 > \cdots > r_{k-1}$ be parameters to be specified later. We will begin by querying a set $S_1 \subset A_1$ of r_1 indices. Next we will search for r_2 indices in A_2 that collide with the indices in S_1. We will remove those indices $K_1 \subset S_1$ for which we did not find a collision, leaving a set S_2 of r_2 2-collisions. Next we will search A_3 for r_3 indices that collide with the 2-collisions in S_2. We will remove those 2-collisions $K_2 \subset S_2$ for which we did not find a collision in A_3, resulting in a set S_3 of r_3 3-collisions. We will continue inductively, at the t^{th} step, finding indices from A_t that collide with some set of $(t-1)$-collisions, until we have a superposition over tuples of sets $S_{k-1}, K_1, \ldots, K_{k-2}$ such that S_{k-1} is a set of $(k-1)$-collisions, and for each j, K_j is a set of j-collisions. We will measure the sets $K_1, \ldots, K_{k-2}$ so that what remains is a superposition over sets S_{k-1} of $(k-1)$-collisions.

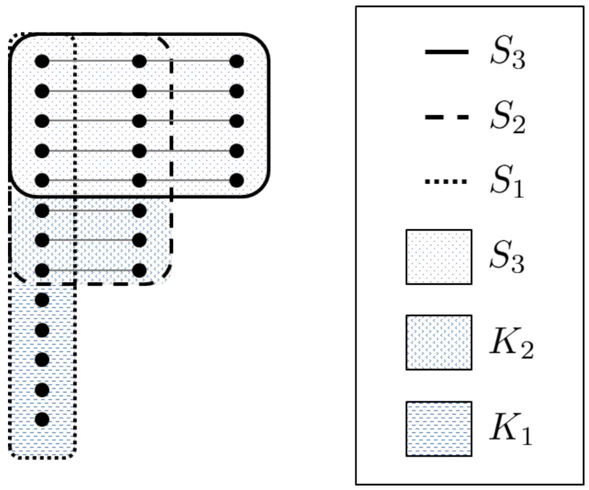


Figure 6.1: After two steps, we have a superposition over sets S_2 of 2-collisions and sets K_1 of single indices. After the third step, we have a state $|\psi_3\rangle$, supported by states of the pictured form, which consist of a set S_3 of 3-collisions, a set K_2 of 2-collisions, and a set K_1 of single indices.

Apart from the details of the time complexity analysis being somewhat technical, a small issue comes up when we attempt to implement this procedure, because whenever we measure the sets K_j of j-tuples with which we did not find collisions, there is some small correlation between K_j and the unmeasured register of $(k-1)$-collisions, S_{k-1}, so the resulting state does not have support on all sets of t-collisions from $\mathcal{C}_t(A_1,\dots,A_t)$, just as we saw in the case of 3-distinctness. We can deal with this by slightly modifying the partition $A_1,\dots,A_k$.

The full setup construction is quite technical, and consists almost entirely in the proof of the Theorem 6.2.5, below. An immediate corollary of Theorem 6.2.5 is that

$$\mathsf{S} = O\left(\left(r_1 + \sum_{i=1}^{k-2} r_{i+1}\sqrt{n/r_i}\right)(\log n + \log q)\right),$$

since we can complete the setup by constructing $\sum_{S_1\subset[n]\setminus\hat{S}_{k-1}:|S_1|=r_1}|S_1\rangle|d'(S_1)\rangle$ using r_1 insertions and queries.

Theorem 6.2.5 (First part of Setup)**.** *We can construct the state*

$$\binom{n_{k-1}}{r_{k-1}}^{-1/2}\sum_{S_{k-1}\subset\mathcal{C}_{k-1}(A_1,\dots,A_{k-1}):|S_{k-1}|=r_{k-1}}|S_{k-1}\rangle|d(S_{k-1})\rangle,$$

in time $O\left((r_1+\sum_{i=1}^{k-2}r_{i+1}\sqrt{n/r_i})(\log n+\log q)\right)$, for $A_1,\dots,A_{k-1}$ random variables such that:

1. *$A_1,\dots,A_{k-1},A_k := [n]\setminus(\bigcup_{j=1}^{k-1}A_j)$ is a k-partition of $[n]$;*
2. *if $\{a_1,\dots,a_k\}$ denotes the unique k-collision of x, then with probability at least $\frac{1}{2k^k}$, $(a_1,\dots,a_k)\in A_1\times\dots\times A_k$; and*

3. the time complexity of determining to which of $A_1, \ldots, A_k$ an index $i \in [n]$ belongs is $O(\log n)$, and the space required to store the partition is $O\left(\sum_{i=1}^{k-2} r_i \log n\right)$.

Proof. Let $B_1, \ldots, B_k$ be a uniform k-wise independent k-partition of $[n]$. For any $j \in \{1, \ldots, k-1\}$, let $n_j := |\mathcal{C}_j(B_1, \ldots, B_j)|$. With high probability, we have $n_j = \Theta(n)$ for all j.

For any $j \in \{1, \ldots, k-1\}$ and any set $S_j \subseteq \mathcal{C}_j(B_1, \ldots, B_j)$, define

$$Z(S_j) := \{i \in B_{j+1} : \exists (i_1, \ldots, i_j) \in S_j \text{ s.t. } x_i = x_{i_1}\},$$

the set of indices that form $(j+1)$-collisions with the j-collisions in S_j. For any $t \in \{1, \ldots, k-1\}$, $j \in \{1, \ldots, t\}$, and $S_t \subseteq \mathcal{C}_t(B_1, \ldots, B_t)$, define

$$S_t^{(j)} := \{(i_1, \ldots, i_j) \in \mathcal{C}_j(B_1, \ldots, B_j) : \exists i_{j+1}, \ldots, i_t \text{ s.t. } (i_1, \ldots, i_t) \in S_t\}.$$

For any $j \in \{1, \ldots, k-1\}$ and $S_j \subseteq \mathcal{C}_j(B_1, \ldots, B_j)$, let $|\tilde{x}(S_j)\rangle$ denote $|S_j\rangle|x(S_j)\rangle$. For $t \in \{1, \ldots, k-1\}$, define (omitting uniform amplitudes)

$$|\psi_t\rangle := \binom{|\mathcal{C}_t(B_1, \ldots, B_t)|}{r_t}^{-1/2} \sum_{\substack{S_t \subset \mathcal{C}_t(B_1, \ldots, B_t): \\ |S_t| = r_t}} |\tilde{x}(S_t)\rangle |\psi_t(S_t)\rangle,$$

where

$$|\psi_t(S_t)\rangle := \sum_{\substack{K_1 \subset \mathcal{C}_1(B_1), \ldots, \\ K_{t-1} \subset \mathcal{C}_{t-1}(B_1, \ldots, B_{t-1}): \\ \forall j, |K_j| = r_j - r_{j+1}, \\ \forall i \neq j, \hat{S}_t \cap \hat{K}_j = \hat{K}_i \cap \hat{K}_j = \emptyset}} \prod_{j=1}^{t-1} \binom{|Z(S_t^{(j)} \cup \bigcup_{i=j}^{t-1} K_i^{(j)})|}{r_j}^{-1/2} |\tilde{x}(K_1), \ldots, \tilde{x}(K_{t-1})\rangle.$$

We will first prove by induction on $t \in \{1, \ldots, k-1\}$ that we can construct $|\psi_t\rangle$ in cost $O\left((r_1 + \sum_{i=1}^{t-1} r_{i+1}\sqrt{n/r_i})(\log n + \log q)\right)$. We will then show how to construct the desired state from $|\psi_{k-1}\rangle$.

Base Case Since $\mathcal{C}_1(B_1) = B_1$, we need only construct the state:

$$|\psi_1\rangle = \binom{|B_1|}{r_1}^{-1/2} \sum_{S_1 \in \binom{B_1}{r_1}} |S_1\rangle |x(S_1)\rangle.$$

We can do this by simply creating a uniform superposition of subsets of B_1, and querying all indices in each set. Using a quantum radix tree to store S_1 and $x(S_1)$, this costs $O(r_1(\log n + \log q))$, as required.

Induction Step Let $t > 1$. We begin by constructing the state $|\psi_{t-1}\rangle$, which has complexity $O((r_1 + \sum_{i=1}^{t-2} r_{i+1}\sqrt{n/r_i})(\log n + \log q))$, by the induction hypothesis. To complete the proof, we will use $|\psi_{t-1}\rangle$ to construct $|\psi_t\rangle$ in time complexity $O(r_t\sqrt{n/r_{t-1}}(\log n + \log q))$. To do this, we will search for r_t indices in $Z(S_{t-1})$, and use them to form r_t t-collisions. The $(t-1)$-tuples in S_{t-1} for which we do not find t-collisions will become K_{t-1}.

The complexity of for indices in $Z(S_{t-1})$ will depend on $|Z(S_{t-1})|$, which varies in S_{t-1}. For a uniform random S_{t-1}, $|Z(S_{t-1})|$ has a hypergeometric distribution with mean $\mu \geq \frac{r_{t-1}n_t}{|B_t|} \geq \frac{r_{t-1}n_t}{n}$, so using tail inequalities from Fact 2.3.1:

$$\Pr\left[|Z(S_{t-1})| \leq \frac{r_{t-1}n_t}{2n}\right] \leq 2\exp\left(-\frac{\mu}{10}\right) = o(1),$$

so we will simply ignore any part of the state $|\psi_{t-1}\rangle$ such that $|Z(S_{t-1})| \leq \frac{r_{t-1}n_t}{2n}$.

We next define a *collision search* operation, which we will apply to the $|S_{t-1}\rangle|x(S_{t-1})\rangle$ registers of $|\psi_{t-1}\rangle$ and some auxiliary space. Let $S_{t-1} \subseteq \mathcal{C}_{t-1}(B_1, \ldots, B_{t-1})$. Then the collision search operator acts as:

$$|S_{t-1}\rangle|x(S_{t-1})\rangle|0\rangle \mapsto |S_{t-1}\rangle|x(S_{t-1})\rangle \sum_{J\subset Z(S_{t-1}):|J|=r_t} |J\rangle|x(J)\rangle.$$

We can implement this mapping using Grover search to look for an index $i \in [n]$ such that $i \in Z(S_{t-1})$. We repeat this r_t times, each time looking for an index not yet found. Let $\tilde{J}$ denote the set of indices in $Z(S_{t-1})$ already found, which we store in a quantum radix tree. Then we can check if an index i is in $Z(S_{t-1}) \setminus \tilde{J}$ in cost $O(\log n + \log q)$ by looking up i in $\tilde{J}$, testing if $i \in A_t$, and looking up x_i in $x(\tilde{J})$. Thus, since we assume $|Z(S_{t-1})| \geq \frac{r_{t-1}n_t}{2n} = \Theta(r_{t-1})$, since $n_t = \Theta(n)$, finding a new i costs $O(\sqrt{n/r_{t-1}}(\log n + \log q))$. Once we have found a new i, we insert i into $\tilde{J}$, and insert (i, x_i) into $x(\tilde{J})$, costing $O(\log n + \log q)$. Thus, the total cost to implement collision search is $O\left(r_t\sqrt{n/r_{t-1}}(\log n + \log q)\right)$.

Applying collision search to $|\psi_{t-1}\rangle|0\rangle$ results in the state (omitting uniform amplitudes):

$$|\psi'_{t-1}\rangle = \sum_{\substack{S_{t-1}\subset\mathcal{C}_{t-1}(B_1,\ldots,B_{t-1}):\\ |S_{t-1}|=r_{t-1}}} |S_{t-1}\rangle|x(S_{t-1})\rangle \binom{|Z(S_{t-1})|}{r_t}^{-1/2} \sum_{J\subset Z(S_{t-1}):|J|=r_t} |J\rangle|x(J)\rangle|\psi_{t-1}(S_{t-1})\rangle.$$

Each index in J forms a t-collision with some $(t-1)$-tuple in S_{t-1}. We want to form t-collisions and insert them into a new set, S_t. We will show how to do this one-by-one for each $j \in J$. Consider a state of the form $|S_{t-1}\rangle|x(S_{t-1})\rangle|j,x_j\rangle|S_t\rangle|x(S_t)\rangle$, where S_t is a (possibly empty) set of elements of the form $(i_1,\dots,i_t)$ with $i_t < j$, stored in a quantum radix tree, and $j \in Z(S_{t-1})$. In particular, let $\vec{i} = (i_1,\dots,i_{t-1})$ be the unique element of S_{t-1} such that $x_j = x_{i_1}$ (it is necessarily unique, because of the assumption that there is only one k-collision and it is in $B_1 \times \cdots \times B_k$). We next describe how to implement the mapping:

$$|S_{t-1}\rangle|x(S_{t-1})\rangle|j,x_j\rangle|S_t\rangle|x(S_t)\rangle \mapsto |S_{t-1}\setminus\{\vec{i}\}\rangle|x(S_{t-1}\setminus\{\vec{i}\})\rangle|0\rangle|S_t\cup\{(\vec{i},j)\}\rangle|x(S_t\cup\{(\vec{i},j)\})\rangle.$$

This is reversible, since $j > i_t$ for all $(i_1,\dots,i_t) \in S_t$. We can implement this mapping roughly as follows:

- Lookup x_j in $x(S_{t-1})$ to find $(i_1,\dots,i_{t-1},x_j)$, and write it into an auxiliary register.
- Delete $(i_1,\dots,i_{t-1})$ from S_{t-1} and $(i_1,\dots,i_{t-1},x_j)$ from $x(S_{t-1})$.
- Insert $(i_1,\dots,i_{t-1},j)$ into S_t and $(i_1,\dots,i_{t-1},j,x_j)$ into $x(S_t)$.
- Lookup x_j in $x(S_t)$ to get some tuple $(\ell_1,\dots,\ell_t,x_j)$, and subtract it from the auxiliary register containing $(i_1,\dots,i_{t-1},x_j)$.
- Lookup the largest index $\ell \in S_t$, and subtract ℓ, x_ℓ from the register containing j, x_j.

Note that we can lookup the largest key in a radix tree by simply traversing the tree as far down as possible, always taking the right child. Since j is the largest, the last step erases j, x_j. The complete procedure costs $O(\log n + \log q)$. For any S_{t-1}, J, define

$$K^{S_{t-1},J} := \{(i_1,\dots,i_{t-1}) : \nexists j \in J \text{ s.t. } x_j = x_{i_1}\},$$

which is the set of tuples in S_{t-1} for which we did not find collisions; and

$$S_t^{S_{t-1},J} := \{(i_1,\dots,i_t) : (i_1,\dots,i_{t-1}) \in S_{t-1}, i_t \in J, x_{i_1} = x_{i_t}\},$$

which is the set of t-collisions we can form from S_{t-1} and J. By repeating the above process $|J| = r_t$ times on $|\psi'_{t-1}\rangle$, in cost $O(r_t(\log n + \log q))$, we get the state $|\psi''_{t-1}\rangle$ defined:

$$\sum_{\substack{S_{t-1}\subset\mathcal{C}_{t-1}(B_1,\dots,B_{t-1}):\\ |S_{t-1}|=r_{t-1}}} \binom{|Z(S_{t-1})|}{r_t}^{-1/2} \sum_{\substack{J\subset Z(S_{t-1}):\\ |J|=r_t}} |\tilde{x}(S_t^{S_{t-1},J})\rangle|\tilde{x}(K^{S_{t-1},J})\rangle|\psi_{t-1}(S_{t-1})\rangle.$$

We will now show that $|\psi''_{t-1}\rangle = |\psi_t\rangle$. First, by observing that $S_{t-1} = (S_t^{S_{t-1},J})^{(t-1)} \cup K^{S_{t-1},J}$, we can rewrite $|\psi''_{t-1}\rangle$ as:

$$\sum_{\substack{S_t \subset \mathcal{C}_t(B_1,\dots,B_t):\\ |S_t|=r_t}} |\tilde{x}(S_t)\rangle \sum_{\substack{K_{t-1}\subset \mathcal{C}_{t-1}(B_1,\dots,B_{t-1}):\\ |K_{t-1}|=r_{t-1}-r_t,\ \hat{S}_t\cap \hat{K}_{t-1}=\emptyset}} \binom{|Z(S_t^{(t-1)}\cup K_{t-1})|}{r_t}^{-1/2} |\tilde{x}(K_{t-1})\rangle |\psi_{t-1}(S_t^{(t-1)}\cup K_{t-1})\rangle.$$

For any S_t, consider

$$|\tilde{\psi}_t(S_t)\rangle := \sum_{\substack{K_{t-1}\subset \mathcal{C}_{t-1}(B_1,\dots,B_{t-1}):\\ |K_{t-1}|=r_{t-1}-r_t,\ \hat{S}_t\cap \hat{K}_{t-1}=\emptyset}} \binom{|Z(S_t^{(t-1)}\cup K_{t-1})|}{r_t}^{-1/2} |\tilde{x}(K_{t-1})\rangle |\psi_{t-1}(S_t^{(t-1)}\cup K_{t-1})\rangle,$$

and recall that $|\psi_{t-1}(S_t^{(t-1)}\cup K_{t-1})\rangle$ is equal to:

$$\sum_{\substack{K_1\subset\mathcal{C}_1(B_1),\dots,\\ K_{t-2}\subset\mathcal{C}_{t-2}(B_1,\dots,B_{t-2}):\\ \forall j, |K_j|=r_j-r_{j+1},\\ \forall i\neq j, (\hat{S}_t^{(t-1)}\cup\hat{K}_{t-1})\cap\hat{K}_j=\hat{K}_i\cap\hat{K}_j=\emptyset}} \prod_{j=1}^{t-2} \binom{|Z((S_t^{(t-1)}\cup K_{t-1})^{(j)})\cup\bigcup_{i=j}^{t-2}K_i^{(j)})|}{r_j}^{-1/2} |\tilde{x}(K_1),\dots,\tilde{x}(K_{t-1})\rangle.$$

Let $\tilde{\mathcal{K}}$ be the set of $(K_1,\dots,K_{t-1})$ on which $|\tilde{\psi}_t(S_t)\rangle$ has nonzero support, and $\mathcal{K}$ the set on which $|\psi_t(S_t)\rangle$ has nonzero support. Note that

$$(\hat{S}_t^{(t-1)}\cup\hat{K}_{t-1})\cap\hat{K}_j = (\hat{S}_t^{(t-1)}\cap\hat{K}_j)\cup(\hat{K}_{t-1}\cap\hat{K}_j) = (\hat{S}_t\cap\hat{K}_j)\cup(\hat{K}_{t-1}\cap\hat{K}_j),$$

since $\hat{S}_t\setminus B_t = \hat{S}_t^{(t-1)}$ and $\hat{K}_j\cap B_t=\emptyset$, so $(\hat{S}_t^{(t-1)}\cup\hat{K}_{t-1})\cap\hat{K}_j=\emptyset$ if and only if $\hat{S}_t\cap\hat{K}_j=\emptyset$ and $\hat{K}_{t-1}\cap\hat{K}_j=\emptyset$, from which it follows that $\mathcal{K}=\tilde{\mathcal{K}}$.

Let $\tilde{\alpha}(K_1,\dots,K_{t-1})$ be the amplitude on $|\tilde{x}(K_1),\dots,\tilde{x}(K_{t-1})\rangle$ in $|\tilde{\psi}_t(S_t)\rangle$. Note that $(S_t^{(t-1)}\cup$

$K_{t-1})^{(j)} = (S_t^{(t-1)})^{(j)} \cup K_{t-1}^{(j)} = S_t^{(j)} \cup K_{t-1}^{(j)}$ We have

$$\begin{aligned}\tilde{\alpha}(K_1, \dots, K_{t-1}) &= \binom{|Z(S_t^{(t-1)} \cup K_{t-1})|}{r_t}^{-1/2} \prod_{j=1}^{t-2} \binom{|Z((S_t^{(t-1)} \cup K_{t-1})^{(j)} \cup \bigcup_{i=j}^{t-2} K_i^{(j)})|}{r_j}^{-1/2} \\ &= \binom{|Z(S_t^{(t-1)} \cup K_{t-1})|}{r_t}^{-1/2} \prod_{j=1}^{t-2} \binom{|Z(S_t^{(j)} \cup \bigcup_{i=j}^{t-1} K_i^{(j)})|}{r_j} \\ &= \prod_{j=1}^{t-1} \binom{|Z(S_t^{(j)} \cup \bigcup_{i=j}^{t-1} K_i^{(j)})|}{r_j},\end{aligned}$$

so $|\tilde{\psi}_t(S_t)\rangle = |\psi(S_t)\rangle$, and thus $|\psi''_{t-1}\rangle = |\psi_t\rangle$, completing the induction step.

Constructing the Desired State from $|\psi_{k-1}\rangle$. To construct the desired state from $|\psi_{k-1}\rangle$, we will simply measure the registers containing $|\psi_{k-1}(S_{k-1})\rangle$ to get some $K = \bigcup_{j=1}^{k-2} \hat{K}_j$. The remaining superposition, $|\psi\rangle$, over states $|S_{k-1}\rangle$ will not have nonzero support on all possible subsets of $\mathcal{C}_{k-1}(B_1, \dots, B_{k-1})$: rather, it will have nonzero support on all $S_{k-1} \subset \mathcal{C}_{k-1}(B_1, \dots, B_{k-1})$ such that $|S_{k-1}| = r_{k-1}$ and $\hat{S}_{k-1} \cap K = \emptyset$. For all $i \in \{1, \dots, k-1\}$, define $A_i := B_i \setminus K$. Then $|\psi\rangle$ has nonzero support on $\binom{\mathcal{C}(A_1, \dots, A_{k-1})}{r_{k-1}}$. To show that $|\psi\rangle$ is a uniform superposition, we will show that the relative amplitudes of each possible $(K_1, \dots, K_{k-2})$ have no dependence on S_{k-1}. Define

$$\alpha(K_1, \dots, K_{k-2}, S_{k-1}) := \prod_{j=1}^{k-2} \binom{|Z(S_{k-1}^{(j)} \cup \bigcup_{i=j}^{k-2} K_i^{(j)})|}{r_j}.$$

For all $j \in \{1, \dots, k-2\}$, define:

$$S_j := S_{k-1}^{(j)} \cup \bigcup_{i=j}^{k-2} K_i^{(j)}.$$

Lemma 6.2.6. *Let $A_k = [n] \setminus (\bigcup_{i=1}^{k-1} A_i)$, and suppose the unique k-collision is in $A_1 \times \cdots \times A_k$. Then for all $j \in \{1, \dots, k-2\}$, $|Z(S_j)| = |Z(S_j \setminus S_{k-1}^{(j)})| + r_{k-1}$.*

Proof. Let $\overline{S_{k-1}^{(j)}} := \hat{S}_{k-1} \cap A_j$, and note that $|\overline{S_{k-1}^{(j)}}| = |S_{k-1}| = |S_{k-1}^{(j)}|$, since $\overline{S_{k-1}^{(j)}}$ contains exactly the last member of each tuple in $S_{k-1}^{(j)}$.

We first observe that $Z(S_{k-1}^{(j)}) = \overline{S_{k-1}^{(j+1)}}$. To see this, since it's clear that $\overline{S_{k-1}^{(j+1)}} \subseteq Z(S_{k-1}^{(j)})$, suppose there is some $i'_{j+1} \in Z(S_{k-1}^{(j)}) \setminus \overline{S_{k-1}^{(j+1)}}$. Since $i'_{j+1} \in Z(S_{k-1}^{(j)})$, let $(i_1, \ldots, i_j) \in S_{k-1}^{(j)}$ be such that $x_{i_1} = x_{i'_{j+1}}$. Let $i_{j+1}, \ldots, i_{k-1}$ be such that $(i_1, \ldots, i_{k-1}) \in S_{k-1}$. Then since $i'_{j+1} \neq i_{j+1}$, $(i_1, \ldots, i_{k-1}, i'_{j+1})$ is a k-collision in $[n] \setminus A_k$, which is a contradiction.

