# On Span Programs and Quantum Algorithms

Álvaro Piedrafita

# On span programs and quantum algorithms

ILLC Dissertation Series DS-2021-XX

Institute for Logic, Language and Computation

For further information about ILLC publications, please contact

Institute for Logic, Language and Computation
Universiteit van Amsterdam
Science Park 107
1098 XG Amsterdam
phone: +31-20-525 6051
e-mail: illc@uva.nl
homepage: http://www.illc.uva.nl/






Cover design by Guillem Galobardes.
Printed and bound by NBD Biblion.

ISBN: 978-90-619-6149-9.

# On span programs and quantum algorithms

Academisch Proefschrift

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. ir. K.I.J. Maex
ten overstaan van een door het College voor Promoties ingestelde commissie,
in het openbaar te verdedigen in de Agnietenkapel
op woensdag 10 november 2021, te 16.00 uur

door

Alvaro Piedrafita Postigo

geboren te Sabadell, Spanje

## Promotiecommissie

| | | |
|---|---|---|
| Promotor: | prof. dr. H.M. Buhrman | Universiteit van Amsterdam |
| Co-promotor: | dr. S.M. Jeffery | Centrum Wiskunde & Informatica |
| Overige leden: | prof. dr. R.M. de Wolf | Universiteit van Amsterdam |
| | prof. dr. C.J.M. Schoutens | Universiteit van Amsterdam |
| | prof. dr. S. Fehr | Universiteit Leiden |
| | dr. S. Kimmel | Middlebury College |
| | dr. M.Ozols | Universiteit van Amsterdam |

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

# List of publications

This dissertation is based on the following papers (in chronological order). In each work, all authors contributed equally unless stated otherwise.

[JJK+18] ***Quantum algorithms for connectivity and related problems***
Michael Jarret, Stacey Jeffery, Shelby Kimmel, and Alvaro Piedrafita,
*26th Annual European Symposium on Algorithms (ESA 2018)*.
Ed. by Yossi Azar, Hannah Bast, and Grzegorz Herman. Vol. 112. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl-Leibniz Zentrum fuer Informatik, 2018, 49:1-49:13.

[CJO+20] ***Span Programs and Quantum Time Complexity***
Arjan Cornelissen, Stacey Jeffery, Maris Ozols, and Alvaro Piedrafita,
*45th International Symposium on Mathematical Foundations of Computer Science (MFCS 2020)*.
Ed. by Javier Esparza and Daniel Král. Vol. 170. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020, 26:1-26:14.

The author has additionally co-authored the following papers, which are not included in the dissertation.

[PR17] ***Reliable Channel-Adapted Error Correction: Bacon-Shor Code Recovery from Amplitude Damping***
Alvaro Piedrafita and Joseph M. Renes,
*Phys. Rev. Lett.* **119** (2017), p. 250501.
Joseph M. Renes is the principal author of this paper.

[PP20] ***An Overview of Quantum Algorithms: From Quantum Supremacy to Shor Factorization***
Subhasree Patro and Alvaro Piedrafita,
*2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2020, pp. 1-5.
Alvaro Piedrafita is the principal author of this review.

# Contents

# Part I

# The one with the introduction and mathematical preliminaries

# Chapter 1

# Introduction

In March of 2021, László Lovász and Avi Wigderson were awarded the Abel prize for their contributions to computer science and discrete mathematics. When my Dad, a geneticist, read this news, he asked me if I knew either of them. I said "I know of them, and I know people who know them". "Have you used any of their work?", he asked. I laughed. "My thesis is all about span programs. Avi Wigderson invented span programs with his student in the nineties".

It's been almost thirty years and span programs are still being studied, having proven to be a useful tool in the quantum computing toolbox. To give a few examples, they have been used to prove the tightness of the general adversary lower bound [Rei09], design quantum algorithms for graph problems [CMB18; BR12; Āri16], $k-$distinctness [Bel12a], and formula evaluation [RŠ12; JK17] and study quantum space complexity [Jef20]. So span programs are useful, message received, but *what exactly are they?*

## Span programs

Span programs are a way of encoding a function of the form $f : X \subseteq [q]^n \to \{0, 1\}$ for some $q, n \in \mathbb{N}$, as a linear algebraic problem.

We begin by associating a subspace $\mathcal{H}_{i,b}$ in some space $\mathcal{H}$ to each pair of indices $i, b : i \in [n], b \in [q]$. That is, for each coordinate and each possible value of the coordinate we have a subspace.

It follows that each input $x = (x_1, \ldots, x_n) \in [q]^n$ has associated a subspace $\mathcal{H}(x) = \bigoplus_{i\in[n]} \mathcal{H}_{i,x_i}$. To give us some more freedom, we now define

another space $\mathcal{V}$ and a map $A : \mathcal{H} \to \mathcal{V}$ so that every input has associated also a subspace $\mathcal{V}(x) = A(\mathcal{H}(x))$.

Next, we choose a vector $|\tau\rangle$ in $\mathcal{V}$, independent of the input, which we call the *target*. The natural question, now that we have a bunch of subspaces and a lonely vector called the *target* is: Given an input $x$, is the target $|\tau\rangle$ in the subspace $\mathcal{V}(x)$?

The answer can only be yes or no, and, once we have fixed our choice of $\mathcal{V}$, $|\tau\rangle$, the map $A$ and the assignment $i, b \mapsto \mathcal{H}_{i,b}$, it only depends on $x \in [q]^n$. We have just described a function $f_P : [q]^n \to \{0, 1\}$ that takes value 1 if and only if the answer to question is: *yes,* $|\tau\rangle \in \mathcal{V}(x)$. We then say that $f_P$ is *encoded* by the tuple $P = (\mathcal{H}, \mathcal{V}, A, \tau)$, which we name[1] the *span program for* $f_P$. We will sometimes say that $P$ *decides* or *computes* $f_P$, but these are misnomers.

One way of deciding whether $|\tau\rangle \in \mathcal{V}(x)$ is to find a vector $|w\rangle$ such that $A\Pi_{\mathcal{H}(x)}|w\rangle = |\tau\rangle$. We call such a vector a *positive witness* for $x$ in $P$, and it only exists if $f_P(x) = 1$. We define the *positive witness size* of $x$ in $P$, denoted as $w_+(x, P)$, as the norm squared of the shortest positive witness. The *positive complexity* of $f_P$, denoted $W_+(P)$ is defined as the largest positive witness size for any 1-input.

If $f_P(x) = 0$, it must be that $|\tau\rangle \notin \mathcal{V}(x)$, and so $|\tau\rangle$ must have a component in $\mathcal{V}(x)^\perp$. In other words, there exists a vector $|\omega\rangle$ in $\mathcal{V}(x)^\perp$ such that $\langle w|\tau\rangle = 1$. We call such a vector a *negative witness* for $x$ in $P$. For reasons that will be explained in Chapter 3, the *negative witness size* for $x$ in $P$ is defined as the minimum of $\|\langle\omega|A\|^2$ such that $|\omega\rangle$ is a negative witness. The *negative complexity* of $f_P$, denoted $W_-(P)$, is defined as the largest negative witness size for any 0-input.

It is sometimes advantageous to relax our requirements in the definition of $f_P$ and accept as "positive" inputs where $|\tau\rangle$ is simply "close enough" to $\mathcal{V}(x)$. This gives rise to the concept of *approximate span programs*. This generalization, first introduced in [IJ19] and further refined in [Jef20], will play a central role in this thesis.

## Span programs and the Adversary Lower Bound

Span programs were first introduced to the field of quantum computing by Reichardt and Špalek in [RŠ12]. In [Rei09; Rei10], Reichardt used span

[1] Arthurian fanfare goes here. Bells and trumpets, maybe some horses.

programs to prove that the *general adversary lower bound* was a tight lower bound on the quantum query complexity of any Boolean function.

Prior to [Rei09], it was known that this lower bound technique, introduced by Ambainis [Amb02] and generalized in [HLS07], gives lower bounds on the quantum query complexity of a function $f$ as the feasible solutions to a semi-definite program $ADV^{\pm}(f)$. What Reichardt did was prove that any feasible solution of the SDP dual to the $ADV^{\pm}(f)$ corresponds to a span program, and that the solution's objective value equals the geometric mean of the positive and negative span program complexities, a quantity known as the *span program complexity*.

He then showed that for every span program for a Boolean function $f$, there exists a general transformation that compiles it into a quantum algorithm computing $f$ whose query complexity is precisely the span program complexity. This proves that any feasible solution to the dual of $ADV^{\pm}(f)$ gives an upper bound on the query complexity of $f$. Therefore, Reichardt concluded that the optimal solution of the semi-definite program $ADV^{\pm}(f)$ tightly characterizes the quantum query complexity of $f$, since solutions to the primal SDP give lower bounds, and solutions to the dual SDP give upper bounds.

In particular, this means that for any Boolean function $f$ there always exists a span program whose span program complexity equals the query complexity of $f$. Hence, there always exists a query-optimal quantum algorithm for $f$ based on the span program framework.

## Quantum algorithms for span programs

We have already said that span programs do not compute a function but rather encode it in linear-algebraic terms. Evaluating the value of a function given access to the input requires further computation. Classically, one would have to solve the $n \times \dim\mathcal{V}$ system of linear equations $A\Pi_{\mathcal{H}(x)}|w\rangle = |\tau\rangle$ to decide this question. This would require a lengthy computation that includes reading the input since it determines the matrix $A\Pi_{\mathcal{H}(x)}$. The flip-side is that evaluating the function produces a positive witness as a byproduct of the computation if the input happens to be a 1-input.

In contrast, Ref. [RŠ12; Rei09; IJ19; Jef20] built quantum algorithms that evaluate a function encoded within a span program $P = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$. These quantum algorithm do not decide the function by solving a linear system of equations or finding a witness. Instead, they uses the elements of

the span program to construct a unitary $U(x,P) = (2\Pi_{\mathcal{H}(x)} - I)(2\Pi_{\ker A} - I)$ and a vector $|w_0\rangle = A^+|\tau\rangle$, or variants thereof. Then, if $x$ is a 0-input, $|w_0\rangle$ will have some overlap with the 0-phase eigenspace of $U(x,P)$, and if $x$ is a 1-input, it will have no overlap with that space and only small overlap with the small-phase eigenspaces of $U(x,P)$. The algorithms distinguish these two cases by performing phase estimation and/or amplitude amplification on $U(x,P)$ to some precision determined by $W_+$ and $W_-$. Regardless of which algorithm one uses, the number of calls to $U(x,P)$ will be the span program complexity $\sqrt{W_+W_-}$. This is precisely its query complexity since the only input-dependent element is the reflection $(2\Pi_{\mathcal{H}(x)} - I)$, which can be implemented with one quantum query to $x$. This is *fundamentally different* from solving a linear system of equations, and allows us to obtain quantum algorithms with better query complexity than any classical algorithm.

For the time complexity, one must analyze the cost of constructing the reflections $(2\Pi_{\ker A} - I)$ and the state $|w_0\rangle$. Naturally this varies from one span program to another, and has been largely ignored in the literature. A notable exception is the span program for $st$-connectivity, whose time complexity was studied in [BR12; JK17].

### The $st$-connectivity span program

Chapters 5 and 6 deal extensively with a span program for $st$-connectivity presented in [BR12]. In a nutshell, the $st$-connectivity problem is this: Fix a graph $G$, two vertices $s$ and $t$ connected in $G$, and let $x$ specify a subgraph $G(x)$. Are $s$ and $t$ still connected in $G(x)$?

The span program that computes this function is rather simple. As the space $\mathcal{H}$, we take the space generated by the edges of $G$, i.e. $\mathcal{H} = \text{span}\{|u,v\rangle : (u,v) \in E(G)\}$. Assume, for simplicity, that $x \in \{0,1\}^{|E(G)|}$ and that every bit of the input corresponds to one edge, which is in $G(x)$ if its bit value is 1. Then $\mathcal{H}(x) = \text{span}\{|u,v\rangle : (u,v) \in E(G(x))\}$. The space $\mathcal{V}$ is the space generated by the vertices, i.e. $\mathcal{V} = \text{span}\{|v\rangle : v \in V(G)\}$, and the span program map is defined as $A|u,v\rangle = |u\rangle - |v\rangle$. Finally, we define the target as $|s\rangle - |t\rangle$. It is not hard to see that any $st$-path in $G(x)$ is a positive witness for $x$. It is a little bit harder to see that an $st$-cut over the edges in $G \setminus G(x)$ is a negative witness. Of course, the *optimal* witnesses are convex combinations of paths or cuts. In [BR12; JK17] the authors characterized the positive witness size of this span program as an electrical quantity on $G(x)$ known as the *effective resistance*. Prior to our work, only upper bounds for

the negative witness size were known in the general case, and a complete characterization for planar graphs was given in [JK17].

This span program has been used as a primitive to construct quantum algorithms for a variety of problems, including cycle detection and bipartiteness testing [Āri16; CMB18], formula evaluation [RŠ12; Rei09; JK17], learning graph evaluation [Bel12b], $k$-distinctness [Bel12a], and maximum bipartite matching [BT20]. In [Āri16], the author gave a span-program-based algorithm for graph connectivity with optimal worst-case query complexity $\mathcal{O}(n^{3/2})$ using $\mathcal{O}(\log n)$ space. This problem, first studied in the context of quantum algorithms in [DHH+06], can be solved by a quantum algorithm that also finds $st$-paths in time $\Theta(n^{3/2})$ — instead of the classical $\Omega(n^2)$ — and space $\Theta(n)$.

## Contributions

The topics we have discussed so far exemplify a few of the features of span programs that make them worthy of study. Namely,

1. Span programs, although related to quantum upper and lower bounds, are classical objects.

2. Through the general construction that compiles a span program into a quantum algorithm, span programs are a way to make new quantum algorithms using classical ideas. As we show in Chapter 4, these algorithms can even be query, time, and space optimal.

3. They can also be used as a theoretical tool, for example in the study of space lower bounds [Jef20].

4. As classical objects, span programs are amenable to composition and classical logic.

There is a general feeling in the community that span programs, despite the features I just mentioned, are a somewhat rigid tool for making quantum algorithms, limited to decision problems,[2] mostly on graphs, and can't say much about time complexity. In this dissertation we challenge both of these statements. I aim to show that span programs are a versatile tool to construct

[2] With the single exception of the span programs for witness size estimation in [IJ19].

quantum algorithms, not only for decision problems but also for function estimation and state generation, and that we can make meaningful time complexity statements about span program algorithms. If you were to ask me "what is this thesis about?", that would be my answer.

This thesis collects the work I have done with my co-authors in the last four years on the topic of span programs, split into five chapters.

**Chapter 2. Preliminaries** In this dissertation I will assume the reader to be familiar with the standard concepts in linear algebra and quantum computing. In Chapter 2 I make concrete what these assumptions are and discuss other mathematical results and definitions that will be used throughout the thesis, such as Jordan's Lemma, quantum query algorithms, graph theoretical results and electrical networks.

**Chapter 3. Theory of span programs** This chapter is split into two parts. The first part is a survey of the previous literature on span programs. I first talk about span programs as presented in [Rei09]. Then I explain how Ref. [IJ19] generalizes Reichardt's construction and introduces *approximate span programs*, and talk about the algorithms given by the authors of those papers.

In Section 3.3, I introduce a generalization of span programs called *reflection programs* and *approximate reflection programs*. This is joint work with Arjan Cornelissen and Maris Ozols, and is part of a paper in preparation. The purpose of defining reflection programs is to improve our intuition of the algorithmic components of a span program and streamline the analysis of span program algorithms. Then, I construct two algorithms for reflection programs that will be of use throughout the dissertation. Although new, these algorithms don't differ much from other algorithms already present in the literature for approximate span programs [IJ19; Jef20] — I could have simply updated those algorithms to the language of reflection programs — but have opted for including them for two reasons. First, they dispense with a procedure called *span program scaling* necessary for the analysis in [IJ19; Jef20]. This is yet another layer of complexity on the subject which these algorithms remove. Second, they are the basis of the two novel algorithms for generating positive witnesses in Section 3.4. Witnesses are qualitatively similar to certificates in that they encode much information about the function they decide. For example, witnesses for *st*-connectivity are weighted super-

positions of $st$-paths. Somewhat like the outputs of quantum linear solvers, they encode the optimal solution of a system $A(x)|w\rangle = |\tau\rangle$ in a quantum state. This is the first time anyone has used span programs to construct algorithms for state generation. This, together with the proof-of-concept algorithm in Section 5.7, is joint work with Stacey Jeffery and Shelby Kimmel and is part of a paper in preparation.

**Chapter 4. Span programs and time complexity** This chapter is based on [CJO+20] and is joint work with Arjan Cornelissen, Stacey Jeffery, and Maris Ozols. We study a map taking any quantum query algorithm $\mathcal{A}$ to a span program $P_{\mathcal{A}}$, first introduced in [Rei09]. In its original formulation, the map $\mathcal{A} \mapsto P_{\mathcal{A}}$ only applied to one-sided error algorithms[3], and the corresponding query complexity of the span program algorithm for $P_{\mathcal{A}}$ was that of $\mathcal{A}$ itself. Ref. [Jef14] showed that the map also extended to bounded (two-sided) error algorithms, mapping them to approximate span programs that could then be turned back into algorithms with the same query complexity as $\mathcal{A}$. In this chapter, we modify the map to allow the span program algorithms to have the same query and time complexities as $\mathcal{A}$. This has theoretical implications, i.e. every function that admits an optimal query, time, and space optimal algorithm admits, too, a query, time and space optimal *span program algorithm*. But it also has practical ones. Combining our results and a novel construction for the OR of span programs, we improve a result of Ambainis [Amb10] on variable-time quantum search. Our construction allows us to give concrete statements on the time, space, as well as query complexities of the variable time search algorithm. The chapter also contains results that are of independent interest such as the analysis of the time complexity of the algorithms in Chapter 3 and the definition of implementing subspaces.

**Chapter 5. Span programs for graph problems** In Chapter 5 we give a battery of results showing the versatility of the span program for $st$-connectivity first introduced in [KW93]. The chapter is largely based on [JJK+18] and is joint work with Michael Jarret, Stacey Jeffery and Shelby Kimmel. We characterize the negative witness size of the span program as an electrical quantity on $G$ known as *effective capacitance*, and give an algorithm that estimates it. We also give three different span program algorithms for

[3] One-sided error algorithms decide 1-inputs with certainty and may err on 0-inputs with probability 1/3 (or vice versa).

graph connectivity. All three algorithms we propose are based on the *st*-connectivity span program, each modifying it in a different way. For all three, we analyze the query and time complexity, and find them pairwise incomparable, outperforming the current algorithms [DHH+06; Āri16] and each other for some assumptions but not others. We also give a first-of-its-kind quantum algorithm for subgraphs of a complete graph that estimates a measure of connectedness called the *algebraic connectivity*.

Finally, In Section 5.7, I use the algorithm that generates the *optimal positive witness* for the *st*-connectivity span program to give a proof-of-concept of how one can use this vector to find *st*-paths in a particular graph. The resulting algorithm outperforms the path finding algorithm in [DHH+06] in query as well as space complexity.

**Chapter 6. Boundary problems** This chapter is joint work with Stacey Jeffery and Maris Ozols, and is part of a paper in preparation. In the last chapter of this thesis, I take a broader look at the *st*-connectivity problem and frame it as a particular case of a boundary problem on a graph. Imagining other boundaries I discuss modifications of the *st*-connectivity algorithm that would solve those other boundary problems and then discuss a generalization of the *st*-connectivity problem to higher dimensions. Surprisingly, this takes us to the realm of topology, simplicial complices and homology, which I define and explain before presenting a span program for simplicial homology in general dimensions that generalizes the *st*-connectivity span program. I conclude with a discussion of a span program for surface homology, with an analysis on the positive and negative witness sizes that allows us to formulate a span-program-based quantum algorithm that decides whether a cycle on a surface is the boundary of a sub-surface.

To the best of my knowledge, span programs have not been used to study topological structures beyond graphs before, and there are only a few quantum algorithms that deal with topological problems. In giving a connection between span programs and simplicial homology, I show that span programs are a useful tool to approach such problems with the goal to construct quantum algorithms in mind, and I hope to spark future research in this direction.

# Chapter 2

# Preliminaries

## 2.1 Mathematical preliminaries

In this thesis, we assume the reader to be familiar with the basics of quantum computing and linear algebra[1].

**Linear algebra** We will use Dirac notation for vectors. We denote the set of linear operators from a vector space $V$ to a space $W$ as $\mathcal{L}(V, W)$. For an operator $A \in \mathcal{L}(V, W)$, we denote the columnspace, rowspace, and kernel of A by col$A$, row$A$, and $\ker A$ respectively.

For any linear operator $A$, let $A = \sum_{i=1}^{r} \sigma_i |\phi_i\rangle\langle\psi_i|$ be a singular value decomposition. We define the *spectral gap* of $A$ as the smallest non-zero singular value of $A$. We define the *pseudo-inverse* of $A$ as the operator $A^+ := \sum_{i=1}^{r} \frac{1}{\sigma_i} |\psi_i\rangle\langle\phi_i|$.

Let $H$ be a finite-dimensional inner product vector space. An operator $U \in \mathcal{L}(H, H)$ is called unitary if it admits an eigenvalue value decomposition of the form

$$U = \sum_{j=1}^{r} e^{i\varphi_j} |\varphi_j\rangle\langle\varphi_j|,$$

where $\varphi_j \in [-\pi, \pi]$ are known as the *eigenphases* of $U$. We define the *phase gap* of $U$ as $\Delta(U) := \min\{|\varphi_j| : \varphi_j \text{ eigenphase of } U, \varphi_j \neq 0\}$.