Next, we observe that $\overline{S_{k-1}^{(j+1)}} \cap Z(S_j \setminus S_{k-1}^{(j)}) = \emptyset$. Suppose not, and let $i'_{j+1} \in \overline{S_{k-1}^{(j+1)}} \cap Z(S_j \setminus S_{k-1}^{(j)})$. Since $i_{j+1} \in Z(S_j \setminus S_{k-1}^{(j)})$, there exists $(i'_1, \ldots, i'_j) \in S_j \setminus S_{k-1}^{(j)}$ such that $x_{i_1} = x_{i'_{j+1}}$. Since $i'_{j+1} \in \overline{S_{k-1}^{(j+1)}}$, let $i_1, \ldots, i_j, i_{j+1}, \ldots, i_{k-1}$ be such that $(i_1, \ldots, i_{k-1}) \in S_{k-1}$. Then since $(i'_1, \ldots, i'_j) \notin S_{k-1}^{(j)}$, $(i_1, \ldots, i_j) \neq (i'_1, \ldots, i'_j)$. Suppose $i_\ell \neq i'_\ell$ for $\ell \in [j]$. Then $(i_1, \ldots, i_{k-1}, i'_\ell)$ is a k-collision in $[n] \setminus A_k$, which is a contradiction.

Then, since for any sets S and S', $Z(S \cup S) = Z(S) \cup S(S')$, we have:

$$\begin{aligned}
|Z(S_j)| &= |Z(S_j \setminus S_{k-1}^{(j)}) \cup Z(S_{k-1}^{(j)})| \\
&= |Z(S_j \setminus S_{k-1}^{(j)})| + |Z(S_{-1}^{(j)})| - |Z(S_j \setminus S_{k-1}^{(j)}) \cap Z(S_{k-1}^{(j)})| \\
&= |Z(S_j \setminus S_{k-1}^{(j)})| + |\overline{S_{k-1}^{(j+1)}}| - |Z(S_j \setminus S_{k-1}^{(j)}) \cap \overline{S_{k-1}^{(j+1)}}| \\
&= |Z(S_j \setminus S_{k-1}^{(j)})| + r_{k-1}.
\end{aligned}$$

Note that this proof relies heavily on the assumption that there is a unique k-collision. This is a barrier to constructing uniform superpositions over sets of t-collisions when $1 < t < k-1$, which would allow us to implement more clever algorithms for k-distinctness with better complexity. $\square$

Thus, we have

$$\alpha(K_1, \ldots, K_{k-2}, S_{k-1}) = \prod_{j=1}^{k-2} \binom{|Z(\bigcup_{i=j}^{k-2} K_i^{(j)})| + r_{k-1}}{r_j},$$

which is independent of S_{k-1}. Thus, under the assumption that the unique k-collision is in $A_1 \times \cdots \times A_k$, $|\psi\rangle$ is a uniform superposition over $\binom{\mathcal{C}_{k-1}(A_1,\ldots,A_{k-1})}{r_{k-1}}$. We complete the proof by arguing that this happens with constant probability.

Let $(a_1, \ldots, a_k)$ be the unique k-collision in x. Since $B_1, \ldots, B_k$ is a uniform k-wise independent partition of $[n]$, $\Pr[a_1 \in B_1, \ldots, a_k \in B_k] = \frac{1}{k^k}$. Suppose $(a_1, \ldots, a_k) \in B_1 \times \cdots \times B_k$. Then the only way we can have $(a_1,\dots,a_k)\notin A_1\times\cdots\times A_k$ is if $a_i\in K$ for some $i\in[k-1]$. Since $|K|=\sum_{j=1}^{k-2}|\hat{K}_j|=\sum_{j=1}^{k-2}j(r_j-r_{j+1})=o(n)$, this happens with probability $o(1)$, so for large enough n, we have $\Pr[a_1\in A_1,\dots,a_n\in A_n]\geq\frac{1}{2k^k}$.

Finally, if we store K in a radix tree, we can efficiently compute to which set $i\in[n]$ belongs as follows. We first check if $i\in K$, in which case, we know $i\in A_k$. Otherwise, we compute to which of $B_1,\dots,B_k$ i belongs. □

Checking Subroutine In order to check if a state S_{k-1} is marked, we will search A_k for an index a_k such that $x_{a_1}=\cdots=x_{a_k}$ for some $(a_1,\dots,a_{k-1})\in S_{k-1}$. We can accomplish this task using Grover search over A_k. For any $a_k\in A_k$, we can check if it has the desired property by looking up x_{a_k} in the quantum radix tree storing $|d(S_{k-1})\rangle$. This has time complexity $O(\log n)$. Since we are searching a set of size $|A_k|=\frac{n}{k}=O(n)$, we can determine if such an a_k exists with bounded error in $O(\sqrt{n})$ such data structure queries. Thus, the checking procedure has total time complexity $\mathsf{C}=O(\sqrt{n}\log n)$.

Inner Update Subroutines To implement the inner update, we need to implement the local diffusion for $\mathcal{W}^{S_{k-1}}$:

$$|S_1,0\rangle\mapsto|S_1\rangle\sum_{i\in S_1,j\in([n]\setminus\hat{S}_{k-1})\setminus S_1}\frac{1}{\sqrt{r_1(n-(k-1)r_{k-1}-r_1)}}|i\rangle|j\rangle;$$

and the data swap for $\mathcal{W}^{S_{k-1}}$:

$$|S_1,S_1'\rangle|d'(S_1)\rangle\mapsto|S_1',S_1\rangle|d'(S_1')\rangle.$$

We will begin with the local diffusion. To construct $\sum_{i\in S_1}\frac{1}{\sqrt{r_1}}|i\rangle$, controlled on $|S_1\rangle$, we simply access the elements of the quantum radix tree storing S_1 in superposition, which we can do in time $O(\log n)$. To construct $\sum_{j\in([n]\setminus\hat{S}_{k-1})\setminus S_1}\frac{1}{\sqrt{n-(k-1)r_{k-1}-r_1}}|j\rangle$, we can construct $\sum_{j\in[n]}\frac{1}{\sqrt{n}}|j\rangle$, and then bring it as close as desired to the required state using a constant number of rounds of amplitude amplification, where at each round we check, in time $O(\log n)$, if j in S_1 or S_{k-1}. Thus the local diffusion costs $O(\log n)$.

To implement the data swap, on an edge (S_1,S_1') with $S_1\setminus S_1'=\{i\}$ and $S_1'\setminus S_1=j$, we first note that we can map $|S_1,S_1'\rangle=|S_1\rangle|i,j\rangle$ to $|S_1',S_1\rangle=|S_1',j,i\rangle$ by inserting j into S_1, and removing i from S_1, in cost $O(\log n)$, and then mapping $|i,j\rangle\mapsto|j,i\rangle$ in cost $O(1)$. Finally, to map $|d(S_1)\rangle$ to $|d(S_1')\rangle$, controlled on $|j,i\rangle$, we query $|x_j,x_i\rangle$ in an auxiliary register, remove (i,x_i) from $|d(S_1)\rangle$ and add (j,x_j), in cost $O(\log q)$, and 2 queries.

Thus, the total update cost of $\mathcal{W}^{S_{k-1}}$ is $\mathsf{U}'=O(\log n+\log q)$.

Inner Checking Subroutine In order to check if a state S_1 is marked, we need to determine if it has at least m $(k-1)$-collisions from $\mathcal{C}_{k-1}$, which we can do by simply checking the collision counter of the augmented quantum radix tree storing $d'(S_1) = x(S_1)$. Thus, the time complexity of the inner checking step is $\mathsf{C}' = O(1)$.

Extraction Subroutine To implement the extraction, we must implement the map (omitting uniform amplitudes):

$$|S_{k-1}, 0\rangle \sum_{S_1 \in M^{S_{k-1}}} |S_1, 0\rangle|d'(S_1)\rangle \mapsto |S_{k-1}\rangle \sum_{\substack{I \subset S_{k-1}: \\ |I|=m}} |I\rangle \sum_{\substack{J \subset \mathcal{C}_{k-1} \setminus S_{k-1}: \\ |J|=m}} |J\rangle|\Gamma\left(S_{k-1}, (S_{k-1} \cup J) \setminus I\right)\rangle,$$

for some states Γ. This extraction will be carried out just as in the case of 3-distinctness from Section 6.2, however, since we are now concerned with time complexity, our analysis must be more detailed.

To construct $\sum_{I \subset S_{k-1}:|I|=m} \frac{1}{\sqrt{\binom{r_{k-1}}{m}}}|I\rangle$, we simply access S_{k-1} in superposition m times, in cost $O(m \log n)$. To construct the second part, we will use the fact that $M^{S_{k-1}}$ consists of subsets of $[n]$ that contain at least m $(k-1)$-collisions not already in S_2, and extract these to form J, letting whatever is leftover be the garbage. We first implement the map:

$$|0\rangle|S_1, d'(S_1)\rangle \mapsto \sum_{J \subset \mathcal{C}_{k-1}(S_1):|J|=m} \frac{1}{\sqrt{\binom{|\mathcal{C}_{k-1}(S_1)|}{m}}}|J\rangle|S_1, d'(S_1)\rangle,$$

extracting every possible set of m $(k-1)$-collisions from S_1 in superposition. We can construct a uniform superposition of $(k-1)$-collisions in S_1 from the augmented quantum radix tree storing $d'(S_1)$ in cost $O(\log q)$. Repeating this m times gives the desired state in time $O(m \log q)$. Applying this procedure to $|M^{S_{k-1}}\rangle$ results in the state (omitting uniform amplitudes):

$$\sum_{S_1 \in M^{S_{k-1}}} \sum_{\substack{J \subseteq \mathcal{C}_{k-1}(S_1): \\ |J|=m}} \frac{1}{\sqrt{\binom{|\mathcal{C}_{k-1}(S_1)|}{m}}}|J\rangle|S_1\rangle|d'(S_1)\rangle$$

$$= \sum_{\substack{J \subseteq \mathcal{C}_{k-1} \setminus S_{k-1}: \\ |J|=m}} |J\rangle \sum_{S_1 \in M^{S_{k-1}}: \hat{J} \subset S_1} \frac{1}{\sqrt{\binom{|\mathcal{C}_{k-1}(S_1)|}{m}}}|S_1\rangle|d'(S_1)\rangle.$$

We can define the garbage as what we happen to have leftover. For any S'_{k-1} such that $S'_{k-1} \setminus S_{k-1} = J$, we define the garbage state:

$$|\Gamma(S_{k-1}, S'_{k-1})\rangle := \sqrt{\frac{|M^{S_{k-1}}|}{\binom{n_{k-1}-r_{k-1}}{m}}} \sum_{S_1 \in M^{S_{k-1}}: \hat{J} \subset S_1} \frac{1}{\sqrt{\binom{|\mathcal{C}_{k-1}(S_1)|}{m}}}|S_1\rangle|d'(S_1)\rangle.$$

With these garbage states, we can implement the extraction in time $\mathsf{E} = O(m(\log n + \log q))$.

Garbage Swap Having defined the garbage states, we can now describe how to implement the garbage swap, which is similar to the 3-distinctness case seen in Section 6.2. This must act as

$$|S_{k-1}, S'_{k-1}\rangle|d(S_{k-1})\rangle|\Gamma(S_{k-1}, S'_{k-1})\rangle \mapsto |S'_{k-1}, S_{k-1}\rangle|d(S'_{k-1})\rangle|\Gamma(S'_{k-1}, S_{k-1})\rangle.$$

Let $I = S_{k-1} \setminus S'_{k-1}$ and $J = S'_{k-1} \setminus S_{k-1}$. To implement the mapping

$$\begin{aligned}|d(S_{k-1})\rangle &= |\{(i_1, \dots, i_{k-1}, x_{i_1}) : (i_1, \dots, i_{k-1}) \in S_{k-1}\}\rangle \\ &\mapsto |d(S'_{k-1})\rangle = |\{(i_1, \dots, i_{k-1}, x_{i_1}) : (i_1, \dots, i_{k-1}) \in S'_{k-1}\}\rangle,\end{aligned}$$

we must remove $x(I) = \{(i_1, \dots, i_{k-1}, x_{i_1}) : (i_1, \dots, i_{k-1}) \in I\}$ and add $x(J) = \{(i_1, \dots, i_{k-1}, x_{i_1}) : (i_1, \dots, i_{k-1}) \in J\}$. To implement this, we will remove J and $x(J)$ from S_1 and $d'(S_1)$, and replace them with I and $x(I)$. We do this using the map (controlled on $|S_{k-1}, S'_{k-1}\rangle$):

$$G : |d(S_{k-1})\rangle|S_1\rangle|d'(S_1)\rangle \mapsto |d((S_{k-1} \setminus I) \cup J\rangle|(S_1 \setminus \hat{J}) \cup \hat{I}\rangle|d'((S_1 \setminus \hat{J}) \cup \hat{I})\rangle.$$

Note that since $|S_{k-1}, S'_{k-1}\rangle$ encodes both I and J, we can reversibly perform this mapping by doing the following for each $(i_1, \dots, i_{k-1}) \in I$:

- Remove $(i_1, \dots, i_{k-1}, x_{i_1})$ from $|d(S_{k-1})\rangle$ ($O(\log q)$).
- For each $t \in [k-1]$, insert i_t into $|S_1\rangle$, and (i_t, x_{i_1}) into $|d(S_1)\rangle$ ($O(\log q + \log n)$).

and a similar (but reversed) set of operations for each $(i_1, \dots, i_{k-1}) \in J$. The total cost of this is $O(m(\log q + \log n))$. Recall from Section 6.2 that the map $\varphi(S_1) = (S_1 \setminus \hat{J}) \cup \hat{I}$ is a bijection from $M^{S_{k-1}}$ to $M^{S'_{k-1}}$ such that $|\mathcal{C}_{k-1}(S_1)| = |\mathcal{C}_{k-1}(\varphi(S_1))|$. Then, applying G results in (omitting uniform amplitudes):

$$\begin{aligned} & |d(S_{k-1})\rangle|\Gamma(S_{k-1}, S'_{k-1})\rangle \\ \mapsto\ & |d((S_{k-1} \setminus I) \cup J)\rangle \sum_{S_1 \in M^{S_{k-1}} : \hat{J} \subset S_1} \frac{1}{\sqrt{\binom{|\mathcal{C}_{k-1}(S_1)|}{m}}} |\varphi(S_1)\rangle|d'(\varphi(S_1))\rangle \\ =\ & |d(S'_{k-1})\rangle \sum_{S_1 \in M^{S'_{k-1}} : \hat{I} \subset S_1} \frac{1}{\sqrt{\binom{|\mathcal{C}_{k-1}(S_1)|}{m}}} |S_1\rangle|d'(S_1)\rangle \ =\ |d(S'_{k-1})\rangle|\Gamma(S'_{k-1}, S_{k-1})\rangle. \end{aligned}$$

Thus we can implement the garbage swap in cost $\mathsf{G} = O(m(\log n + \log q))$.

Final Analysis

Applying Theorem 6.1.1, we can compute the time complexity of our nested-update quantum walk algorithm, (neglecting logarithmic factors, and assuming $r_1^{k-1} > n^{k-2}$):

$$\begin{aligned}
& \mathsf{S} + \frac{1}{\sqrt{\varepsilon}}\left(\frac{1}{\sqrt{\delta}}\left(\frac{1}{\sqrt{\varepsilon'}}\left(\frac{1}{\sqrt{\delta'}}\mathsf{U}' + \mathsf{C}'\right) + \mathsf{E} + \mathsf{G}\right) + \mathsf{C}\right) \\
= \quad & r_1 + \sum_{i=1}^{k-2} r_{i+1}\sqrt{\frac{n}{r_i}} + \sqrt{\frac{n}{r_{k-1}}}\left(\sqrt{\frac{r_{k-1}}{m}}\left(\left(\sqrt{r_1}+1\right) + m + m\right) + \sqrt{n}\right) \\
= \quad & r_1 + \sum_{i=1}^{k-2} r_{i+1}\sqrt{\frac{n}{r_i}} + \frac{n^{(k-1)/2}}{r_1^{(k-2)/2}} + \frac{r_1^{(k-1)/2}}{n^{(k-3)/2}} + \frac{n}{\sqrt{r_{k-1}}}.
\end{aligned}$$

3-Distinctness In optimizing the parameters, there are now two cases: $k = 3$ or $k > 3$. We first suppose $k = 3$. In that case, the complexity expression becomes:

$$r_1 + r_2\sqrt{\frac{n}{r_1}} + \frac{n}{r_1^{1/2}} + r_1 + \frac{n}{\sqrt{r_2}}.$$

Just as in the case of the query complexity, this is optimized by setting $r_1 = n^{5/7}$ and $r_2 = n^{4/7}$. This gives total time complexity $\widetilde{O}\left(n^{5/7}\right)$. In this case, the dominating terms are the first, second, and fourth terms.

***k*-Distinctness** When $k > 3$, the optimization of terms works out differently. In that case, we set $r_1 = n^{\frac{k-1}{k}}$, optimizing the first and third-last terms, and set $r_i = \sqrt{n}$ for all $i > 1$, so that all but the first term in the sum become $n^{3/4}$. This gives time complexity $\widetilde{O}\left(n^{\frac{k-1}{k}}\right)$, completing the proof of Theorem 6.2.1.

Part III

Approximate Span Programs

Chapter 7

Approximate Span Programs

Span programs are a model of computation first used to study logspace complexity [KW93], and more recently, introduced to the study of quantum query complexity in [RŠ12]. The main practical use of span programs stems from a general construction that takes a span program and constructs a quantum algorithm for the associated decision problem with quantum query complexity matching the "complexity" of the span program [Rei09, Rei11]. When we construct and run this quantum algorithm, we say that we *evaluate* the span program.

It is known that span programs are equivalent to bounded error quantum query complexity of Boolean functions, in the sense that, for every span program, we can construct an algorithm that decides the related function in bounded error query complexity that is within a constant factor of the span program complexity, and for every Boolean function, there is an optimal span program for that function, whose complexity is within a constant of the bounded error quantum query complexity of the function. However, there is no known construction that takes a bounded error quantum query algorithm and converts it to a span program with matching complexity (up to a constant) compared with a simple construction, discussed in Section 7.2, that takes an algorithm with *one-sided* error and converts it to a span program with matching complexity [Rei09, Section 3]. Interestingly, this conversion even preserves the time complexity, in the sense that the time complexity of evaluating the resulting span program is within a constant factor of the time complexity of the original algorithm. Although nothing practical is gained from this construction, it is unsatisfying that despite the known relationship between bounded error query complexity and span programs, this construction only works for one-sided error, suggesting that perhaps the definition of span programs currently used is not the ideal definition for modeling bounded error quantum query complexity.

The problem in extending this idea beyond 1-sided error seems to be that span programs have a somewhat *exact* nature, in the sense that an input is said to be "accepted" by a span program (that is, the associated function is 1 on that input) if and only if the target is in the span of the available vectors (see Definition 7.0.7), and is "rejected" even if it is *almost* in the span of these vectors.

Motivated by this unsatisfying state of affairs, we study the concept of *approximate span programs*. We begin by giving a slightly modified definition of a span program that has the advantage of working with non-Boolean alphabets without incurring any logarithmic (in the input alphabet size) factors, in contrast to the previous definition. We then relax the notion of the associated function of a span program, allowing us to consider a single span program $\mathcal{P}$ with a number of possible functions f, with which $\mathcal{P}$ has a more approximate relationship. We then view this pair $(\mathcal{P}, f)$ as an *approximate span program*. We show that this new definition of approximate span programs is actually meaningful by giving an algorithm to evaluate an approximate span program. That is, for any $(\mathcal{P}, f)$ with the approximate relationship we will define, we can construct an algorithm from $\mathcal{P}$, that solves f, and whose query complexity is equal to the complexity of $\mathcal{P}$. The contents of this chapter are based on joint work with Tsuyoshi Ito.

Span Programs for non-Boolean Alphabets To begin, we slightly modify the definition of a span program to facilitate its extension to non-Boolean alphabets. We use the following definition.

Definition 7.0.7. *A span program $\mathcal{P} = (H, V, \tau, A)$ on $[q]^n$ consists of*

1. *finite-dimensional complex inner product spaces $H = H_1 \oplus \cdots \oplus H_n \oplus H_{\text{true}} \oplus H_{\text{false}}$, and $\{H_{j,a} \subseteq H_j\}_{j\in[n],a\in[q]}$ such that $H_j = H_{j,1} + \cdots + H_{j,q}$,*
2. *a vector space V,*
3. *a* target vector *$\tau \in V$, and*
4. *a linear operator $A \in \mathcal{L}(H, V)$.*[1]

Furthermore, for a string $x \in [q]^n$, we define $H(x) := H_{1,x_1} \oplus \cdots \oplus H_{n,x_n} \oplus H_{\text{true}}$ and say that these vectors are available *to x (in a slight abuse of terminology, since typically $V(x) := AH(x)$ are called the available vectors for x).*

For a function $f : D \to \{0, 1\}$, $D \in [q]^n$, we say that $\mathcal{P}$ decides *f if*

1. *for every $x \in f^{-1}(1)$ there exists $|w\rangle \in H(x)$ such that $A|w\rangle = \tau$ (such a $|w\rangle$ is called a* positive witness*), and*
2. *for every $x \in f^{-1}(0)$ there is no $|w\rangle \in H(x)$ such that $A|w\rangle = \tau$, or equivalently, there exists a linear mapping $\omega \in \mathcal{L}(V, \mathbb{C})$ such that $\omega(\tau) = 1$ and $\left\|\omega A \Pi_{H(x)}\right\| = 0$ (such an ω is called a* negative witness*).*

[1]Wlog, we can assume $V = \text{im}(A)$.

For each $x \in f^{-1}(1)$, we define

$$\mathrm{wsize}_+(x,\mathcal{P}) := \min_{|w\rangle \in H(x) : A|w\rangle = \tau} \||w\rangle\|^2 \qquad \textit{and} \qquad \mathrm{wsize}_-(x,\mathcal{P}) := \infty.$$

For each $x \in f^{-1}(0)$, we define

$$\mathrm{wsize}_-(x,\mathcal{P}) := \min_{\omega \in \mathcal{L}(V,\mathbb{C}) : \omega(\tau)=1, \omega A \Pi_{H(x)}=0} \|\omega A\|^2 \qquad \textit{and} \qquad \mathrm{wsize}_+(x,\mathcal{P}) := \infty.$$

We define the positive and negative complexities of $\mathcal{P}$ as a span program for f as

$$W_+(\mathcal{P},f) := \max_{x \in f^{-1}(1)} \mathrm{wsize}_+(x,\mathcal{P}) \qquad \textit{and} \qquad W_-(\mathcal{P},f) := \max_{x \in f^{-1}(0)} \mathrm{wsize}_-(x,\mathcal{P}).$$

The complexity *of $\mathcal{P}$ as a span program for f is defined $C(\mathcal{P},f) := \sqrt{W_+(\mathcal{P},f)W_-(\mathcal{P},f)}$.*

There are a number of cosmetic differences between Definition 7.0.7 and previous span program definitions. One that we note is the use of spaces H_{true} and H_{false}. We can view these spaces in the following manner: for every input $x \in [q]^n$, we append a 1 to x to get $x' = (x,1) \in [q]^{n+1}$. Then every string x' has a 1 in the $(n+1)$th position, so H_{true} functions as $H_{n+1,1}$, available to every input x' obtained in this manner, whereas H_{false} acts as $H_{n+1,0}$, never available to any of the x'. Thus, it is never necessary, but often convenient, to use these spaces. The image of H_{true} under A is similar to the space V_{free} often used in earlier definitions of span programs (though also for convenience, not necessity). For convenience, we do count the part of a positive witness on H_{true} in the witness size.

Aside from these cosmetic differences, the only substantial difference between our definition and previous definitions for non-Boolean input alphabets is that the spaces $\{H_{j,a}\}_a$ need not be orthogonal. This allows us to prove the following theorem, in Section 7.1, which was known before only for the case $q = 2$ [Rei09, Rei11].

Theorem 7.0.8. *Let $f : D \to \{0,1\}$ for $D \subseteq [q]^n$ be a function with bounded error quantum query complexity $Q(f)$. There exists a span program on $[q]^n$, $\mathcal{P}$, that decides f, such that $C(\mathcal{P},f) = \Theta(Q(f))$.*

The rest of this chapter is organized as follows. In Section 7.1, we prove Theorem 7.0.8. In Section 7.2, we present a method for converting a quantum algorithm `A` that decides $f : D \to \{0,1\}$, $D \subseteq [q]^n$, with one-sided error, to a span program $\mathcal{P}$ on $[q]^n$, so that the quantum query complexity of `A` is equal to $C(\mathcal{P},f)$, up to a multiplicative constant. This construction only works when `A` has one-sided error, which motivates us to explore approximate notions of span programs in Section 7.3. In Section 7.3, we discuss several possible notions of approximate span programs, and show that they are all, in fact, the same. In Section 7.3.2, we show how the construction for converting a one-sided algorithm to a span program can be adapted to convert a two-sided error algorithm to an approximate span program.

To show that these new approximate span programs are meaningful, we need to present an algorithm to evaluate them. With that in mind, in Section 7.4, we explore the structure of witnesses in order to better understand span programs, and in Section 7.5, we look at various transformations we can perform on a span program. The results of these two sections allow us to give a fairly intuitive algorithm for evaluating span programs in Section 7.6.1. A variant of this span program evaluation algorithm, shown in Section 7.6.2, can be used to approximately evaluate a span program in a sense that makes our approximate definition of span programs meaningful.

7.1 Span Programs on non-Boolean Alphabets

We now prove that with our slightly modified definition of span program, there exists an optimal span program for any decision problem f. Previously, this was only known to be possible for functions over Boolean alphabets; for more general alphabets, it was only known up to logarithmic factors in q [Rei09]. Our proof will use the dual adversary bound, which we define shortly. The adversary lower bound is a semidefinite program parameterized by any function f, which gives a lower bound on $Q(f)$, the bounded error quantum query complexity of f [Amb00, HLŠ07]. As we formally state in Theorem 7.1.2, it was shown that the adversary lower bound can, in fact, give an optimal (up to a constant) lower bound for f [Rei09, Rei11], and so its dual formulation, stated below, gives an upper bound on $Q(f)$. It can be defined for non-decision problems, but we give the decision version, as it is sufficient, and well-stated, for our purposes.

Definition 7.1.1 (Dual Adversary Bound)**.** *Let $f : D \to \{0,1\}$ be some problem with domain $D \subseteq [q]^n$. The* dual adversary bound *of f is the following optimization problem, where $F_0 = f^{-1}(0)$ and $F_1 = f^{-1}(1)$.*

$$\begin{aligned}
\mathrm{Adv}(f) \quad &= \quad \text{minimize} \sqrt{\left(\max_{x\in F_0}\sum_{j\in[n]}\||v_{x,j}\rangle\|^2\right)\left(\max_{x\in F_1}\sum_{j\in[n]}\||u_{x,j}\rangle\|^2\right)} \\
\text{subject to} \quad & \quad \{|u_{x,j}\rangle\}_{j\in[n],x\in F_1}, \{|v_{x,j}\rangle\}_{j\in[n],x\in F_0} \subset \mathbb{R}^m \textit{ for some } m \in \mathbb{N}, \\
& \quad \forall (x,y) \in F_0 \times F_1, \sum_{j:x_j\neq y_j} \langle v_{x,j}|u_{y,j}\rangle = 1.
\end{aligned}$$

The following theorem tells us that the optimal solution to the dual adversary bound is equal to the quantum query complexity of f, up to a constant.

Theorem 7.1.2 ([Rei09, Rei11])**.** *Let $f : D \to \{0,1\}$ be some problem with domain $D \subseteq [q]^n$. Then $Q(f) = \Theta(\mathrm{Adv}(f))$.*

We will make use of the following useful set of vectors, from [LMR+11].

Fact 7.1.3. *For any $k \in \mathbb{N}$, there exist unit vectors $\{|\mu_i\rangle\}_{i=1}^k$ and $\{|\nu_j\rangle\}_{j=1}^k$ such that*

$$\langle \mu_i|\nu_j\rangle = \begin{cases} \frac{1}{2}\frac{k}{k-1} & \text{if } i \neq j \\ 0 & \text{else.} \end{cases}$$

Proof. We can define $\mu_i = -\alpha|i\rangle + \sqrt{\frac{1-\alpha^2}{k-1}}\sum_{j\neq i}|j\rangle$ and $\nu_j = \sqrt{1-\alpha^2}|i\rangle + \frac{\alpha}{\sqrt{k-1}}\sum_{j\neq i}|j\rangle$, for $\alpha = \sqrt{\frac{1}{2} - \frac{\sqrt{k-1}}{k}}$ to satisfy the desired property. $\square$

We are now ready to prove the main theorem of this section, which immediately proves Theorem 7.0.8.

Theorem 7.1.4. *Let $f : D \to \{0,1\}$ be some problem with domain $D \subseteq [q]^n$. Then there exists a span program $\mathcal{P}$ on $[q]^n$ such that*

$$C(\mathcal{P}, f) \leq 2\mathrm{Adv}(f).$$

Proof. Let $\{|u_{x,j}\rangle\}_{j\in[n],x\in F_1}$ and $\{|v_{x,j}\rangle\}_{j\in[n],x\in F_0}$ in $\mathbb{R}^m$ be a dual adversary solution. We will construct a span program $\mathcal{P}$ for f that has complexity at most

$$2\sqrt{\left(\max_{x\in F_0}\sum_{j\in[n]}\||v_{x,j}\rangle\|^2\right)\left(\max_{x\in F_1}\sum_{j\in[n]}\||u_{x,j}\rangle\|^2\right)}.$$

We will make use of the vectors from Fact 7.1.3, $\{|\nu_a\rangle, |\mu_a\rangle\}_{a\in[q]}$.

The Span Program We define $\mathcal{P}$ as follows. We will work in the inner product space

$$H := \mathrm{span}\{|\alpha, j, a\rangle : \alpha \in [m], j \in [n], a \in [q]\},$$

which we decompose into

$$H_j := \mathrm{span}\{|\alpha, j, a\rangle : \alpha \in [m], a \in [q]\}.$$

Within each H_j, we define, for each $a \in [q]$,

$$H_{j,a} := \mathrm{span}\{|\alpha, j\rangle : \alpha \in [m]\} \otimes (\mathrm{span}\{|\nu_a\rangle\})^{\perp}.$$

Note that the $H_{j,a}$ and $H_{j,b}$ for $a \neq b$ are not orthogonal as is required in the usual definition of span programs. With our modification, however, this is allowed. This is the only place that we use our modification to the definition; the rest of the proof is identical to the Boolean case.

Next, we define:

$$V := \text{span}\{|x\rangle : x \in F_0\},\ |\tau\rangle := \sum_{x \in F_0} |x\rangle$$

and finally, we define $A \in \mathcal{L}(H, V)$ by $A|\alpha, j, a\rangle = \sum_{x\in F_0} \langle \upsilon_{x,j}|\alpha\rangle\langle \nu_{x_j}|a\rangle|x\rangle$. Then we can write:

$$\begin{aligned}
A &= \sum_{\alpha\in[m], j\in[n], a\in[q]} \sum_{x\in F_0} \langle \upsilon_{x,j}|\alpha\rangle\langle \nu_{x_j}|a\rangle|x\rangle\langle \alpha, j, a| \\
&= \sum_{j\in[n]} \sum_{x\in F_0} |x\rangle \left(\sum_{\alpha\in[m]} \langle \upsilon_{x,j}|\alpha\rangle\langle\alpha|\right)\langle j| \left(\sum_{a\in[m]} \langle \nu_{x_j}|a\rangle\langle a|\right) \\
&= \sum_{j\in[n]} \sum_{x\in F_0} |x\rangle\langle \upsilon_{x,j}|\langle j|\langle \nu_{x_j}|.
\end{aligned}$$

Negative Analysis Suppose $x \in F_0$. Then let $\omega_x := \langle x|$. So we have:

$$\omega_x A = \langle x|A = \sum_{j\in[n]} \langle \upsilon_{x,j}|\langle j|\langle \nu_{x_j}|.$$

Since for all j, $H_{j,x_j} = \text{span}\{|\alpha, j\rangle\} \otimes \left(\text{span}\{|\nu_{x_j}\rangle\}\right)^{\perp}$, we have $\langle x|A\Pi_{H(x)} = 0$. Furthermore, we have

$$\omega_x|\tau\rangle = \langle x| \sum_{y\in F_0} |y\rangle = 1,$$

so ω_x is a negative witness for x. It has complexity:

$$\left\|\langle x|A\right\|^2 = \left\|\sum_{j\in[n]} \langle \upsilon_{x,j}|\langle j|\langle \nu_{x_j}|\right\|^2 = \sum_{j\in[n]} \left\||\upsilon_{x,j}\rangle\right\|^2,$$

since $\left\||\nu_{x_j}\rangle\right\| = 1$. Thus,

$$W_-(\mathcal{P}) \leq \max_{x\in F_0} \sum_{j\in[n]} \left\||\upsilon_{x,j}\rangle\right\|^2.$$

Positive Analysis On the other hand, suppose $x \in F_1$. Then define

$$|w_x\rangle = \frac{2(q-1)}{q} \sum_{j\in[n]} |u_{x,j}\rangle|j\rangle|\mu_{x_j}\rangle.$$

Note that since $\langle \mu_{x_j}|\nu_{x_j}\rangle = 0$, $|u_{x,j}\rangle|j\rangle|\mu_{x_j}\rangle \in H_{j,x_j}$, so $|w_x\rangle \in H(x)$. Furthermore, we have

$$\begin{aligned} A|w_x\rangle &= \frac{2(q-1)}{q} \sum_{j\in[n]} \sum_{y\in F_0} |y\rangle\langle v_{y,j}|\langle j|\langle \nu_{y_j}| \sum_{j\in[n]} |u_{x,j}\rangle|j\rangle|\mu_{x_j}\rangle \\ &= \frac{2(q-1)}{q} \sum_{y\in F_0} |y\rangle \sum_{j\in[n]} \langle v_{y,j}|u_{x,j}\rangle\langle \nu_{y_j}|\mu_{x_j}\rangle \\ &= \sum_{y\in F_0} |y\rangle \sum_{j\in[n]:x_j\neq y_j} \langle v_{y,j}|u_{x,j}\rangle, \text{ since } \langle \nu_{y_j}|\mu_{x_j}\rangle = \frac{1}{2}\frac{q}{q-1}(1-\delta_{x_j,y_j}) \\ &= \sum_{y\in F_0} |y\rangle = |\tau\rangle. \end{aligned}$$

The last line follows from the feasibility condition $\sum_{j\in[n]:x_j\neq y_j} \langle v_{y,j}|u_{x,j}\rangle = 1$ for all $(x,y) \in F_1 \times F_0$. Thus, $|w_x\rangle$ is a positive witness for x in $\mathcal{P}$. It has complexity

$$\||w_x\rangle\|^2 = \frac{4(q-1)^2}{q^2} \sum_{j\in[n]} \||u_{x,j}\rangle\|^2 \leq 4 \sum_{j\in[n]} \||u_{x,j}\rangle\|^2 .$$

Thus,

$$W_+(\mathcal{P}) \leq 4 \max_{x\in F_1} \sum_{j\in[n]} \||u_{x,j}\rangle\|^2 .$$

We thus have:

$$C(\mathcal{P}) = \sqrt{W_-(\mathcal{P})W_+(\mathcal{P})} \leq 2\sqrt{\max_{x\in F_0} \sum_{j\in[n]} \||v_{x,j}\rangle\|^2 \max_{x\in F_1} \sum_{j\in[n]} \||u_{x,j}\rangle\|^2}.$$

□

7.2 Motivation: One-Sided Error Algorithms to Span Programs

It is possible to convert a span program $\mathcal{P}$ for a problem f to a quantum algorithm that computes f with bounded error query complexity $\Theta(C(\mathcal{P}, f))$ by Theorem 7.0.8. Similarly, given a quantum algorithm that computes f with bounded error query complexity C, there exists a span program for f with complexity C, by Theorem 7.0.8. However, there is no general conversion that takes an arbitrary quantum algorithm and converts it to a span program for the same problem, preserving the query complexity. It is not immediately clear that such a conversion would be useful. It is clear why we would want to convert span programs to quantum algorithms, since quantum algorithms are useful, whereas our main use for span programs is to generate algorithms, so if we already have a quantum algorithm, a span program is of little use. However, it is somewhat unpleasant that there is no conversion from algorithms to span programs, especially given that there *is* a very natural conversion, to be presented shortly, that works for any quantum algorithm with *one-sided* error [Rei09, Section 3 of full version]. Since span programs seem to correspond so perfectly to (two-sided) bounded error quantum algorithms, why does this nice conversion only work for one-sided algorithms?

An Arbitrary One-sided Error Algorithm Fix a decision problem f, and let $\mathtt{A}$ be a quantum algorithm that solves f with one-sided error using T queries. Suppose without loss of generality that $\mathtt{A}$ works on the space $H_{\mathtt{A}} := \text{span}\{|i,z,a\rangle : i \in \{0,\dots,n\}, z \in Z, a \in \{0,1\}\}$ for some finite set Z with $0 \in Z$. A query operates as

$$\mathcal{O}_x : |i,z,a\rangle \mapsto (-1)^{x_i}|i,z,a\rangle.$$

Suppose without loss of generality that $\mathtt{A}$ acts as $U_{T+1}\mathcal{O}_x U_T \mathcal{O}_x \dots U_2 \mathcal{O}_x U_1$ on starting state $|0,0,0\rangle$, before measuring the bit in the last register.

Define unitaries $\{U'_t\}_{t=1}^{2T+1}$ such that:

$$U'_{2t-1} := U_t \ \forall t \in \{1,\dots,T+1\} \quad \text{and} \quad U'_{2t} := \mathcal{O}_x \ \forall t \in \{1,\dots,T\}.$$

Then we can rewrite the unitary enacted by $\mathtt{A}$ as $U'_{2T+1}U'_{2T}\dots U'_1$. For any $t \in \{0,\dots,2T+1\}$, write the state of the algorithm after t steps on input x as

$$|\Psi_t(x)\rangle := U'_t \dots U'_1|0,0,0\rangle.$$

For all x, we have $|\Psi_0(x)\rangle = |\Psi_0\rangle := |0,0,0\rangle$. Suppose that $\mathtt{A}$ ends in a state $|\Psi_{2T+1}(x)\rangle$ such that $\langle\Psi_{2T+1}(x)|0,0,1\rangle$ is the probability of measuring 1 in the answer register. We can impose this condition with at most a doubling of T by starting with an arbitrary algorithm, copying the output register, and then running the algorithm in reverse. In particular, this implies that when $f(x) = 1$, $|\Psi_{2T+1}(x)\rangle = |0,0,1\rangle =: |\Psi^1_{2T+1}\rangle$, since $\mathtt{A}$ has one-sided error.

A Corresponding Span Program We will now define a span program for f based on $\mathtt{A}$. The span program will have

$$H := \text{span}\{|t\rangle : t \in \{0,\dots,2T\}\} \otimes H_{\mathtt{A}} \otimes \mathbb{C}^2$$

and

$$V := \text{span}\{|t\rangle : t \in \{0, \dots, 2T+1\}\} \otimes H_{\mathtt{A}},$$

and in particular, for each $j \in [n]$, we will set

$$H_{j,b} := \text{span}\left\{|t, j, z, a, b\rangle : t \text{ odd}\right\},$$

$$H_{\text{true}} := \text{span}\left\{|t, j, z, a, b\rangle : t \text{ even}\right\},$$

$$\text{and} \qquad A|t, j, z, a, b\rangle = \begin{cases} |t, j, z, a\rangle - (-1)^b|t+1, j, z, a\rangle & \text{if } t \text{ odd} \\ |t, j, z, a\rangle - |t+1\rangle U'_{t+1}|j, z, a\rangle & \text{else.} \end{cases}$$

Notice that for any t, j, z, and a, we have $A|t, j, z, a, x_j\rangle = |t, j, z, a\rangle - |t+1\rangle U'_{t+1}|j, z, a\rangle$. The target vector will be

$$|\tau\rangle := |0\rangle|\Psi_0\rangle - |2T+1\rangle|\Psi^1_{2T+1}\rangle.$$

Notice that we can always use available vectors in $H(x)$ to construct $|t\rangle|\Psi_t(x)\rangle - |t+1\rangle|\Psi_{t+1}(x)\rangle$ in V. Adding these up, we can construct $|0\rangle|\Psi_0(x)\rangle - |2T+1\rangle|\Psi_{2T+1}(x)\rangle$. We have, for every x, $|\Psi_0(x)\rangle = |0, 0, 0\rangle = |\Psi_0\rangle$. Furthermore, when $f(x) = 1$, we have $|\Psi_{2T+1}(x)\rangle = |0, 0, 1\rangle = |\Psi^1_{2T+1}\rangle$, so we will have constructed $|\tau\rangle$. We can make this idea rigorous by constructing positive and negative witnesses.

Positive Witnesses Suppose $f(x) = 1$. We construct the positive witness:

$$|w_x\rangle := \sum_{t=0}^{2T} \sum_{j \in [n]} \alpha_{j,t}(x)|t\rangle|j\rangle|\psi^j_t(x)\rangle|x_j\rangle, \quad \text{where} \quad |\Psi_t(x)\rangle = \sum_{j \in [n]} \alpha_{j,t}(x)|j\rangle|\psi^j_t(x)\rangle.$$

To see that this is a positive witness, we must show $|w_x\rangle \in H(x)$, and $A|w_x\rangle = |\tau\rangle$. Note that

$$H(x) = \bigoplus_{j \in [n]} H_{j,x_j} \oplus H_{\text{true}} = \text{span}\left\{|t, j, z, a, x_j\rangle : t \text{ odd}\right\} \cup \{|t, j, z, a, b\rangle : t \text{ even}\},$$

so clearly $|w_x\rangle \in H(x)$. Next we compute:

$$\begin{aligned}
A|w_x\rangle &= \sum_{t=0}^{2T}\sum_{j\in[n]} \alpha_{j,t}(x)\left(|t\rangle|j\rangle|\psi_t^j(x)\rangle - |t+1\rangle U'_{t+1}|j\rangle|\psi_t^j(x)\rangle\right)\\
&= \sum_{t=0}^{2T}\left(|t\rangle\sum_{j\in[n]}\alpha_{j,t}(x)|j\rangle|\psi_t^j(x)\rangle - |t+1\rangle U'_{t+1}\sum_{j\in[n]}\alpha_{j,t}(x)|j\rangle|\psi_t^j(x)\rangle\right)\\
&= \sum_{t=0}^{2T}\left(|t\rangle|\Psi_t(x)\rangle - |t+1\rangle U'_{t+1}|\Psi_t(x)\rangle\right)\\
&= \sum_{t=0}^{2T}\left(|t\rangle|\Psi_t(x)\rangle - |t+1\rangle|\Psi_{t+1}(x)\rangle\right) = |0\rangle|\Psi_0(x)\rangle - |2T+1\rangle|\Psi_{2T+1}(x)\rangle\\
&= |0\rangle|\Psi_0\rangle - |2T+1\rangle|\Psi_{2T+1}^1\rangle = |\tau\rangle.
\end{aligned}$$

Thus, $|w_x\rangle$ is indeed a positive witness for x in P. So we have:

$$\text{wsize}_+(x) \leq \||w_x\rangle\|^2 = \sum_{t=0}^{2T}\sum_{j\in[n]}|\alpha_{j,t}(x)|^2\left\||\psi_t^j(x)\rangle\right\|^2 = \sum_{t=0}^{2T}1 = 2T+1. \tag{7.1}$$

Thus $W_+(\mathcal{P}) \leq 2T+1$.

Negative Witnesses We now turn to the case where $f(x) = 0$. In that case, consider the linear mapping:

$$\omega_x := \sum_{t=0}^{2T+1}\langle t|\langle\Psi_t(x)|.$$

To show that this is a negative witness for x, we need to show that $\left\|\omega_x A\Pi_{H(x)}\right\| = 0$, and $\omega_x(|\tau\rangle) = 1$. To see the former, first note that for odd t:

$$\begin{aligned}
\omega_x A|t,j,z,a,x_j\rangle &= \omega_x\left(|t,j,z,a\rangle - (-1)^{x_j}|t+1\rangle|j,z,a\rangle\right)\\
&= \langle\Psi_t(x)|j,z,a\rangle - \langle\Psi_{t+1}(x)|(-1)^{x_j}|j,z,a\rangle\\
&= \langle\Psi_{t+1}(x)|U'_{t+1}|j,z,a\rangle - \langle\Psi_{t+1}(x)|(-1)^{x_j}|j,z,a\rangle\\
&= \langle\Psi_{t+1}(x)|(-1)^{x_j}|j,z,a\rangle - \langle\Psi_{t+1}(x)|(-1)^{x_j}|j,z,a\rangle = 0
\end{aligned}$$

since $U'_{t+1}|j,z,a\rangle = \mathcal{O}_x|j,z,a\rangle = (-1)^{x_j}|j,z,a\rangle$. Next note that for even t:

$$\begin{aligned}
\omega_x A|t,j,z,a,b\rangle &= \omega_x(|t,j,z,a\rangle - |t+1\rangle U'_{t+1}|j,z,a\rangle)\\
&= \langle\Psi_t(x)|j,z,a\rangle - \langle\Psi_{t+1}(x)|U'_{t+1}|j,z,a\rangle\\
&= \langle\Psi_{t+1}(x)|U'_{t+1}|j,z,a\rangle - \langle\Psi_{t+1}(x)|U'_{t+1}|j,z,a\rangle = 0.
\end{aligned}$$

Thus, $\left\|\omega_x A\Pi_{H(x)}\right\| = 0$. Next, we compute:

$$\omega_x|\tau\rangle = \left(\sum_{t=0}^{2T+1}\langle t|\langle\Psi_t(x)|\right)\left(|0\rangle|\Psi_0\rangle - |2T+1\rangle|\Psi_{2T+1}^1\rangle\right) = \langle\Psi_0(x)|\Psi_0\rangle - \langle\Psi_{2T+1}(x)|\Psi_{2T+1}^1\rangle.$$

For all x we have $|\Psi_0(x)\rangle = |\Psi_0\rangle$ so $\langle\Psi_0(x)|\Psi_0\rangle = 1$. Furthermore, by assumption, $\langle\Psi_{2T+1}(x)|\Psi_{2T+1}^1\rangle$ is the probability of measuring a 1 in the answer register, so $p_1(x) := \langle\Psi_{2T+1}(x)|\Psi_{2T+1}^1\rangle$ is a real number that is at most ε, where ε is the error of $\mathtt{A}$, since it is the probability that the algorithm errs on input x. Let $\bar{\omega}_x := \frac{1}{1-p_1(x)}\omega_x$. Clearly, $\bar{\omega}_x$ is a negative witness for x in P, and

$$\mathrm{wsize}_-(x) \leq \|\bar{\omega}_x A\|^2 = \frac{\|\omega_x A\|^2}{(1-p_1(x))^2} \leq \frac{\|\omega_x A\|^2}{(1-\varepsilon)^2}. \tag{7.2}$$

To compute $\|\omega_x A\|^2$, note that

$$\begin{aligned} A &= \sum_{j,z,a,b}\Big(\sum_{t=0}^{T}\Big(|2t,j,z,a\rangle - (-1)^b|2t+1,j,z,a\rangle\Big)\langle 2t,j,z,a,b| \\ &\quad + \sum_{t=0}^{T}\left(|2t+1,j,z,a\rangle - |2t+2\rangle U'_{2t+2}|j,z,a\rangle\right)\langle 2t+1,j,z,a,b|\Big). \end{aligned}$$

We already know that for t even or $b = x_j$, we have $\omega_x A|t,j,z,a,b\rangle = 0$. Suppose on the other hand that t is even, and $b \neq x_j$. Then

$$\begin{aligned} \omega_x A|t,j,z,a,b\rangle &= \left(\sum_{t=0}^{2T+1}\langle t|\langle\Psi_t(x)|\right)\Big(|t,j,z,a\rangle - (-1)^b|t+1\rangle|j,z,a\rangle\Big) \\ &= \langle\Psi_t(x)|j,z,a\rangle + \langle\Psi_t(x)||j,z,a\rangle = 2\langle\Psi_t(x)|j,z,a\rangle, \end{aligned}$$

so we have:

$$\begin{aligned} \|\omega_x A\|^2 &= \sum_{t=0}^{T}\sum_{j,z,a}|\omega_x A|j,z,a,x_j\oplus 1\rangle|^2 \\ &= \sum_{t=0}^{T}4\sum_{j,z,a}|\langle\Psi_t(x)|j,z,a\rangle|^2 = 4(T+1). \end{aligned}$$

Thus, we have $\mathrm{wsize}_-(x) \leq \frac{4(T+1)}{(1-\epsilon)^2}$. Thus $W_-(\mathcal{P}) \leq \frac{4(T+1)}{(1-\varepsilon)^2}$. So the complexity of the span program is

$$\sqrt{W_+W_-} \leq \sqrt{\frac{(2T+1)4(T+1)}{(1-\varepsilon)^2}} = \Theta(T),$$

matching the query complexity of A, up to a constant, as desired.