[1] Sorry Mom and Dad.

We say a unitary $\widetilde{U}$ approximates $U$ with error $\varepsilon$ if $\left\|\widetilde{U} - U\right\|_\infty \leq \varepsilon$. That is, $\max_{|\psi\rangle} \left\|(\widetilde{U} - U)|\psi\rangle\right\| \leq \varepsilon \||\psi\rangle\|$.

For a finite set $X$, we denote the space $\mathbb{C}^{|X|}$ with orthonormal basis $\{|x\rangle : x \in X\}$ by $\mathbb{C}^X$, which we call the *space generated by* $X$. For any subspace $H' \subseteq H$, we write $\Pi_{H'} \in \mathcal{L}(H)$ to denote the projector onto $H'$.

**Quantum circuits** All the algorithms in this thesis are in the circuit model, and we assume the reader to be familiar with the concepts of quantum states, qubits, unitaries, projective measurements, and with the most common one and two-qubit gates, namely, the bit-flip, phase-flip and bit-phase-flip gates $X, Z, Y$, as well as the Hadamard gate $H$ and the CNOT gate. When we talk about gate complexity, we will implicitly assume it to be relative to a particular universal gate set. For simplicity, we assume the gates $X, Y, Z, H,$ CNOT to be part of the universal gate set. Any unitary in the universal gate set is said to be *elementary*. See [NC02] for an in-depth discussion of these concepts.

**Miscellaneous notation** For any $n \in \mathbb{N}$, we denote the set $\{1, 2, \ldots, n\}$ by $[n]$, and the set $\{0, 1, 2, \ldots, n\}$ by $[n]_0$. We assume the reader to be familiar with the asymptotic notations $\mathcal{O}(\cdot)$, $o(\cdot)$, $\Theta(\cdot)$ and $\widetilde{\mathcal{O}}(\cdot)$. By $f(x, y) = \mathcal{O}(\text{polylog}(x, y))$, we mean that there exist constants $C_1, C_2 > 0$ such that $f(x, y) = \mathcal{O}\left(\log^{C_1}(x) \log^{C_2}(y)\right)$, in the limit $x, y \to \infty$.

### 2.1.1 Jordan's Lemma

In this section, we study the geometry of the eigenspaces of a particular set of unitaries of great importance for quantum computing and for us. Imagine that one has two linear spaces $A$ and $B$ which are subspaces of a larger, finite-dimensional Hilbert space $\mathcal{H}$. Consider the unitary $U = (2\Pi_A - I)(2\Pi_B - I)$ which is a product of two reflections around $A$ and $B$ respectively. Jordan's Lemma states that $U$ induces a partition of $\mathcal{H}$ into a direct sum of 2-dimensional spaces in which $U$ acts as a rotation operator, and a collection of 1-dimensional spaces that are either $(+1)$- or $(-1)$-eigenspaces of $U$. Jordan's Lemma has been at the core of quantum algorithms since Grover's unstructured search algorithm [Gro96] and is the basis for the analysis of span program algorithms, quantum random walks (see [San08] for a compre-

hensive survey), and many other things, some of which we wot not of. The lemma has been discovered and rediscovered several times, and is sometimes referred to in the mathematics community as the CS decomposition, but in the present formulation it was first rediscovered in the context of quantum algorithms by Szegedy in [Sze04] and applied to the study of quantum random walks. We state it now without proof, although the interested reader is encouraged to read the original proof since it is rather short and elegant.

**Lemma 1** ([Sze04])**.** *Let $A, B$ be two subspaces of $\mathcal{H}$ and let $U = (2\Pi_A - I)(2\Pi_B - I)$ be a unitary with discriminant $D = \Pi_A\Pi_B$.*

*Let $D = \sum_{j=1}^{d} \cos\varphi_j |\theta_j\rangle\langle\psi_j|$ be a singular value decomposition with $\varphi_j \in [0, \pi/2]$ for all $j \in [d]$. Then the vectors $|\theta_j\rangle - e^{\pm i\varphi_j}|\psi_j\rangle$ are eigenvectors of $U$ with eigenvalue $e^{\mp i2\varphi_j}$ respectively. Furthermore, the $(+1)$-eigenspace of $U$ is $(A \cap B) \oplus (A^\perp \cap B^\perp)$ and the $(-1)$-eigenspace of $U$ is $(A \cap B^\perp) \oplus (A^\perp \cap B)$.*

A quick corollary of this lemma which will be of great use later relates the phase gap of $-U$ with the smallest spectral value of $D$.

**Corollary 2.** *Let $U = (2\Pi_A - I)(2\Pi_B - I)$ and $D = \Pi_A\Pi_B$ be its discriminant. Then $\Delta(-U) = 2\sin^{-1}(\sigma_{\min}(D))$.*

*Proof.* Assume without loss of generality that the singular values of $D$ are ordered such that $\theta_i \geq \theta_j$ for $i < j$. Since cos is a decreasing function in the interval $[0, \frac{\pi}{2}]$, it follows that $\sigma_{\min}(D) = \cos\theta_1$. By Lemma 1, the spectrum of $U$ outside of its $(\pm 1)$-eigenspace is $\{e^{\pm 2i\theta_j}\}_{j=1}^{d}$, hence the biggest eigenphase smaller than $\pi$ in absolute value must be $2\theta_1$. In other words, $\pi - 2\theta_1$ is the phase gap of $-U$.

Let $\alpha$ be the complementary angle to $\theta_1$, i.e. $\pi/2 = \alpha + \theta_1$. Then $2\alpha = \pi - 2\theta_1 = \Delta(-U)$, but $\cos(\theta_1) = \sin(\alpha)$, which means $\sigma_{\min}(D) = \sin(\alpha) = \sin(\Delta(-U)/2)$. The first result follows from applying the arcsine function on both sides of the last equality. □

That Jordan's Lemma is at the core of so many quantum algorithms speaks to the power of the family of unitaries composed of the product of two reflections. Indeed, we shall see later in Chapter 3 that all Boolean functions admit an optimal quantum algorithm that consists of phase and amplitude amplification of a unitary of this form. But the utility of Lemma 1 is not limited to describing the action of $U$, it is also useful to better understand the interrelation of $A$, $B$ and the eigenspaces of $U$. The following results will be of use to us and are all consequences of Jordan's Lemma.

**Lemma 3.** *Let $A, B \subset \mathcal{H}$ be two subspaces, let $U = (2\Pi_A - I)(2\Pi_B - I)$, let $\varphi_j \in (0, \pi)$, $j \in [d]$, be positive eigenphases of $U$, let $E_\varphi := E_{\varphi^+} \oplus E_{\varphi^-}$ be the 2-dimensional spaces spanned by $(\pm\varphi)$-eigenvalue eigenvectors of $U$, and let $E_0$, $E_\pi$ be the $(+1)$ and $(-1)$-eigenspaces of $U$ respectively. Then,*

$$B = (B \cap E_0) \bigoplus_{j \in [d]} \left(B \cap E_{\varphi_j}\right) \oplus (B \cap E_\pi).$$

*Proof.* We will prove that the subspaces $E_0 \cap B$, $E_\pi$ and $E_{\varphi_j}$ are invariant subspaces for $\Pi_B$. Since $\mathcal{H} = E_0 \oplus E_\pi \bigoplus_j E_{\varphi_j}$ and every linear operator decomposes its image into invariant subspaces, we will have the result.

Let us begin with $E_0$. Proving that $E_0 \cap B$ is an invariant subspace of $\Pi_B$ is equivalent to proving that $\Pi_{E_0}\Pi_B = \Pi_B\Pi_{E_0}$. Now, remember that $\Pi_{E_0}$ is

$$\Pi_{E_0} = \Pi_{A \cap B} + \Pi_{A^\perp \cap B^\perp},$$

so $\Pi_B$ trivially commutes with both terms because one of them projects into a proper subspace of $B$ and the other one projects into a subspace orthogonal to it. The proof for $E_\pi$ is identical and is left for the reader.

All there is left is to prove that $\Pi_{E_{\varphi_j}}\Pi_B = \Pi_B\Pi_{E_{\varphi_j}}$ for all positive eigenphases of $U$ different from $\pi$. For the remainder, fix any such eigenphase. We will give a basis of $E_{\varphi_j}$ that is composed of a vector in $B$ and a vector in $B^\perp$.

First, remember that by Jordan's Lemma,

$$\Pi_A\Pi_B = \sum_i^d \cos\frac{\varphi_i}{2} |\theta_i\rangle\langle\psi_i|$$
$$|\eta_j\rangle = |\theta_j\rangle - e^{i\frac{\varphi_j}{2}}|\psi_j\rangle$$
$$|\nu_j\rangle = |\theta_j\rangle - e^{-i\frac{\varphi_j}{2}}|\psi_j\rangle$$

where $|\eta_j\rangle$ and $|\nu_j\rangle$ are eigenvectors of $U$ with eigenvalues $e^{i\varphi_j}$ and $e^{-i\varphi_j}$ respectively, and so they form a basis for $E_{\varphi_j}$. Consider the vector

$$\frac{|\nu_j\rangle - |\eta_j\rangle}{2} = i \sin\frac{\varphi_j}{2}|\psi_j\rangle.$$

By Jordan's Lemma, $|\psi_j\rangle \in B$ so we simply need to find a vector $|\psi_j^\perp\rangle$ in $E_{\varphi_j}$ that is orthogonal to $|\psi_j\rangle$ and to $B$. We define the vector

$$|\psi_j^\perp\rangle := |\theta_j\rangle - \cos\frac{\varphi_j}{2}|\psi_j\rangle. \tag{2.1}$$

Observe that from $\Pi_A\Pi_B = \sum_{j=1}^d \cos\left(\frac{\varphi_j}{2}\right)|\theta_j\rangle\langle\psi_j|$ follows that $\langle\theta_j|\psi_j\rangle = \cos\frac{\varphi_j}{2}$, so we have:

$$\langle\psi_j|\psi_j^\perp\rangle = \langle\psi_j|\theta_j\rangle - \cos\frac{\varphi_j}{2}\langle\psi_j|\psi_j\rangle = 0.$$

To conclude the proof, we see that this vector is orthogonal to $B$.

$$\Pi_B|\psi_j^\perp\rangle = \Pi_B|\theta_j\rangle - \cos\frac{\varphi_j}{2}\Pi_B|\phi_j\rangle = \cos\frac{\varphi_j}{2}|\psi_j\rangle - \cos\frac{\varphi_j}{2}|\psi_j\rangle = 0,$$

where we have used that $\Pi_B|\theta_j\rangle = \cos\frac{\varphi_j}{2}|\psi_j\rangle$. □

Observe that switching the roles of $A$ and $B$, and $|\theta_j\rangle$ and $|\psi_j\rangle$ in the proof we obtain the following corollary:

**Corollary 4.** *Let $\varphi_j, \in (0,\pi)$, $j \in [d]$ be positive eigenphases of $U = (2\Pi_A - I)(2\Pi_B - I)$, $E_{\varphi_j} := E_{\varphi_j^+} \oplus E_{-\varphi_j^-}$, and let $E_0$, $E_\pi$ be defined as above. Then,*

$$A = (A \cap E_0) \bigoplus_{j\in[d]} \left(A \cap E_{\varphi_j}\right) \oplus (A \cap E_\pi),$$

*and*

$$A^\perp = \left(A^\perp \cap E_0\right) \bigoplus_{j\in[d]} \left(A^\perp \cap E_{\varphi_j}\right) \oplus \left(A^\perp \cap E_\pi\right).$$

**Lemma 5** (Effective spectral gap lemma [LMR+11])**.** *Let $|\phi\rangle$ be a unit vector such that $\Pi_B|\phi\rangle = 0$, let $P_\theta$ be the projector onto the eigenvectors of $U = (2\Pi_A - I)(2\Pi_B - I)$ with eigenvalues $e^{i\omega}$ with $|\omega| < \theta$ for some $\theta \in [0,\pi)$, then if $\Pi_B|\phi\rangle = 0$, we have $\|P_\theta\Pi_A|\phi\rangle\| \leq \theta/2$.*

## 2.2 Quantum algorithms

The Encyclopaedia Britannica defines algorithm as follows:

"Algorithm: *Systematic procedure that produces — in a finite number of steps — the answer to a question or the solution to a problem.*"

The term itself comes from the 8th century Muslim mathematician Al-Khwārizmī[2], but the idea is as old as hills. Nowadays, we understand that these questions or problems are mathematical in nature, as are the steps taken. Modern computers are machines that implement *classical* algorithms — procedures composed entirely of arithmetic operations — and are based on classical physics.

In contrast, quantum computers implement quantum algorithms — procedures composed of quantum operations, typically unitaries and measurements — and are based on quantum physics. Quantum algorithms are of interest both theoretically and practically. As we all know by now, quantum computers can solve certain computational problems much faster than classical computers. But they also inform us about the absolute limits of computation, since they rely on quantum mechanics, our best understanding of nature. In a very real sense, the study of quantum algorithms is "*the study of the power and limitations of the strongest-possible computational devices that Nature allows us*" [deW19].

In order to characterize that power and/or limitations, we need a metric, a criterion for measuring the performance of an algorithm. And once you have a metric, you have a notion of *complexity*. Classical computers often use total number of operations, but processor time, input queries, and other metrics are common. In quantum mechanics, a usual metric is the number of queries to the input of the algorithm. There are many reasons for this. First, this metric is often times the simplest and most tractable. Second, it gives us a lower bound on the total number of operations that an algorithm makes. That is, it tells us about the limitations of the algorithm. Third, there exist techniques to give lower bounds on the query complexity of computing Boolean functions which are intimately related to span programs.

Query complexity is not the only measure of complexity that is used in the quantum computing community, but, for reasons that will become clear in Section 3.2.3, it is the primary measure of complexity when dealing with span programs. In Chapter 5, and especially, Chapter 4, we will also care about the *time complexity* of algorithms. We understand by the time complexity of an algorithm the total number of time steps it takes, where each time step the algorithm makes either a call to the input oracle $\mathcal{O}_x$ from Eq. (2.2), or an elementary gate from a universal gate set. We will often refer to this as

[2]Whose works also introduced Hindu-Arabic decimal numerals into European mathematics. Lest we forget that we used to be bad at math.

the *gate* complexity or *cost* of an algorithm, in a slight abuse of language.

The lion's share of this work will be devoted to algorithms that *decide* Boolean functions, as is natural in a work so much centred around span programs. In Chapter 3, we give algorithms that approximate a quantum state, which we revisit at the end of Chapter 5, where we also give a query algorithm that *estimates* a real function. Let us begin by formalizing the notion of quantum query algorithms for decision and estimation problems. We will discuss state generation algorithms later when they appear.

### 2.2.1 Quantum query algorithms

Let $n \in \mathbb{N}$, $X \subseteq \{0,1\}^n$. A quantum query algorithm $\mathcal{A}$ with input $x \in X$ is a sequence of unitaries $U_1, \ldots, U_T$ acting on $\mathbb{C}^{[n]\times\mathcal{W}}$, where $T \in \mathbb{N}$ is the total number of time steps, $[n] := \{1, \ldots, n\}$ and $\mathcal{W}$ is a finite set that labels the workspace states, together with an initial state $|\Psi_0\rangle \in \mathbb{C}^{[n]\times\mathcal{W}}$.

The algorithm $\mathcal{A}$ makes queries to an input string $x \in X$ by having a subset of the unitaries be (controlled) calls to an oracle $\mathcal{O}_x$ defined by its action on the computational basis of $\mathbb{C}^{[n]\times\mathcal{W}}$ as:

$$\forall i \in [n], \forall j \in \mathcal{W}, \qquad \mathcal{O}_x : |i,j\rangle \mapsto (-1)^{x_i}|i,j\rangle, \tag{2.2}$$

where the two registers correspond to the input bit index and the workspace, respectively. The only dependence on $x$ of the unitaries that make up $\mathcal{A}$ is through some of the $U_t$'s being $\mathcal{O}_x$. We denote the set $\mathcal{S} \subset [T]$ to be the set that contains all $t \in \mathcal{S}$, such that $U_t = \mathcal{O}_x$. Then $S = |\mathcal{S}|$ is the *query complexity* of $\mathcal{A}$.

In the standard definition of a quantum query algorithm, every second unitary is a query, so $\mathcal{S}$ would be the set of odd indices. This is appropriate when we are only interested in the query complexity of the algorithm, since we can combine any consecutive non-query unitaries into a single unitary. However, since we are also interested in the time complexity, we want to restrict the non-query unitaries to some universal gate set. Thus, we do not assume that every other unitary is a query, and we explicitly allow for sequences of non-query unitaries between any two queries, as well as at the beginning and the end of the algorithm.

We take the initial state to be a computational basis state. We can assume that $U_1$ and $U_T$ are not queries without loss of generality. Indeed, if the first unitary is a query, then it only introduces a global phase and hence it is

redundant. Similarly, we assume that any measurement at the end of the algorithm is a computational basis measurement, which implies that if $U_T$ is a query, then it is also redundant as it does not influence the measurement probabilities. Finally, we also assume without loss of generality that no two consecutive time steps are query time steps, as then the resulting operation on the state space would reduce to $\mathcal{O}_x^2 = I$, rendering both queries redundant.

For every $x \in \{0,1\}^n$ we define the state of the system at time $t \in [T]_0 := \{0, \ldots, T\}$ on input $x$ as

$$|\Psi_t(x)\rangle := U_t U_{t-1} \cdots U_1 |\Psi_0\rangle, \tag{2.3}$$

where $|\Psi_0\rangle \in \mathbb{C}^{[n]\times\mathcal{W}}$ is the initial state. Note that the right-hand side of Eq. (2.3) has an implicit dependence on $x$, since for some indices $t$, $U_t = \mathcal{O}_x$.

After its final step, the state of the system is $|\Psi_T(x)\rangle$. Depending on the kind of problem that the algorithm solves, we do different things with this final state.

### Algorithms for decision problems

We say that an algorithm $\mathcal{A}$ *decides* a (partial) Boolean function $f : X \subseteq \{0,1\}^n \to \{0,1\}$, with error probability $\varepsilon \in [0, 1/2)$ if, for all $x \in X$, it outputs $f(x)$ with probability $p_{f(x)}(x) \geq 1 - \varepsilon$.

If $\mathcal{A}$ is a quantum algorithm for a decision problem, we can assume that there is a single-qubit answer register used to indicate the output of the computation, and the algorithm ends with a measurement of that register (see Section 2.2.2). If $\Pi_b$ denotes the orthogonal projector onto states with $|b\rangle$ in the answer register for $b \in \{0,1\}$, then $p_b(x) := \|\Pi_b |\Psi_T(x)\rangle\|^2$ is the probability that the algorithm outputs $b$ on input $x$. We say that a quantum algorithm $\mathcal{A}$ *decides* $f$ *with bounded error* if it decides $f$ with error probability $\varepsilon = 1/3$ for all $x \in X$.

The query complexity of a function $f$, denoted as $Q(f)$, is defined as the smallest query complexity among query algorithms that decide $f$ with bounded error.

### Algorithms for estimation problems

As we have already said at the beginning of this section, part of Chapter 5 will be devoted to algorithms that estimate functions mapping a domain

$X \subseteq \{0,1\}^n$ into the real numbers. Typical examples include algorithms for estimating graph properties like effective resistance or algebraic connectivity (more of which later) as well as the *phase* and *amplitude estimation* subroutines, assuming an implicit relation between the input oracle and the inputs to both subroutines. We define quantum query algorithms for estimation problems in the following way.

Let $n \in \mathbb{N}$, $X \subseteq \{0,1\}^n$ and $f : X \to \mathbb{R}^+$ be a real function. In the same spirit as algorithms that decide Boolean functions, let $\mathcal{A}$ be a quantum algorithm acting on $\mathbb{C}^{[n]\times\mathcal{W}}$ that applies the sequence of unitaries $U_1, \ldots, U_T$, of which some are queries to the oracle $\mathcal{O}_x$, to the state $|\Psi_0\rangle \in \mathbb{C}^{[n]\times\mathcal{W}}$. We can make the same assumptions as before and assume that the first and last unitaries are not queries, and that the initial state is a computational basis state. We can also assume that the last $m$ qubits of $\mathbb{C}^{[n]\times\mathcal{W}}$ contain the output of the computation written as a binary number $\tilde{f}$. Strictly speaking, the output of the computation will be an entangled state of the form

$$|\Psi_T(x)\rangle = \sum_{\tilde{f}\in\mathcal{D}} |\Psi_{\tilde{f}}(x)\rangle|\tilde{f}\rangle,$$

where $\mathcal{D}$ is some domain and $|\Psi_{\tilde{f}}(x)\rangle$ are unnormalized states. We say that a quantum algorithm $\mathcal{A}$ *estimates $f$ to relative accuracy $\varepsilon$ with bounded error* if, for any $x \in X$ we have

$$\left\| \left( I \otimes \sum_{\tilde{f}:|\tilde{f}-f(x)|\leq\varepsilon f(x)} |\tilde{f}\rangle\langle\tilde{f}| \right) |\Psi_T(x)\rangle \right\|^2 \geq \frac{2}{3}. \tag{2.4}$$

In other words, $\mathcal{A}$ estimates $f$ if the result $\tilde{f}$ of measuring the outcome register is such that $|\tilde{f} - f(x)| \leq \varepsilon f(x)$ with probability at least $2/3$.