There is one more intriguing detail of this construction that motivates even further the desire to understand why such a beautiful construction does not apply to general quantum algorithms: If the time complexity of this algorithm is also T — that is, each U_i can be implemented with a constant number of gates — then if we convert the algorithm to a span program using this construction, and then back to an algorithm by a span program evaluation algorithm like the one to be presented in Section 7.6.1, the resulting algorithm can be made to have time complexity T as well! This is because the span program evaluation algorithm is constructed from a pair of reflections, one of which has 0 query complexity, but possibly large time complexity. In this case that reflection ends up being sufficiently structured that a time-efficient implementation is possible (see [Rei09]). This actually says that, in general, any one-sided algorithm with time complexity T can be converted to a span program in this way and then back to an algorithm with time complexity T: simply break the algorithm into constant-time sub-unitaries U_i, and put empty queries between those U_i and U_{i+1} that do not have a query between them.

If we try to apply the same idea to an algorithm with two-sided error, the conversion fails because we cannot specify an exact target $|\tau\rangle$: every $x \in f^{-1}(1)$ has a different "target" they can obtain $|0\rangle|\Psi_0\rangle - |2T+1\rangle|\Psi_{2T+1}(x)\rangle$. All are close to $|0\rangle|\Psi_0\rangle - |2T+1\rangle|\Psi^1_{2T+1}\rangle$, but we have no guarantee that we can hit this target *exactly*. This exact nature of span programs does not seem necessary at all, and in fact, as we will soon see, it is not. This motivates us to consider approximate versions of the span program formalism.

7.3 Definition of Approximate Span Program

Section 7.2 motivates us to consider approximate versions of the span program definition, but as we will see in this section, there are a number of very natural sounding definitions. Fortunately, as we will show here, these definitions are all equivalent.

Approximation 1: Approximate Acceptance If we naively try to apply the construction of Section 7.2 to a two-sided error algorithm, we see that it fails for precisely the reason that even when $f(x) = 1$, we can not exactly predict the final state $|\Psi_{2T+1}(x)\rangle$ — it can vary slightly for different x, and all we can say is that the state must be *close to* $|\Psi^1_{2T+1}\rangle = |0,0,1\rangle$, the final state that outputs 1 with probability 1. A natural way to modify the definition of a span program then is to say a span program $\mathcal{P}$ accepts an input x if there is some witness $|w_x\rangle \in H(x)$ such that $A|w_x\rangle$ is close to τ, or rather, since V has no notion of distance, we modify the definition by saying that $\mathcal{P}$ accepts x if there is a witness $|w_x\rangle$ that is *almost* in $H(x)$ (since H comes equipped with an orthonormal basis, this "almost" can be quantified) such that $A|w_x\rangle = \tau$. This idea gives rise to our first definition of an approximate span program.

Definition 7.3.1 (Approximate Positive Witness)**.** *Let $\mathcal{P}$ be a span program on $[q]^n$. For any $x \in [q]^n$, the* positive error of x *is defined*

$$\mathrm{error}_+(x, \mathcal{P}) := \min_{|w\rangle : A|w\rangle = |\tau\rangle} \left\| \Pi_{H(x)^\perp} |w\rangle \right\|^2 .$$

Definition 7.3.2 (Span Program with Approximate Positive Witnesses)**.** *Let $f : D \to \{0,1\}$ be a decision problem on $D \subseteq [q]^n$. Let $\mathcal{P}$ be a span program on $[q]^n$. We say that $\mathcal{P}$* solves f with approximate positive witness error gap λ *if*

$$\frac{\max_{x \in f^{-1}(1)} \mathrm{error}_+(x, \mathcal{P})}{\min_{x \in f^{-1}(0)} \mathrm{error}_+(x, \mathcal{P})} \leq \lambda.$$

We can think of $\mathcal{P}$ as accepting inputs with small positive error, and rejecting those with large positive error. We require that there is some relative gap between what we consider small enough to accept, and what we consider large enough to reject.

Of course, the main reason that the original definition of span programs is useful is that they can be used to construct quantum algorithms. In Section 7.6.2, we will prove that our definition is similarly useful by giving an algorithm for evaluating approximate span programs. However, we first consider alternative definitions of approximate span programs.

Approximation 2: Approximate Rejection Naturally, we can consider the same type of approximation in the negative witnesses, rather than the positive witnesses — that is, having an almost negative witness for x is enough for x to be rejected. We formalize this by defining the negative error of an input.

Definition 7.3.3 (Approximate Negative Witness)**.** *Let $\mathcal{P}$ be a span program on $[q]^n$. For any $x \in [q]^n$, the* negative error of x *is defined*

$$\mathrm{error}_-(x, \mathcal{P}) := \min_{\omega \in \mathcal{L}(V, \mathbb{C}) | \omega(\tau) = 1} \left\| \omega A \Pi_{H(x)} \right\|^2 .$$

This yields our next notion of approximate span program.

Definition 7.3.4 (Span Program with Approximate Negative Witnesses)**.** *Let $f : D \to \{0,1\}$ be a decision problem on $D \subseteq [q]^n$. Let $\mathcal{P}$ be a span program on $[q]^n$. We say that $\mathcal{P}$* solves f with approximate negative witness error gap λ *if*

$$\frac{\max_{x \in f^{-1}(0)} \mathrm{error}_-(x, \mathcal{P})}{\min_{x \in f^{-1}(1)} \mathrm{error}_-(x, \mathcal{P})} \leq \lambda.$$

Approximation 3: Positive Witness Size Gap The first (and second) definitions are very well motivated by the construction in Section 7.2, however, in coming up with an approximate notion of span programs, there is another possibly even more natural definition: to accept only those strings with a *small* positive witness, below some threshold, and reject any strings with positive witnesses above some threshold. This motivates our third definition:

Definition 7.3.5 (Span Program with Positive Witness Size Gap)**.** *Let $f : D \to \{0,1\}$ be a decision problem on $D \subseteq [q]^n$. Let $\mathcal{P}$ be a span program on $[q]^n$. We say that $\mathcal{P}$* solves f with positive witness gap λ *if*

$$\frac{\max_{x\in f^{-1}(1)} \mathrm{wsize}_+(x,\mathcal{P})}{\min_{x\in f^{-1}(0)} \mathrm{wsize}_+(x,\mathcal{P})} \leq \lambda.$$

Approximation 4: Negative Witness Size Gap Finally, we may similarly consider a gap in *negative* witness size, where we reject only those strings with very small negative witness, and accept any strings with negative witness size above a certain threshold.

Definition 7.3.6 (Span Program with Negative Witness Size Gap)**.** *Let $f : D \to \{0,1\}$ be a decision problem on $D \subseteq [q]^n$. Let $\mathcal{P}$ be a span program on $[q]^n$. We say that $\mathcal{P}$* solves f with negative witness gap λ *if*

$$\frac{\max_{x\in f^{-1}(0)} \mathrm{wsize}_-(x,\mathcal{P})}{\min_{x\in f^{-1}(1)} \mathrm{wsize}_-(x,\mathcal{P})} \leq \lambda.$$

Comparison of Definitions We will now prune the unwieldy landscape of definitions by showing them all to be equivalent.

Theorem 7.3.7 (Equivalence of Definitions 7.3.2 and 7.3.6)**.** *Let $\mathcal{P}$ be a span program on $[q]^n$. For all $x \in [q]^n$ such that* $\mathrm{wsize}_-(x,\mathcal{P}) < \infty$*, we have:*

$$\mathrm{error}_+(x,\mathcal{P}) = \frac{1}{\mathrm{wsize}_-(x,\mathcal{P})}.$$

Furthermore, if $\mathrm{wsize}_-(x,\mathcal{P}) = \infty$*, we have* $\mathrm{error}_+(x,\mathcal{P}) = 0$*. Therefore, for any $f : D \to \{0,1\}$ with $D \subseteq [q]^n$, $\mathcal{P}$ solves f with approximate positive witness error gap λ if and only if $\mathcal{P}$ solves f with negative witness gap λ.*

Proof. To begin, suppose $\mathrm{wsize}_-(x,\mathcal{P}) = \infty$. Then there does not exist an exact negative witness for x, so x is positive for $\mathcal{P}$, and thus, there exists an exact positive witness, $|w\rangle$ such that $\left\|\Pi_{H(x)^\perp}|w\rangle\right\|^2 = 0$, and thus $\mathrm{error}_+(x) = 0$.

Now suppose that $\mathrm{wsize}_-(x) < \infty$. Let $\omega \in \mathcal{L}(V,\mathbb{C})$ be an optimal negative witness for x, so $\omega(\tau) = 1$, $\left\|\omega A\Pi_{H(x)}\right\|^2 = 0$, and $\|\omega A\|^2 = \mathrm{wsize}_-(x)$. Let $|w\rangle$ be a minimum-error positive witness for x, so $A|w\rangle = \tau$, and $\left\|\omega A \Pi_{H(x)^\perp}\right\|^2 = \mathrm{error}_+(x)$. We have

$$\begin{aligned}\mathrm{wsize}_-(x,\mathcal{P}) = \|\omega A\|^2 \ &\geq\ \frac{\left\|(\omega A)(\Pi_{H(x)^\perp}|w\rangle)\right\|^2}{\left\|\Pi_{H(x)^\perp}|w\rangle\right\|^2} \geq \frac{\left\|\omega(\tau) - \omega(A\Pi_{H(x)}|w\rangle)\right\|^2}{\mathrm{error}_+(x,\mathcal{P})}\\ &=\ \frac{1}{\mathrm{error}_+(x,\mathcal{P})} \text{ since } \omega(\tau)=1 \text{ and } \left\|\omega A \Pi_{H(x)}\right\|^2 = 0.\end{aligned}$$

Now consider $|w_{\mathrm{err}}\rangle = \Pi_{H(x)^\perp}|w\rangle$, the error part of $|w\rangle$. Define

$$|w'\rangle = |w\rangle - \Pi_{\ker A}|w_{\mathrm{err}}\rangle \quad \text{and note} \quad A|w'\rangle = A|w\rangle = |\tau\rangle.$$

Then by the assumption that $|w\rangle$ has minimal error, we have

$$\left\|\Pi_{H(x)^\perp}|w\rangle\right\|^2 \leq \left\|\Pi_{H(x)^\perp}|w'\rangle\right\|^2 \leq \left\|\Pi_{H(x)^\perp}|w\rangle - \Pi_{\ker A}\Pi_{H(x)^\perp}|w\rangle\right\|^2 = \left\|\Pi_{\ker A^\perp}|w_{\mathrm{err}}\rangle\right\|^2,$$

so we must have $\Pi_{\ker A^\perp}|w_{\mathrm{err}}\rangle = |w_{\mathrm{err}}\rangle$, or in other words, $|w_{\mathrm{err}}\rangle \in \ker A^\perp$. Thus, the linear function $\langle w_{\mathrm{err}}|$ has $\ker \langle w_{\mathrm{err}}| \subseteq \ker A$, so (by the fundamental homomorphism theorem) there exists a linear function $\tilde{\omega} : \mathrm{Im}A \to \mathrm{Im}\langle w_{\mathrm{err}}|$ such that $\tilde{\omega}A = \langle w_{\mathrm{err}}|$. We can assume without loss of generality that $\mathrm{Im}A = V$, so $\tilde{\omega} \in \mathcal{L}(V,\mathbb{C})$. We have

$$\tilde{\omega}(\tau) = \tilde{\omega}A|w\rangle = \langle w|\Pi_{H(x)^\perp}|w\rangle,$$

so $\omega = \frac{1}{\left\|\Pi_{H(x)^\perp}|w\rangle\right\|^2}\tilde{\omega}$ has $\omega(\tau) = 1$ and

$$\mathrm{wsize}_-(x,\mathcal{P}) \leq \|\omega A\|^2 = \frac{\|\tilde{\omega}A\|^2}{\left\|\Pi_{H(x)^\perp}|w\rangle\right\|^4} = \frac{\left\|\Pi_{H(x)^\perp}|w\rangle\right\|^2}{\left\|\Pi_{H(x)^\perp}|w\rangle\right\|^4} = \frac{1}{\mathrm{error}_+(x,\mathcal{P})}.$$

□

We can similarly show that span programs with approximate negative witnesses are equivalent to span programs with positive witness gap.

Theorem 7.3.8 (Equivalence of Definitions 7.3.4 and 7.3.5)**.** *Let P be a span program on $[q]^n$. For all $x \in [q]^n$ such that $\mathrm{wsize}_+(x,\mathcal{P}) < \infty$, we have:*

$$\mathrm{error}_-(x,\mathcal{P}) = \frac{1}{\mathrm{wsize}_+(x,\mathcal{P})}.$$

Furthermore, if $\mathrm{wsize}_+(x,\mathcal{P}) = \infty$, we have $\mathrm{error}_-(x,\mathcal{P}) = 0$. Therefore, for any $f : D \to \{0,1\}$

with $D \subseteq [q]^n$, $\mathcal{P}$ solves f with approximate negative witnesses with gap λ if and only if $\mathcal{P}$ solves f with positive witness gap λ.

Proof. To begin, suppose $\mathrm{wsize}_+(x,\mathcal{P}) = \infty$. Then there does not exist an exact positive witness for x, $|w\rangle \in H(x)$ such that $A|w\rangle = \tau$. So x is negative for $\mathcal{P}$, and thus there exists an exact negative witness for x, $\omega \in \mathcal{L}(V,\mathbb{C})$ such that $\omega\tau = 1$, and $\left\|\langle\omega|A\Pi_{H(x)}\right\|^2 = 0$, thus $\mathrm{error}_-(x,\mathcal{P}) = 0$.

Now suppose that $\mathrm{wsize}_+(x,\mathcal{P}) < \infty$. Let $|w\rangle \in H(x)$ be an optimal witness for x, so $A|w\rangle = \tau$ and $\left\||w\rangle\right\|^2 = \mathrm{wsize}_+(x,\mathcal{P})$. For any $\omega \in \mathcal{L}(V,\mathbb{C})$ such that $\omega\tau = 1$ we have

$$\frac{\left\|\omega A|w\rangle\right\|^2}{\left\||w\rangle\right\|^2} = \frac{\left\|\omega\tau\right\|^2}{\mathrm{wsize}_+(x,\mathcal{P})} = \frac{1}{\mathrm{wsize}_+(x,\mathcal{P})},$$

so certainly

$$\mathrm{error}_-(x,\mathcal{P}) = \min_{\omega:\omega(\tau)=1} \left\|\omega A\Pi_{H(x)}\right\|^2 \geq \frac{1}{\mathrm{wsize}_+(x,\mathcal{P})}, \tag{7.3}$$

since $|w\rangle \in H(x)$.

For the other direction, let $\omega \in \mathcal{L}(V,\mathbb{C})$ such that $\omega\tau = 1$ minimize $\left\|\omega A\Pi_{H(x)}\right\|^2$. Note that $\omega A \in H^\dagger$, so its dual is in H. Define

$$|w\rangle := \frac{\Pi_{H(x)}(\omega A)^\dagger}{\left\|\omega A\Pi_{H(x)}\right\|^2}.$$

Clearly $|w\rangle \in H(x)$. We will show that it is a positive witness for x by showing that $A|w\rangle = \tau$, by contradiction. Assume that $A|w\rangle$ and τ are linearly independent. In that case, let $\alpha \in \mathcal{L}(V,\mathbb{C})$ be such that $\alpha(A|w\rangle) = 0$ and $\alpha(\tau) = 1$. Then we have

$$\alpha(A\Pi_{H(x)}(\omega A)^\dagger) = (\alpha A)\left(\left\|\omega A\Pi_{H(x)}\right\|^2 |w\rangle\right) = \left\|\omega A\Pi_{H(x)}\right\|^2 \alpha(A|w\rangle) = 0, \tag{7.4}$$

and since $\alpha(\tau) = 1$, for any $\varepsilon \in [0,1]$, we have $(\varepsilon\omega + (1-\varepsilon)\alpha)(\tau) = 1$, so by the optimality of ω, we have

$$\begin{aligned}
\left\|\omega A\Pi_{H(x)}\right\|^2 &\leq \left\|(\varepsilon\omega + (1-\varepsilon)\alpha)A\Pi_{H(x)}\right\|^2 \\
&= \varepsilon^2\left\|\omega A\Pi_{H(x)}\right\|^2 + (1-\varepsilon)^2\left\|\alpha A\Pi_{H(x)}\right\|^2 \text{ by (7.4)} \\
(1-\varepsilon^2)\left\|\omega A\Pi_{H(x)}\right\|^2 &\leq (1-\varepsilon)^2\left\|\alpha A\Pi_{H(x)}\right\|^2.
\end{aligned}$$

In particular, this must hold for $\varepsilon = \frac{\left\|\alpha A\Pi_{H(x)}\right\|^2}{\left\|\alpha A\Pi_{H(x)}\right\|^2+\left\|\omega A\Pi_{H(x)}\right\|^2}$, yielding

$$\left\|\omega A\Pi_{H(x)}\right\|^6 + \left\|\omega A\Pi_{H(x)}\right\|^4 \left\|\omega A\Pi_{H(x)}\right\|^2 \leq 0,$$

a contradiction, since $\left\|\omega A\Pi_{H(x)}\right\| > 0$. Thus, we must have $A|w\rangle = r\tau$ for some scalar r, so $\omega(A|w\rangle) = r\omega(\tau)$. We then have

$$\omega(A|w\rangle) = \omega A \frac{\Pi_{H(x)}(\omega A)^\dagger}{\left\|\omega A\Pi_{H(x)}\right\|^2} = 1,$$

and so we have $r = 1$, and thus $A|w\rangle = \tau$. So $|w\rangle$ is a positive witness for x, and we can compute

$$\mathrm{wsize}_+(x,\mathcal{P}) \leq \left\||w\rangle\right\|^2 = \frac{\left\|\Pi_{H(x)}(\omega A)^\dagger\right\|^2}{\left\|\omega A\Pi_{H(x)}\right\|^4} = \frac{1}{\mathrm{error}_-(x,\mathcal{P})}.$$

Combining this with (7.3) completes the proof that $\mathrm{error}_-(x,\mathcal{P}) = \frac{1}{\mathrm{wsize}_+(x,\mathcal{P})}$.

Thus, we can conclude that $\mathcal{P}$ solves f with approximate negative witness gap λ if and only if $\mathcal{P}$ solves f with positive witness gap λ, since

$$\frac{\max_{x\in f^{-1}(1)} \mathrm{wsize}_+(x,\mathcal{P})}{\min_{x\in f^{-1}(0)} \mathrm{wsize}_+(x,\mathcal{P})} = \frac{\max_{x\in f^{-1}(0)} \mathrm{error}_-(x,\mathcal{P})}{\min_{x\in f^{-1}(1)} \mathrm{error}_-(x,\mathcal{P})}.$$

□

By Theorem 7.3.7, Span Programs with Approximate Positive Witness and Span Programs with Negative Witness Gap are actually the same thing, so we can merge Definitions 7.3.2 and 7.3.6 into a new definition, which we will call *Approximate Negative Span Programs*, since all $x \in f^{-1}(0)$ have negative witnesses.

Definition 7.3.9 (Approximate Negative Span Program)**.** *Let $f : D \to \{0,1\}$ be a decision problem on $D \subseteq [q]^n$. Let $\mathcal{P}$ be a span program on $[q]^n$. For $\lambda \in (0,1)$, we say that $\mathcal{P}$ is a* λ-negative span program for f *if*

$$\frac{\max_{x\in f^{-1}(0)} \mathrm{wsize}_-(x,\mathcal{P})}{\min_{x\in f^{-1}(1)} \mathrm{wsize}_-(x,\mathcal{P})} \leq \lambda.$$

We define the negative complexity of $\mathcal{P}$ for f as

$$W_-^0(\mathcal{P},f) := \max_{x\in f^{-1}(0)} \mathrm{wsize}_-(x,\mathcal{P}).$$

We can also define

$$W_-^1(\mathcal{P},f) := \min_{x\in f^{-1}(1)} \mathrm{wsize}_-(x,\mathcal{P}).$$

Note that we have $W_-^1(\mathcal{P},f) \geq W_-^0(\mathcal{P},f)/\lambda$.

Let $\widetilde{\mathrm{wsize}}_+(x,\mathcal{P},\varepsilon) := \min\{\|\omega A\|^2 : \omega(\tau)=1, \left\|\omega A \Pi_{H(x)}\right\|^2 \leq \varepsilon\}$*. In particular, let* $\widetilde{\mathrm{wsize}}_+(x,\mathcal{P}) := \widetilde{\mathrm{wsize}}_+(x,\mathcal{P},\mathrm{error}_+(x,\mathcal{P}))$ *denote the smallest* min-error witness *for* x *in* $\mathcal{P}$*. We define the* λ*-positive complexity of* $\mathcal{P}$ *for* f *as*

$$\widetilde{W}_+^1(\mathcal{P},f) := \max_{x\in f^{-1}(1)} \widetilde{\mathrm{wsize}}_+\left(x,\mathcal{P},\frac{\lambda}{W_-^0(\mathcal{P},f)}\right).$$

Finally, define the complexity of $\mathcal{P}$ *on* f *as:*

$$C(\mathcal{P},f) := \sqrt{\widetilde{W}_+^1(\mathcal{P},f)W_-^0(\mathcal{P},f)}.$$

Similarly, by Theorem 7.3.8, we can merge Definitions 7.3.4 and 7.3.5 into one definition, which we call an *approximate positive span program* for f, since all $x \in f^{-1}(1)$ have a positive witness.