### 2.2.2 Clean quantum algorithms

In addition to the standard assumptions that we outlined above, we will also make some non-standard assumptions on the structure of quantum query algorithms for decision problems. We refer to the query algorithms that satisfy both the standard and the non-standard assumptions as *clean algorithms*. We first define this object and then show that, in fact, we can assume without loss of generality that every quantum query algorithm is clean incurring at most a constant overhead in both queries and time.

**Definition 6** (Clean quantum algorithm)**.** Let $\mathcal{A}$ be a quantum query algorithm acting on $\mathbb{C}^{[n]\times\mathcal{W}} = \mathbb{C}^{[n]\times\mathcal{W}'\times\{0,1\}}$ with the last register being the answer register. Suppose that the time complexity of $\mathcal{A}$ is $T$, the query complexity is $S$, and the initial state has $|0\rangle$ in the answer register, so it can be expressed as $|\Psi_0\rangle = |\psi_0\rangle|0\rangle$ for some $|\psi_0\rangle \in \mathbb{C}^{[n]\times\mathcal{W}'}$. Define the *final accepting state* as $|\Psi_T\rangle := |\psi_0\rangle|1\rangle$. $\mathcal{A}$ is a *clean quantum algorithm* if it satisfies the following properties.

1. *Consistency:* For all inputs $x \in \{0,1\}^n$,

$$\langle\Psi_T|\Psi_T(x)\rangle = p_1(x), \quad \text{and} \quad \langle\Psi_T|(I\otimes X)|\Psi_T(x)\rangle = p_0(x),$$

   where $p_b(x) = \|(I\otimes|b\rangle\langle b|)|\Psi_T(x)\rangle\|^2$ is the probability that $\mathcal{A}$ outputs $b$ on input $x$, and $X$ denotes the Pauli matrix implementing the logical NOT.
2. *Commutation:* $(I\otimes X)$ commutes with every unitary $U_t$ of the algorithm, where $X$ acts on the answer register.
3. *Query-uniformity:* Two consecutive queries are not more than $\lfloor 3T/S\rfloor$ time steps apart, and the first and last queries are separated by at most $\lfloor 3T/S\rfloor$ time steps from the start and the finish of the algorithm, respectively.

We proceed by showing that restricting our attention to clean algorithms only incurs a constant multiplicative overhead in the query and time complexities and constant additive overhead in the space complexity.

We prove this in two steps. First, we show that we can satisfy conditions 1 and 2 by modifying the algorithm in the following sense: we first run it once, then we copy out the answer register, and subsequently, we run it backwards. This constitutes Lemma 7. After that, we insert some queries and identity gates into the resulting algorithm, such that we also satisfy condition 3, which is the objective of Lemma 8.

**Lemma 7.** *Fix $f : X \subseteq \{0,1\}^n \to \{0,1\}$. Let $\mathcal{A}$ be a quantum query algorithm with initial state $|\Psi_0\rangle \in \mathbb{C}^{[n]\times\mathcal{W}}$, unitaries $U_1,\ldots,U_T$, time complexity $T$ and query complexity $S$ and suppose that it computes $f$ with error probability $\varepsilon > 0$. Now, let $\mathcal{A}'$ be a quantum algorithm acting on $\mathbb{C}^{[n]\times\mathcal{W}\times\{0,1\}}$, with initial state $|\Psi_0'\rangle = |\Psi_0\rangle|0\rangle$ and consisting of the following sequence of unitaries:*

$$(U_1^\dagger\otimes I)\cdots(U_T^\dagger\otimes I)(I\otimes \mathrm{CNOT})(U_T\otimes I)\cdots(U_1\otimes I),$$

*where the* CNOT *is a controlled-not gate with the answer qubit of* $\mathcal{A}$ *acting as control qubit and the last qubit of* $\mathcal{A}'$ *acting as the target. Then* $\mathcal{A}'$ *fulfills conditions 1 and 2 in Definition 6 with final accepting state* $|\Psi'_{T'}\rangle = |\Psi_0\rangle|1\rangle$*, time complexity* $T' = 2T + 1 = \Theta(T)$*, query complexity* $S' = 2S = \Theta(S)$*, uses one more qubit than* $\mathcal{A}$ *and evaluates* $f$ *with error probability* $\varepsilon$*.*

*Proof.* Since an $X$-gate on the target qubit of a CNOT gate commutes with the CNOT-gate itself, we find that all operations in $\mathcal{A}'$ commute with $I \otimes X$, thus the commutation condition is fulfilled.

Next we check the consistency condition. To that end, we let $|\Psi_T(x)\rangle = |\Phi_0(x)\rangle + |\Phi_1(x)\rangle$, where $|\Phi_b(x)\rangle = \Pi_b|\Psi_T(x)\rangle$ is the projection of $|\Psi_T(x)\rangle$ onto the part of the state with $|b\rangle$ in the answer register of $\mathcal{A}$. Then the state of $\mathcal{A}'$ after $T$ steps on input $x$ is

$$|\Psi'_T(x)\rangle = |\Psi_T(x)\rangle|0\rangle = |\Phi_0(x)\rangle|0\rangle + |\Phi_1(x)\rangle|0\rangle,$$

and the state of $\mathcal{A}'$ after $T+1$ steps on input $x$ is

$$|\Psi'_{T+1}(x)\rangle = \mathrm{CNOT}|\Psi'_T(x)\rangle = |\Phi_0(x)\rangle|0\rangle + |\Phi_1(x)\rangle|1\rangle.$$

Let $U_{\mathcal{A}} = U_T \cdots U_1$, so that $|\Psi_T(x)\rangle = U_{\mathcal{A}}|\Psi_0\rangle$, and

$$|\Psi'_{2T+1}(x)\rangle = (U_{\mathcal{A}}^\dagger \otimes I)|\Psi'_{T+1}(x)\rangle = (U_{\mathcal{A}}^\dagger|\Phi_0(x)\rangle)|0\rangle + (U_{\mathcal{A}}^\dagger|\Phi_1(x)\rangle)|1\rangle. \quad (2.5)$$

Since $U_{\mathcal{A}}^\dagger|\Phi_b(x)\rangle = U_{\mathcal{A}}^\dagger\Pi_b|\Psi_T(x)\rangle$, for $b \in \{0,1\}$, the success probability of $\mathcal{A}'$ is equal to the success probability of $\mathcal{A}$:

$$\left\|(I \otimes |b\rangle\langle b|)|\Psi'_{2T+1}(x)\rangle\right\|^2 = \left\|U_{\mathcal{A}}^\dagger\Pi_b|\Psi_T(x)\rangle\right\|^2 = \|\Pi_b|\Psi_T(x)\rangle\|^2 = p_b(x).$$

Moreover, from Eq. (2.5), we have for all $b \in \{0,1\}$,

$$\langle\Psi_0, b|\Psi'_{2T+1}(x)\rangle = \langle\Psi_0|U_{\mathcal{A}}^\dagger\Pi_b U_{\mathcal{A}}|\Psi_0\rangle = \|\Pi_b U_{\mathcal{A}}|\Psi_0\rangle\|^2 = \|\Pi_b|\Psi_T(x)\rangle\|^2 = p_b(x).$$

In particular that implies that

$$\begin{aligned}\langle\Psi'_{T'}|\Psi'_{2T+1}(x)\rangle = \langle\Psi_0, 1|\Psi'_{2T+1}(x)\rangle = p_1(x) \quad \text{and} \\ \langle\Psi'_{T'}|(I \otimes X)|\Psi'_{2T+1}(x)\rangle = \langle\Psi_0, 0|\Psi'_{2T+1}(x)\rangle = p_0(x).\end{aligned}$$

Hence, $\mathcal{A}'$ satisfies the consistency condition as well. □

**Lemma 8.** *Fix $f : X \subseteq \{0,1\}^n \to \{0,1\}$. Let $\mathcal{A}$ be a quantum query algorithm with time complexity $T$ and query complexity $S$ that computes $f$ with error probability $\varepsilon > 0$. Then, there exists an algorithm $\mathcal{A}'$ with time complexity $T' = \Theta(T)$ and query complexity $S' \leq 3S$ such that two consecutive queries are no more than $\lfloor 3T'/S' \rfloor$ times steps apart. In addition if $\mathcal{A}$ fulfills conditions 1 and 2 in Definition 6, then $\mathcal{A}'$ is a clean quantum algorithm evaluating $f$ with error $\varepsilon$.*

*Proof.* First, if $S \in \{1,2\}$, we note that $\lfloor 3T/S \rfloor > T$, and hence the third condition in Definition 6 is trivially satisfied without any modifications to $\mathcal{A}$. Hence, we restrict to the case where $S \geq 3$. We insert a sequence of operations $I\mathcal{O}_x I\mathcal{O}_x I$ into $\mathcal{A}$ between time steps $\lceil kT/S \rceil$ and $\lceil kT/S \rceil + 1$ where $k \in [S-1]$. This increases the number of queries to $S' \leq 3S$ and the number of time steps to $T' = T + 5(S-1)$. The number of time steps between two consecutive queries is at most

$$\left\lceil \frac{T}{S} \right\rceil + 2 \leq \frac{T}{S} + 3 = \frac{T' - 5(S-1)}{S} + 3 \leq 3\frac{T'}{S'} - 5 + \frac{5}{S} + 3 < 3\frac{T'}{S'}.$$

As the left-hand side is an integer, we can just as well take the floor on the right-hand side. Similarly, the distance of the first query from the start is at most $\lceil T/S \rceil + 1 < 3T'/S'$, and the number of time steps between the last query and the end of the algorithm is at most $T - \lceil (S-1)T/S \rceil + 1 \leq T/S + 1 < 3T'/S'$. Thus, we have satisfied the query-uniformity condition from Definition 6.

Furthermore, the second statement follows immediately from the fact that the unitaries that we are inserting amount to the identity, and hence if $\mathcal{A}$ evaluates $f$ with error probability $\varepsilon$, so does $\mathcal{A}'$. This completes the proof. □

By Lemma 7 and Lemma 8, we can assume without loss of generality that any quantum algorithm is a clean quantum algorithm, namely, that it does a computation, copies out the answer, and then reverses the computation. The overhead of putting an algorithm into this form is only a constant factor in the query and time complexity, and a single auxiliary qubit in the space complexity.

For clarity, we emphasize that in a clean quantum algorithm with non-zero error, while in some sense the algorithm uncomputes everything but the answer, this uncomputation does not succeed fully – we do not return the

non-answer registers of the algorithm to the fixed state $|\Psi_0\rangle$. The weight of the final state $|\Psi_T(x)\rangle$ on $|\Psi_0\rangle$ in the non-answer registers is

$$|\langle\Psi_0, 0|\Psi_T(x)\rangle|^2 + |\langle\Psi_0, 1|\Psi_T(x)\rangle|^2 = p_0(x)^2 + p_1(x)^2,$$

which is strictly less than 1 whenever $0 < p_0(x) < 1$.

### 2.2.3 Four useful quantum subroutines

In this thesis we will present several algorithms that use in one way or another the phase estimation and amplitude estimation subroutines. To say that these subroutines appear in every quantum algorithm would only be a slight exaggeration and, certainly, the original papers to which we refer, [Kit95; CEM+98] and [BHM+02] must be among the most cited papers in the field of quantum computing. In addition to the more standard forms of phase and amplitude estimation, we will occasionally use *gapped* versions of them, all of which which we describe below.

**Theorem 9** (Phase Estimation [Kit95; CEM+98])**.** *Let $U$ be a unitary with eigenvectors $|\theta_j\rangle$ satisfying $U|\theta_j\rangle = e^{i\theta_j}|\theta_j\rangle$ and assume $\theta_j \in [-\pi, \pi]$. For any $\Theta \in (0, \pi)$ and $\varepsilon \in (0, 1)$, there exists a quantum algorithm, call it $PE(U, \Theta, \varepsilon)$, that makes $\mathcal{O}\left(\frac{1}{\Theta}\log\frac{1}{\varepsilon}\right)$ calls to $U$ and, on input $|\theta_j\rangle$ outputs a state $|\theta_j\rangle|w\rangle_P$ such that if $\theta_j = 0$, then $|w\rangle_P = |0\rangle_P$ and if $|\theta_j| \geq \Theta$, $|\langle 0|w\rangle_P|^2 \leq \varepsilon$. If $U$ acts on $s$ qubits, the algorithm uses $\mathcal{O}\left(s + \log\frac{1}{\Theta}\right)$ qubits and $\mathcal{O}\left(\left(\log\frac{1}{\theta} + \log\frac{1}{\varepsilon}\right)^2\right)$ extra elementary operations.*

We will use the following corollary of Theorem 9, which is a slight generalization of an algorithm introduced in [CKS17], also called Gapped Phase Estimation.

**Theorem 10** (Gapped Phase Estimation)**.** *Let $U$ be a unitary with eigenvectors $|\theta\rangle$ satisfying $U|\theta\rangle = e^{i\pi\theta}|\theta\rangle$ and assume $\theta \in [-1, 1]$. Let $\varphi \in (0, 1)$, let $\varepsilon > 0$ and let $\delta \in (0, 1 - \varphi]$. Then, there exists a unitary procedure $GPE(\varphi, \varepsilon, \delta)$ making $\mathcal{O}(\varphi^{-1}\log\varepsilon^{-1})$ queries to $U$ that on input $|0\rangle_C|0\rangle_P|\theta\rangle$ prepares a state $(\beta_0|0\rangle_C|\gamma_0\rangle_P + \beta_1|1\rangle_C|\gamma_1\rangle_P)|\theta\rangle$ where $|\gamma_0\rangle$ and $|\gamma_1\rangle$ are some unit vectors, $\beta_0^2 + \beta_1^2 = 1$ and such that*

- *if $|\theta| \leq \delta$, then $|\beta_1| \leq \varepsilon$, and*
- *if $\delta + \varphi \leq |\theta|$, then $|\beta_0| \leq \varepsilon$.*

*The registers $C$ and $P$ have 1 and $\mathcal{O}(\log(\varphi^{-1})\log(\varepsilon^{-1}))$ qubits respectively. In addition to the queries to $U$, the algorithm uses $\mathcal{O}\left((\log\frac{1}{\varphi}+\log\frac{1}{\varepsilon})^2\right)$ elementary gates.*

*Proof.* Standard phase estimation [Kit95; CEM+98] (but see, in particular [CEM+98, Appendix C]) on input $|\theta\rangle$ with precision $\varphi/2$ and error $\varepsilon$ prepares a state $|\tilde{\theta}\rangle_P|\theta\rangle$ such that upon measuring $|\tilde{\theta}\rangle_P$, with probability at least $1-\varepsilon$, we measure some $\bar{\theta}$ that is within $\varphi/2$ of $\theta$, meaning that if $|\theta| \leq \delta$, then $|\bar{\theta}| < \delta + \varphi/2$, and if $|\theta| \geq \delta + \varphi$, then $|\bar{\theta}| > \delta + \varphi/2$.

Instead of measuring, assume that we have an extra bit register $C$. Apply to registers $C, P$ the unitary that maps $|0\rangle_C|\bar{\theta}\rangle_P$ to $|0\rangle_C|\bar{\theta}\rangle_P$ if $|\bar{\theta}| \leq \delta + \varphi/2$, and to $|\bar{\theta}\rangle_{P_i}|1\rangle_{C_i}$ otherwise. This unitary can be done with $O(\log^2\frac{1}{\varphi})$ elementary gates.

Grouping all phase states with phases with $|0\rangle$ (resp. $|1\rangle$) in the $C$ register into $|\gamma_0\rangle$ (resp. $|\gamma_1\rangle$) we have that the state produced will be $(\beta_0|0\rangle_C|\gamma_0\rangle_P + \beta_1|1\rangle_C|\gamma_1\rangle_P)|\theta\rangle$, with $|\beta_1| \leq \varepsilon$ whenever $|\theta| \leq \delta$, and $|\beta_0| \leq \varepsilon$ whenever $|\theta| \geq \delta + \varphi$.

The number of elementary gates used in standard phase estimation with precision $\varphi/2$ and error $\varepsilon$ is $\mathcal{O}((\log 1/\varphi + \log 1/\varepsilon)^2)$, in addition to $\mathcal{O}\left(\frac{1}{\varphi}\log\frac{1}{\varepsilon}\right)$ calls to $U$, from which the result follows. □

**Theorem 11** (Amplitude Estimation [BHM+02])**.** *Let $\mathcal{A}$ be a quantum algorithm that, on input $x$, outputs*

$$\sqrt{p(x)}|0\rangle|\Psi_0(x)\rangle + \sqrt{1-p(x)}|1\rangle|\Psi_1(x)\rangle.$$

*Then there exists a quantum algorithm that estimates $p(x)$ to precision $\varepsilon$ using $O\left(\frac{1}{\varepsilon\sqrt{p(x)}}\right)$ calls to $\mathcal{A}$. If $s$ is the number of qubits that $\mathcal{A}$ uses, then the amplitude estimation algorithm uses $O\left(s + \log\left(\frac{1}{\varepsilon\sqrt{p(x)}}\right)\right)$ qubits and $O\left(\log^2\left(\frac{1}{\varepsilon\sqrt{p(x)}}\right)\right)$ additional elementary gates.*

We will make use of the following corollary (see [IJ19] for a proof).

**Corollary 12.** *Let $\mathcal{A}$ be a quantum algorithm that outputs the $s$ qubit state $\sqrt{p(x)}|0\rangle|\Psi_0(x)\rangle + \sqrt{1-p(x)}|1\rangle|\Psi_1(x)\rangle$ on input $x$ such that either $p(x) \leq p_0$, or $p(x) \geq p_1$ for $p_1 > p_0$. Then there exists a quantum algorithm*

*that, with bounded error, decides if* $p(x) \leq p_0$ *using* $\mathcal{O}\left(\frac{\sqrt{p_1}}{p_1-p_0}\right)$ *calls to* $\mathcal{A}$, $\mathcal{O}\left(s + \log\left(\frac{\sqrt{p_0}}{p_1-p_0}\right)\right)$ *qubits and* $\mathcal{O}\left(\log^2\left(\frac{\sqrt{p_0}}{p_1-p_0}\right)\right)$ *extra gates.*

## 2.3 Graph theory

A graph $G$ is a tuple formed by a set $V(G)$, called the *vertex set*, and a set $E(G)$ of pairs of elements in $V(G)$, called the *edge set*. In more colloquial terms, graphs are collections of vertices and edges, also known as links. Defined as such, they don't seem much, but graphs play a central role in discrete mathematics and computer science. That is because they are a natural way to describe sets and binary relations between elements of such sets. As a result, classical algorithms for graph problems are plentiful, and there is a large literature on quantum algorithms for graph problems. Graph theory is one of the first applications of span programs, with one particular span program standing out, the *st*-connectivity span program [BR12]. This span program is at the core of many quantum algorithms for other problems, like graph bipartiteness [Āri16], or cycle detection [CMB18], to name a few. In Chapter 5 we will give applications of the *st*-connectivity span program to graph connectivity and other problems.

### 2.3.1 Multigraphs

We will consider graphs which may have multiple edges between a pair of vertices, also known as multigraphs. Thus, to differentiate edges that share vertices, we associate a unique identifying label $\ell$. We refer to each edge in the graph using its endpoints and the label $\ell$, $(u, v, \ell)$, where the order of $u$ and $v$ denote the direction of the edge. We will sometimes write $(\{u, v\}, \ell)$ for an undirected edge, the curly brackets denoting the lack of order in the pair $u, v$, since, for us, an ordered set is a list. Given the nature of the span program for *st*-connectivity, it will be advantageous to treat undirected graphs as directed ones. We construct the set of directed edges $\overrightarrow{E}(G)$ of a graph $G$ by ascribing two directed edges $(u, v, \ell), (v, u, \ell) \in \overrightarrow{E}(G)$ to each edge $(\{u, v\}, \ell) \in E(G)$. In a slight abuse of notation we will sometimes drop the curly brackets while talking about undirected graphs with the convention that $(u, v, \ell) = (v, u, \ell)$.

**Networks** A *network* $\mathcal{N} := (G, c)$ consists of an undirected graph $G$ combined with a positive real-valued *weight* function $c : E(G) \longrightarrow \mathbb{R}^+$. Since $c$ is a map on undirected edges, we can easily extend it to a map on directed edges such that $c(u, v, \ell) = c(v, u, \ell)$, and we overload our notation accordingly. We will often assume that some $c$ is implicit for a graph $G$ and denote its adjacency matrix as:

$$\mathcal{A}_G = \sum_{(u,v,\ell)\in E(G)} c(u, v, \ell)(|u\rangle\langle v| + |v\rangle\langle u|). \tag{2.6}$$

Note that $\mathcal{A}_G$ only depends on the total weight of edges from $u$ to $v$, and is independent of the number of edges across which this weight is distributed.

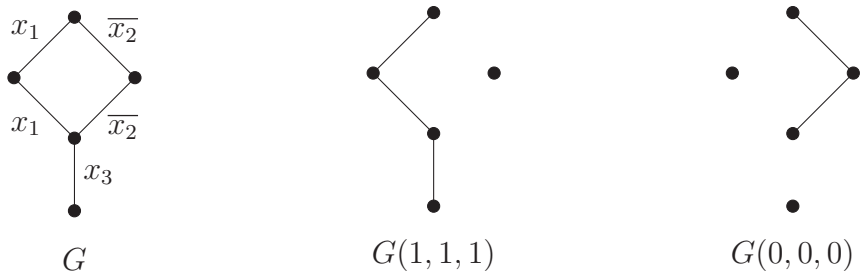



Figure 2.1: Example of a map from a 5-edge graph to a 3-bit string. The map $L$, depicted on the left, associates the edges of $G$ with a bit of $x$ and a value of that bit. For edges labeled by $x_i$ we include the edge in $G(x)$ if $x_i = 1$, while for edges labeled by $\overline{x_i}$, we include the edge in $G(x)$ if $x_i = 0$.

**Subgraphs** We will be concerned with certain subgraphs of a graph $G$, associated with bit strings of length $N$. We assume that there exists a map $L : E(G) \to \{x_1, \ldots, x_N, \bar{x}_1, \ldots, \bar{x}_N\}$ and denote by $G(x)$ the subgraph associated with the string $x \in \{0, 1\}^N$. In particular, each edge in $G$ is associated with a variable $x_i$ or its negation $\overline{x}_i$ and is included in $G(x)$ if and only if the associated variable evaluates to 1. Here $x_i$ is the $i^{\text{th}}$ bit of $x$. Observe that $N$ and $|E(G)|$ need not be the same in general. Precisely how this map $L$ is chosen depends on the problem of interest, so we will leave the description implicit, and often assume for simplicity that there is a one-to-one mapping between the edges and the literals $x_i$.

### 2.3.2 Laplacians

Typically, quantum algorithms focus on topological properties of networks, but networks are much more than fixed topologies, they are often the substrate of dynamic processes. These dynamic processes, in turn, are influenced by the topology of the network and their dynamics described by matrix representations. In our discussion of span program algorithms for graph connectivity we will make use of a matrix representation known as the *Laplacian* of a graph, and its first non-zero eigenvalue, called the *algebraic connectivity* or *Fiedler value*. This matrix representation was introduced to model the process of *diffusion* on a graph, but is not limited to the diffusion process. For example, Laplacians play important roles in other processes like synchronization or random walks, to mention a few.

Let $d_G(u) = \sum_{v,\ell:(u,v,\ell)\in E(G)} c(u,v,\ell)$ denote the weighted degree of $u$ in $G$, under the implicit weight function $c$, and define the degree matrix $\mathcal{D}_\mathcal{G}$ as:

$$\mathcal{D}_G = \sum_{u\in V(G)} d_G(u)|u\rangle\langle u| \tag{2.7}$$

We define the *Laplacian of* $G$ as:

$$L_G := \mathcal{D}_G - \mathcal{A}_G. \tag{2.8}$$

Different processes will have different matrix representations, the Laplacian being the right one for the process of diffusion. Matrix representations encode both dynamical as well as topological properties of graphs, which are accessed through their eigenvalues and eigenvectors. Observe that the Laplacian is always positive semi-definite, so its eigenvalues are real and non-negative. For $|\mu\rangle = \sum_{u\in V(G)}|u\rangle$, it is always the case that $L_G|\mu\rangle = 0$, so the smallest eigenvalue of $L_G$ is 0. Of particular importance is the second smallest eigenvalue of $L_G$ including multiplicity denoted as $\lambda_2(G)$. This value is called the *algebraic connectivity* or the *Fiedler value* of $G$, and it is non-zero if and only if $G$ is connected. For connected networks it governs the speed of diffusion and its associated eigenvector, called the *Fiedler vector* is used for community detection and spectral clustering [Fie89].

A variant of the Laplacian with direct connections to random walks, isoperimetric problems and expander graphs (and many other things too, see [Chu97] ) is the *symmetric normalized Laplacian* defined as:

$$L_G^{\text{sym}} = \mathcal{D}_G^{-1/2} L_G \mathcal{D}_G^{-1/2}. \tag{2.9}$$

Then we let $\delta(G)$ denote the second smallest eigenvalue of $L_G^{\text{sym}}$. One can check that $\delta(G)$ is the spectral gap of the random walk operator $P$ on $G$ [Chu97], and in the case that $G$ is regular, so that $d_G(u) = d$ does not depend on $u$, $\delta(G) = \frac{1}{d}\lambda_2(G)$.

**Bounds on the algebraic connectivity** In some of the algorithms that we will design in Chapter 5 for graph connectivity, the complexity of the algorithms will depend on the algebraic connectivity. Computing the algebraic connectivity, even approximately, is in itself a non-trivial task equivalent to solving the symmetric eigenvalue problem of the Laplacian matrix. However, it is possible to give bounds on the algebraic connectivity of a connected graph that depend only on topological invariants. In particular, we will be interested in lower bounds. An extensive discussion of bounds for the algebraic connectivity is found in [Abr07]. A very simple bound can be obtain in terms of the diameter of a graph, which is defined as the maximum distance between any two vertices. Then the algebraic connectivity is lower bounded as:

$$\lambda_2(G) \geq \frac{4}{\text{diam}(G)n}. \tag{2.10}$$

### 2.3.3 Electrical networks

Considering the definition of graphs and networks, with nodes and weighted edges, it is natural to attempt to model electrical networks. An electrical network is in essence a network in which each edge is a conductor of a certain conductance and through which electrons flow once an electric potential difference is set between two points. These electrons flow through the different elements of the circuit in an attempt to avoid each other and minimize the energy dissipated as much as possible following the well known laws of Kirchhoff and Ohm. Therefore, if we want to import notions of graph theory to the study of electrical networks we first need to define language to model these flow and energy exchanges. The first concept we generalize is that of electrical flow. One can consider a fluid that enters a graph $G$ at a node $s$, flows along the edges of the graph, and exits the graph at a different node $t$. The fluid can spread out along some number of the $st$-paths in $G$. An *$st$-flow* is any linear combination of $st$-paths. More precisely:

**Definition 13** (Unit *st*-flow)**.** Let $G$ be an undirected graph with $s, t \in V(G)$, and $s$ and $t$ connected. Then a *unit st-flow* on $G$ is a function $\theta : \overrightarrow{E}(G) \to \mathbb{R}$ such that:

1. For all $(u, v, \ell) \in \overrightarrow{E}(G)$, $\theta(u, v, \ell) = -\theta(v, u, \ell)$;
2. $\sum_{v,\ell:(s,v,\ell)\in\overrightarrow{E}(G)} \theta(s, v, \ell) = \sum_{v,\ell:(v,t,\ell)\in\overrightarrow{E}(G)} \theta(v, t, \ell) = 1$; and
3. for all $u \in V(G) \setminus \{s, t\}$, $\sum_{v,\ell:(u,v,\ell)\in\overrightarrow{E}(G)} \theta(u, v, \ell) = 0$.

This function models the amount of fluid flowing through each edge. In our electrical analogy that means the *intensity*. No fluid is lost and all that enters through $s$ exits through $t$. Not every *st*-flow models the flow of electrons on an electrical network, but every electron flow is an *st*-flow. In order to model the flow of electrons we need to define the concept of energy of the flow.

**Definition 14** (Unit Flow Energy)**.** Given a graph $G$ with implicit weighting $c$ and a unit *st*-flow $\theta$ on $G(x)$, the *unit flow energy* of $\theta$ on $E' \subseteq E(G(x))$, is:

$$J_{E'}(\theta) = \frac{1}{2} \sum_{e\in\overrightarrow{E'}} \frac{\theta(e)^2}{c(e)}. \tag{2.11}$$

In our electrical analogy, this function corresponds exactly to the energy dissipated in the network by a unit intensity current flowing from $s$ to $t$ and is sometimes referred to as Watt's Law, Power = Resistance $\times$ Intensity$^2$. The weight function $c(e)$ here plays the role of the inverse of the electrical resistance of an edge, also known as *conductance*. As we said, electrons flow in a network in an attempt to minimize energy losses. In the spirit of Watt's law, the minimizing unit flow defines then the *effective resistance*.

**Definition 15** (Effective resistance)**.** Let $G$ be a graph with implicit weighting $c$ and $s, t \in V(G)$. If $s$ and $t$ are connected in $G$, the *effective resistance* of $G$ between $s$ and $t$ is $R_{s,t}(G) = \min_\theta J_{E(G)}(\theta)$, where $\theta$ runs over all unit *st*-unit flows of $G$. If $s$ and $t$ are not connected in $G$, $R_{s,t}(G) = \infty$.

Intuitively, $R_{s,t}$ characterizes "how connected" the vertices $s$ and $t$ are in a network. The more, shorter paths connecting $s$ and $t$, and the more weight on those paths, the smaller the effective resistance.

Our interest in this analogy lies in the many applications of effective resistance. For example, $R_{s,t}(G) \cdot \left(\sum_{e \in E(G)} c(e)\right)$ is equal to the *commute time* between $s$ and $t$, or the expected time a random walker starting from $s$ takes to reach $t$ and then return to $s$ [CRR+96]. Also, electrical networks and effective resistance play an important role in understanding the $st$-connectivity span program of Section 5.2.

The resistance is a measure of how well connected two particular nodes in a graph are. Averaging over all pairs of nodes then gives a global measure of connectedness. For a connected graph $G$, we define the *average resistance* as:

$$R_{\text{avg}}(G) := \frac{1}{n(n-1)} \sum_{s,t \in V : s \neq t} R_{s,t}(G). \tag{2.12}$$

Now that we have a measure of the connectedness of $s$ and $t$ in a graph $G$, we next introduce a measure of how disconnected $s$ and $t$ are in a subgraph $G(x)$ of $G$. In an electrical network, when a voltage difference is applied to a pair of connected nodes, a flow through the network appears. When the nodes are not connected there will be no flow but electrons (and electron deficits) are going to accumulate in the different connected components of the network, giving rise to different potential energies in each component. This potential function is the relevant quantity, which we define in the context of networks as follows.

**Definition 16** (Unit $st$-potential)**.** Let $G$ be an undirected weighted graph with $s, t \in V(G)$, and $s$ and $t$ connected. For $G(x)$ such that $s$ and $t$ are not connected, a *unit $st$-potential* on $G(x)$ is a function $\mathsf{V} : V(G) \to \mathbb{R}^+$ such that $\mathsf{V}(s) = 1$ and $\mathsf{V}(t) = 0$ and $\mathsf{V}(u) = \mathsf{V}(v)$ if $(u, v, \ell) \in E(G(x))$ for some $\ell$.

Note that this is a different definition from the typical potential function. Usually, if we have a flow from a vertex $s$ to a vertex $t$, we define the potential difference between $u$ and $v$ for an edge $(u, v, \ell)$ to be the amount of flow across that edge divided by the weight of the edge. In our definition, the potential difference across all edges in $E(G(x))$ is zero, and we have potential difference across edges that are in $E(G) \setminus E(G(x))$.

A unit $st$-potential is a witness of the disconnectedness of $s$ and $t$ in $G(x)$ through its missing edges, in the sense that it is a generalization of the notion of an $st$-cut, which is simply a unit potential that only takes values 0 and 1.

Just like flows have energy, we can define the energy of a unit potential. Again, this has a tight correspondence with electrical network concepts.

**Definition 17** (Unit Potential Energy)**.** Given a graph $G$ with implicit weighting $c$ and a unit $st$-potential $\mathsf{V}$ on $G(x)$, the *unit potential energy* of $\mathsf{V}$ on $E' \subseteq E(G)$ is defined as:

$$\mathcal{J}_{E'}(\mathsf{V}) = \frac{1}{2} \sum_{(u,v,\ell) \in \overrightarrow{E'}} (\mathsf{V}(u) - \mathsf{V}(v))^2 c(u,v,\ell). \tag{2.13}$$

**Definition 18** (Effective capacitance)**.** Let $G$ be a graph with implicit weighting $c$ and $s, t \in V(G)$. If $s$ and $t$ are not connected in $G(x)$, the *effective capacitance* between $s$ and $t$ of $G(x)$ is $C_{s,t}(G(x)) = \min_{\mathsf{V}} \mathcal{J}_{E(G) \setminus E(G(x))}(\mathsf{V})$, where $\mathsf{V}$ runs over all unit $st$-potentials on $G(x)$. If $s$ and $t$ are connected, $C_{s,t}(G(x)) = \infty$.

In physics, capacitance is a measure of how well a system stores electric charge. The simplest capacitor is just two metal plates facing each other at a certain distance and separated by an insulating layer. Within each connected component of the circuit, all points have the same voltage. Therefore, each connected component behaves as a single node with respect to voltage, and only the total capacitance between components determines the voltage on those components. Consider a graph $G(x)$ in which a 0-resistance wire is connected between vertices whenever there is an edge in $G(x)$, and a $c(e)$-unit capacitor is connected between vertices whenever there is an edge $e \in E(G) \setminus E(G(x))$. If $s$ and $t$ are not connected in $G(x)$, it is as though $s$ and $t$ are on separate "plates" (with some complicated geometry) that can accumulate charge relative to each other. Then the effective capacitance given in Definition 18 is precisely the ratio of charge (accumulated on the plates corresponding to $s$ and $t$) to voltage (on those plates) that is achieved when electrical energy is stored in this configuration.

Definitions 15 and 18 may seem unwieldy for actually calculating the effective resistance and effective capacitance. Luckily for us, there are simple and well known ways of computing the combined resistance of a network of resistors. Any reader with a basic knowledge in physics or electrical engineering might recall that resistances in series add, while for resistors connected in parallel it is the inverse resistances that add. Capacitors follow the same relations with parallel and series switched. Most importantly, these relations

remain valid for the effective resistances and capacitances of networks as the following proposition shows.

**Proposition 19.** *Let two networks $(G_1, c_1)$ and $(G_2, c_2)$ each have connected nodes $s$ and $t$. Let $G(x_1)$ and $G(x_2)$ be subgraphs of $G_1$ and $G_2$ respectively. Then we consider a new graph $G$ by identifying the $s$ nodes and the $t$ nodes of $G_1$ and $G_2$ (i.e. connecting the graphs in parallel) and define $c : E(G) \to \mathbb{R}^+$ by $c(e) = c_1(e)$ if $e \in E(G_1)$ and $c(e) = c_2(e)$ if $e \in E(G_2)$. Similarly, we set $G(x)$ to be the subgraph of $G$ that includes the corresponding edges $e$ such that $e \in E(G_1(x_1))$ or $e \in E(G_2(x_2))$. Then,*

$$\frac{1}{R_{s,t}(G(x))} = \frac{1}{R_{s,t}(G_1(x))} + \frac{1}{R_{s,t}(G_2(x))} \tag{2.14}$$

$$C_{s,t}(G(x)) = C_{s,t}(G_1(x_1)) + C_{s,t}(G_2(x_2)) \tag{2.15}$$

*If we create a new graph $G$ by identifying the $t$ node of $G_1$ with the $s$ node of $G_2$, relabeling this node $v \notin \{s, t\}$ (i.e. connecting the graphs in series) and define $c$ and $G(x)$ as before, then,*

$$R_{s,t}(G(x)) = R_{s,t}(G_1(x)) + R_{s,t}(G_2(x)), \tag{2.16}$$

$$\frac{1}{C_{s,t}(G(x))} = \frac{1}{C_{s,t}(G_1(x_1))} + \frac{1}{C_{s,t}(G_2(x_2))}. \tag{2.17}$$

*Proof.* We deal first with effective resistances. In plain English, the statement says that the effective resistance of two graphs connected in parallel follow an inverse addition law but a direct addition law for effective capacitance. Let $(G_1, c_1)$ and $(G_2, c_2)$ be two networks connected in parallel as in the statement of the Proposition. We begin with the definition of effective resistance for a graph of this form.

$$R_{s,t}(G(x)) = \min_{\theta} \sum_{e \in E(G(x))} \frac{\theta^2(e)}{c(e)} = \min_{\theta} \left( \sum_{e \in E(G_1(x))} \frac{\theta^2(e)}{c(e)} + \sum_{e \in E(G_2(x))} \frac{\theta^2(e)}{c(e)} \right),$$

where $\theta$ is an $st$-flow in $G(x)$. Observe that this flow breaks into two non-unit $st$-flows $\phi_1\theta_1$ and $\phi_2\theta_2$ defined in $G(x_1)$ and $G(x_2)$, where $\theta_1$, $\theta_2$ are unit $st$-flows, and the factors $\phi_1$, $\phi_2$ are defined as:

$$\phi_1 := \sum_{u,\ell:(s,u,\ell) \in E(G(x_1))} \theta_1(u, v, \ell), \quad \phi_2 := \sum_{u,\ell:(s,u,\ell) \in E(G(x_2))} \theta_2(u, v, \ell) = 1 - \phi_1.$$

From the definition of flow energy it follows that the energy of $\phi_i\theta_i$, $i=1,2$ is $\phi_i^2$ times that of $\theta_i$. Therefore, the effective resistance can be written as:

$$\begin{aligned} R_{s,t}(G(x)) &= \min_{\theta_1} \sum_{e\in E(G_1(x))} \phi_1^2 \frac{\theta_1^2(e)}{c(e)} + \min_{\theta_2} \sum_{e\in E(G_2(x))} \phi_2^2 \frac{\theta_2^2(e)}{c(e)} \\ &= \min_{\phi_1,\phi_2} \phi_1^2 R_{s,t}(G(x_1)) + \phi_2^2 R_{s,t}(G(x_2)). \end{aligned} \tag{2.18}$$

Since $\phi_1+\phi_2=1$, this minimization problem reduces to taking a derivative and equating to zero. The solution one arrives at is:

$$\frac{1}{R_{s,t}(G(x))} = \frac{1}{R_{s,t}(G_1(x))} + \frac{1}{R_{s,t}(G_2(x))}. \tag{2.19}$$

Now, let $(G_1,c_1)$ and $(G_2,c_2)$ be two networks connected in series and joined through a single vertex called $v$. Then the effective resistance of the new graph is:

$$R_{s,t}(G(x)) = \min_{\theta} \sum_{e\in E(G(x))} \frac{\theta^2(e)}{c(e)} = \min_{\theta} \left( \sum_{e\in E(G_1(x))} \frac{\theta^2(e)}{c(e)} + \sum_{e\in E(G_2(x))} \frac{\theta^2(e)}{c(e)} \right).$$

Notice now that any $st$-flow in $G(x)$ is composed of an $sv$-flow defined in the vertices of $G(x_1)$ and a $vt$-flow in the vertices of $G(x_2)$ which are independent. We conclude that the effective resistance is simply:

$$R_{s,t}(G(x)) = R_{s,v}(G(x)) + R_{v,t}(G(x)) = R_{s,t}(G_1(x)) + R_{s,t}(G_2(x)). \tag{2.20}$$

Let us now concern ourselves with effective capacitances of graphs connected in series and in series. Let $G$ be a graph composed of connecting two graphs $G_1$ and $G_2$ in series joined at a single vertex which we label $v$. Let $G(x)$ be a subgraph corresponding to taking $G(x_1)$ in $G_1$ and $G(x_2)$ in $G_2$ and let $s\in G_1$ $t\in G_2$. By definition, the effective capacitance between the vertices $s$ and $t$ of $G(x)$ is:

$$C_{s,t}(G(x)) = \min_{\mathsf{V}} \mathcal{J}_{E(G)\setminus E(G(x))}(\mathsf{V}) \tag{2.21}$$

$$\begin{aligned} &= \min_{\mathsf{V}} \left\{ \frac{1}{2} \sum_{(u,w,\ell)\in E(G_1)\setminus E(G(x_1))} (\mathsf{V}(u)-\mathsf{V}(w))^2 c(u,w,\ell) \right. \\ &\left. + \frac{1}{2} \sum_{(u,w,\ell)\in E(G_2)\setminus E(G(x_2))} (\mathsf{V}(u)-\mathsf{V}(w))^2 c(u,w,\ell) \right\}, \end{aligned}$$

where we have used that $E(G) \setminus E(G(x))$ is the disjoint union of $E(G_1) \setminus E(G(x_1))$ and $E(G_2) \setminus E(G(x_2))$. Now consider any unit potential function $\mathsf{V}$ in $G(x)$, and consider its restrictions $\mathsf{V}|_{G(x_1)}$ and $\mathsf{V}|_{G(x_2)}$. Observe that $\mathsf{V}|_{G(x_1)}(s) = 1$, and $(u, w, \ell) \in E(G(x_1)) \Rightarrow \mathsf{V}|_{G(x_1)}(u) = \mathsf{V}|_{G(x_1)}(w)$. That is, edges connected in $G(x)$ have the same potential. Since $\mathsf{V}(v) \neq 0$, $\mathsf{V}|_{G(x_1)}$ is not a unit $sv$-potential in $G(x_1)$, but $\mathsf{V}' := \left(\mathsf{V}|_{G(x_1)} - \mathsf{V}(v)\right)/(1 - \mathsf{V}(v))$ is. Similarly, $\widetilde{\mathsf{V}} = \mathsf{V}(v) \cdot \mathsf{V}|_{G(x_2)}$ is a unit $vt$-potential in $G(x_2)$. We conclude that the effective capacitance can be written as

$$
\begin{aligned}
&C_{s,t}(G(x)) = \\
&\min_{\mathsf{V}(v)\in[0,1]} \Bigg\{ (1-\mathsf{V}(v))^2 \min_{\mathsf{V}'} \frac{1}{2} \sum_{(u,w,\ell)\in E(G_1)\setminus E(G(x_1))} (\mathsf{V}'(u) - \mathsf{V}'(w))^2 c(u,w,\ell) \\
&\qquad + \mathsf{V}^2(v) \min_{\widetilde{\mathsf{V}}} \frac{1}{2} \sum_{(u,w,\ell)\in E(G_2)\setminus E(G(x_2))} (\mathsf{V}(u) - \mathsf{V}(w))^2 c(u,w,\ell) \Bigg\} \\
&\quad = \min_{\mathsf{V}(v)\in[0,1]} \{(1-\mathsf{V}(v))^2 C_{st}(G(x_1)) + \mathsf{V}^2(v) C_{st}(G(x_2))\}. \qquad (2.22)
\end{aligned}
$$

Notice how this equation is equivalent to (2.18), and so its solution has the same form. We conclude that the capacitance of graphs connected in series is:

$$
C_{s,t}(G(x)) = \frac{1}{\frac{1}{C_{s,t}(G_1(x_1))} + \frac{1}{C_{s,t}(G_2(x_2))}}.
$$

At last we turn to the capacitance of networks connected in parallel. It is trivial to see that in this case the restrictions $\mathsf{V}|_{G(x_1)}$ and $\mathsf{V}|_{G(x_2)}$ of any unit potential in $G(x)$ must be themselves unit potentials, and therefore the capacity is the sum of minimizing the potential energy over $G(x_1)$ and $G(x_2)$ independently. □

# Part II

# The one where we discuss the theory of span programs

# Chapter 3

# Theory of span programs

## 3.1 Overview

In this chapter we will introduce, develop and eventually expand the notion of span programs. Span programs are a model of computation first introduced by Karchmer and Wigderson [KW93] for the study of classical counting branching programs, and imported to the quantum setting by Reichardt and Špalek in [RŠ12; Rei09] for the study of formula evaluation and the dual adversary lower bound. Since then, they have been applied to the design of quantum algorithms, lower bounds, and have been reformulated and generalized.