Definition 7.3.10 (Approximate Positive Span Program)**.** *Let* $f : D \to \{0,1\}$ *be a decision problem on* $D \subseteq [q]^n$*. Let* $\mathcal{P}$ *be a span program on* $[q]^n$*. For* $\lambda \in (0,1)$*, we say that* $\mathcal{P}$ *is a* λ-positive span program for f *if*

$$\frac{\max_{x\in f^{-1}(1)} \mathrm{wsize}_+(x,\mathcal{P})}{\min_{x\in f^{-1}(0)} \mathrm{wsize}_+(x,\mathcal{P})} \leq \lambda.$$

We define the positive complexity of $\mathcal{P}$ *for* f *as*

$$W_+^1(\mathcal{P},f) := \max_{x\in f^{-1}(1)} \mathrm{wsize}_+(x,\mathcal{P}).$$

We can also define

$$W_+^0(\mathcal{P},f) := \min_{x\in f^{-1}(0)} \mathrm{wsize}_+(x,\mathcal{P}).$$

Note that we have $W_+^0(\mathcal{P},f) \geq W_-^0(\mathcal{P},f)/\lambda$.

Let $\widetilde{\mathrm{wsize}}_-(x,\mathcal{P},\varepsilon) := \min\{\||w\rangle\|^2 : A|w\rangle = \tau, \left\|\Pi_{H(x)^\perp}|w\rangle\right\|^2 \leq \varepsilon\}$*. In particular, let* $\mathrm{wsize}_-(x,\mathcal{P}) := \widetilde{\mathrm{wsize}}_-(x,\mathcal{P},\mathrm{error}_-(x,\mathcal{P}))$ *denote the smallest min-error witness for* x *in* $\mathcal{P}$*. We define the* λ*-negative complexity of* P *for* f *as*

$$\widetilde{W}_-^0(\mathcal{P},f) := \max_{x\in f^{-1}(0)} \widetilde{\mathrm{wsize}}_-\left(x,\mathcal{P},\frac{\lambda}{W_+^1}\right).$$

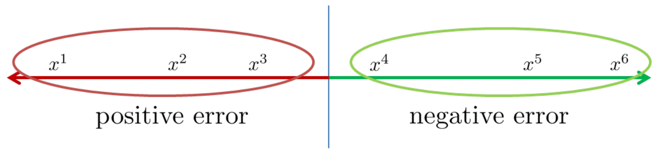


Figure 7.1: We can order inputs from those with largest positive error, to those with largest negative error (exactly one of these is non-zero). A span program algorithm can be used to distinguish between those inputs with positive error, and those inputs with negative error.

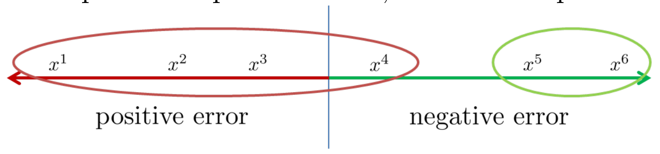


Figure 7.2: Approximate span programs allow us to more generally distinguish between inputs on either side of any threshold. When the threshold is set somewhere on the positive side, we call this a positive span program.

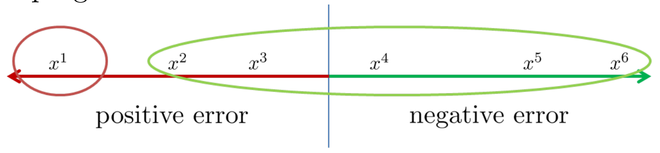


Figure 7.3: When the threshold is on the negative side, we call this a negative span program.

Finally, define the complexity of $\mathcal{P}$ on f as:

$$C(\mathcal{P},f) := \sqrt{W_+^1(\mathcal{P},f)\widetilde{W}_-^0(\mathcal{P},f)}.$$

We now have only two definitions, which is certainly preferable to the previous four. Even these two definitions are on the same continuum. For any span program $\mathcal{P}$ on $[q]^n$, we can order the set $[q]^n$ as $\{x^1 < \dots < x^{q^n}\}$ with "<" defined so that for all x such that $\mathcal{P}$ accepts x, and y such that $\mathcal{P}$ rejects y, we have $y < x$; for all x and y such that $\mathcal{P}$ accepts both x and y, if $\text{wsize}_+(x) < \text{wsize}_+(y)$, then $y < x$, and for all x and y such that $\mathcal{P}$ rejects both x and y, if $\text{wsize}_-(x) < \text{wsize}_-(y)$ then $x < y$. This ordering captures the idea that some inputs are more accepted than others, and some inputs are more rejected than others. The usual notion of span programs defines a threshold between those inputs that $\mathcal{P}$ accepts and those inputs that $\mathcal{P}$ rejects. Our idea is simply to allow this threshold to fall at any point in the ordering of inputs. An approximate negative span program places this threshold somewhere on the side of the rejected inputs, whereas an approximate positive span program places this threshold somewhere on the side of the accepted inputs.

For good measure, we can further strengthen the equivalence of these definitions by giving conversions between the two that mostly preserve the parameters.

Theorem 7.3.11 (Conversion from Positive to Negative). *Let $\mathcal{P} = (H, V, \tau, A)$ be a λ-positive span program for $f : D \subseteq [q]^n \to \{0,1\}$. For any positive constant d, there exists a λ'-negative span program for f, $\mathcal{P}'$, with*

$$\lambda' = \frac{d^2+1}{d^2}\lambda, \quad \widetilde{W}_+^1(\mathcal{P}', f) \leq \left(1 + \frac{\lambda}{d^2 C(\mathcal{P}, f)^2}\right) W_+^1(\mathcal{P}, f), \quad \textit{and} \quad W_-^0(\mathcal{P}', f) \leq (d^2+1)\widetilde{W}_-^0(\mathcal{P}, f).$$

Proof. Our goal is to define a span program that rejects all inputs in $x \in f^{-1}(0)$, even those that $\mathcal{P}$ accepts. Our strategy will be to define a span program $\mathcal{P}'$ that rejects *everything*, but that penalizes most those things that $\mathcal{P}$ accepted the most. In order to accomplish this, we will define A' so that it adds a small error, orthogonal to τ', to every vector, proportional to its size. In this way, every positive witness from $\mathcal{P}$ gets an error term proportional to its positive witness complexity in $\mathcal{P}$. Let $N = d\frac{1}{\sqrt{\lambda}}C(\mathcal{P}, f)$.Concretely, we define $\mathcal{P}' = (\{H'_{j,a}\}, V', \tau', A')$ as follows:

$$H' = H \oplus H, \qquad H'_{j,a} = H_{j,a} \otimes |0\rangle, \qquad H'_{\text{true}} = H_{\text{true}} \otimes |0\rangle \qquad H'_{\text{false}} = H_{\text{false}} \oplus H$$

$$V' = V \oplus H, \qquad \tau' = \tau \otimes |0\rangle, \qquad A'\left(|h_0, 0\rangle + |h_1, 1\rangle\right) = (A|h_0\rangle)|0\rangle + \frac{1}{N}|h_0, 1\rangle + |h_1, 1\rangle$$

Then we have $H'(x) = H(x)|0\rangle$ and $H'(x)^\perp = H(x)^\perp \otimes |0\rangle + H \otimes |1\rangle$.

Let $x \in f^{-1}(0)$. We will show that it has a (small) exact negative witness in $\mathcal{P}'$. Let $\tilde{\omega} \in \mathcal{L}(V, \mathbb{C})$ be a (possibly approximate) negative witness for x in $\mathcal{P}$ with

$$\|\tilde{\omega}A\|^2 \leq \widetilde{W}_-^0(\mathcal{P}, f), \quad \text{and} \quad \left\|\tilde{\omega}A\Pi_{H(x)}\right\|^2 \leq \frac{\lambda}{W_+^1(\mathcal{P}, f)}.$$

We will define a negative witness ω' in $\mathcal{P}'$ so that ω' agrees with $\tilde{\omega}$ on V, and ω' on H cleans up any errors in $\tilde{\omega}$. Specifically, if Π_0 is the orthogonal projector from V' into $V|0\rangle$, and Π_1 is the orthogonal projector from V' into H, we define

$$\omega' = \tilde{\omega}\Pi_0 - N\tilde{\omega}A\Pi_{H(x)}\Pi_1.$$

We can see that this has zero error for x:

$$\begin{aligned}
\left\|\omega' A' \Pi_{H'(x)}\right\|^2 &= \max_{|h\rangle\in H(x):\||h\rangle\|=1} \left\|\omega' A'|h,0\rangle\right\|^2 \\
&= \max_{|h\rangle\in H(x):\||h\rangle\|=1} \left\|\omega'(A|h\rangle|0\rangle + \frac{1}{N}|h\rangle|1\rangle)\right\|^2 \\
&= \max_{|h\rangle\in H(x):\||h\rangle\|=1} \left\|\tilde{\omega}(A|h\rangle) - N\frac{1}{N}\tilde{\omega}(A\Pi_{H(x)}|h\rangle)\right\|^2 = 0.
\end{aligned}$$

We can compute the negative witness size:

$$\begin{aligned}
\left\|\omega' A\right\|^2 &= \left\|\omega' A\Pi_0\right\|^2 + \left\|\omega' A\Pi_1\right\|^2 \\
&= \max_{|h\rangle\in H:\||h\rangle\|=1} \left\|\omega' A'|h,0\rangle\right\|^2 + \max_{|h\rangle\in H:\||h\rangle\|=1} \left\|\omega' A'|h,1\rangle\right\|^2 \\
&= \max_{|h\rangle\in H:\||h\rangle\|=1} \left\|\omega'((A|h\rangle)|0\rangle + \frac{1}{N}|h\rangle|1\rangle)\right\|^2 + \max_{|h\rangle\in H:\||h\rangle\|=1} \left\|\omega'|h,1\rangle\right\|^2 \\
&= \max_{|h\rangle\in H:\||h\rangle\|=1} \left\|\tilde{\omega}(A|h\rangle) - N\frac{1}{N}\tilde{\omega}(A\Pi_{H(x)}|h\rangle)\right\|^2 + \max_{|h\rangle\in H:\||h\rangle\|=1} \left\|-N\tilde{\omega}A\Pi_{H(x)}|h\rangle\right\|^2 \\
&= \left\|\tilde{\omega}A\Pi_{H(x)^\perp}\right\|^2 + N^2\left\|\tilde{\omega}A\Pi_{H(x)}\right\|^2 \\
&\leq \widetilde{W}_-^0(\mathcal{P},f) + d^2\frac{C(\mathcal{P},f)^2}{\lambda}\frac{\lambda}{W_+^1(\mathcal{P})} \leq (1+d^2)\widetilde{W}_-^0(\mathcal{P},f).
\end{aligned}$$

So we have $W_-^0(\mathcal{P}',f) \leq (d^2+1)\widetilde{W}_-^0(\mathcal{P},f)$.

We now turn to the case $x \in f^{-1}(1)$. In that case, we know that there is an exact positive witness for x in $\mathcal{P}$. Let $|w\rangle$ be an optimal positive witness for x in $\mathcal{P}$, so $\left\|\Pi_{H(x)^\perp}|w\rangle\right\|^2 = 0$ and $\||w\rangle\|^2 \leq W_+^1(\mathcal{P},f)$. Define $|w'\rangle = |w\rangle|0\rangle - \frac{1}{N}|w\rangle|0\rangle$. Then $A'|w'\rangle = (A|w\rangle)|0\rangle + \frac{1}{N}|w\rangle|1\rangle - \frac{1}{N}|w\rangle|1\rangle$, and

$$\begin{aligned}
\left\|\Pi_{H'(x)^\perp}|w'\rangle\right\|^2 &= \left\|\left(\Pi_{H(x)^\perp}|w\rangle\right)|0\rangle - \frac{1}{N}|w\rangle|1\rangle\right\|^2 = \frac{1}{N^2}\||w\rangle\|^2 \\
&\leq \frac{\lambda}{d^2 W_+^1(\mathcal{P},f)\widetilde{W}_-^0(\mathcal{P},f)} W_+^1(\mathcal{P},f) \\
&\leq \frac{d^2+1}{d^2}\frac{\lambda}{W_-^0(\mathcal{P}',f)} = \frac{\lambda'}{W_-^0(\mathcal{P}',f)},
\end{aligned}$$

where the last line follows from $W_-^0(\mathcal{P}',f) \leq (d^2+1)\widetilde{W}_-^0(\mathcal{P},f)$. Thus, $|w'\rangle$ has error at most λ' for x, so we have

$$\mathrm{wsize}_+(x,\mathcal{P}',\lambda'/W_-^0(\mathcal{P}',f)) \leq \left\||w'\rangle\right\|^2 = \||w\rangle\|^2 + \frac{1}{N^2}\||w\rangle\|^2 \leq (1+\frac{1}{N^2})W_+^1(\mathcal{P},f)$$

so we have $\widetilde{W}_+^1(\mathcal{P}',f) \leq \left(1+\frac{\lambda}{d^2C(\mathcal{P},f)^2}\right)W_+^1(\mathcal{P},f)$, as desired. □

We can show a similar conversion in the other direction.

Theorem 7.3.12 (Conversion from Negative to Positive). *Let $\mathcal{P}=(H,V,\tau,A)$ be a λ-negative span program for $f:D\subseteq[q]^n\to\{0,1\}$. For any positive constant $d\geq\sqrt{\frac{\lambda}{1-\lambda}}$, there exists a λ'-positive span program for f, $\mathcal{P}'$, with*

$$\lambda' = \frac{d^2+1}{d^2}\lambda,\quad \widetilde{W}_-^0(\mathcal{P}',f) \leq (1+\frac{\lambda}{d^2C(\mathcal{P},f)^2})W_-^0(\mathcal{P},f),\quad \textit{and}\quad W_+^1(\mathcal{P}',f)\leq(d^2+1)\widetilde{W}_+^1(\mathcal{P},f).$$

Proof. We will construct a span program in which everything is positive, so in particular, every input in $f^{-1}(1)$ is positive. To do this, we will make a complete copy of H available in each $H'(x)$, by adding it to H_{true}. In order to ensure that inputs in $f^{-1}(0)$ have large positive witnesses, we will construct A' so that it imposes a penalty whenever we use these vectors, so that it will always be preferable to use other vectors in $H(x)$, when possible. In this way, small error positive witnesses in $\mathcal{P}$ will have small positive witnesses in $\mathcal{P}'$.

The New Span Program Let $N=\frac{dC(\mathcal{P},f)}{\sqrt{\lambda}}$. The new span program $\mathcal{P}'$ will have the same target as $\mathcal{P}$, so we set $V'=V$ and $\tau'=\tau$. We will define the inner product spaces so that all are the same as in $\mathcal{P}$, except for the space H'_{true}, which gets an additional copy of H:

$$H' = H|0\rangle\oplus H|1\rangle,\qquad H'_{j,a}=H_{j,a}|0\rangle,\qquad H'_{\text{true}}=H_{\text{true}}|0\rangle\oplus H|1\rangle,\qquad H'_{\text{false}}=H_{\text{false}}|0\rangle$$

Note that $H'(x)=H(x)|0\rangle\oplus H|1\rangle$ and $H'(x)^\perp = H(x)^\perp|0\rangle$. Finally, we define A' so that it acts as A, except that it ensures that vectors in the new space $H|1\rangle$ are expensive to use, by penalizing these vectors by a factor of $\frac{1}{N}$:

$$A'(|h_0,0\rangle+|h_1,1\rangle) = A|h_0\rangle + \frac{1}{N}A|h_1\rangle$$

Positive Complexity To begin, suppose that $x\in f^{-1}(1)$, and let $|w\rangle$ be an approximate positive witness for x in $\mathcal{P}$ with $\||w\rangle\|^2 \leq \widetilde{\mathrm{wsize}}_+(x,\mathcal{P},\frac{\lambda}{W_-^0(P)}) \leq \widetilde{W}_+^1(\mathcal{P},f)$ and $\left\|\Pi_{H(x)^\perp}|w\rangle\right\|^2 \leq$

$\frac{\lambda}{W_-^0(\mathcal{P},f)}$. Define

$$|w'\rangle = N(\Pi_{H(x)^\perp}|w\rangle)|1\rangle + (\Pi_{H(x)}|w\rangle)|0\rangle.$$

We can see that $|w'\rangle \in H'(x) = H(x)|0\rangle \oplus H|1\rangle$. Furthermore we have

$$A'|w'\rangle = N\frac{1}{N}A\Pi_{H(x)^\perp}|w\rangle + A\Pi_{H(x)}|w\rangle = A|w\rangle = \tau,$$

so $|w'\rangle$ is a positive witness for x in $\mathcal{P}'$, and we have:

$$\text{wsize}_+(x,\mathcal{P}') \leq \left\||w'\rangle\right\|^2 = N^2\left\|\Pi_{H(x)^\perp}|w\rangle\right\|^2 + \left\|\Pi_{H(x)}|w\rangle\right\|^2 \leq N^2\frac{\lambda}{W_-^0(\mathcal{P},f)} + \widetilde{W}_+^1(\mathcal{P},f).$$

We therefore have

$$W_+^1(\mathcal{P},f) \leq \frac{d^2\widetilde{W}_+^1(\mathcal{P},f)W_-^0(\mathcal{P},f)}{\lambda}\frac{\lambda}{W_-^0(\mathcal{P},f)} + \widetilde{W}_+^1(\mathcal{P},f) = (d^2+1)\widetilde{W}_+^1(\mathcal{P},f),$$

completing the first part of the proof.

Negative Complexity Next, suppose $x \in f^{-1}(0)$, and let $\omega \in \mathcal{L}(V,\mathbb{R})$ be an optimal negative witness for x in $\mathcal{P}$, so $\left\|\omega A\Pi_{H(x)}\right\|^2 = 0$ and $\|\omega A\|^2 \leq W_-^0(\mathcal{P},f)$. Since $V' = V$ and $\tau' = \tau$, we have $\omega(\tau') = 1$, so we can consider ω as a negative witness in $\mathcal{P}'$. Let Π_0 and Π_1 be the orthogonal projectors from H' into $H|0\rangle$ and $H|1\rangle$ respectively, and note that $A' = A(\Pi_0 + \frac{1}{N}\Pi_1)$. As a negative witness for x in $\mathcal{P}'$, ω has error

$$\begin{aligned}
\left\|\omega A'\Pi_{H'(x)}\right\|^2 &= \left\|\omega A\left(\Pi_0 + \frac{1}{N}\Pi_1\right)\Pi_{H'(x)}\right\|^2 = \left\|\omega A\left(\Pi_{H(x)}\Pi_0 + \frac{1}{N}\Pi_1\right)\right\|^2 \\
&= \left\|\omega A\Pi_{H(x)}\right\|^2 + \frac{1}{N}\|\omega A\|^2 = \frac{1}{N^2}\|\omega A\|^2 \\
&\leq \frac{\lambda}{d^2\widetilde{W}_+^1(\mathcal{P},f)W_-^0(\mathcal{P},f)}W_-^0(\mathcal{P},f) \leq \frac{\lambda}{d^2(W_+^1(\mathcal{P}',f)/(d^2+1))} = \frac{\lambda'}{W_+^1(\mathcal{P}',f)},
\end{aligned}$$

where we use the fact that $W_+^1(\mathcal{P}',f) \leq (d^2+1)\widetilde{W}_+^1(\mathcal{P},f)$. We therefore have

$$\widetilde{\text{wsize}}_-\left(x,\mathcal{P}',\frac{\lambda'}{W_+^1(\mathcal{P}',f)}\right) \leq \|\omega A\|^2 = \|\omega A\|^2 + \frac{1}{N^2}\|\omega A\|^2 \leq \left(1+\frac{1}{N^2}\right)W_-^0(\mathcal{P},f),$$

so $\widetilde{W}_-^0(\mathcal{P}',f) \leq \left(1 + \frac{\lambda}{d^2C(\mathcal{P},f)^2}\right)W_-^0(\mathcal{P},f)$. □

7.3.1 Example: Gapped t-threshold

To illustrate that this construction may in fact be useful for coming up with quantum algorithms and not just in strengthening the theoretical correspondence between bounded error query complexity and span programs, consider the following very natural example of an approximate span program.

Let f be the problem of Gapped t-threshold: given an input $x \in \{0,1\}^n$, if $|x| \geq 2t$ output 1, and if $|x| \leq t$ output 0, under the promise that one of these conditions holds. We will give a positive approximate span program for this problem.

Define a span program $\mathcal{P}$ by:

$$V = \mathbb{C}, \qquad \tau = 1, \qquad H_j = \mathbb{C}|j\rangle, \qquad H_{j,1} = H_j, \qquad H_{j,0} = \{0\}, \quad \text{and} \quad A = \sum_{j=1}^{n} \langle j|.$$

Then for any input x, we have $H(x) = \text{span}\{|j\rangle : x_j = 1\}$, and it's easy to see that the optimal positive witness for any x is $|w_x\rangle = \frac{1}{|x|}\sum_{j:x_j=1}|j\rangle$, where $|x|$ is the Hamming weight of x, so $\text{wsize}_+(x,\mathcal{P}) = \frac{1}{|x|}$. We therefore have

$$W_+^1(\mathcal{P},f) = \max_{x\in f^{-1}(1)} \frac{1}{|x|} = \frac{1}{2t}.$$

On the other hand, the *only* negative witness is the identity on $\mathbb{R}$, $\omega = 1$, and it has negative witness complexity $\|\omega A\|^2 = \left\|\sum_{j=1}^n \langle j|\right\|^2 = n$. For any x, this has error $\left\|\omega A \Pi_{H(x)}\right\| = |x|$, so for $\lambda = 1/2$, any $x \in f^{-1}(0)$ has negative witness error at most $t = \frac{\lambda}{W_+^1(\mathcal{P},f)}$. We then have $\widetilde{W}_-^0(\mathcal{P},f) = n$.

By Theorem 7.6.9 in Section 7.6.2, this 1/2-positive approximate span program yields a $\Theta\left(\sqrt{n/t}\right)$ algorithm for Gapped t-threshold (by first converting it to a negative span program via Theorem 7.3.11), which is known to be optimal.

7.3.2 Two-Sided Error Algorithms to Approximate Span Programs

We will now show how to turn a two-sided error algorithm for f into a 3/4-negative approximate span program for f. Fix an algorithm $\mathtt{A}$ as in Section 7.2, except now we only suppose that $\mathtt{A}$ has error at most $\varepsilon = 1/3$ for all inputs, whereas in Section 7.2, we required the stronger condition that $\mathtt{A}$ had error 0 on 1-inputs.

Recall the span program $\mathcal{P}=(H,V,|\tau\rangle,A)$ from Section 7.2. We will use a span program $\mathcal{P}'=(H',V,|\tau\rangle,A')$ nearly identical to $\mathcal{P}$, but we now add some never-available vectors to H:

$$H_0' := |2T+1\rangle \otimes H_{\mathtt{A}},$$

to get $H' = H \oplus H_0'$. The operator A' acts on these vectors as:

$$A'|2T+1\rangle|\psi\rangle = \sqrt{c(T+1)}|2T+1\rangle|\psi\rangle,$$

for some c to be specified later, and A' acts as A everywhere else. The new vectors in H_0' allow us to put any desired state as the final state, but when we do this, we will be penalized, because these vectors are never in $H'(x)=H(x)$, and so always contribute to positive witness error. The multiplicative factor $\sqrt{c(T+1)}$ ensures that this error is not too large.