Span programs, regardless of the chosen formulation, *encode* a function $f : [q]^n \to \{0, 1\}$ (see Section 2.2) in terms of linear algebra by framing it as a problem of vectors being contained in input-dependent subspaces of an inner product space. We sometimes abuse the English language and say that they *compute* $f$. By themselves, span programs are not quantum objects. They have classical input and classical output, and consist of a collection of vectors and input-dependent vector spaces. Their connection to quantum computing comes in two steps.

First, every span program that computes a function $f$ corresponds to a feasible solution for the semi-definite program (SDP) dual to the *general adversary bound* SDP for $f$ whose objective value is a quantity known as the *span program complexity*, $C(P)$ (see Section 3.2.3 and [Rei09]). The general adversary bound is a technique for computing *lower bounds* to the

quantum query complexity of a function through feasible solutions to an SDP. Therefore, span programs give *upper bounds* to the objective value of the solution that is optimal for both the dual and primal SDP.

Second, every span program deciding a function $f$ can be turned into a quantum algorithm that computes the same function $f$, and whose query complexity is $\mathcal{O}(C(P))$. Furthermore, this process of turning a span program into a quantum algorithm is constructive and universally applicable (although not unique, stay tuned). Among other things, this means that *the quantum query complexity of a function $f$ is equal to the minimum complexity of any span program computing $f$.* This transformation allows us to design quantum algorithms using classical thought. The goal of this chapter is to define span programs and understand in detail their geometry and structure in order to construct such *span program algorithms*.

What is even more exciting is that span programs encode quite a bit more than the single output bit of the function they decide. For example, in [IJ19], Ito and Jeffery build an algorithm that evaluates a positive real-valued function related to any given span program $P$ called the *witness size*, and we show in Section 3.4.2 how one can modify the span program algorithm to generate certain quantum states as outputs. In Chapter 5 we build on this idea and make a three-course meal out of the *st*-connectivity span program.

**A first definition** We begin by defining a span program in Section 3.2.1 in the manner of [Rei09]. There, span programs are defined as tuples formed by a vector space $\mathcal{V}$ over $\mathbb{C}$, a target vector $|\tau\rangle \in \mathcal{V}$ and a set of input vectors $\{|v_i\rangle\}_{i\in I} \subseteq \mathcal{V}$ over an index set $I$. To every $x \in [q]^n$ corresponds a subset $I(x) \subseteq I$. The program is said to evaluate the function $f_P : [q]^n \to \{0,1\}$ defined as

$$f_P(x) = \begin{cases} 1 & \text{if } |\tau\rangle \in \text{span}\,\{|v_j\rangle : j \in I(x)\} \\ 0 & \text{otherwise.} \end{cases}$$

The relation between these available vectors and the inputs is found in Definition 20. This formulation, although sufficient to show equivalence between span programs and dual adversary solutions (see Section 3.2.3) is somewhat unsatisfying because: 1) it puts the emphasis in the space $\mathcal{V}$ (which is *not* where the span program algorithms operate), and, 2) it allows for non-binary alphabets $[q]$, $q \in \mathbb{N}$, but at an extra cost of $\mathcal{O}(\log q)$ in the query complexity of the associated algorithms.

**An alternative definition** An alternative definition of span programs originally proposed by Ito and Jeffery [IJ19] comes in Section 3.2.2. The main innovations in this definition are the introduction of a new space $\mathcal{H}$ that replaces the index set $I$, and a map $A : \mathcal{H} \to \mathcal{V}$ that replaces the notion of "available" vectors and simplifies the analysis. The input $x \in [q]^n$ does not define a subset $I(x)$ of the index set but a subspace $\mathcal{H}(x) \subseteq \mathcal{H}$ of a particular form (see Definition 22).

In this notation, a span program is a tuple $P = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$ that encodes the function $\{f(x) = 1 \Leftrightarrow \exists\, |w\rangle \in \mathcal{H}(x) : A|w\rangle = |\tau\rangle\}$. Any such vector $|w\rangle$ is called a *positive witness* for $x$ and acts as a sort of certificate for positivity. As we just said, the original definition of span programs allows for non-binary alphabets $[q]$ with logarithmic overhead. This definition improves on it by removing that logarithmic factor. Other than this concrete improvement, this alternative definition puts the emphasis on $\mathcal{H}$, where the actual span program algorithms act, rather than $\mathcal{V}$.

It is using this notation that the authors of [IJ19] define *approximate span programs*. Approximate span programs are an extension of the span program formalism that will play an important role in this thesis, particularly in Chapter 4, so let us explain them up front. Consider the space $\mathcal{H}(x)$ and its image under $A$, $A(\mathcal{H}(x)) \subseteq \mathcal{V}$. Just like a span program $P$ computes the function $\{f(x) = 1 \Leftrightarrow |\tau\rangle \in A(\mathcal{H}(x))\}$, we say that $P$ *approximately computes* a function $\{g(x) = 1 \Leftrightarrow |\tau\rangle$ is *approximately in* $A(\mathcal{H}(x))\}$, for a carefully chosen notion of closeness (see Section 3.3.3 and Definitions 36 to 39). The gap between what is and isn't *close enough* depends on a parameter $\lambda \in [0, 1)$ called the *approximation factor*. Then, we say that $P$ $\lambda$-approximates $g$ or that it is a $\lambda$-*approximate span program* for $g$.

In this way, a span program decides a family of functions, rather than just one. In some sense, it makes a span program more powerful because it decides more functions. This is useful because it makes it easier to find a span program that computes (maybe approximately) any given decision problem. But approximate span programs are *not* a more powerful framework than exact (as in, not approximate) span programs. For every decision problem there exists a span program that exactly computes it with optimal complexity, it's just that that span program is hard to find, and an approximate span programs might be easier to design.

**A few algorithms for span programs** We wrap up the literature review with Section 3.2.3, where we discuss the adversary lower bound and its relation to span programs, and give a quick overview of the algorithms presented by Reichardt and Ito & Jeffery.

**Reflection programs** Next we present a novel reformulation of span programs, the first original contribution of this dissertation, called *reflection programs*. Reflection programs are a generalization of the span program formalism that aims to strip down all the parts that are not essential for constructing an algorithm. We define reflection programs in Section 3.3, give geometrical interpretations, and then define $\lambda$-*approximate reflection programs* and functions.

It is important to remark that any span program can be repackaged as a reflection program for the purposes of turning them into algorithms, but that does not make the language of span programs unnecessary or obsolete since the choices of the spaces $\mathcal{H}$, $\mathcal{V}$, the map $A$, and the target vector $|\tau\rangle$ are much more intuitive in a span program than in a reflection program. This will become apparent in Chapter 5 when we describe in depth the span program for $st$-connectivity. Like span programs, reflection programs lend themselves well to compositions and transformations. We finish the section by explaining how to turn a positively $\lambda$-approximating reflection program into a negative one, and how to write span programs as reflection programs.

**Algorithms for reflection programs** We proceed in Section 3.4 with providing an algorithm for positively $\lambda$-approximating reflection programs which is also applicable to span programs (since these are a special case of reflection programs). The algorithm generalizes the span program algorithm in [IJ19] to reflection programs and somewhat simplifies the analysis by following the approach of [Chi21].

Finally, we provide two new algorithms that generate the optimal positive witness for $x$ in a reflection program $\mathcal{R}$. In the span programs setting, that is the smallest vector $|w_x\rangle \in \mathcal{H}(x)$ such that $A|w_x\rangle = |\tau\rangle$. These are the first span program algorithms of their kind, and the first to have quantum output. In Section 5.7 we will use one of these algorithms on the $st$-connectivity span program to find an $st$-path that outperforms all known quantum algorithms for path-finding and the classical query lower bound for the problem it solves.

The original contributions in this chapter are contained in Section 3.3

and Section 3.4. Only Section 3.2.2 within Section 3.2 is strictly necessary to understand these results, but we include the others for context and in the interest of giving a complete account of how span programs have appeared in the field of quantum computing.

## 3.2 Span programs

### 3.2.1 Span programs: a first definition

We begin by defining a span program in the manner of [Rei09]. There, span programs are defined as tuples formed by a vector space $\mathcal{V}$ over $\mathbb{C}$, a target vector $|\tau\rangle \in \mathcal{V}$ and a set of input vectors $\{|v_i\rangle\}_{i\in I} \subseteq \mathcal{V}$ over an index set $I$, or rather, a subset $I(x) \subseteq I$ that depends on $x \in [q]^n$.

**Definition 20** (Span program [Rei09])**.** Let $n$, $q \in \mathbb{N}$, $[q]^n$. A span program $P$ over $[q]^n$ is a tuple $P = (|\tau\rangle, \mathcal{V}, I, \{|v_j\rangle\}_{j\in I})$ where $|\tau\rangle \in \mathcal{V}$ is called the *target* vector, $\mathcal{V}$ is a finite-dimensional inner product space over $\mathbb{C}$ and the vectors in $\{|v_j\rangle\}_{j\in I}$ are called *input vectors*. The index set $I$ is the disjoint union $I = I_{\text{free}} \sqcup \bigsqcup_{i\in[n],b\in[q]} I_{i,b}$ for some sets $I_{\text{free}}$ and $I_{i,b}$, and given an input $x = (x_1, \ldots, x_n) \in [q]^n$, the subset $I(x)$ is defined as $I(x) = I_{\text{free}} \sqcup \bigsqcup_{i\in[n]} I_{i,x_i}$.

We say $P$ *computes* the total function $f_P : [q]^n \to \{0, 1\}$ defined by

$$f_P(x) = \begin{cases} 1 & \text{if } |\tau\rangle \in \operatorname{span}\{|v_j\rangle : j \in I(x)\} \\ 0 & \text{otherwise.} \end{cases} \tag{3.1}$$

The vectors in $\{|v_j\rangle : j \in I(x)\}$ are called *available vectors*, and so the function $f$ reduces to "Is $|\tau\rangle$ in the span of available vectors?". Let us look at the index sets $I$ and $I(x)$ more closely. For every coordinate $i \in [n]$ and possible value $b \in [q]$, we have a set of indices $I_{i,b}$. We might have some other indices that are available to all inputs which make the free set $I_{\text{free}}$. An input $x = (x_1, \ldots, x_n)$ makes *available* the index sets $I_{i,x_i}$ for $i \in [n]$, which form the available index set $I(x)$, meaning that all vectors $|v_j\rangle$ with indices in the available set may be used to construct $|\tau\rangle$.

The intuition that connects span programs to quantum query complexity is that an input oracle $\mathcal{O}_x$ acting as $\mathcal{O}_x|i, b\rangle = |i, b \oplus x_i\rangle$, acts on the space $\mathbb{C}^I$ —typically spanned by vectors $|i, b\rangle$, $i \in [n], b \in [q]$— by "marking" the vectors corresponding to available indices. Classically, we would say that the

available index set is built by querying the input. Quantumly, the intuition is that $\mathcal{O}_x$ gives us access to $\mathbb{C}^{I(x)}$ in some way.

The space $\mathbb{C}^I$ generated by the index set is much more important than it looks. The algorithms we mention in Section 3.2.3 for span programs in this formulation operate on $\mathbb{C}^I$, rather than $\mathcal{V}$, and the positive and negative witness sizes that we will define presently are computed in $\mathbb{C}^I$. First, let us define a pair of maps ancillary to those definitions.

$$\begin{aligned} &A : \mathbb{C}^I \to \mathcal{V} && \Pi(x) : \mathbb{C}^I \to \mathbb{C}^I \\ &A|j\rangle = |v_j\rangle && \Pi(x) = \sum_{j \in \bigsqcup_{i\in[n]} I_{i,x_i} \sqcup I_{\text{free}}} |j\rangle\langle j|. \end{aligned} \tag{3.2}$$

The first map makes explicit the correspondence between inputs and input vectors, while the second selects the indices that are available for a given input $x$. We define witness sizes as follows:

**Definition 21** (Witness sizes [Rei09])**.** Consider a span program $P$ over $[q]^n$, and let $f_P$ be the function defined in Equation (3.1). For each input $x \in [q]^n$, we define its witness size as follows:

- If $f_P(x) = 1$, there exists a state $|w\rangle \in \mathbb{C}^I$ such that $A\Pi(x)|w\rangle = |\tau\rangle$. Such state is called a *positive witness*. Then,

$$\text{wsize}(P, x) = \min\{\||w\rangle\|^2 : A\Pi(x)|w\rangle = |\tau\rangle\}. \tag{3.3}$$

- If $f_P(x) = 0$, then one can show that there always exists a witness $|w'\rangle \in \mathcal{V}$ such that $\langle w'|\tau\rangle = 1$ but $\langle w'|A\Pi(x) = 0$. Such vector is called a *negative witness*. Then,

$$\text{wsize}(P, x) = \min\{\|A^\dagger|w'\rangle\|^2 : \langle w'|\tau\rangle = 1, \langle w'|A\Pi(x) = 0\}. \tag{3.4}$$

The *witness size of* $P$, also known as the *span program complexity of* $P$, is defined as:

$$\text{wsize}(P) = \max_{x\in[q]^n} \text{wsize}(P, x). \tag{3.5}$$

We define positive and negative witnesses for every input to the total function $f_P$ from Equation (3.1), which is in some sense the natural function

of $P$. But $P$ can also decide any partial functions of $f_P$, that is, functions $f : X \subseteq [q]^n \to \{0,1\}$ such that $f(x) = f_P(x)$ for all $x \in X$, just by restricting the inputs to $X$. These are promise versions of the decision problem on $f_P$, which might have smaller witness size. For such a partial function we define the *span program complexity of $f$ with respect to $P$* as:

$$\mathrm{wsize}(f, P) = \max_{x \in X} \{\mathrm{wsize}(x, P)\}. \tag{3.6}$$

In [Rei09], Reichardt defines the witnesses with costs associated to evaluating different inputs. This can be easily incorporated in this definition and all the following by changing the norm in which the witness size is evaluated from $\|\cdot\|^2$ to $\|S \cdot\|^2$ where $S$ is a matrix of weights diagonal in the computational basis of $\mathbb{C}^I$ and constant among all indices $j$ consistent with a given coordinate $i$, i.e. all $j \in \bigsqcup_{b \in [q]} I_{i,b}$. That said, we will forget costs of inputs and assume that all inputs are uniform in cost for the rest of this thesis.

At this point we want to make a few key observations to motivate the next definition.

1. First, it looks like $\mathbb{C}^I$ is more important than $\mathcal{V}$ and that it supersedes $I$ itself.

2. Any two legs of the trio $\langle$input set – input vectors – $A : \mathbb{C}^I \to \mathcal{V}\rangle$ determines the third.

3. According to these definitions, $\mathcal{V}$ need not be an inner-product space, just a vector space. In fact, [Rei09, Lemma 4.12] says that the span program complexity is invariant under any linear transformations of $|\tau\rangle$ and $\{|v_i\rangle\}_{i \in I}$. Therefore, the norms of $|\tau\rangle$ and $\{|v_i\rangle\}_{i \in I}$ do not matter, only their relation to each other, and the norm of vectors in $\mathbb{C}^I$ (which remains invariant under transformations of $\mathcal{V}$), matter.

### 3.2.2 An alternative definition of span programs

We have talked about the caveats of Definition 20, mainly that the importance of the input space $\mathbb{C}^I$ is understated and the importance of $\mathcal{V}$ is overstated. In that definition, the input vectors $\{|v_j\rangle\}_{j \in I}$ are the ones that are chosen by the user (along with the target), while the index set is derived from the input vectors. This can create confusion because it is not in $\mathcal{V}$ but in $\mathbb{C}^I$ where the witnesses live and their norm is computed. The emphásis

is on the wrong sylláble. In this setting, the map $A$ is entirely descriptive, i.e. it is determined by the choice of input vectors. A better definition for span programs would be one that puts the input space front and center. In this new definition, the map $A$ becomes normative, meaning that the user chooses the index vectors $|j\rangle$ and the map $A$, and the vectors $\{|v_j\rangle\}_j$ become determined by these two things.

In this section we follow [IJ19] and redefine span programs by introducing an inner product space $\mathcal{H}$ that takes the role of $\mathbb{C}^I$ and a map $A : \mathcal{H} \to \mathcal{V}$ that replaces the choice of input states $\{|v_j\rangle\}_{j\in I}$. The advantage of this change of perspective is that it gives us tools to better study the anatomy of witnesses and define approximate versions of them. This is crucial in Chapter 4 when we describe span programs for two-sided, bounded-error algorithms. Additionally, the shift from input vectors to span program map $A$ has the advantage that linear maps are more structured than sets. This will become obvious later when we describe algorithms for span programs and their time complexity, where the span program map $A$ and its kernel will play a crucial role.

Following [IJ19], we define a span program for evaluating a decision problem as follows.

**Definition 22** (Span program)**.** A *span program* $P = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$ on $[q]^n$ consists of:

1. A finite-dimensional inner-product space $\mathcal{H}$ that decomposes as
$$\mathcal{H} = \mathcal{H}_1 \oplus \cdots \oplus \mathcal{H}_n \oplus \mathcal{H}_{\text{true}} \oplus \mathcal{H}_{\text{false}},$$
where each $\mathcal{H}_i$, $i \in [n]$, decomposes further as $\mathcal{H}_i = \sum_{b\in[q]} \mathcal{H}_{i,b}$.
2. A vector space $\mathcal{V}$.
3. A linear operator $A \in \mathcal{L}(\mathcal{H}, \mathcal{V})$.
4. A target vector $|\tau\rangle \in \mathcal{V}$.

With each string $x \in \{0,1\}^n$, we associate the subspace
$$\mathcal{H}(x) = \mathcal{H}_{1,x_1} \oplus \cdots \oplus \mathcal{H}_{n,x_n} \oplus \mathcal{H}_{\text{true}}.$$

This definition is equivalent to that of Section 3.2.1 except for one thing. By demanding that $\mathcal{H}_i = \sum_{b\in[q]} \mathcal{H}_{i,b}$ rather than $\mathcal{H}_i = \bigoplus_{b\in[q]} \mathcal{H}_{i,b}$ we save a factor of $\log q$ in the correspondence between query complexity and span

program complexity, see [Jef14].

Intuitively, a span program encodes the question "Is $|\tau\rangle \in A(\mathcal{H}(x))$?". To answer this question in the affirmative, it is sufficient to provide a preimage of $|\tau\rangle$ under $A$ in $\mathcal{H}(x)$, called a *positive witness*. In the negative case, one would like to find an object, called a *negative witness*, that precludes the existence of such a positive witness. In the previous definition, $f_P(x) = 1$ iff $|\tau\rangle$ is a combination of available vectors, and one would expect the positive witness that certifies this to be that combination of available vectors, but it is not. Positive witnesses are defined as linear combinations of indices in $j \in I(x)$ whose vectors $|v_j\rangle$ sum up to $|\tau\rangle$. Confusing indeed. The present formulation of span program addresses that confusion. We formally define witnesses and witness sizes in [IJ19] notation as:

**Definition 23** (Positive and negative witnesses)**.** Fix a span program $P = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$ and an input $x \in [q]^n$. We call a vector $|w\rangle \in \mathcal{H}$ a *positive witness* for $x$ if $|w\rangle \in \mathcal{H}(x)$, and $A|w\rangle = |\tau\rangle$. If there exists a positive witness for $x$, the *positive witness size* of $x$ is

$$w_+(x, P) = w_+(x) := \min_{|w\rangle \in \mathcal{H}(x)} \{\||w\rangle\|^2 : A|w\rangle = |\tau\rangle\},$$

and $w_+(x) = \infty$ otherwise. We say that $|\omega\rangle \in \mathcal{V}$ is a *negative witness* for $x$ if $\langle\omega|A\Pi_{\mathcal{H}(x)} = 0$ and $\langle\omega|\tau\rangle = 1$. If there exists a negative witness, the *negative witness size* of $x$ is

$$w_-(x, P) = w_-(x) := \min_{|\omega\rangle \in \mathcal{V}} \{\|\langle\omega|A\|^2 : \langle\omega|A\Pi_{\mathcal{H}(x)} = 0, \langle\omega|\tau\rangle = 1\},$$

and $w_-(x) = \infty$ otherwise.