Negative Analysis Since we have $H'(x)=H(x)$ for any x, all exact negative witnesses for $\mathcal{P}$ are still exact negative witnesses for $\mathcal{P}'$, but their complexity may have changed slightly. Using the negative witnesses $\bar{\omega}_x$ defined in Section 7.2, we recompute, for any x such that $f(x)=0$:

$$\begin{aligned}\mathrm{wsize}_-(x,\mathcal{P}') \leq \left\|\bar{\omega}_x A'\right\|^2 &= \|\bar{\omega}_x A\|^2 + \left\|\bar{\omega}_x\sqrt{c(T+1)}|2T+1\rangle\langle 2T+1|\otimes I_{H_{\mathtt{A}}}\right\|^2\\ &\leq \frac{4}{(1-p_1(x))^2} + c(T+1)\left\|\langle 2T+1|\langle\Psi_{2T+1}(x)|\right\|^2\\ &= \left(\frac{4}{(1-p_1(x))^2}+c\right)(T+1),\end{aligned}$$

(see eq. (7.2)) where $p_1(x)$ is the probability that $\mathtt{A}(x)=1$. As $p_1(x)$ gets larger (x becomes less negative) the negative witness size grows unboundedly. However, when $f(x)=0$, $p_1(x)\leq\varepsilon$, so

$$W_-^0(\mathcal{P}',f) \leq \left(\frac{4}{(1-\varepsilon)^2}+c\right)(T+1).$$

Positive Analysis We now consider positive witnesses, which may now be approximate. Suppose $f(x)=1$. In the exact case, we had positive witnesses $|w_x\rangle\in H(x)$ such that $A|w_x\rangle=|\tau\rangle$. Now we may have $A'|w_x\rangle = A|w_x\rangle \neq |\tau\rangle$, since equality relied on the exactness of $\mathtt{A}$ on 1-inputs. We have:

$$A'|w_x\rangle = A|w_x\rangle = |0\rangle|0,0,0\rangle - |2T+1\rangle|\Psi_{2T+1}(x)\rangle.$$

Then $A|w_x\rangle - |\tau\rangle = |2T+1\rangle(|0,0,1\rangle - |\Psi_{2T+1}(x)\rangle)$, so we define a new approximate witness:

$$|\tilde{w}_x\rangle := |w_x\rangle - \frac{1}{\sqrt{c(T+1)}}|2T+1\rangle(|0,0,1\rangle - |\Psi_{2T+1}(x)\rangle),$$

so clearly $A'|\tilde{w}_x\rangle = |\tau\rangle$. We can compute the positive witness error:

$$\begin{aligned}
\mathrm{error}_+(x,\mathcal{P}') \leq \left\|\Pi_{H'(x)^\perp}|\tilde{w}_x\rangle\right\|^2 &= \left\|\frac{1}{\sqrt{c(T+1)}}|2T+1\rangle(|0,0,1\rangle - |\Psi_{2T+1}(x)\rangle)\right\|^2 \\
&= \frac{1}{c(T+1)}\left(2 - 2\mathrm{Re}\langle\Psi_{2T+1}(x)|0,0,1\rangle\right) \\
&= \frac{1}{c(T+1)}\left(2 - 2p_1(x)\right) \\
&\leq \frac{2-2(1-\varepsilon)}{c(T+1)} = \frac{2\varepsilon}{c(T+1)}.
\end{aligned}$$

Thus, for all x such that $f(x)=1$, $\mathrm{wsize}_-(x,\mathcal{P}') \geq \frac{c(T+1)}{2\varepsilon}$, so $W_-^1(\mathcal{P}',f) \geq \frac{c(T+1)}{2\varepsilon}$. Thus

$$\frac{W_-^0(\mathcal{P}',f)}{W_-^1(\mathcal{P}',f)} \leq \left(\frac{4}{(1-\varepsilon)^2} + c\right)(T+1)\frac{2\varepsilon}{c(T+1)} = \frac{8\varepsilon}{c(1-\varepsilon)^2} + 2\varepsilon =: \lambda.$$

When $\varepsilon = 1/3$, setting $c \geq 72$ gives $\lambda \leq 3/4$. We can also compute the positive witness size:

$$\begin{aligned}
\widetilde{\mathrm{wsize}}_+(x,\mathcal{P}') \leq \||\tilde{w}_x\rangle\|^2 &\leq \||w_x\rangle\|^2 + \frac{2\varepsilon}{c(T+1)} \\
&\leq 2T+1+\frac{2\varepsilon}{c(T+1)}. \quad \text{(See eq. (7.1).)}
\end{aligned}$$

Thus $\widetilde{W}_+^1(\mathcal{P}',f) \leq 2T+1+\frac{2\varepsilon}{c(T+1)}$. Then $\mathcal{P}'$ is a 3/4-negative approximate span program for f with complexity

$$\sqrt{\widetilde{W}_+^1(\mathcal{P}',f)W_-^0(\mathcal{P}',f)} = \Theta(T).$$

7.4 Witness Anatomy

In this section, we explore the structure of span program witnesses. Understanding this structure will allow us to present more intuitive algorithms in Section 7.6.

In general, a positive witness is any $|w\rangle \in H$ such that $A|w\rangle = \tau$. Assume the set of all such vectors is non-empty, and let $|w\rangle$ be any vector in H such that $A|w\rangle = \tau$. Then the set is exactly

$$T := |w\rangle + \ker A = \{|w\rangle + |h\rangle : |h\rangle \in \ker A\}.$$

We prove here, for completeness, that the unique shortest vector in T is the unique vector in $T \cap (\ker A)^\perp$ (See also Figure 7.4).

Lemma 7.4.1. *Let $T = |w\rangle + U$ be an affine subspace of inner product space H, for some subspace U of H, and some vector $|w\rangle \notin U$. Then $\Pi_{U^\perp}|w\rangle$ is the unique shortest vector in T.*

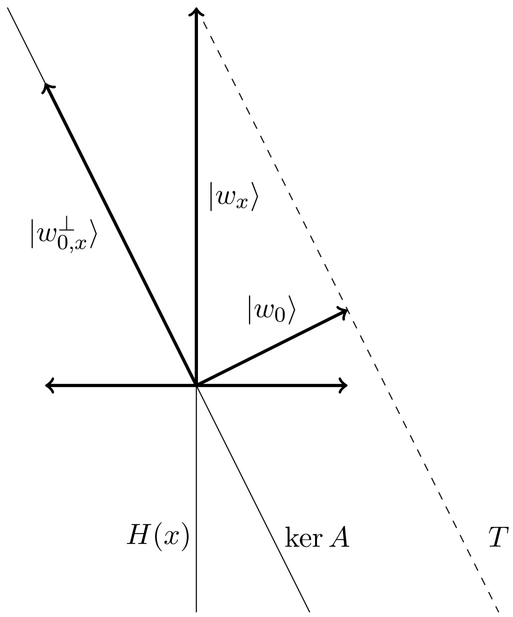


Figure 7.4: The smallest witness $|w_0\rangle$ is the unique vector in T orthogonal to $\ker A$. We can write any optimal witness for an input x as $|w_x\rangle = |w_0\rangle + |w_{0,x}^{\perp}\rangle$ for some $|w_{0,x}^{\perp}\rangle \in \ker A$.

Proof. We first notice that $\Pi_{U^\perp}|w\rangle = |w\rangle - \Pi_U|w\rangle$, and $-\Pi_U|w\rangle \in U$, so $\Pi_{U^\perp}|w\rangle \in T$, so we can write $T = \Pi_{U^\perp}|w\rangle + U$. Let $|w'\rangle \in T$ be such that $\||w'\rangle\| \leq \|\Pi_{U^\perp}|w\rangle\|$. Write $|w'\rangle = \Pi_{U^\perp}|w\rangle + |u\rangle$ for some $|u\rangle \in U$. We have

$$\left\||w'\rangle\right\|^2 = \|\Pi_{U^\perp}|w\rangle\|^2 + \||u\rangle\|^2$$

since $\langle u|\Pi_{U^\perp}|w\rangle = 0$. Then, since $\||w'\rangle\| \leq \|\Pi_{U^\perp}|w\rangle\|$, we must have $\||u\rangle\| = 0$, so $|w'\rangle = \Pi_{U^\perp}|w\rangle$. □

We can therefore talk about the unique smallest positive witness, whenever T is non-empty.

Definition 7.4.2. *Fix a span program $\mathcal{P}$, and suppose $T = \{|h\rangle \in H : A|h\rangle = \tau\}$ is non-empty. We define the* globally optimal positive witness *to be the vector $|w_0\rangle \in T$ with smallest norm. We define $G_+(\mathcal{P}) := \||w_0\rangle\|^2$.*

By Lemma 7.4.1, we know that $|w_0\rangle$ is in $(\ker A)^\perp$. We can write any positive witness $|w_x\rangle$ as $|w_0\rangle + |w_{0,x}^{\perp}\rangle$ for some $|w_{0,x}^{\perp}\rangle \in \ker A$.

Negative Global Witness Just as we can talk about a globally optimal positive witness, we can also talk about a globally optimal negative witness: any $\omega_0 \in \mathcal{L}(V,\mathbb{C})$ such that $\omega_0(\tau) = 1$, that minimizes $\|\omega_0 A\|$. Note that unlike $|w_0\rangle$, ω_0 might not be unique. There may be distinct $\omega_0, \omega_0' \in \mathcal{L}(V,\mathbb{C})$ that map τ to 1 and have minimal complexity. However, the following lemma tells us that in that case, $\omega_0 A = \omega_0' A$.

Lemma 7.4.3. *Suppose $\omega_0, \omega_0' \in \mathcal{L}(V, \mathbb{C})$ are such that $\omega_0(\tau) = \omega_0'(\tau) = 1$, $A\omega_0 \neq A\omega_0'$ and $\|\omega_0 A\| = \|\omega_0' A\|$. Then there exists $\omega_0'' \in \mathcal{L}(V, \mathbb{C})$ such that $\omega_0''(\tau) = 1$ and $\|\omega_0'' A\| < \|\omega_0 A\|$.*

Proof. For any choice of $\delta \in (0,1)$, $\omega_0'' := \delta\omega_0 + (1-\delta)\omega_0'$ has $\omega^\delta(\tau) = 1$.

Note that if $(\omega_0 A)^\dagger$ and $(\omega_0' A)^\dagger$ are linearly independent, then we have $|\omega_0 A(\omega_0' A)| < \|\omega_0 A\| \, \|\omega_0' A\| = \|\omega_0 A\|^2$ by Cauchy-Schwarz. On the other hand, if they are linearly dependent, we must have $\omega_0 A = -\omega_0' A$ since $\|\omega_0 A\| = \|\omega_0' A\|$ and $\omega_0 A \neq \omega_0' A$. In that case, $\omega_0 A(\omega_0' A)^\dagger = -\omega_0 A(\omega_0 A)^\dagger = -\|\omega_0 A\|^2 < \|\omega_0 A\|^2$. Thus, we can compute:

$$\begin{aligned}
\left\|\omega_0'' A\right\|^2 &= \left\|\delta\omega_0 A + (1-\delta)\omega_0' A\right\|^2 \\
&= \delta^2 \left\|\omega_0 A\right\|^2 + (1-\delta)^2 \left\|\omega_0' A\right\|^2 + 2\delta(1-\delta)\omega_0 A(\omega_0' A)^\dagger \\
&< (2\delta^2 - 2\delta + 1)\left\|\omega_0 A\right\|^2 + 2\delta(1-\delta)\left\|\omega_0 A\right\|^2 \\
&= (2\delta^2 - 2\delta + 1 + 2\delta - 2\delta^2)\left\|\omega_0 A\right\|^2 = \left\|\omega_0 A\right\|^2.
\end{aligned}$$

□

For any global optimal negative witness, ω_0, $\omega_0 A$ is conveniently related to the global optimal positive witness $|w_0\rangle$, as the following lemma shows.

Lemma 7.4.4. *Let $|w_0\rangle$ be the globally optimal positive witness in $\mathcal{P}$, and ω_0 a globally optimal negative witness in $\mathcal{P}$. Then $(\omega_0 A)^\dagger = \frac{|w_0\rangle}{G_+(\mathcal{P})}$, and $G_+(\mathcal{P}) = \frac{1}{G_-(\mathcal{P})}$.*

Proof. We can compute

$$\omega_0 A|w_0\rangle = \omega_0\tau = 1,$$

so write $(\omega_0 A)^\dagger = \frac{|w_0\rangle}{\||w_0\rangle\|^2} + |u\rangle$ for some $|u\rangle$ with $\langle u|w_0\rangle = 0$. Let $|v\rangle = \frac{|w_0\rangle}{\||w_0\rangle\|^2}$. Then $|v\rangle \in \ker A^\perp$, so $\ker A \subseteq \ker\langle v|$, so by the fundamental homomorphism theorem, there exists a linear function $\omega_0' \in \mathcal{L}(\mathrm{Im}A, \mathrm{Im}\langle v|) = \mathcal{L}(V, \mathbb{C})$ such that $\langle v| = \omega_0' A$. Furthermore, we have

$$\|\omega_0 A\|^2 = \left\|\frac{|w_0\rangle}{\||w_0\rangle\|^2} + |u\rangle\right\|^2 = \left\|\omega_0' A\right\|^2 + \||u\rangle\|^2,$$

so since by optimality of ω_0, we must have $\|\omega_0 A\| \leq \|\omega_0' A\|$, it must be the case that $|u\rangle = 0$. That is, $\omega_0 A = \frac{\langle w_0|}{\||w_0\rangle\|^2} = \frac{\langle w_0|}{G_+(\mathcal{P})}$. Thus:

$$G_-(\mathcal{P}) = \|\omega_0 A\|^2 = \frac{\||w_0\rangle\|^2}{G_+(\mathcal{P})^2} = \frac{1}{G_+(\mathcal{P})},$$

|completing the proof. □

A related fact shows a similar relationship between the positive error and negative witness for a particular input x. The following lemma is actually proven in the proof of Theorem 7.3.7.

Lemma 7.4.5. *Let ω_x be an optimal negative witness for x and $|\tilde{w}_x\rangle$ an optimal min-error positive witness for x. Then the error of $|\tilde{w}_x\rangle$, $\Pi_{H(x)^\perp}|\tilde{w}_x\rangle$, is exactly*

$$\Pi_{H(x)^\perp}|\tilde{w}_x\rangle = \frac{(\omega_x A)^\dagger}{\|\omega_x A\|^2}.$$

7.5 Span Program Scaling

When presenting an algorithm for evaluating approximate span programs in the next section, it will be much clearer what is going on in the algorithm if we can make certain assumptions about the span program, such as that the optimal positive witness is a unit vector. Although we can't hope to effect a scaling that decreases $W_+(\mathcal{P},f)W_-(\mathcal{P},f)$, it is simple to scale all positive witnesses up, while simultaneously scaling negative witnesses down by the same factor, or vice versa, by simply scaling the target vector. However, it will be desirable to perform such a scaling, while independently scaling the optimal positive witness. In this section, we show how to nearly accomplish this independent scaling, showing how to normalize $G_+(\mathcal{P})$, the optimal positive witness size, to 1, with a nearly independent relative scaling of the positive and negative witnesses.

Theorem 7.5.1 (Scaling G_+). *Let $\mathcal{P} = (H, V, A, \tau)$ be a span program on $[q]^n$. Then for any positive real number β, define a span program on $[q]^n$, $\mathcal{P}^\beta = (H', V', A', \tau')$, by*

$$H' = H \oplus \mathbb{C}|0\rangle \oplus \mathbb{C}|1\rangle, \quad \forall j \in [n], a \in [q], H'_{j,a} = H_{j,a}, \quad H'_{\text{true}} = H_{\text{true}} \oplus \mathbb{C}|1\rangle, \quad H'_{\text{false}} = H_{\text{false}} \oplus \mathbb{C}|0\rangle$$

$$V' = V \oplus \mathbb{C}|1\rangle, \quad \tau' = \tau + |1\rangle,$$

and $A' \in \mathcal{L}(H', V')$ defined:

$$\forall |h\rangle \in H,\ A'|h\rangle = \beta A|h\rangle, \quad A'|0\rangle = \tau, \quad \textit{and} \quad A'|1\rangle = \frac{\sqrt{\beta^2 + G_+(\mathcal{P})}}{\beta}|1\rangle.$$

Then for all $x \in [q]^n$ with a positive witness in $\mathcal{P}$,

$$\text{wsize}_+(x, \mathcal{P}^\beta) = \frac{1}{\beta^2}\text{wsize}_+(x, \mathcal{P}) + \frac{\beta^2}{G_+(\mathcal{P}) + \beta^2};$$

for all $x \in [q]^n$ with a negative witness in $\mathcal{P}$,

$$\text{wsize}_-(x, \mathcal{P}^\beta) = \beta^2 \text{wsize}_-(x, \mathcal{P}) + 1;$$

and $G_+(\mathcal{P}^\beta) = 1$.

Proof. Let $|w_0\rangle$ be the globally optimal positive witness of $\mathcal{P}$. Then it's clear that the globally optimal positive witness of $\mathcal{P}'$ must have the form:

$$|w_0'\rangle = \frac{1-\alpha}{\beta}|w_0\rangle + \alpha|0\rangle + \frac{\beta}{\sqrt{\beta^2 + G_+(\mathcal{P})}}|1\rangle,$$

for some parameter α. In particular, since $|w_0'\rangle$ is assumed to be optimal, it must be the α that minimizes:

$$\left\||w_0'\rangle\right\|^2 = \frac{(1-\alpha)^2}{\beta^2}G_+(\mathcal{P}) + \alpha^2 + \frac{\beta^2}{\beta^2 + G_+(\mathcal{P})}.$$

We can compute this as follows:

$$\begin{aligned}
-2\frac{1-\alpha}{\beta^2}G_+(\mathcal{P}) + 2\alpha &= 0\\
\alpha\left(1 + \frac{G_+(\mathcal{P})}{\beta^2}\right) &= \frac{G_+(\mathcal{P})}{\beta^2}\\
\alpha &= \frac{G_+(\mathcal{P})}{\beta^2 + G_+(\mathcal{P})}.
\end{aligned}$$

Plugging in the computed value for α, we get:

$$\begin{aligned}
|w_0'\rangle &= \frac{\frac{\beta^2}{\beta^2+G_+(\mathcal{P})}}{\beta}|w_0\rangle + \frac{G_+(\mathcal{P})}{\beta^2 + G_+(\mathcal{P})}|0\rangle + \frac{\beta}{\sqrt{\beta^2 + G_+(\mathcal{P})}}|1\rangle\\
&= \frac{\beta}{\beta^2 + G_+(\mathcal{P})}|w_0\rangle + \frac{G_+(\mathcal{P})}{\beta^2 + G_+(\mathcal{P})}|0\rangle + \frac{\beta}{\sqrt{\beta^2 + G_+(\mathcal{P})}}|1\rangle.
\end{aligned}$$

From which we can compute:

$$\begin{aligned}
G_+(\mathcal{P}^\beta) = \left\||w_0'\rangle\right\|^2 &= \frac{\beta^2 G_+(\mathcal{P})}{(\beta^2 + G_+(\mathcal{P}))^2} + \frac{G_+(\mathcal{P})^2}{(\beta^2 + G_+(\mathcal{P}))^2} + \frac{\beta^2}{\beta^2 + G_+(\mathcal{P})}\\
&= \frac{\beta^2 G_+(\mathcal{P}) + G_+(\mathcal{P})^2 + \beta^2(\beta^2 + G_+(\mathcal{P}))}{(\beta^2 + G_+(\mathcal{P}))^2} = 1.
\end{aligned}$$

Similarly, let $|w_x\rangle$ be the optimal positive witness for some x in $\mathcal{P}$. Since $|0\rangle \notin H(x)$, it's clear that the optimal positive witness for x has the form:

$$|w_x'\rangle = \frac{1}{\beta}|w_x\rangle + \frac{\beta}{\sqrt{\beta^2 + G_+(\mathcal{P})}}|1\rangle.$$

We thus have

$$\mathrm{wsize}_+(x,\mathcal{P}^\beta) = \left\||w'_x\rangle\right\|^2 = \frac{1}{\beta^2}\mathrm{wsize}_+(x,\mathcal{P}) + \frac{\beta^2}{\beta^2+G_+(\mathcal{P})}.$$

Finally, let ω_x be the optimal negative witness for some x in $\mathcal{P}$. Suppose ω'_x is an optimal negative witness for x in $\mathcal{P}^\beta$. Then $\omega'_x A\Pi_{H'(x)} = 0$, so in particular $\omega'_x A'|1\rangle = 0$, so $\omega'_x|1\rangle = 0$. We also need $\omega'_x(\tau + |1\rangle) = 1$, so $\omega'_x\tau = 1$. Thus, ω'_x is a vector in V' such that $\omega'_x\tau = 1$ and $\omega'_x A'\Pi_{H(x)} = 0$ that minimizes:

$$\left\|\omega'_x A'\right\|^2 = \left\|\omega'_x A\Pi_H + \omega'_x\tau\langle 0|\right\|^2 = \left\|\omega'_x A\right\|^2 + 1,$$

and thus ω'_x minimizes $\|\omega'_x A\|^2$, and so we have $\|\omega'_x A\|^2 = \|\omega_x A\|^2 = \mathrm{wsize}_-(x,\mathcal{P})$. Thus:

$$\mathrm{wsize}_-(x,\mathcal{P}^\beta) = \mathrm{wsize}_-(x,\mathcal{P}) + 1,$$

completing the proof. □

When we give our algorithm for evaluating a negative approximate span program, in Section 7.6.2, we will also make use of the following properties of the scaling described in Theorem 7.5.1.

Corollary 7.5.2. *For any $f : D \to \{0,1\}$ for some $D \subseteq [q]^n$, the span program defined in Theorem 7.5.1 has*

$$W_-^0(\mathcal{P}^\beta, f) = \beta^2 W_-^0(\mathcal{P},f)+1, \quad W_-^1(\mathcal{P}^\beta, f) = \beta^2 W_-^1(\mathcal{P},f)+1, \;\textit{and}\; \widetilde{W}_+(\mathcal{P},f) \leq \frac{1}{\beta^2}\widetilde{W}_+(\mathcal{P},f)+2.$$

Proof. The first two statements are easy to see, since $\mathrm{wsize}_-(x,\mathcal{P}^\beta) = \beta^2\mathrm{wsize}_-(x,\mathcal{P}) + 1$, we have

$$W_-^0(\mathcal{P}^\beta,f) = \max_{x\in f^{-1}(0)} \mathrm{wsize}_-(x,\mathcal{P}^\beta) = \max_{x\in f^{-1}(0)}\{\beta^2\mathrm{wsize}_-(x,\mathcal{P})+1\} = \beta^2 W_-^0(\mathcal{P},f)+1,$$

$$W_-^1(\mathcal{P}^\beta,f) = \min_{x\in f^{-1}(1)} \mathrm{wsize}_-(x,\mathcal{P}^\beta) = \min_{x\in f^{-1}(1)}\{\beta^2\mathrm{wsize}_-(x,\mathcal{P})+1\} = \beta^2 W_-^1(\mathcal{P},f)+1.$$

For the final fact, let $|\tilde{w}_x\rangle$ be an optimal min-error approximate witness for x in $\mathcal{P}$. Define

$$|\tilde{w}'_x\rangle := \frac{\beta\mathrm{wsize}_-(x,\mathcal{P})}{1+\beta^2\mathrm{wsize}_-(x,\mathcal{P})}|\tilde{w}_x\rangle + \frac{1}{1+\beta^2\mathrm{wsize}_-(x,\mathcal{P})}|0\rangle + \frac{\beta}{\sqrt{\beta^2+G_+(\mathcal{P})}}|1\rangle.$$

We have

$$A'|\tilde{w}'_x\rangle \;=\; \frac{\beta^2\mathrm{wsize}_-(x,\mathcal{P})}{1+\beta^2\mathrm{wsize}_-(x,\mathcal{P})}A|\tilde{w}_x\rangle + \frac{1}{1+\beta^2\mathrm{wsize}_-(x,\mathcal{P})}\tau + |1\rangle = \tau + |1\rangle = \tau'$$

so $|\tilde{w}_x'\rangle$ is an approximate witness for x. Since $H'(x)^\perp = H(x)^\perp \oplus \mathrm{span}\{|0\rangle\}$, it has error:

$$\begin{aligned}
&\left\|\Pi_{H'(x)^\perp}|\tilde{w}_x'\rangle\right\|^2\\
=\;& \frac{\beta^2\mathrm{wsize}_-(x,\mathcal{P})^2}{(1+\beta^2\mathrm{wsize}_-(x,\mathcal{P}))^2}\left\|\Pi_{H(x)^\perp}|\tilde{w}_x\rangle\right\|^2 + \frac{1}{(1+\beta^2\mathrm{wsize}_-(x,\mathcal{P}))^2}\\
=\;& \frac{\beta^2\mathrm{wsize}_-(x,\mathcal{P})}{(1+\beta^2\mathrm{wsize}_-(x,\mathcal{P}))^2} + \frac{1}{(1+\beta^2\mathrm{wsize}_-(x,\mathcal{P}))^2} \quad \text{since } \mathrm{error}_+(x,\mathcal{P}) = \frac{1}{\mathrm{wsize}_-(x,\mathcal{P})}\\
=\;& \frac{1}{1+\beta^2\mathrm{wsize}_-(x,\mathcal{P})} = \frac{1}{\mathrm{wsize}_-(x,\mathcal{P}^\beta)},
\end{aligned}$$

which we know is optimal. Thus $|\tilde{w}_x'\rangle$ is a min-error approximate witness for x in $\mathcal{P}^\beta$. Its complexity is

$$\begin{aligned}
\widetilde{\mathrm{wsize}}_+(x,\mathcal{P}^\beta) &\leq \left\||\tilde{w}_x'\rangle\right\|^2\\
&= \frac{\beta^2\mathrm{wsize}_-(x,\mathcal{P})^2}{(1+\beta^2\mathrm{wsize}_-(x,\mathcal{P}))^2}\left\||\tilde{w}_x\rangle\right\|^2 + \frac{1}{(1+\beta^2\mathrm{wsize}_-(x,\mathcal{P}))^2} + \frac{\beta^2}{\beta^2+G_+(\mathcal{P})}\\
&\leq \frac{\beta^2\mathrm{wsize}_-(x,\mathcal{P})^2\widetilde{\mathrm{wsize}}_+(x,\mathcal{P})+1}{(1+\beta^2\mathrm{wsize}_-(x,\mathcal{P}))^2}+1\\
&\leq \frac{\beta^2\mathrm{wsize}_-(x,\mathcal{P})^2\widetilde{\mathrm{wsize}}_+(x,\mathcal{P})}{1+2\beta^2\mathrm{wsize}_-(x,\mathcal{P})+\beta^4\mathrm{wsize}_-(x,\mathcal{P})^2}+2\\
&= \frac{\beta^2\mathrm{wsize}_-(x,\mathcal{P})^2\widetilde{\mathrm{wsize}}_-(x,\mathcal{P})}{\beta^4\mathrm{wsize}_-(x,\mathcal{P})^2}+2 = \frac{\widetilde{\mathrm{wsize}}_+(x,\mathcal{P})}{\beta^2}+2.
\end{aligned}$$

Thus, we can compute

$$\widetilde{W}_+^1(\mathcal{P}^\beta,f) = \max_{x\in f^{-1}(1)}\widetilde{\mathrm{wsize}}_+(x,\mathcal{P}^\beta) \leq \max_{x\in f^{-1}(1)}\left(\frac{\widetilde{\mathrm{wsize}}_+(x,\mathcal{P})}{\beta^2}+2\right) = \frac{1}{\beta^2}\widetilde{W}_+^1(\mathcal{P},f)+2.$$

□

7.6 Algorithm for evaluating approximate span programs

In this section, we show that our span program definitions are actually useful by giving algorithms for "evaluating" span programs. These algorithms are similar to previous algorithms for evaluating span programs, such as [Rei09], however, the understanding of the structure of span programs gained in Section 7.4 allows us to present a more transparent view of the structure of the algorithm, and why it works.

We begin in Section 7.6.1 by presenting and analyzing an algorithm that decides if input x is accepted by $\mathcal{P}$ or rejected. This does not show any new result, as it does not use any notion of approximate span programs, and for our other modification, that $H_{j,a}$ and $H_{j,a'}$ need not be orthogonal, it is trivial to see that the old results on evaluating span programs still hold. However, our presentation is somewhat novel, and our later algorithms build on this first algorithm. Later, in Section 7.6.2, we give an algorithm for *approximately evaluating span programs*; that is, we present an algorithm that decides if input x has negative witness size above a given threshold, or below a given threshold. Applying this construction to a span program $\mathcal{P}$ that is a λ-negative approximate span program for a decision problem f gives an algorithm that decides f with bounded error, whose quantum query complexity is $O\left(\sqrt{W_-^0(\mathcal{P},f)\widetilde{W}_+^1(\mathcal{P},f)}\right)$ for any constant λ, or more generally, scales as $O\left(\sqrt{W_-^0(\mathcal{P},f)\widetilde{W}_+^1(\mathcal{P},f)}\frac{1}{(1-\lambda^{1/4})^2}\log\frac{1}{1-\lambda^{1/4}}\right)$.

The following algorithm will be the basis of both the algorithm in Section 7.6.1 and the algorithm in Section 7.6.2.

The Basic Algorithm Let $\mathcal{P}$ be a span program with $G_+(\mathcal{P})=1$. Let $|w_0\rangle$ be the globally optimal positive witness for $\mathcal{P}$ (which is a unit vector by assumption). Define $U(\mathcal{P},x) := (2\Pi_{H(x)^\perp}-I)(2\Pi_{\ker A^\perp}-I)$. Then the algorithm $\texttt{BasicSpan}_{\mathcal{P},\Theta,\epsilon}(x)$ consists of running phase estimation, from Theorem 3.1.7, on $U(\mathcal{P},x)$ applied to $|w_0\rangle$ to precision Θ and error ϵ. If we measure 0, we reject, and otherwise, we accept.

We can easily analyze the quantum query complexity of $\texttt{BasicSpan}$. The phase estimation makes $O(\frac{1}{\Theta}\log\frac{1}{\epsilon})$ controlled calls to $U(\mathcal{P},x)$. We have the following.

Lemma 7.6.1. *The operator $U(\mathcal{P},x)$ can be implemented using 2 queries to x.*

Proof. The reflection $2\Pi_{\ker A^\perp}-I$ is independent of the input, so it has query complexity 0, although its time complexity may be quite high. We have $H(x)=\bigoplus_{j\in[n]}H_{j,x_j}$. For every $j\in[n]$ and $a\in[q]$, let $R_{j,a}$ reflect states in $H_{j,a}^\perp\cap H_j$. Implementing $R_{j,a}$ is also input-independent, and so has query complexity 0, although again it may have high time complexity. Since the H_j are orthogonal, we can map $|0\rangle|0\rangle|\psi_{j,a,\ell}\rangle\mapsto|j\rangle|0\rangle|\psi_{j,a,\ell}\rangle\mapsto|j\rangle|x_j\rangle|\psi_{j,a,\ell}\rangle$ using one query to the input. We can then apply R_{j,x_j}, controlled on the first two registers, in 0 queries, before uncomputing the first two registers using one more query, and accomplishing $2\Pi_{H(x)^\perp}-I$. Thus $U(\mathcal{P},x)$ can be implemented using 2 queries to x. $\square$

Thus, the quantum query complexity of the algorithm $\texttt{BasicSpan}_{\mathcal{P},\Theta,\epsilon}$ is $O\left(\frac{1}{\Theta}\log\frac{1}{\epsilon}\right)$.

We will make use of the following lemma, which can be derived from Theorem 4.1.4, to analyze the action of $U(\mathcal{P},x)$ on $|w_0\rangle$.

Lemma 7.6.2 (Effective Spectral Gap Lemma [LMR+11]). *Let $U = (2\Pi_A - I)(2\Pi_B - I)$ be the product of two reflections, and let Π_Θ be the orthogonal projector onto*

$$\mathrm{span}\{|u\rangle : U|u\rangle = e^{i\theta}|u\rangle, |\theta| \leq \Theta\}.$$

Then if $\Pi_A|u\rangle = 0$,

$$\|\Pi_\Theta \Pi_B |u\rangle\| \leq \frac{\Theta}{2} \||u\rangle\|.$$

7.6.1 Non-approximate Span Programs

We first show how the algorithm `BasicSpan` can be used to evaluate span programs in the traditional sense, in which we want to accept all inputs that are positive for $\mathcal{P}$, and reject all inputs that are negative for $\mathcal{P}$. Specifically, we prove the following theorem.

Theorem 7.6.3. *Let $f : D \to \{0,1\}$, $D \subseteq [q]^n$ be any decision problem such that $\mathcal{P}$ decides f. Then the query complexity of f is at most $O\left(\sqrt{W_-(\mathcal{P},f)W_+(\mathcal{P},f)}\right)$.*

Upper Bound on Rejection Error We start by showing an upper bound on the probability that a positive input is rejected. In order to do this, we simply show that the overlap with small phase subspaces can be upper bounded in terms of the positive witness size. We notice then that the rejection probability goes down with the witness size.

Lemma 7.6.4. *For any $x \in [q]^n$,*

$$\|\Pi_\Theta |w_0\rangle\|^2 \leq \frac{\Theta^2}{4} \mathrm{wsize}_+(x,\mathcal{P}).$$

Proof. If there is no positive witness for x, then $\mathrm{wsize}_+(x,\mathcal{P}) = \infty$, and the statement is vacuously true. Otherwise, let $|w_x\rangle = |w_0\rangle + |w_{0,x}^\perp\rangle$, with $|w_{0,x}^\perp\rangle \in \ker A$, be an optimal positive witness for x, so $\||w_x\rangle\|^2 = \mathrm{wsize}_+(x,\mathcal{P})$. Since $\Pi_{H(x)^\perp}|w_x\rangle = 0$, by the Effective Spectral Gap Lemma, we have $\|\Pi_\Theta \Pi_{\ker A^\perp}|w_x\rangle\| \leq \frac{\Theta}{2}\||w_x\rangle\|$. Thus, we have:

$$\|\Pi_\Theta|w_0\rangle\| = \|\Pi_\Theta \Pi_{\ker A^\perp}|w_x\rangle\| \leq \frac{\Theta}{2}\||w_x\rangle\| = \frac{\Theta}{2}\sqrt{\mathrm{wsize}_+(x,\mathcal{P})}.$$

The statement follows immediately. □

Corollary 7.6.5. *If $f(x) = 1$, the probability $\texttt{BasicSpan}_{\mathcal{P},\Theta,\epsilon}$ rejects x is at most $\frac{\Theta^2}{4}W_+(\mathcal{P},f)+\epsilon$.*

Proof. By Lemma 7.6.4, if $f(x)=1$, then $\|\Pi_\Theta|w_0\rangle\|^2 \leq \frac{\Theta^2}{4}W_+(\mathcal{P},f)$. Let Φ be the unitary action of applying phase estimation on $U(\mathcal{P},x)$ to precision Θ. Then the probability the algorithm rejects is:

$$\begin{aligned}|\langle 0|\Phi|w_0\rangle|^2 &= |\langle 0|\Phi\Pi_\Theta|w_0\rangle|^2 + |\langle 0|\Phi(I-\Pi_\Theta)|w_0\rangle|^2 \\ &\leq \frac{\Theta^2}{4}W_+(\mathcal{P},f)+\epsilon, \text{ by Theorem 3.1.7.}\end{aligned}$$

□

Acceptance Error Next we show an upper bound on the probability that a negative input is accepted. In order to do this, we simply show that when the input has a negative witness, $|w_0\rangle$ has high overlap with the 0-phase space.

Lemma 7.6.6. *For any input $x\in[q]^n$,*

$$\|\Pi_0|w_0\rangle\|^2 \geq \frac{1}{\mathrm{wsize}_-(x,\mathcal{P})}.$$

Proof. If there is no negative witness then $\mathrm{wsize}_-(x,\mathcal{P})=\infty$ and the statement is vacuously true. Otherwise, let ω_x be an optimal negative witness for x in $\mathcal{P}$, so $\omega_x\tau=1$, $\left\|\omega_x A\Pi_{H(x)}\right\|=0$, and $\|\omega_x A\|^2 = \mathrm{wsize}_-(x,\mathcal{P})$. Define $|u\rangle = (\omega_x A)^\dagger$. We clearly have $\Pi_{\ker A^\perp}|u\rangle = |u\rangle$, and $\Pi_{H(x)^\perp}|u\rangle = |u\rangle$, since $\left\|\omega_x A\Pi_{H(x)}\right\|=0$, so $U(\mathcal{P},x)|u\rangle = |u\rangle$ — i.e., $|u\rangle$ is in the 0-phase space of $U(\mathcal{P},x)$. We have:

$$\langle u|w_0\rangle = \omega_x A|w_0\rangle = \omega_x\tau = 1$$

and

$$\||u\rangle\|^2 = \|\omega_x A\|^2 = \mathrm{wsize}_-(x,\mathcal{P}),$$

so we have

$$\|\Pi_0|w_0\rangle\|^2 \geq \left\|\frac{|u\rangle\langle u|}{\||u\rangle\|^2}|w_0\rangle\right\|^2 = \frac{1}{\mathrm{wsize}_-(x,\mathcal{P})}.$$

□

Corollary 7.6.7. *If $f(x)=0$, then the probability* $\texttt{BasicSpan}_{\mathcal{P},\Theta,\epsilon}$ *rejects x is at least $\frac{1}{W_-(\mathcal{P},f)}$.*

Proof. By Lemma 7.6.6, if $f(x)=0$, then $\|\Pi_0|w_0\rangle\|^2 \geq \frac{1}{W_-(\mathcal{P},f)}$. Since phase estimation outputs 0 with certainty on input $\Pi_0|w_0\rangle$, the probability phase estimation outputs 0, and thus the algorithm rejects, is at least $\frac{1}{W_-(\mathcal{P},f)}$. □

Final Algorithm We now give the final algorithm to evaluate any span program.

```
ExactSpan$_{\mathcal{P},\varepsilon}(x)$

1. Let $\mathcal{P} = (H, V, A, \tau)$, and define $\mathcal{P}'$ to be $\mathcal{P}^\beta$ from Theorem 7.5.1, for $\beta = \sqrt{\frac{\varepsilon}{W_-(\mathcal{P},f)(1-\varepsilon)}}$
2. Define $\Theta = \sqrt{\frac{2\varepsilon}{\frac{(1-\varepsilon)}{\varepsilon}W_-(\mathcal{P},f)W_+(\mathcal{P},f)+1}}$ and $\epsilon = \frac{\varepsilon}{2}$
3. Run BasicSpan$_{\mathcal{P}',\Theta,\epsilon}(x)$ and output the outcome
```

Theorem 7.6.8. *Let $\mathcal{P}$ be a span program on $[q]^n$, and let f be any decision problem with domain $D \subseteq [q]^n$ such that if $f(x) = 1$, then $\mathcal{P}$ accepts x and if $f(x) = 0$ then $\mathcal{P}$ rejects x. Then for any $\varepsilon \in (0, 1/2)$,* `ExactSpan`$_{\mathcal{P},\varepsilon}(x)$ *solves f with bounded error ε in $O\left(\sqrt{W_-(\mathcal{P})W_+(\mathcal{P})}\frac{1}{\varepsilon}\log\frac{1}{\varepsilon}\right)$ quantum query complexity.*

Proof. The complexity of `BasicSpan`$_{\mathcal{P},\Theta,\epsilon}$ is $O(1/\Theta \log \frac{1}{\epsilon})$ quantum queries, so the query complexity of `ExactSpan` follows.

We now show correctness. We first note that by Theorem 7.5.1, $G_+(\mathcal{P}') = 1$, so we can apply `BasicSpan` to $\mathcal{P}'$. Also by Theorem 7.5.1, we have

$$W_+(\mathcal{P}^\beta, f) = \frac{1}{\beta^2}W_+(\mathcal{P}, f) + \frac{\beta^2}{G_+(\mathcal{P}) + \beta^2} \quad \text{and} \quad W_-(\mathcal{P}^\beta, f) = \beta^2 W_-(\mathcal{P}, f) + 1,$$

so plugging in $\beta = \sqrt{\frac{\varepsilon}{W_-(\mathcal{P},f)(1-\varepsilon)}}$, and using $\beta^2 < G_+(\mathcal{P}) + \beta^2$, we get

$$W_+(\mathcal{P}', f) \leq \frac{1-\varepsilon}{\varepsilon}W_-(\mathcal{P}, f)W_+(\mathcal{P}, f) + 1, \quad \text{and} \quad W_-(\mathcal{P}', f) = \frac{\varepsilon}{1-\varepsilon} + 1 = \frac{1}{1-\varepsilon}.$$

Suppose $x \in f^{-1}(1)$. Then by Corollary 7.6.5, `BasicSpan`$_{\mathcal{P}',\Theta,\epsilon}$ errs (in this case, rejects) with probability at most

$$\frac{\Theta^2}{4}W_+(\mathcal{P}', f) + \epsilon \leq \frac{1}{2}\frac{\varepsilon}{\frac{1-\varepsilon}{\varepsilon}W_-(\mathcal{P}, f)W_+(\mathcal{P}, f) + 1}\left(\frac{1-\varepsilon}{\varepsilon}W_-(\mathcal{P}, f)W_+(\mathcal{P}, f) + 1\right) + \frac{\varepsilon}{2} = \varepsilon.$$

Next, suppose $x \in f^{-1}(0)$, then by Corollary 7.6.7, `BasicSpan`$_{\mathcal{P}',\Theta,\epsilon}$ errs (in this case, accepts) with probability at most

$$1 - \frac{1}{W_-(\mathcal{P}')} = 1 - (1-\varepsilon) = \varepsilon.$$

Thus, the error probability on any input is at most ε, completing the proof. $\square$

7.6.2 Evaluating Approximate Span Programs

In this section, we will use the algorithm `BasicSpan` to prove the following theorem.

Theorem 7.6.9. *If $\mathcal{P}$ is a λ-negative approximate span program for f, then there exists an algorithm that decides f with bounded error in quantum query complexity:*

$$O\left(\frac{\sqrt{W_-^0(\mathcal{P},f)\widetilde{W}_+^1(\mathcal{P},f)}}{(1-\lambda^{1/4})^2}\log\frac{1}{1-\lambda^{1/4}}\right).$$

To prove this theorem, we will analyze the behaviour of `BasicSpan` on a span program with $G_+(\mathcal{P}) = 1$, as in the previous section, but this time, we will consider the case when positive witnesses may have some error. Just as in the previous section, we will use the results of Section 7.5 to transform an arbitrary span program to a span program $\mathcal{P}'$ with $G_+(\mathcal{P}') = 1$, and run `BasicSpan` on $\mathcal{P}'$.

Upper Bound on Rejection Error The following lemma is very similar to Lemma 7.6.4, except that we now consider positive witnesses that may have some error.

Lemma 7.6.10. *For any $x \in [q]^n$,*

$$\|\Pi_\Theta|w_0\rangle\|^2 \leq \left(\frac{\Theta}{2}\sqrt{\widetilde{\mathrm{wsize}}_+(x,\mathcal{P})} + \frac{1}{\sqrt{\mathrm{wsize}_-(x,\mathcal{P})}}\right)^2.$$

Proof. Let $|w_x\rangle = |w_0\rangle + |w_0^\perp\rangle$ be an optimal min-error approximate witness for x, so $\left\|\Pi_{H(x)^\perp}|w\rangle\right\|^2 = \frac{1}{\mathrm{wsize}_-(x,\mathcal{P})}$, and minimal witness size $\||w_x\rangle\|^2 = \widetilde{\mathrm{wsize}}_+(x,\mathcal{P})$. We have:

$$|w_0\rangle = \Pi_{\ker A^\perp}|w_x\rangle = \Pi_{\ker A^\perp}\Pi_{H(x)}|w_x\rangle + \Pi_{\ker A^\perp}\Pi_{H(x)^\perp}|w_x\rangle.$$

By the effective spectral gap Lemma,

$$\text{since } \Pi_{H(x)^\perp}\Pi_{H(x)}|w_x\rangle = 0, \text{ we have } \left\|\Pi_\Theta\Pi_{\ker A^\perp}\Pi_{H(x)}|w_x\rangle\right\| \leq \frac{\Theta}{2}\left\|\Pi_{H(x)}|w_x\rangle\right\|.$$

Then we have:

$$\begin{aligned}
\|\Pi_\Theta|w_0\rangle\| &= \left\|\Pi_\Theta\Pi_{\ker A^\perp}\Pi_{H(x)}|w_x\rangle + \Pi_\Theta\Pi_{\ker A^\perp}\Pi_{H(x)^\perp}|w_x\rangle\right\| \\
&\leq \left\|\Pi_\Theta\Pi_{\ker A^\perp}\Pi_{H(x)}|w_x\rangle\right\| + \left\|\Pi_\Theta\Pi_{\ker A^\perp}\Pi_{H(x)^\perp}|w_x\rangle\right\| \\
&\leq \frac{\Theta}{2}\left\|\Pi_{H(x)}|w_x\rangle\right\| + \left\|\Pi_{H(x)^\perp}|w_x\rangle\right\| \\
&\leq \frac{\Theta}{2}\sqrt{\widetilde{\mathrm{wsize}}_+(x,\mathcal{P})} + \frac{1}{\sqrt{\mathrm{wsize}_-(x,\mathcal{P})}}.
\end{aligned}$$

The result follows. □

Corollary 7.6.11. *For all $x \in f^{-1}(1)$ the probability that* `BasicSpan`$_{\mathcal{P},\Theta,\epsilon}$ *rejects on input x is at most:*

$$\left(\frac{\Theta}{2}\sqrt{\widetilde{W}_+^1(\mathcal{P},f)} + \frac{1}{\sqrt{W_-^1(\mathcal{P},f)}}\right)^2 + \epsilon.$$

Proof. By Lemma 7.6.10, if $f(x)=1$, then $\|\Pi_\Theta|w_0\rangle\|^2 \leq \left(\frac{\Theta}{2}\sqrt{\widetilde{W}_+^1(\mathcal{P},f)} + \frac{1}{\sqrt{W_-^1(\mathcal{P},f)}}\right)^2$. Let Φ be the unitary action of applying phase estimation on $U(\mathcal{P},x)$ to precision Θ. Then the probability we reject is:

$$\begin{aligned}
|\langle 0|\Phi|w_0\rangle|^2 &= |\langle 0|\Phi\Pi_\Theta|w_0\rangle|^2 + |\langle 0|\Phi(I-\Pi_\Theta)|w_0\rangle|^2 \\
&\leq \left(\frac{\Theta}{2}\sqrt{\widetilde{W}_+^1(\mathcal{P},f)} + \frac{1}{\sqrt{W_-^1(\mathcal{P},f)}}\right)^2 + \epsilon, \text{ by Theorem 3.1.7.}
\end{aligned}$$

□

Upper Bound on Acceptance Error By Lemma 7.6.6, for all $x \in [q]^n$, $\|\Pi_0|w_0\rangle\|^2 \geq \frac{1}{\mathrm{wsize}_-(x,\mathcal{P})}$. As a corollary, we get the following.