This definition is the analogue of Definition 20 restated in the new formalism and in greater detail. We define the set of *positive* and *negative inputs* of $P$, respectively, as:

$$P_1 := \{x \in [q]^n : w_+(x) < \infty\}, \qquad P_0 := \{x \in [q]^n : w_-(x) < \infty\}.$$

It can be shown that every string $x$ has either a positive or a negative witness, but never both, hence $P_0 \cup P_1 = [q]^n$ and $P_0 \cap P_1 = \emptyset$. Therefore, the span program $P$ decides the total function

$$f_P(x) = \begin{cases} 1 & \text{if} \quad x \in P_1 \\ 0 & \text{if} \quad x \in P_0 \end{cases}.$$

This is the same function as the one defined in Equation (3.1). As we discussed in the previous section, any partial function $f : X \subseteq [q]^n \to \{0,1\}$ such that $f(x) = f_P(x)$ for all $x \in X$ is also decided by $P$.

**Definition 24** (Span program complexity)**.** Let $P = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$ be a span program on $[q]^n$, and let $f : X \to \{0,1\}$ be a function decided by $P$. Let $W_-(f,P) := \max_{x \in f^{-1}(0)} w_-(x,P)$ be the *negative complexity of* $P$, and let $W_+(f,P) := \max_{x \in f^{-1}(1)} w_+(x,P)$ be the *positive complexity of* $P$ . Then, the *span program complexity* of $P$ with respect to $f$ is defined as:

$$C(f,P) = \sqrt{W_-(f,P)W_+(f,P)}.$$

It is not obvious that this quantity is the same as the span program complexity $\mathrm{wsize}(f,P)$ from Equation (3.6). There, the complexity is defined as

$$\mathrm{wsize}(f,P) = \max_{x \in X}\{\mathrm{wsize}(f,P)\} = \max\{W_-(f,P),\, W_+(f,P)\}.$$

Strictly speaking, $\mathrm{wsize}(f,P) \geq C(f,P)$, but we can modify the span program $P$ in either formalism by rescaling the target $|\tau\rangle \mapsto \alpha|\tau\rangle$. This multiplies all positive witnesses sizes by $\alpha^2$ and all negative witness sizes by $1/\alpha^2$. Choosing $\alpha = (W_-(f,P)/W_+(f,P))^{1/4}$ we obtain

$$\mathrm{wsize}(f,P') = \max\{\alpha^2 W_+(f,P),\, 1/\alpha^2 W_-(f,P)\} = \sqrt{W_-(f,P)W_+(f,P)}.$$

This transformation doesn't affect the class of functions decided by $P$, nor the new span program complexity $C(f,P)$. Hence, choosing the right scaling of $|\tau\rangle$ (which we can do because norms in $\mathcal{V}$ don't matter) renders both definitions of span program complexity equal. From this point on we will stick to $C(f,P) = \sqrt{W_-W_+}$.

The concept of witness is qualitatively similar to that of a *certificate*, in the sense that its existence is proof that the instance is either positive or negative, and the bigger the witness that certifies that, the harder it is to find. This justifies the notion that the larger the witness sizes, the bigger the span program complexity is. But why do we define the span program complexity in this way? Why the square root? Why not $W_- + W_+$?

The reason is (and we will come back to this again and again) that the smallest span program complexity $C(f,P)$ out of all span programs $P$ that decide $f$ equals the quantum query complexity of $f$, up to constant factors. That is because every span program corresponds to a dual adversary solution (See Section 3.2.3), and for every span program $P$ we can construct

a quantum algorithm with query complexity $\mathcal{O}(C(f,P))$ (and we will, in Section 3.4).

**Approximate span programs** As [IJ19] illustrates, it can be advantageous to relax the constraints in Definition 23 and simply require that the target be sufficiently close to $A(\mathcal{H}(x))$ for an instance $x$ to be considered positive, or so close to having a valid negative witness that the input is considered negative. Since there is no meaningful distance in $\mathcal{V}$, the notion of closeness has to be defined in $\mathcal{H}$. The definitions we give of approximate witnesses come from [Jef20] and improve on the original ones given in [IJ19]. We defer their in-depth discussion to Section 3.3.

**Definition 25** (Approximate positive witness size [Jef20])**.** For any span program $P$ on $[q]^n$ and $x \in [q]^n$, we define the $\lambda$-approximate positive witness size of $f$ as

$$\widetilde{w}_+(x,f) = \min\left\{ \||w\rangle\|^2 : A|w\rangle = |\tau\rangle, \left\|\Pi_{\mathcal{H}(x)^\perp}|w\rangle\right\|^2 \leq \frac{\lambda}{W_-(f,P)} \right\}.$$

If the set over which we minimize is empty, we say that $\widetilde{w}_+(x,f) = \infty$.

Any vector yielding a solution to this minimization problem is called an *approximate positive witness*, where $\lambda$ is implicit. Note that if $\lambda = 0$ we recover the usual definition of positive witness size, and that if $\left\|\Pi_{\mathcal{H}(x)^\perp}|w\rangle\right\| > 0$ for every positive witness $|w\rangle$, there exists an exact negative witness.

The approximate negative witnesses are defined in a somewhat similar fashion by relaxing the requirement that the witnesses be orthogonal to $\mathcal{H}(x)$.

**Definition 26** (Approximate negative witness size [Jef20])**.** For any span program $P$ on $[q]^n$ and $x \in [q]^n$, we define the $\lambda$-approximate negative witness size as

$$\widetilde{w}_-(x,P) = \min\left\{ \|\langle\omega|A\|^2 : \langle\omega|\tau\rangle = 1, \left\|\langle\omega|A\Pi_{\mathcal{H}(x)}\right\|^2 \leq \frac{\lambda}{W_+(f,P)} \right\}.$$

If the set over which we minimize is empty, we say that $\widetilde{w}_-(x,f) = \infty$.

Like before, we call a vector solving this minimization an *approximate negative witness*. If $\left\|\langle\omega|A\Pi_{\mathcal{H}(x)}\right\| > 0$ for all approximate negative witnesses $\langle\omega|$, there exists an exact positive witness.

These relaxations of the notions of positive and negative witness give rise to the concept of *approximate span programs*, which allow a span program to decide a broader class of functions, (see Definition 39). This does not mean that approximate span programs are a more *powerful* model of computation. Every two-outcome function $f$ admits an exact (as in non-approximate) span program deciding it that can be turned to an algorithm with optimal query complexity. But extending the range of functions that a span program decides can (and will) be very useful. We define these things formally in Section 3.3.3.

**Minimal witness and normalization** The notion of positive and negative witnesses need not be tied to specific inputs $x \in [q]^n$. For a given span program $P = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$, a positive witness is any vector $|w\rangle \in \mathcal{H}$ such that $A|w\rangle = |\tau\rangle$. Moreover, assuming that a positive witnesses $|w\rangle$ exists, it is a simple exercise to show that the set $W$ of positive witnesses is exactly

$$W = \{|w\rangle + |h\rangle : |h\rangle \in \ker A\}.$$

Let $A^+$ be the Moore-Penrose pseudo-inverse of $A$. Then $A^+|\tau\rangle$ is the unique shortest vector in $W$ and the only one orthogonal to $\ker A$ (another simple exercise). Therefore, all witnesses are actually of the form $|w\rangle = A^+|\tau\rangle + |h\rangle$ for some $|h\rangle \in \ker A$. This vector will be crucial in our analysis of span programs, reflection programs and the algorithms for them.

**Definition 27** (Minimal witness, [IJ19])**.** Let $P = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$ be any span program. We define the *minimal positive witness* $|w_0\rangle$ to be the unique shortest positive witness in $P$ for $|\tau\rangle$. That is, $|w_0\rangle = A^+|\tau\rangle$. We define the *minimal witness size* to be $\||w_0\rangle\|^2$.

A span program is *normalized* if $\||w_0\rangle\|^2 = 1$. It is possible to scale all positive witnesses by a factor of $\alpha$ by redefining the target to be $|\tau'\rangle = \alpha|\tau\rangle$. This would also scale all negative witnesses by a factor $1/\alpha$, which leaves the span program complexity $\sqrt{W_+W_-}$ invariant. This way we can normalize any span program by redefining $|\tau'\rangle = |\tau\rangle/\||w_0\rangle\|^2$. However, it is also possible to normalize and scale a span program independently (see [IJ19, Theorem 2.14]). This is necessary for the algorithms presented there but will not play any role in the algorithms we define in Section 3.4.

**Span programs and quantum algorithms** Span programs are confusing. They encode a function $f$ in terms of a geometric question "Is $|\tau\rangle$ in

$A(\mathcal{H}(x))$?", but their complexity also corresponds to the query complexity of an algorithm that decides $f$. Moreover, the smallest complexity corresponds to the query complexity of $f$. *"That's a lot to process, please explain".*

When we say that we can construct an algorithm for a function $f$ out of a span program that decides $f$ with complexity $\mathcal{O}(C(f,P))$, we mean that we can use the parts of $P = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$ to make a quantum algorithm. It works like this.

Recall that all positive witnesses are of the form $|w\rangle = |w_0\rangle + |h\rangle$, including positive witnesses $|w_x\rangle$ for $x \in f^{-1}(1)$, which are of that form and such that $|w_x\rangle \in \mathcal{H}(x)$. In other words, if $f(x) = 1$, $|w_0\rangle = |w_x\rangle - |h\rangle$ where $|w_x\rangle \in \mathcal{H}(x)$ and $|h\rangle \in \ker A$. In fact we can show that $|w_0\rangle \in \mathcal{H}(x) + \ker A$ iff $f(x) = 1$. If $f(x) = 0$, then $|w_0\rangle \in \mathcal{H}^\perp(x) \cap (\ker A)^\perp$. So the crux of the algorithm is in distinguishing $|w_0\rangle \in \mathcal{H}(x) + \ker A$ from $|w_0\rangle \in \mathcal{H}^\perp(x) \cap (\ker A)^\perp$.

All span program algorithms make queries to a unitary $U(x,P) = (2\Pi_{\ker A} - I)(2\Pi_{\mathcal{H}(x)} - I)$ by running phase estimation and/or amplitude amplification on $U(x,P)$ with initial state $|w_0\rangle$ (or some variants thereof) to distinguish those two cases. The number of calls to $U(x,P)$ being (drumrolls) $\mathcal{O}(C(f,P))$. Moreover, this unitary can be implemented with a single quantum query to $x$.

To show that all this is true and works and extends to approximate span programs, we present an in-depth analysis of the geometrical meaning of witnesses, witness sizes and approximate witnesses in Section 3.3 in the context of reflection programs. This will be followed by an algorithm that decides any approximate reflection program.

In the next section we explain the connection between span programs and dual adversary solutions (whatever they are), and why that implies that the smallest possible span program complexity of a function $f$ equals its query complexity, up to constant factors. We then give a brief exposition of span program algorithms in the literature before diving into reflection programs.

### 3.2.3 A few algorithms for span programs

As we have started to see, and will become clear in Section 3.3.2, span programs are a natural way of expressing decision problems in terms of linear algebra. This model of computation has evolved over time to include non-Boolean input alphabets and alternative notations. We have already said that span programs give rise to algorithms and that they correspond to *dual adversary solutions*. Let's explain the second claim first and then focus our

attention to how span programs have been turned into algorithms in the literature.

The adversary bound $ADV^{\pm}(f)$ [Amb02; HLS07] is a semi-definite program (SDP) whose feasible solutions give lower bounds for the quantum query complexity of a function $f : X \in [q]^n \to \{0,1\}$ over finite alphabet $[q]$. Its dual is the SDP:

$$\begin{aligned} ADV^{\pm}(f) = \min \quad & \max_{z \in X} \sum_{j \in [n]} \langle \psi_{j,z} | \psi_{j,z} \rangle \\ \text{s.t.} \quad & \sum_{j : x_j \neq y_j} \langle \psi_{j,x} | \psi_{j,y} \rangle = 1 \qquad \forall x \in f^{-1}(1), y \in f^{-1}(0), \\ & |\psi_{j,z}\rangle \in \mathbb{C}^m \quad \text{for some } m \in \mathbb{N}, \forall j \in [n], z \in X \subseteq [q]^n. \end{aligned} \tag{3.7}$$

Each set $\{|\psi_{j,z}\rangle\}_{j,z}$ satisfying the constraints of Eq. (3.7) is called a *dual adversary solution* and exactly corresponds to a span program $P$ that decides $f$ in what is known as *canonical* form. Moreover, the objective value of the dual adversary solution, $\max_{z \in D} \sum_{j \in [n]} \langle \psi_{j,z} | \psi_{j,z} \rangle$, equals the span program complexity for $f$, $C(f,P)$. See [Rei09] or [Chi21] for a proof of this fact.

This is a very profound result in and of itself. Yet, the most important feature of span programs, the one that justifies our interest, is not that a span program encodes a decision problem, or a dual adversary solution. The most important feature of span programs is that they can be used to compile quantum algorithms for $f$ with query complexity $\mathcal{O}(C(f,P))$.

This was first observed by Reichardt [Rei09], who used this fact to show that the adversary lower bound actually characterizes quantum query complexity. It was already known that feasible solutions to the primal semi-definite program were lower bounds on the query complexity of a Boolean function $f$. Using the correspondence between dual adversary solutions, span programs, and quantum algorithms, and the strong duality of the Adversary Lower Bound, Reichardt showed that dual adversary solutions *were* algorithms, and that the optimal solution to the primal and the dual was a tight lower bound.

Among other things, this means that the quantum query complexity of a function $f$ is the minimum complexity of any span program computing $f$.

We show what the span program looks like for functions $f$ with binary inputs in our second notation. For non-binary alphabets, we refer to the generalization in [Jef14, Section 7.1]. Let $\{|\psi_{j,z}\rangle\}_{j,z}$ be a dual adversary

solution that satisfies the constraints of Equation (3.7) with objective value $W$, and $F_0 = f^{-1}(0)$. We define the span program $P = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$ as:

$$\mathcal{H} = \text{span}\{|i, b, z\rangle : i \in [n] \times \{0,1\} \times [m]\}, \quad \mathcal{V} = \mathbb{C}^{F_0}$$
$$A = \sum_{\substack{x \in F_0 \\ j \in [m]}} |x\rangle\langle j, \bar{x}_j|\langle\psi_{j,x}|, \quad |\tau\rangle = \frac{1}{3\sqrt{W}} \sum_{x \in F_0} |x\rangle.$$

Then, it is shown in [Rei09] that $P$ decides $f$ with span program complexity $\mathcal{O}(W)$. Several algorithms are possible now [Rei10], but what they all have in common is that they make $\mathcal{O}(W)$ queries to a unitary $U_x = (2\Pi_x - I)(2\Lambda - I)$, for some projectors $\Pi_x$ and $\Lambda$. The first reflection, $(2\Pi_x - I)$, is a reflection around the subspace of available indices for $x$, $\text{span}\{|i, x_i\rangle\} \otimes \mathbb{C}^m$, and requires only two queries to the input oracle $\mathcal{O}_x$, while the other reflection space is related to the Kernel of the map $A$ and is input independent.

In their endeavour to clarify and generalize span programs, Ito and Jeffery introduced in [IJ19] a variant algorithm to evaluate approximate span programs. Their algorithm performs phase estimation of the unitary[1] $U(x, P) := (2\Pi_{\ker A} - I)(2\Pi_{\mathcal{H}(x)} - I)$ with initial state $|w_0\rangle$, and then estimates the amplitude on a 0 phase to distinguish positive from negative inputs. At first, their algorithm only works for *normalized* span programs and has complexity $\widetilde{\mathcal{O}}\left(\frac{W_-(f,P)}{(1-\lambda)^{3/2}}\sqrt{\widehat{W}_+(f,P)}\right)$, where $\widehat{W}_+(f,P)$ is the approximate positive witness complexity of $f$, and $\lambda$ is the approximation factor (see Equation (3.16) and Definition 39). So they supplement it with a transformation that simultaneously normalizes a span program and scales its witnesses to obtain:

**Lemma 28** ([IJ19], Corollary 3.7)**.** *Let $P$ be a $\lambda$-approximating span program for $f : X \subseteq [q]^n \to \{0,1\}$. Then the quantum query complexity of $f$ is $\mathcal{O}\left(\frac{1}{(1-\lambda)^{3/2}}\sqrt{W_-(f,P)\widetilde{W}_+(f,P)}\log\frac{1}{1-\lambda}\right)$. In particular, if $P$ is an exact span program for $f$, then the quantum query complexity of $f$ is at most $\mathcal{O}\left(\sqrt{W_-W_+}\right)$.*

[1] $U(x,P)$ and $U_x$ are not the same unitary, although they are intimately related.

In Section 3.4 we present a generalization of this algorithm for reflection programs that bypasses the scaling procedure by baking it into the algorithm.

Using standard algorithmic techniques, Ito and Jeffery were also able to modify their algorithm for decision problems and turn it into an algorithm that estimates the positive or negative witness size of an input. We will need this result later in Section 5.3.1, so we might as well introduce it now. The algorithm is based on the direct connection between the minimal witness $|w_0\rangle$, the 0 phase eigenspace of $U(x,P)$ and the negative witness size that we will discuss in Section 3.3.2.

**Theorem 29** ([IJ19], Theorem 2.8)**.** *Let $U(x,P) = (2\Pi_{\ker A} - I)(2\Pi_{\mathcal{H}(x)} - I)$. Fix $X \subseteq [q]^n$ and $f : X \to \mathbb{R}_{\geq 0}$. Let $P$ be a span program on $[q]^n$ such that for all $x \in X$, $f(x) = w_-(x,P)$ and define $\widetilde{W}_+ = \widetilde{W}_+(P) = \max_{x\in X} \widetilde{w}_+(x,P)$. Then there exists a quantum algorithm that estimates $f$ to accuracy $\varepsilon$ and that uses $\widetilde{\mathcal{O}}\left(\frac{1}{\varepsilon^{3/2}}\sqrt{w_-(x)\widetilde{W}_+}\right)$ queries.*

*Similarly, let $P$ be a span program such that for all $x \in X$, $f(x) = w_+(x,P)$, and define $\widetilde{W}_- = \widetilde{W}_-(P) = \max_{x\in X} \widetilde{w}_-(x,P)$. Then there exists a quantum algorithm that estimates $f$ to accuracy $\varepsilon$ and uses $\widetilde{\mathcal{O}}\left(\frac{1}{\varepsilon^{3/2}}\sqrt{w_+(x)\widetilde{W}_-}\right)$ queries.*

## 3.3 Reflection programs

In this section we present an abstraction of span programs, called *reflection programs*, that constitutes our first original contribution to the field. Reflection programs are a strict generalization of span programs. They originate from the authors' desire to strip a span program of its non-algorithmic parts so that we can build intuition towards constructing an algorithm.

First, we define reflection programs and witnesses in Section 3.3.1 and give geometrical and operational interpretations in Section 3.3.2. Then we define approximate witnesses in Section 3.3.3 and give an intuition on reflection programs approximating functions before deriving some important characterizations of approximate and exact witnesses.

We top off the section with a transformation that turns approximate and exact positive witnesses into negative ones and vice versa, and comment on the connection between reflection and span programs.

### 3.3.1 Definitions

We've already motivated reflection programs, so let's define them.

**Definition 30** (Reflection program)**.** Fix $X \subseteq [q]^n$. A reflection program $\mathcal{R} = (\mathcal{H}, \{\mathcal{H}(x)\}_{x\in X}, \mathcal{K}, |w_0\rangle)$ on $X$ consists of:

1. A finite-dimensional inner-product space $\mathcal{H}$.
2. A subspace $\mathcal{H}(x) \subseteq \mathcal{H}$ for every $x \in X$.
3. A subspace $\mathcal{K} \subseteq \mathcal{H}$.
4. A unit vector $|w_0\rangle \in \mathcal{K}^\perp$.

Moreover, we say that $\mathcal{R}$ *evaluates a function* $f : X \to \{0, 1\}$, if

$$f(x) = 1 \Leftrightarrow |w_0\rangle \in \mathcal{K} + \mathcal{H}(x).$$

At first it seems like most elements in this definition have changed with respect to Definition 22. This is not as dramatic a change as it seems. We have kept the space $\mathcal{H}$ but disposed of the space $\mathcal{V}$, the target $|\tau\rangle$ and the map $A$. That is because we don't need them to make an algorithm. All the algorithms we will present, and all algorithms in the literature, operate on $\mathcal{H}$ and only ever use reflections around $\mathcal{H}(x)$, $\ker A$ and the state $|w_0\rangle$. As we will shortly see, the relations between $|\tau\rangle$, $A$ and the witnesses in Definition 23 can be lifted to relations exclusively in $\mathcal{H}$ between $\mathcal{H}(x)$, $\ker A$, $|w_0\rangle$ and appropriately defined witnesses.

For this reason we have substituted $\ker A$ by $\mathcal{K}$ and $|\tau\rangle$ by $|w_0\rangle$ in the definitions.

We have also generalized the way in which the spaces $\mathcal{H}(x)$ sit inside $\mathcal{H}$ and left this map $x \mapsto \mathcal{H}(x)$ as a component of the reflection program chosen by the user. Of course, this map is also defined for span programs in Definition 22, but we don't include it as a component to the span program because it follows from the decomposition of $\mathcal{H}$ there. Last but not least, we now define reflection programs over domains $X \subseteq [q]^n$, rather than all $[q]^n$. That is because we have lifted any and all restrictions on how $\mathcal{H}(x)$ sits in $\mathcal{H}$. It might be that we can define these spaces for all $x \in X$ without having to determine what $\mathcal{H}(x)$ is for $x \in [q]^n \setminus X$. In span programs, $\mathcal{H}(x)$ is constructed in such a way that we can reflect around it with one quantum query to $x$. In reflection programs we loose the straightforward relation between query complexity and reflection program complexity.

We have restricted ourselves to subsets $X \subseteq [q]^n$ out of laziness but there is nothing stopping $X$ from being any finite set. It is even possible that under certain conditions, $X$ could be an infinite set[2] and our proofs would still hold. Would that be a way to make quantum algorithms deciding functions with continuous input and binary output? That is a very interesting open question.

Just like in the case of span programs, we can give definitions for positive and negative witnesses. These positive and negative witnesses have very clear geometric interpretations, which lead to a clearer understanding of the reflection program algorithm, and thus, the span program algorithm, which is a special case.

**Definition 31** (Positive and negative reflection witnesses)**.** Fix an arbitrary $x \in X$.

1. A *positive witness* is a vector $|w_x\rangle \in \mathcal{H}(x)$, such that $|w_0\rangle - |w_x\rangle \in \mathcal{K}$. Such a vector exists if and only if $|w_0\rangle \in \mathcal{K} + \mathcal{H}(x)$.
   We define the positive witness size of $x$ as:
   $$w_+(x, \mathcal{R}) = \min\{\||w_x\rangle\|^2 : |w_x\rangle \in \mathcal{H}(x), |w_0\rangle - |w\rangle \in \mathcal{K}\}.$$

2. A *negative witness* is a vector $|\omega_x\rangle \in \mathcal{H}(x)^\perp \cap \mathcal{K}^\perp$, with the property that $\langle \omega_x | w_0 \rangle = 1$. Such a vector exists if and only if $|w_0\rangle \notin \mathcal{K} + \mathcal{H}(x)$.
   We define the negative witness size of $x$ as:
   $$w_-(x, \mathcal{R}) = \min\{\||\omega\rangle\|^2 : |\omega_x\rangle \in \mathcal{H}(x)^\perp \cap \mathcal{K}^\perp, \langle \omega | w_0 \rangle = 1\}.$$

As was the case with span programs, $\mathcal{R}$ splits $X$ into two parts:

$$P_1 = \{x \in X : |w_0\rangle \in \mathcal{K} + \mathcal{H}(x)\} \qquad P_0 = \{x \in X : |w_0\rangle \notin \mathcal{K} + \mathcal{H}(x)\}.$$

$\mathcal{R}$ decides any function $f : \mathcal{D} \subseteq X \to \{0, 1\}$ such that $f^{-1}(0) \subseteq P_0$ and $f^{-1}(1) \subseteq P_1$. We define the *positive* and *negative complexity* of $\mathcal{R}$ as:

$$W_+(\mathcal{R}) = \max_{x \in P_1} w_+(x, \mathcal{R}) \qquad \text{and} \qquad W_-(\mathcal{R}) = \max_{x \in P_0} w_-(x, \mathcal{R}), \tag{3.8}$$

[2] My money is on Compact Hausdorff. It's always Compact Hausdorff.

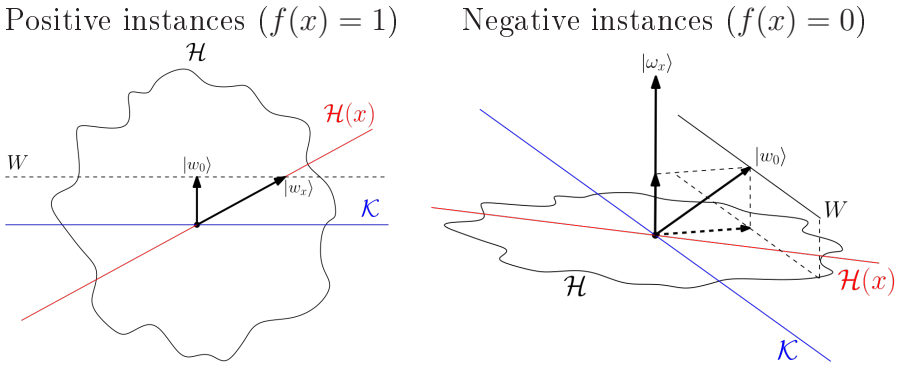



Figure 3.1: Representation of the subspace generated by $\mathcal{H}(x)$ and $\mathcal{K}$, the space of positive witnesses $W$, and the relevant vectors $|w_0\rangle$, $|w_x\rangle$ and $|\omega_x\rangle$.

and we define the *positive* and *negative reflection program complexities* of $f$ as:

$$W_+(f,\mathcal{R}) = \max_{x\in f^{-1}(1)} w_+(x,\mathcal{R}) \qquad \text{and} \qquad W_-(f,\mathcal{R}) = \max_{x\in f^{-1}(0)} w_-(x,\mathcal{R}). \tag{3.9}$$

In Figure 3.1 we can see visualizations of positive and negative witnesses. They seem to suggest that the positive and negative witnesses can be seen as a vector proportional to the projection of $|w_0\rangle$ on $\mathcal{H}(x)$ and $\mathcal{K}^\perp \cap \mathcal{H}(x)^\perp$ respectively. Such claim is not true for the positive witness (as we shall see in the next section), because representing $\mathcal{K}$ and $\mathcal{H}(x)$ as 1-dimensional spaces oversimplifies the problem. Our intuition of negative witnesses, on the other hand, turns out to be true, and gives rise to the following characterization of the negative witnesses and negative witness size.

**Claim 1.** Let $|\omega_x\rangle$ be a minimal-size negative witness for $x \in X$. Then $|\omega_x\rangle = \frac{\Pi_{\mathcal{K}^\perp\cap\mathcal{H}(x)^\perp}|w_0\rangle}{\|\Pi_{\mathcal{K}^\perp\cap\mathcal{H}(x)^\perp}|w_0\rangle\|^2}$ and $w_-(x,\mathcal{R}) = \|\Pi_{\mathcal{K}^\perp\cap\mathcal{H}(x)^\perp}|w_0\rangle\|^{-2}$.

*Proof.* Let $|\omega\rangle \in \mathcal{H}(x)^\perp \cap \mathcal{K}^\perp$ be a negative witness. Since $1 = \langle\omega|w_0\rangle = \langle\omega|\Pi_{\mathcal{H}(x)^\perp\cap\mathcal{K}^\perp}|w_0\rangle$, any component of $\omega$ that is not parallel to $\Pi_{\mathcal{H}(x)^\perp\cap\mathcal{K}^\perp}|w_0\rangle$ does not contribute to making $\langle\omega|w_0\rangle = 1$. We conclude that the optimal negative witness must be parallel to $\Pi_{\mathcal{H}(x)^\perp\cap\mathcal{K}^\perp}|w_0\rangle$. Its norm is then determined by the condition $\langle\omega|w_0\rangle = 1$. □

### 3.3.2 Operational interpretations

Together with the geometric interpretation of witnesses, we also give an operational interpretation based on what we call the *reflection program unitary*. This is a unitary operator similar to the span program unitary in [IJ19]. Let $\mathcal{R}$ be a reflection program over $X \subseteq [q]^n$. For every $x \in X$, we define:

$$U(x, \mathcal{R}) = \left(2\Pi_{\mathcal{H}(x)} - I\right)\left(2\Pi_{\mathcal{K}} - I\right). \tag{3.10}$$

Any unitary $U(x, \mathcal{R})$ has a eigenvalue decomposition:

$$U(x, \mathcal{R}) = \sum_{j=1}^{d} e^{\pm i\varphi_j} |\varphi_j\rangle\langle\varphi_j|,$$

for some $d$ and some states $|\varphi_j\rangle$. By Jordan's Lemma (Lemma 1), a unitary operator that is a product of two reflections naturally splits $\mathcal{H}$ into three different parts:

1. The $(+1)$-eigenspace, or 0-phase space:

$$E_0 = (\mathcal{K} \cap \mathcal{H}(x)) \oplus (\mathcal{K}^\perp \cap \mathcal{H}(x)^\perp).$$

2. The $(-1)$-eigenspace, or $\pi$-phase space:

$$E_\pi = (\mathcal{K} \cap \mathcal{H}(x)^\perp) \oplus (\mathcal{K}^\perp \cap \mathcal{H}(x)).$$

3. Everything else, which we refer to as the remaining space: $E_{\text{rem}} := \text{span}\{|\varphi\rangle : U(x, \mathcal{R})|\varphi\rangle = e^{i\varphi}|\varphi\rangle, \varphi \neq 0, \pi\}$.

Also from Jordan's lemma follows that $E_{\text{rem}}$ can be completely decomposed into two-dimensional subspaces, on which $U(x, \mathcal{R})$ acts as a rotation operator. For every eigenphase $\varphi \in (0, \pi)$, then all these two-dimensional subspaces are of the form $E_\varphi := E_{\varphi^+} \oplus E_{-\varphi^-}$ as defined in Section 2.1.1, and the angle of rotation in these subspaces is $\varphi$. We refer the reader to Lemmas 1 and 3 for further detail on the definitions of these eigenvectors and spaces.

We let $\Phi$ be the random variable whose probability distribution is given by the outcome distribution of the phase estimation algorithm of Theorem 9, if we were to run it with $U(x, \mathcal{R})$ on $|w_0\rangle$, with infinite precision. In other words,

$$\mathbb{P}(\Phi = \varphi) = \left\|\Pi_{E_\varphi}|w_0\rangle\right\|^2. \tag{3.11}$$

Let $x \in f^{-1}(0)$, by Claim 1 we readily find:

$$w_-(x,\mathcal{R})^{-1} = \left\|\Pi_{\mathcal{K}^\perp \cap \mathcal{H}(x)^\perp}|w_0\rangle\right\|^2 = \|\Pi_{E_0}|w_0\rangle\|^2 = \mathbb{P}\left(\Phi = 0\right).$$

The operational interpretation of $w_+(x,\mathcal{R})$ will require a bit more work and will depend critically on the results of Lemma 3 and Corollary 4, which imply that $\mathcal{H}(x)$ and $\mathcal{K}$ decompose nicely into the $(+1)$ and $(-1)$-eigenspaces of $U(x,\mathcal{R})$ and the 2-D spaces $E_{\varphi_j}$. For convenience, we restate them now without proof and rewritten in the language of reflection programs.

**Lemma 32.** *Let $\varphi_j > 0$, $j \in [d]$ for some $d$, be the positive eigenphases of $U(x,\mathcal{R})$ in $(0,\pi)$, $E_\varphi := E_{\varphi^+} \oplus E_{\varphi^-}$ be the 2-dimensional spaces spanned by $(\pm\varphi)$-eigenphase eigenvectors of $U(x,\mathcal{R})$, and let $E_0$, $E_\pi$ be the 0-phase and $\pi$-phase eigenspaces of $U(x,\mathcal{R})$. Then,*

$$\mathcal{H}(x) = (\mathcal{H}(x) \cap E_0) \bigoplus_{j=1}^{d} \left(\mathcal{H}(x) \cap E_{\varphi_j}\right) \oplus (\mathcal{H}(x) \cap E_\pi)$$

**Corollary 33.** *Let $\varphi_j > 0$, $j \in [d]$ for some $d$, be the positive eigenphases of $U(x,\mathcal{R})$ in $(0,\pi)$, $E_\varphi := E_{\varphi^+} \oplus E_{\varphi^-}$ be the 2-dimensional spaces spanned by $(\pm\varphi)$-eigenphase eigenvectors of $U(x,\mathcal{R})$, and let $E_0$, $E_\pi$ be the 0-phase and $\pi$-phase eigenspaces of $U(x,\mathcal{R})$. Then,*

$$\mathcal{K} = (\mathcal{K} \cap E_0) \bigoplus_{j=1}^{d} \left(\mathcal{K} \cap E_{\varphi_j}\right) \oplus (\mathcal{K} \cap E_\pi),$$

*and*

$$\mathcal{K}^\perp = \left(\mathcal{K}^\perp \cap E_0\right) \bigoplus_{j=1}^{d} \left(\mathcal{K}^\perp \cap E_{\varphi_j}\right) \oplus \left(\mathcal{K}^\perp \cap E_\pi\right).$$

With these, we are ready to give an operational characterization of the positive witness both in terms of the eigenphases of $U(x,\mathcal{R})$, and in geometrical terms.

**Lemma 34.** *Let $\mathcal{R}$ be a reflection program on $X \subseteq [q]^n$, and $x$ a positive instance of $\mathcal{R}$, that is, $|w_0\rangle \in \mathcal{K} + \mathcal{H}(x)$. Let $\Phi$ be the random variable distributed according to Equation (3.11). Then,*

$$w_+(x,\mathcal{R}) = \mathbb{E}\left[\frac{1}{\sin^2\left(\frac{\Phi}{2}\right)}\right].$$

*Proof.* By Lemma 32 we have that $\mathcal{H}(x)$ decomposes nicely into its intersection with the rotation spaces and the $(\pm 1)$-eigenspaces of $U(x, \mathcal{R})$. Hence, we break the optimization problem in the definition of $w_+(x, \mathcal{R})$ in the following way:

$$\begin{aligned} w_+(x,\mathcal{R}) &= \min\{\||w\rangle\|^2 : |w\rangle \in \mathcal{H}(x), |w_0\rangle - |w\rangle \in \mathcal{K}\} \\ &= \min\{\||w\rangle\|^2 : |w\rangle \in \mathcal{H}(x), \Pi_{\mathcal{K}^\perp}|w\rangle = |w_0\rangle\} \\ &= \min\left\{ \||u\rangle + |v\rangle + |w\rangle\|^2 : \begin{array}{l} |u\rangle \in \mathcal{H}(x) \cap E_0, \\ |v\rangle \in \mathcal{H}(x) \cap E_\pi, \\ |w\rangle \in \mathcal{H}(x) \cap E_{\mathrm{rem}}, \\ \Pi_{\mathcal{K}^\perp}(|u\rangle + |w\rangle + |v\rangle) = |w_0\rangle \end{array} \right\}. \end{aligned} \tag{3.12}$$

Observe now that the last condition can be written as:

$$\Pi_{\mathcal{K}^\perp}(|u\rangle + |w\rangle + |v\rangle) = (\Pi_{E_0} + \Pi_{E_{\mathrm{rem}}} + \Pi_{E_\pi})\,|w_0\rangle.$$

Observe, too, that on the one hand $E_0 \cap \mathcal{H}(x) = \mathcal{K} \cap \mathcal{H}(x)$, so $\Pi_{\mathcal{K}^\perp}|u\rangle = 0$, and since $|w_0\rangle \in \mathcal{K} \oplus \mathcal{H}(x) = (\mathcal{K}^\perp \cap \mathcal{H}(x)^\perp)^\perp$ and $|w_0\rangle \in \mathcal{K}^\perp$, then $|w_0\rangle$ is perpendicular to both subspaces in $E_0$ and $\Pi_{E_0}|w_0\rangle = 0$. On the other hand we have that $|v\rangle \in E_\pi \cap \mathcal{H}(x) = \mathcal{K}^\perp \cap \mathcal{H}(x)$ so $|v\rangle \in \mathcal{K}^\perp$ and $\Pi_{\mathcal{K}^\perp}|v\rangle = |v\rangle$. All things considered, we have that this last condition reduces to:

$$\Pi_{\mathcal{K}^\perp}|w\rangle + |v\rangle = (\Pi_{E_{\mathrm{rem}}} + \Pi_{E_\pi})\,|w_0\rangle.$$

On the right-hand side we have two vectors, $\Pi_{E_{\mathrm{rem}}}|w_0\rangle$ and $\Pi_{E_\pi}|w_0\rangle$ which are clearly orthogonal to each other because one is in $E_{\mathrm{rem}}$ and the other is in $E_\pi$. On the left hand we have also two vectors, $|v\rangle$ and $\Pi_{\mathcal{K}^\perp}|w\rangle$, and, by definition, $|v\rangle \in E_\pi$. If we could show that the other vector is in $E_{\mathrm{rem}}$, then we could split this constraint into two independent ones relating to $|v\rangle$ and $|w\rangle$ respectively. Again, Corollary 33 comes to our aid by asserting that $E_{\mathrm{rem}}$ is an invariant subspace of $\Pi_{\mathcal{K}^\perp}$, meaning that $\Pi_{\mathcal{K}^\perp}|w\rangle \in \Pi_{E_{\mathrm{rem}}}$.

Let us summarize what we have done up to this point. We have split the minimization problem in the definition of the positive witness into two independent minimization problems of the form

$$\begin{aligned} w_+(x,\mathcal{R}) = \min\{\||w\rangle\|^2 : |w\rangle \in \mathcal{H}(x) \cap E_{\mathrm{rem}}, \Pi_{\mathcal{K}^\perp}|w\rangle = \Pi_{E_{\mathrm{rem}}}|w_0\rangle\} \\ + \min\{\||v\rangle\|^2 : |v\rangle \in \mathcal{H}(x) \cap E_\pi, |v\rangle = \Pi_{E_\pi}|w_0\rangle\}. \end{aligned}$$

Observe that in fact $E_{\text{rem}}$ is the direct sum of spaces $E_{\varphi_j}$, and by Lemma 32 we have $\mathcal{H}(x) \cap E_{\text{rem}} = \bigoplus_{j=1}^{d} (\mathcal{H}(x) \cap E_{\varphi_j})$. That, in turn, means that the first term in the right-hand side of the last equation splits into several independent minimization problems, leading us to the expression

$$w_+(x, \mathcal{R}) = \sum_j \min\{\||w\rangle\|^2 : |w\rangle \in \mathcal{H}(x) \cap E_{\varphi_j}, \Pi_{\mathcal{K}^\perp}|w\rangle = \Pi_{E_{\varphi_j}}|w_0\rangle\} + \|\Pi_{E_\pi}|w_0\rangle\|^2 .$$

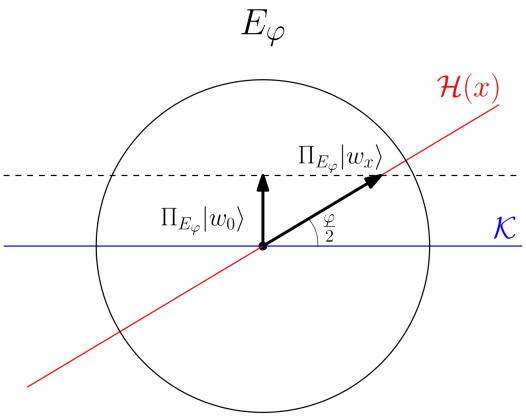



Figure 3.2: How fortunate that Jordan's Lemma gives us two-dimensional subspaces to draw neatly and accurately.

Let us now fix a single eigenphase of $U(x, \mathcal{R})$, $\varphi \in (0, \pi)$. By Jordan's Lemma the space $E_\varphi$ is a two dimensional space spanned by the left- and right-singular vectors of $\Pi_{\mathcal{K}}\Pi_{\mathcal{H}(x)}$ with singular value $\cos\frac{\varphi}{2}$, $|\theta\rangle \in \mathcal{K} \cap E_\varphi$ and $|\psi\rangle \in \mathcal{H}(x) \cap E_\varphi$, which form an angle of $\frac{\varphi}{2}$. Figure 3.3.2 depicts an accurate representation of what is going on in that subspace.

Notice how $\Pi_{E_\varphi}|w_0\rangle$ is still orthogonal to $\mathcal{K}$, and the only solution to the minimization problem is the vector denoted as $\Pi_{E_\varphi}|w_x\rangle$ in the image. To conclude with the proof we note that $\Pi_{\mathcal{K}^\perp}\Pi_{E_\varphi}|w_x\rangle = \Pi_{\mathcal{K}^\perp \cap E_\varphi}\Pi_{E_\varphi}|w_x\rangle$, and since $\mathcal{K} \cap E_\varphi$ and $\mathcal{H}(x) \cap E_\varphi$ form an angle of $\varphi/2$ and $\Pi_{E_\varphi}|w_x\rangle \in \mathcal{H}(x) \cap E_\varphi$, then $\left\|\Pi_{E_\varphi}|w_x\rangle\right\|^2 = \frac{\left\|\Pi_{E_\varphi}|w_0\rangle\right\|^2}{\sin^2\frac{\varphi}{2}}$ (the picture helps), so we arrive

at the expression

$$w_+(x,\mathcal{R}) = \sum_{j=1}^{d} \frac{\left\|\Pi_{E_{\varphi_j}}|w_0\rangle\right\|^2}{\sin^2\frac{\varphi_j}{2}} + \|\Pi_{E_\pi}|w_0\rangle\|^2$$
$$= \sum_{\varphi} \frac{\mathbb{P}(\Phi=\varphi)}{\sin^2\left(\frac{\varphi}{2}\right)} = \mathbb{E}\left[\frac{1}{\sin^2\left(\frac{\Phi}{2}\right)}\right],$$

where the sum is over *all* eigenphases $\varphi$ of $U(x,\mathcal{R})$. □

From this operational interpretation of the positive witness size, we can now deduce an upper bound on the probability of measuring a small phase were we to perform the phase estimation procedure of Theorem 9 to $U(x,\mathcal{R})$ with initial state $|w_0\rangle$ and infinite precision. Take any $\delta \in (0,\pi)$, then by Markov's inequality we have:

$$\mathbb{P}(\Phi \le \delta) = \mathbb{P}\left(\frac{1}{\sin^2\left(\frac{\Phi}{2}\right)} \ge \frac{1}{\sin^2\left(\frac{\delta}{2}\right)}\right) \le \sin^2\left(\frac{\delta}{2}\right)\mathbb{E}\left[\frac{1}{\sin^2\left(\frac{\Phi}{2}\right)}\right]$$
$$= \sin^2\left(\frac{\delta}{2}\right) w_+(x,\mathcal{R}) \le \frac{\delta^2 w_+(x,\mathcal{R})}{4}.$$

Of course, this is not a realistic process. A more realistic setting would be to perform the phase estimation on $U(x,\mathcal{R})$ with precision $\delta$ and accuracy $\varepsilon$ to $|w_0\rangle$. If we now measured the phase register, the probability of observing a phase $\le \delta$ would be upper bounded by $\mathbb{P}(\Phi \le \delta) + \varepsilon$. Together with the fact that for negative instances, the probability of measuring a phase $\le \delta$ would be at least $\mathbb{P}(\Phi = 0) = w_-(x,\mathcal{R})^{-1}$ we could obtain an algorithm for reflection programs. We will not work out the details here since this analysis does not account for approximate witnesses.