Corollary 7.6.12. *For all $x \in f^{-1}(0)$, the probability that* `BasicSpan`$_{\mathcal{P},\Theta,\epsilon}$ *rejects on input x is at least $\frac{1}{W_-^0(\mathcal{P},f)}$.*

Proof. The algorithm `BasicSpan` rejects whenever a 0 is measured, which happens with probability at least $\|\Pi_0|w_0\rangle\|^2$. The result then follows from Lemma 7.6.6. □

Full Algorithm We are now ready to construct an algorithm that evaluates *any* λ-negative span program, $\mathcal{P}$, with no assumption on $G_+(\mathcal{P})$. We first give an algorithm that may not have very high success probability, and later describe how to amplify the success probability to get bounded error.

```
NegativeSpan_{$\mathcal{P},\varepsilon$}($x$)

  1. Let $\mathcal{P} = (H, V, A, \tau)$, and define $\mathcal{P}'$ to be $\mathcal{P}^\beta$ from Theorem 7.5.1, for $\beta = \frac{\lambda^{1/4}}{\sqrt{W_-^0(\mathcal{P},f)}}$
  2. Define $\Theta = \frac{1-\lambda^{1/4}}{\sqrt{W_-^0(\mathcal{P}',f)\widetilde{W}_+^1(\mathcal{P}',f)}}$, and $\epsilon = \frac{1-\lambda^{1/4}}{8}$
  3. Run BasicSpan_{$\mathcal{P}',\Theta,\epsilon$}($x$) and output the outcome
```

In the following theorem we restrict to $\lambda \geq \frac{1}{2}$, but this is not really a restriction at all, since if $\mathcal{P}$ is a λ-negative approximate span program for f with $\lambda < \frac{1}{2}$, then $\mathcal{P}$ is also a $\frac{1}{2}$-negative span program for f, since

$$\frac{W_-^0(\mathcal{P},f)}{W_-^1(\mathcal{P},f)} \leq \lambda \leq \frac{1}{2}.$$

Theorem 7.6.13. *For any $\lambda \in [1/2, 1)$, if $\mathcal{P}$ is a λ-negative span program for f, then the algorithm* `NegativeSpan` *has quantum query complexity*

$$O\left(\frac{\sqrt{W_-^0(\mathcal{P},f)\widetilde{W}_+^1(\mathcal{P},f)}}{1-\lambda^{1/4}}\log\frac{1}{1-\lambda^{1/4}}\right).$$

Furthermore, for any x such that $f(x) = 0$, `NegativeSpan` *rejects with probability at least $p_0 = \frac{1}{\sqrt{\lambda}+1}$, and for any x such that $f(x) = 1$,* `NegativeSpan` *rejects with probability at most $p_1 = \frac{(1/2+1/2\lambda^{1/4})^2}{1+\sqrt{\lambda}} + \frac{1-\lambda^{1/4}}{8}$. Furthermore, we have $p_0 - p_1 \geq \frac{1-\lambda^{1/4}}{4} > 0$.*

Proof. The query complexity of `BasicSpan` is $O\left(\frac{1}{\Theta}\log\frac{1}{\epsilon}\right)$, and thus the query complexity of `NegativeSpan` is $O\left(\frac{1}{\Theta}\log\frac{1}{\epsilon}\right)$. By Corollary 7.5.2, we have

$$W_-^0(\mathcal{P}',f) = \frac{\sqrt{\lambda}}{W_-^0(\mathcal{P},f)}W_-^0(\mathcal{P},f) + 1 = \sqrt{\lambda} + 1,$$

$$W_-^1(\mathcal{P}',f) \geq \frac{\sqrt{\lambda}}{W_-^0(\mathcal{P},f)} W_-^1(\mathcal{P},f) + 1 \geq \frac{1}{\sqrt{\lambda}} + 1, \text{ since } \lambda \geq \frac{W_-^0(\mathcal{P},f)}{W_-^1(\mathcal{P},f)},$$

$$\text{and} \quad \widetilde{W}_+^1(\mathcal{P}',f) \leq \frac{W_-^0(\mathcal{P},f)}{\sqrt{\lambda}} \widetilde{W}_+^1(\mathcal{P},f) + 2.$$

We can thus compute the query complexity of `NegativeSpan` as order of:

$$\begin{aligned}
\frac{1}{\Theta}\log\frac{1}{\epsilon} &= \frac{\sqrt{\frac{\sqrt{\lambda}+1}{\sqrt{\lambda}} W_-^0(\mathcal{P},f)\widetilde{W}_+^1(\mathcal{P},f) + 2(\sqrt{\lambda}+1)}}{1-\lambda^{1/4}} \log\frac{8}{1-\lambda^{1/4}} \\
&= O\left(\frac{\sqrt{W_-^0(\mathcal{P},f)\widetilde{W}_+^1(\mathcal{P},f)}}{1-\lambda^{1/4}}\log\frac{1}{1-\lambda^{1/4}}\right).
\end{aligned}$$

Next, by Corollary 7.6.12, for any $x \in f^{-1}(0)$, the probability of rejection is at least

$$\frac{1}{W_-^0(\mathcal{P}',f)} = \frac{1}{\sqrt{\lambda}+1}.$$

By Corollary 7.6.11, for any $x \in f^{-1}(1)$, the probability of rejection is at most

$$\begin{aligned}
&\left(\frac{\Theta}{2}\sqrt{\widetilde{W}_+^1(\mathcal{P}',f)} + \frac{1}{\sqrt{W_-^1(\mathcal{P}',f)}}\right)^2 + \epsilon \\
= &\left(\frac{1-\lambda^{1/4}}{2\sqrt{W_-^0(\mathcal{P}',f)\widetilde{W}_+^1(\mathcal{P}',f)}}\sqrt{\widetilde{W}_+^1(\mathcal{P}',f)} + \frac{1}{\sqrt{1/\sqrt{\lambda}+1}}\right)^2 + \frac{1-\lambda^{1/4}}{8} \\
\leq &\left(\frac{1-\lambda^{1/4}}{2\sqrt{\sqrt{\lambda}+1}} + \sqrt{\frac{\sqrt{\lambda}}{1+\sqrt{\lambda}}}\right)^2 = \frac{\left(\frac{1}{2}+\frac{1}{2}\lambda^{1/4}\right)^2}{1+\sqrt{\lambda}} + \frac{1-\lambda^{1/4}}{8}.
\end{aligned}$$

Finally, we can compute:

$$\begin{aligned} p_0 - p_1 &= \frac{1-\frac{1}{4}(1+\lambda^{1/4})^2}{\sqrt{\lambda}+1} - \frac{1-\lambda^{1/4}}{8} = \frac{\frac{3}{4}-\frac{1}{4}\sqrt{\lambda}-\frac{1}{2}\lambda^{1/4}}{\sqrt{\lambda}+1} - \frac{1-\lambda^{1/4}}{8} \\ &\geq \frac{3}{8}\left(1-\lambda^{1/4}\right) - \frac{1-\lambda^{1/4}}{8} = \frac{1-\lambda^{1/4}}{4}. \end{aligned}$$

□

Since $p_0 > p_1$ whenever $\lambda < 1$, `NegativeSpan` can distinguish between 0- and 1-inputs of f with possibly small success probability. In order to turn `NegativeSpan` into a bounded error algorithm for f for any $\lambda < 1$, we will make use of the following theorem.

Theorem 7.6.14 (Quantum Amplitude Estimation [BHMT02]). *Let* `A` *be a quantum procedure that outputs a state $\sqrt{p}|0\rangle|\Psi(0)\rangle + \sqrt{1-p}|1\rangle|\Psi(1)\rangle$. Then there exists a quantum procedure that uses T calls to* `A` *and outputs an estimate of p, $\tilde{p}$, such that with probability at least $2/3$,*

$$|p - \tilde{p}| < 2\pi\frac{\sqrt{p(1-p)}}{T} + \frac{\pi^2}{T^2}.$$

Using amplitude estimation and Theorem 7.6.13, we get Theorem 7.6.9:

Corollary 7.6.15 (Theorem 7.6.9). *If $\mathcal{P}$ is a λ-negative approximate span program for f, then there exists an algorithm that decides f with bounded error in quantum query complexity:*

$$O\left(\frac{\sqrt{W_-^0(\mathcal{P},f)\widetilde{W}_+^1(\mathcal{P},f)}}{(1-\lambda^{1/4})^2}\log\frac{1}{1-\lambda^{1/4}}\right).$$

Proof. We will apply amplitude estimation with $T = \frac{4\pi}{\sqrt{p_1}(p_0-p_1)}$ to the output of `NegativeSpan` if we delay all measurements, which we can write as $\sqrt{p(x)}|0\rangle|\Psi(0,x)\rangle + \sqrt{1-p(x)}|1\rangle|\Psi(1,x)\rangle$ for some states $|\Psi(0,x)\rangle, |\Psi(1,x)\rangle$, where $p(x)$ is the probability `NegativeSpan` rejects input x. If the estimate $\tilde{p}$ is $< p_1 + p_1\frac{p_0-p_1}{p_0+p_1}$, then we accept, and otherwise, we reject.

Suppose $f(x) = 1$. Then $p(x) \leq p_1$, so with probability at least 2/3:

$$\begin{aligned}
\tilde{p} &< 2\pi\frac{\sqrt{p(x)(1-p(x))}}{T} + \frac{\pi^2}{T^2} + p(x) \\
&\leq 2\pi\frac{\sqrt{p_1}}{4\pi/(\sqrt{p_1}(p_0-p_1))} + \frac{\pi^2}{16\pi^2/(p_1(p_0-p_1)^2)} + p_1 \\
&= \frac{p_1(p_0-p_1)}{2} + \frac{p_1(p_0-p_1)^2}{16} + p_1 \\
&\leq p_1 + \frac{9}{16}p_1(p_0-p_1), \qquad \text{since } p_0 - p_1 > (p_0-p_1)^2, \\
&< p_1 + \frac{p_1(p_0-p_1)}{p_0+p_1},
\end{aligned}$$

where the last line follows from the fact that since $\lambda \in (0,1)$, $p_0 \in (1/2, 1)$ and $p_1 \in (3/8, 1/2)$, and so $p_0 + p_1 \leq 3/2 < 16/9$.

On the other hand, suppose $f(x) = 0$. Then $p(x) \geq p_0 \geq 1/2$, so with probability at least 2/3:

$$\begin{aligned}
\tilde{p} &> p(x) - 2\pi\frac{\sqrt{p(x)(1-p(x))}}{T} - \frac{\pi^2}{T^2} \\
&\geq p_0 - 2\pi\frac{\sqrt{1/2}}{4\pi/(\sqrt{p_1}(p_0-p_1))} - \frac{\pi^2}{16\pi^2/(p_1(p_0-p_1)^2)} \\
&= p_0 - \frac{\sqrt{p_1}(p_0-p_1)}{2\sqrt{2}} - \frac{p_1(p_0-p_1)^2}{16} \\
&\geq p_0 - \frac{\sqrt{1/2}(p_0-p_1)}{2\sqrt{2}} - \frac{p_0(p_0-p_1)}{16}, \qquad \text{since } p_1 < 1/2 < p_0 \text{ and } (p_0-p_1)^2 \leq p_0 - p_1, \\
&\geq p_0 - \frac{p_0}{1/2}\frac{(p_0-p_1)}{4} - \frac{p_0(p_0-p_1)}{16}, \qquad \text{since } p_0 > 1/2, \\
&\geq p_0 - \frac{9}{16}p_0(p_0-p_1) \\
&\geq p_0 - \frac{p_0(p_0-p_1)}{p_0+p_1}, \qquad \text{since } p_0 + p_1 \leq 3/2 < 16/9, \\
&= \frac{p_0(p_0+p_1) - p_0(p_0-p_1)}{p_0+p_1} = \frac{2p_0p_1}{p_0+p_1} = p_1 + \frac{p_1(p_0-p_1)}{p_0+p_1}.
\end{aligned}$$

Thus, applying amplitude estimation to `NegativeSpan` decides f with bounded error. Since this procedure calls `NegativeSpan` T times, using the bound on $p_0 - p_1$ from Theorem 7.6.13 and the fact that $p_1 \geq 3/8$, the complexity of this procedure grows as

$$\begin{aligned}
T\frac{\sqrt{W_-^0(\mathcal{P},f)\widetilde{W}_+^1(\mathcal{P},f)}}{1-\lambda^{1/4}}\log\frac{1}{1-\lambda^{1/4}} &= \frac{4\pi}{\sqrt{p_1}(p_0-p_1)}\frac{\sqrt{W_-^0(\mathcal{P},f)\widetilde{W}_+^1(\mathcal{P},f)}}{1-\lambda^{1/4}}\log\frac{1}{1-\lambda^{1/4}}\\
&\leq \frac{4\pi}{\sqrt{3/8}(1-\lambda^{1/4})/4}\frac{\sqrt{W_-^0(\mathcal{P},f)\widetilde{W}_+^1(\mathcal{P},f)}}{1-\lambda^{1/4}}\log\frac{1}{1-\lambda^{1/4}}\\
&= O\left(\frac{\sqrt{W_-^0(\mathcal{P},f)\widetilde{W}_+^1(\mathcal{P},f)}}{(1-\lambda^{1/4})^2}\log\frac{1}{1-\lambda^{1/4}}\right).
\end{aligned}$$

□

We remark that it is likely that `NegativeSpan` is not optimal for non-constant λ. There are a number of parameters that could be optimized in order to obtain a more favourable scaling in the rejection gap, when it is non-constant, however, we leave this improvement for future work.

References

[AJJ14] G. Alagic, S. Jeffery, and S. Jordan. Partial-indistinguishability obfuscation using braids. In *Proceedings of the 9th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2014)*, 2014. `arXiv:cs.CR/1212.6458`.

[Amb00] A. Ambainis. Quantum lower bounds by quantum arguments. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC 2000)*, pages 636–643, 2000.

[Amb04] A. Ambainis. Quantum walk algorithm for element distinctness. In *Proceedings of the 45th IEEE Symposium on Foundations of Computer Science (FOCS 2004)*, pages 22–31, 2004.

[Amb05] A. Ambainis. Polynomial degree and lower bounds in quantum complexity: Collision and element distinctness with small range. *Theory of Computing*, 1:37–46, 2005.

[AS04] S. Aaronson and Y. Shi. Quantum lower bounds for the collision and element distinctness problems. *Journal of the ACM*, 51:595–605, 2004.

[BBHT98] M. Boyer, G. Brassard, P. Høyer, and A. Tapp. Tight bounds on quantum searching. *Fortschritte der Physik*, 46(4-5):493–505, 1998.

[BCJ$^+$13] A. Belovs, A. M. Childs, S. Jeffery, R. Kothari, and F. Magniez. Time efficient quantum walks for 3-distinctness. In *Proceedings of the 40th International Colloquium on Automata, Languages and Programming (ICALP 2013)*, pages 105–122, 2013.

[BDH$^+$05] H. Buhrman, C. Dürr, M. Heiligman, P. Høyer, M. Santha, F. Magniez, and R. de Wolf. Quantum algorithms for Element Distinctness. *SIAM Journal of Computing*, 34(6):1324–1330, 2005.

[Bel12a] A. Belovs. Learning-graph-based quantum algorithm for k-distinctness. In *Prooceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2012)*, pages 207 – 216, 2012.

[Bel12b] A. Belovs. Span programs for functions with constant-sized 1-certificates. In *Proceedings of the 44th Symposium on Theory of Computing (STOC 2012)*, pages 77–84, 2012.

[Bel13] A. Belovs. Quantum walks and electric networks, 2013. `arXiv:1302.3143`.

[Bha93] R. Bhatia. *Matrix Analysis*. Springer-Verlag, 1993.

[BHMT02] 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: A Millennium Volume*, volume 305 of *AMS Contemporary Mathematics Series Millennium Volume*, pages 53–74. AMS, 2002. `arXiv:quant-ph/0005055v1`.

[BHT98] G. Brassard, P. Høyer, and A. Tapp. Quantum counting. In *Proceedings of the 25th International Colloquium on Automata, Languages and Programming (ICALP 1998)*, pages 820–831, 1998.

[BJLM13] D. J. Bernstein, S. Jeffery, T. Lange, and A. Meurer. Quantum algorithms for the subset sum problem. In *Proceedings of the 5th International Conference on Post-Quantum Cryptography (PQCrypto 2013)*, pages 16–33, 2013.

[BR12] A. Belovs and B. Reichardt. Span programs and quantum algorithms for *st*-connectivity and claw detection. In *Proceedings of the 20th European Symposium on Algorithms (ESA 2012)*, pages 193–204, 2012.

[CEMM98] R. Cleve, A. Ekert, C. Macchiavello, and M. Mosca. Quantum algorithms revisited. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 454(1969):339–354, 1998.

[Chi13] A. M. Childs. Lecture notes for course CO781/CS867/QIC823, 2013. Available at `http://www.math.uwaterloo.ca/~amchilds/teaching/w13/a2.pdf`.

[CJKM13] A. M. Childs, S. Jeffery, R. Kothari, and F. Magniez. A time-efficient quantum walk for 3-distinctness using nested updates, 2013. `arXiv:1302.7316`.

[CK11] A. M. Childs and R. Kothari. Quantum query complexity of minor-closed graph properties. In *Proceedings of the 28th Symposium on Theoretical Aspects of Computer Science (STACS 2011)*, pages 661–672, 2011.

[DS84] P. G. Doyle and J. L. Snell. *Random Walks and Electrical Networks*, volume 22 of *The Carus Mathematical Monographs*. The Mathematical Association of America, 1984.

[FLT14] H. Nishimura F. Le Gall and S. Tani. Quantum algorithms for finding constant-sized sub-hypergraphs. In *Proceedings of the 20th Annual International Computing and Combinatorics Conference (COCOON 2014)*, 2014. `arXiv:1310.4127`.

[God10] C. Godsil. Association schemes, 2010. Available at `http://quoll.uwaterloo.ca/pdfs/assoc2.pdf`.

[Gro96] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th ACM Symposium on Theory of Computing (STOC 1996)*, pages 212–219, 1996.

[HGJ10] N. Howgrave-Graham and A. Joux. New generic algorithms for hard knapsacks. In *Advances in Cryptology — EUROCRYPT 2014 — 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques (Eurocrypt 2010)*, 2010.

[HLŠ07] P. Høyer, T. Lee, and R. Špalek. Negative weights make adversaries stronger. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC 2007)*, pages 526–535, 2007.

[JKM13a] S. Jeffery, R. Kothari, and F. Magniez. Improving quantum query complexity of Boolean matrix multiplication using graph collision. In *Proceedings of the 39th International Colloquium on Automata, Languages and Programming (ICALP 2012)*, pages 522–532, 2013. `arXiv:1112.5855v2`.

[JKM13b] S. Jeffery, R. Kothari, and F. Magniez. Nested quantum walks with quantum data structures. In *Proceedings of the 24th ACM-SIAM Symposium On Discrete Algorithms (SODA 2013)*, pages 1474–1485, 2013. `arXiv:1210.1199`.

[JLR11] S. Janson, T. Luczak, and A. Rucinski. *Random Graphs*. Wiley Series in Discrete Mathematics and Optimization. John Wiley & Sons, 2011.

[JMd14] S. Jeffery, F. Magniez, and R. de Wolf. Optimal parallel quantum query algorithms. In *Proceedings of the 22nd European Symposium on Algorithms (ESA 2014)*, 2014.

[Jor75] C. Jordan. Essai sur la géométrie à n dimensions. *Bulletin de la Société Mathématique de France*, 3:103–174, 1875.

[Kit95] A. Kitaev. Quantum measurements and the Abelian stabilizer problem, 1995. `arXiv:quant-ph/9511026`.

[KLM06] P. Kaye, R. Laflamme, and M. Mosca. *An Introduction to Quantum Computing*. Oxford University Press, 2006.

[Kut05] S. Kutin. Quantum lower bound for the collision problem with small range. *Theory of Computing*, 1:29–36, 2005.

[KW93] M. Karchmer and A. Wigderson. On span programs. In *Proceedings of the IEEE 8th Annual Conference on Structure in Complexity Theory*, pages 102–111, 1993.

[Le 14] F. Le Gall. Improved quantum algorithm for triangle finding via combinatorial arguments. In *Proceedings of the 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2014)*, 2014. `arXiv:1407.0085`.

[LMR$^+$11] T. Lee, R. Mittal, B. Reichardt, R. Špalek, and M. Szegedy. Quantum query complexity of state conversion. In *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2011)*, pages 344–353, 2011.

[LMS13] T. Lee, F. Magniez, and M. Santha. Improved quantum query algorithms for triangle finding and associativity testing. In *Proceedings of the 24th ACM-SIAM Symposium On Discrete Algorithms (SODA 2013)*, 2013. `arXiv:1109.5135`.

[LPW09] D. A. Levin, Y. Peres, and E. L. Wilmer. *Markov Chains and Mixing Times*. American Mathematical Society, 2009.

[Mey96a] D. Meyer. From quantum cellular automata to quantum lattice gases. *Journal of Statistical Physics*, 85(5–6):551–574, 1996.

[Mey96b] D. Meyer. On the absence of homogeneous scalar unitary cellular automata. *Physical Letter A*, 223(5):337–340, 1996.

[MNRS11] F. Magniez, A. Nayak, J. Roland, and M. Santha. Search via quantum walk. *SIAM Journal on Computing*, 40(1):142–164, 2011.

[MSS07] F. Magniez, M. Santha, and M. Szegedy. Quantum algorithms for the triangle problem. *SIAM Journal on Computing*, 37(2):413–424, 2007.

[NC00] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information (Cambridge Series on Information and the Natural Sciences)*. Cambridge University Press, 1 edition, January 2000.

[Rei09] B. Reichardt. Span programs and quantum query complexity: The general adversary bound is nearly tight for every Boolean function. In *Proceedings of the 50th IEEE Symposium on Foundations of Computer Science (FOCS 2009)*, pages 544–551, 2009. `arXiv:quant-ph/0904.2759`.

[Rei11] B. Reichardt. Reflections for quantum query algorithms. In *Proceedings of the 22nd ACM-SIAM Symposium on Discrete Algorithms (SODA 2011)*, pages 560–569, 2011.

[RŠ12] B. Reichardt and R. Špalek. Span-program-based quantum algorithm for evaluating formulas, 2012.

[Sho97] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26:1484–1509, October 1997.

[SKW03] N. Shenvi, J. Kempe, and K. B. Whaley. Quantum random-walk search algorithm. *Physical Review A*, 67, 2003. Article no. 052307.

[Sze04] M. Szegedy. Quantum speed-up of Markov chain based algorithms. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2004)*, pages 32–41, 2004.

[Wat01] J. Watrous. Quantum simulations of classical random walks and undirected graph connectivity. *Journal of Computer System Sciences*, 62(2):376–391, 2001.

[WC81] M. N. Wegman and L. Carter. New hash functions and their use in authentication and set equality. *Journal of Computer System Sciences*, 22(3):265–279, 1981.