Using similar ideas it is possible to derive other identities such as

$$\mathbb{P}(\Phi=\pi) = \left\|\Pi_{\mathcal{K}^\perp\cap\mathcal{H}(x)}|w_0\rangle\right\|^2 \qquad \text{and} \qquad \mathbb{E}\left[\sin^2\left(\frac{\Phi}{2}\right)\right] = \left\|\Pi_{\mathcal{H}(x)}|w_0\rangle\right\|^2.$$

These identities don't mean anything algorithmically speaking because the random variable $\Phi$ cannot be measured exactly, there is no such thing as infinite precision phase estimation. However, they illustrate the relation between the decomposition of $|w_0\rangle$ into eigenvectors of $U(x,\mathcal{R})$, and how $|w_0\rangle$ sits in $\mathcal{H}$ geometrically. The algorithms that we will give in Section 3.4 exploit this connection.

### 3.3.3 Approximate reflection programs

Geometrically speaking, the trait that distinguishes positive from negative instances in a reflection program is the fact that $|w_0\rangle$ has a component in $\mathcal{K}^\perp \cap \mathcal{H}(x)^\perp$ in the latter case. The length of that component is then related to the negative witness size and the probability of measuring zero if we do phase estimation on $U(x, \mathcal{R})$. This has the advantage of creating a clean partition on $X$ that we could exploit to make a quantum algorithm that decides it. The flip-side is that, just like exact span programs, the algorithm decides only that one partition. If you want a different one, you need to get a different reflection program.

A natural way of extending the notion of reflection programs to accommodate more decision problems to each reflection program is to substitute the requirement that $|w_0\rangle \in \mathcal{K} + \mathcal{H}(x)$ by a closeness condition. We already started doing this in Definition 22, but didn't go into detail. Partly, that is because a full exposition of approximate span programs is found in [IJ19; Jef20], and partly because we wanted to defer the discussion on approximation until after we had defined reflection programs. Remember from Section 3.2.2 that positive and negative witnesses can be defined independently of the input $x$. For example, we could say that any vector $|w\rangle$ such that $\Pi_{\mathcal{K}^\perp}|w\rangle = |w_0\rangle$ is a positive witness. However, we don't want to lose track of $\mathcal{H}(x)$ since it is integral to the geometrical understanding of $|w_0\rangle$ that we have built. It's fine to lift the restriction that a witness $|w\rangle$ is in $\mathcal{H}(x)$, but we should at least keep track of how far from $\mathcal{H}(x)$ it is.

In that spirit, let $\epsilon \geq 0$. Any vector $|\tilde{w}\rangle$ such that $\Pi_{\mathcal{K}^\perp}|\tilde{w}\rangle = |w_0\rangle$ and $\left\|\Pi_{\mathcal{H}(x)^\perp}|\tilde{w}\rangle\right\|^2 \leq \epsilon$ is called an *$\epsilon$-approximate positive witnesses* for $x$ in $\mathcal{R}$. The component of any approximate positive witness $|\tilde{w}\rangle$ in $\mathcal{H}(x)^\perp$ is called the *error* of $|\tilde{w}\rangle$. The smallest possible error for an approximate positive witness given an $x$ is called the *positive error*, formally defined as:

**Definition 35** (Positive error)**.** For any reflection program $\mathcal{R}$ on $X$ and any $x \in X$, we define the *positive error* of $x$ as

$$e_+(x, \mathcal{R}) = \min\left\{\left\|\Pi_{\mathcal{H}(x)^\perp}|\tilde{w}\rangle\right\|^2 : |\tilde{w}\rangle - |w_0\rangle \in \mathcal{K}\right\}.$$

In the original definition of approximate witnesses found in [IJ19], the *approximate witness size* was the minimum norm squared over the set of approximate positive witnesses with error exactly $e_+(x, \mathcal{R})$. This last condition turns out to be unnecessarily restrictive. The point of defining approximate

witness sizes is to bound them (and hope that they mean something useful). Requiring every witness to have error exactly $e_+(x, \mathcal{R})$ to give an upper bound is very harsh, and also unnecessary as proven in [Jef20].

Therefore, we define *approximate positive witnesses size* with respect to an error $\epsilon$.

**Definition 36** (Approximate positive witness size)**.** For any reflection program $\mathcal{R}$ on $X$ and $x \in X$, we define the $\epsilon$-approximate positive witness size for $x$ as:

$$\widetilde{w}_+(x, \mathcal{R}, \epsilon) = \min \left\{ \||\tilde{w}\rangle\|^2 : |\tilde{w}\rangle - |w_0\rangle \in \mathcal{K}, \left\|\Pi_{\mathcal{H}(x)^\perp}|\tilde{w}\rangle\right\|^2 \leq \epsilon \right\}. \quad (3.13)$$

If the set over which we minimize is empty, we say that $\widetilde{w}_+(x, \mathcal{R}, \epsilon) = \infty$.

Note that if $\epsilon = 0$ we recover the usual definition of positive witness size, and the set is not empty if and only if $e_+(x, \mathcal{R}) \leq \epsilon$. Note also that if $\left\|\Pi_{\mathcal{H}(x)^\perp}|\tilde{w}\rangle\right\| > 0$ for all approximate positive witnesses then $|w_0\rangle \in \mathcal{K}^\perp \cap \mathcal{H}(x)^\perp$ and there exists an exact negative witness.

The approximate negative witnesses are defined in a somewhat similar fashion by relaxing the requirement that the witnesses be orthogonal to $\mathcal{H}(x)$. We call a vector $|\tilde{\omega}\rangle$ an *$\epsilon$-approximate negative witness* for $x$ in $\mathcal{R}$ if $\langle\tilde{\omega}|w_0\rangle = 1$, $|\tilde{\omega}\rangle \in \mathcal{K}^\perp$, and $\left\|\Pi_{\mathcal{H}(x)}|\tilde{\omega}\rangle\right\|^2 \leq \epsilon$. The component of an approximate negative witness in $\mathcal{H}(x)$ is its error, and just like in the positive case, we define the *negative error* as follows:

**Definition 37** (Negative error)**.** For any reflection program $\mathcal{R}$ on $X$ and any $x \in X$, we define the *negative error* of $x$ as

$$e_-(x, \mathcal{R}) = \min_{|\tilde{\omega}\rangle} \left\{ \left\|\Pi_{\mathcal{H}(x)}|\tilde{\omega}\rangle\right\|^2 : \langle\tilde{\omega}|w_0\rangle = 1, |\tilde{\omega}\rangle \in \mathcal{K}^\perp \right\}.$$

The $\epsilon$-approximate negative witness size of $x$ is defined as:

**Definition 38** (Approximate negative witness size)**.** For any reflection program $\mathcal{R}$ on $X$ and $x \in X$, we define the *$\epsilon$-approximate negative witness size* of $x$ in $\mathcal{R}$ as:

$$\widetilde{w}_-(x, \mathcal{R}, \epsilon) = \min \left\{ \||\tilde{\omega}\rangle\|^2 : \langle\tilde{\omega}|w_0\rangle = 1, |\tilde{\omega}\rangle \in \mathcal{K}^\perp, \left\|\Pi_{\mathcal{H}(x)}|\tilde{\omega}\rangle\right\|^2 \leq \epsilon \right\}. \quad (3.14)$$

If the set over which we minimize is empty, we say that $\widetilde{w}_-(x, \mathcal{R}, \epsilon) = \infty$.

If $\left\|\Pi_{\mathcal{H}(x)^\perp}|\omega\rangle\right\| > 0$ for all approximate negative witnesses, there exists an exact positive witness. The set is not empty if and only if $e_-(x,\mathcal{R}) \leq \epsilon$.

Finally, we define the functions that are computed by $\mathcal{R}$ in the following manner:

**Definition 39** (Functions approximated by $\mathcal{R}$)**.** Let $\mathcal{R}$ be a reflection program on $X$ and $f : X \to \{0,1\}$ be a decision problem. Let $W_+(f,\mathcal{R})$, $W_-(f,\mathcal{R})$ be the positive and negative complexities for $f$ defined in Eq. (3.9). For any $\lambda \in [0,1)$ we say that $\mathcal{R}$ *positively* $\lambda$*-approximates* $f$ if, $W_+(f,\mathcal{R}) < \infty$, and for all $x \in f^{-1}(0)$, $w_+(x,\mathcal{R}) \geq \frac{1}{\lambda}W_+(f,\mathcal{R})$. We say that $\mathcal{R}$ *negatively* $\lambda$*-approximates* $f$ if, $W_-(f,\mathcal{R}) < \infty$, and for all $x \in f^{-1}(1)$, $w_-(x,\mathcal{R}) \geq \frac{1}{\lambda}W_-(f,\mathcal{R})$.

*This is maximally confusing.* We defined approximate witnesses with respect to an error parameter $\epsilon$, and now we define approximate functions with respect to a different parameter $\lambda$, and no mention of approximate witnesses! We spent a lot of time suggesting that a reflection program should allow inputs $x \in P_0$ to be counted as positive input for $f$ if they had witnesses with small enough error, but this definition counts an input $x \in P_0$ as positive for $f$ if the negative witness size is large enough. *Very confusing indeed.*

Our immediate goal now is to show that these two notions are the same. First, we try to motivate why Definition 39 is a good definition. Then we shall relate the minimal positive and negative errors to the witness sizes. Last we shall see that, if we choose an error $\epsilon \leq \lambda/W_-(f,\mathcal{R})$, the existence of $\epsilon$-approximate positive witness sizes for every $x \in f^{-1}(1)$ implies that $\mathcal{R}$ $\lambda$-approximates $f$. Of course, there is an analogous statement for negative witnesses, and so on. We'll come back to this later.

We have said before that the witness sizes are inversely proportional to the ease with which a quantum algorithm can decide a function. Let us attempt to formalize this notion by defining a function that orders the elements of the input set $x$. Fix a reflection program $\mathcal{R}$ and let $\mathbb{V} : X \to \mathbb{R}$ be such that

$$\mathbb{V}(x,\mathcal{R}) = \frac{1}{w_+(x,\mathcal{R})} - \frac{1}{w_-(x,\mathcal{R})}.$$

We call this function the *value* of an instance. This function characterizes how easy an instance is to decide. Notice how one of these numbers is always zero, since having a finite positive (resp. negative) witness size necessarily

implies having an infinite negative (positive) witness size. An instance $x$ with very high positive value is a positive instance with very small witness size, while an instance with very high negative value is a negative instance with small negative witness size, both "easy" instances. Conversely instances with value close to zero are hard to identify because they have large witnesses. The hardest instances are those closest to zero, hence, we expect that the gap between those tells us something about the difficulty of telling positive from negative instances. Indeed, the gap is simply

$$\begin{aligned}\delta :=& \min_{\mathbb{V}(x)\geq 0} \mathbb{V}(x,\mathcal{R}) - \max_{\mathbb{V}(x)\leq 0} \mathbb{V}(x,\mathcal{R}) \\ =& \frac{1}{\max_{f(x)=1} w_+(x,\mathcal{R})} + \frac{1}{\max_{f(x)=0} w_-(x,\mathcal{R})},\end{aligned}$$

And so, if $W_+ \approx W_-$, its inverse $\delta^{-1} = \frac{W_+W_-}{W_++W_-} = \mathcal{O}(\sqrt{W_+W_-})$ reflects the complexity of the reflection program.

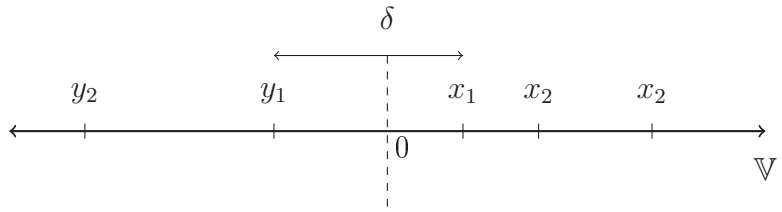



Figure 3.3: The value function orders positive and negative instances on a line.

By definition, the Value function characterizes the function $f(x) = 1 \Leftrightarrow |w_0\rangle \in \mathcal{K} + \mathcal{H}(x)$ by splitting all inputs into two groups; those with positive value and those with negative value. Those above the threshold $\mathbb{V} = 0$ and those below. But, if there was any way to compute the value function, we could just as easily change the function it characterizes by setting a different threshold and saying that the reflection program $\mathcal{R}$ also computes the function $g_\gamma(x) = 1 \Leftrightarrow \mathbb{V}(x) \geq \gamma$ for some $\gamma \in \mathbb{R}$.

From the definition of the Value function and Definition 39, it follows that the functions $g_\gamma$ are $\lambda$-approximated by $\mathcal{R}$ for any $\lambda \geq \frac{\max_{x\in g_\gamma^{-1}(0)} \mathbb{V}(x)}{\min_{x\in g_\gamma^{-1}(1)} \mathbb{V}(x)}$.

Conversely, any function that is positively $\lambda$-approximated by $\mathcal{R}$ is a threshold function $g_\gamma$ with the threshold somewhere in the interval

$$\left(\frac{\lambda}{W_+(f,\mathcal{R})}, \frac{1}{W_+(f,\mathcal{R})}\right).$$

For a threshold $\gamma = 0$, the funcion $g_0$ decides the partition $P_0, P_1$, and we have already seen that the inverse of the Value gap reflects well its complexity. Why should that be different for functions $g_\gamma$ for $\gamma \neq 0$?

Following the example of Figure 3.4, let us imagine that we choose $\gamma$ such that it lies between the values of $x_1$ and $x_2$, and let $\lambda$ be the number such that $\mathbb{V}(x_1) = \lambda\mathbb{V}(x_2)$. The attentive reader might have noticed that we just defined $\lambda$ to be the approximation factor. Indeed, we are committing an abuse of notation here, but only for the purpose of foreshadowing, since we will shortly prove that both notions of $\lambda$ are related. Then all instances with value higher (lower) than $\gamma$ become positive (negative) instances for $g_\gamma$; $x_1$, $x_2$ now become the closest to the threshold, and their value gap should reflect the complexity of deciding $g_\gamma(x)$. Since the positive and negative set has changed, let us redefine the positive and negative complexities as

$$W_-(g_\gamma,\mathcal{R}) = \left\{\max_{x\in g_\gamma^{-1}(0)} \mathbb{V}(x)\right\} \quad \text{and} \quad W_+(g_\gamma,\mathcal{R}) = \left\{\min_{x\in g_\gamma^{-1}(1)} \mathbb{V}(x)\right\}.$$

The value gap between positive and negative instances now becomes $\delta = \mathbb{V}(x_2) - \mathbb{V}(x_1) = (1-\lambda)\mathbb{V}(x_2) = \frac{(1-\lambda)}{W_+(g_\gamma,\mathcal{R})} = \frac{1-\lambda}{\sqrt{\lambda}\sqrt{W_+(g_\gamma,\mathcal{R})W_-(g_\gamma,\mathcal{R})}}$, and its inverse scales as $\frac{\sqrt{W_-W_+}}{1-\lambda}$.

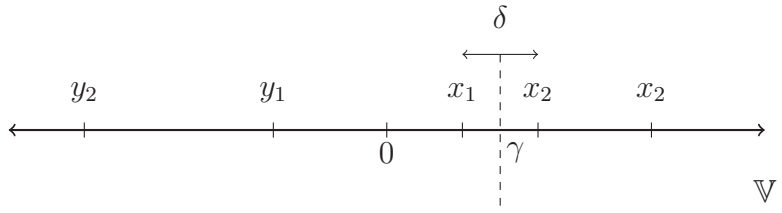



Figure 3.4: $X$ can be partitioned in many ways as long as the order induced by $\mathbb{V}$ is respected.

In Section 3.4, we give an algorithm that decides $\lambda$-approximate reflection programs making $\widetilde{\mathcal{O}}\left(\frac{\sqrt{\widetilde{W}_- W_+}}{(1-\lambda)^{3/2}}\right)$ queries to a unitary $U_x$ similar to $U(x,\mathcal{R})$ for some quantities $\widetilde{W}_-$, $W_+$ related to the witness sizes. We conjecture that the exponent in $(1-\lambda)$ can be brought down to 1. More formally, we have:

**Conjecture 40.** *Let $\mathcal{R}$ be a reflection program that positively $\lambda$-approximates a function $f$. Then there exists an algorithm that decides $f$ with query complexity $\mathcal{O}\left(\frac{1}{1-\lambda}\sqrt{W_-(f,\mathcal{R})\widetilde{W}_+(f,\mathcal{R})}\right)$.*

The notion of the value function allows us to define a whole class of partitions of $X$ that are possibly decidable by $\mathcal{R}$, and also gives an operational interpretation to the approximation parameter. We hope this helps the reader understand why functions approximated by $\mathcal{R}$ are defined so. Let us now move on to relating approximate witness with exact witnesses.

The following theorems characterize the positive and negative errors in terms of the negative and positive witness sizes. In simpler terms, they justify why we defined approximate witnesses in terms of witness errors. These results were already known for span programs and can be found in [IJ19]. The proofs for these results in the reflection program framework are essentially the same as those for span programs, so we will not repeat them here.

**Theorem 41.** *Let $\mathcal{R}$ be a reflection program, $x \in X \subseteq [q]^n$ and let $|\tilde{w}\rangle$ be an approximate positive witness for $x$ such that $\|\Pi_{\mathcal{H}(x)^\perp}|\tilde{w}\rangle\|$ is minimal. Then $\frac{\Pi_{\mathcal{H}(x)^\perp}|\tilde{w}\rangle}{\left\|\Pi_{\mathcal{H}(x)^\perp}|\tilde{w}\rangle\right\|^2}$ is an optimal exact negative witness and $\left\|\Pi_{\mathcal{H}(x)^\perp}|\tilde{w}\rangle\right\|^2 = e_+(x,\mathcal{R}) = 1/w_-(x,\mathcal{R})$.*

So the minimal positive error of an instance $x$ is the negative witness size of that instance. By definition, *any* $\epsilon$-approximating positive witness for $x$ proves that $e_+(x,\mathcal{R}) \leq \epsilon$. This theorem shows that $\epsilon$ also gives us a lower bound for $w_-(x,\mathcal{R})$. Exactly what we need in Definition 39.

**Theorem 42.** *Let $\mathcal{R}$ be a reflection program, $x \in X \subseteq [q]^n$ and let $|\tilde{\omega}\rangle$ be an approximate negative witness for $x$ such that $\|\Pi_{\mathcal{H}(x)}|\tilde{\omega}\rangle\|$ is minimal. Then $|w\rangle = \frac{\Pi_{\mathcal{H}(x)}|\tilde{\omega}\rangle}{\left\|\Pi_{\mathcal{H}(x)}|\tilde{\omega}\rangle\right\|^2}$ is an optimal exact positive witness and $\left\|\Pi_{\mathcal{H}(x)}|\tilde{\omega}\rangle\right\|^2 = e_-(x,\mathcal{R}) = 1/w_+(x,\mathcal{R})$.*

Imagine that we have a function $f : X \to \{0,1\}$ with negative witness size $W_-(f,\mathcal{R}) = \max_{x\in f^{-1}(0)} w_-(x,\mathcal{R}) < \infty$. In particular, this means that $f^{-1}(0) \subseteq P_0$. Fix $\epsilon = \frac{\lambda}{W_-(f,\mathcal{R})}$. If we knew there exist $\epsilon$-approximate positive witnesses for all $x \in f^{-1}(1)$, then we would have $e_+(x,\mathcal{R}) \leq \epsilon = \frac{\lambda}{W_-(f,\mathcal{R})}$, which by Theorem 41, means that $\frac{1}{w_-(x,\mathcal{R})} \leq \epsilon = \frac{\lambda}{W_-(f,\mathcal{R})}$. This is precisely the definition of a function negatively $\lambda$-approximated by $\mathcal{R}$. Let us rephrase.

This means that if there exists an $f : X \to \{0,1\}$ such that $\widetilde{w}_+(x,\mathcal{R},\epsilon) < \infty$ for all $x \in f^{-1}(1)$ with $\epsilon = \frac{\lambda}{W_-(f,\mathcal{R})}$, then $\mathcal{R}$ negatively $\lambda$-approximates $f$. An analogous statement follows from swapping positives and negatives in that argument.

So far, these two ways of looking at approximate reflection programs have successfully accounted for approximation factors, as well as approximated functions, witnesses and errors, but the approximate positive and negative *witness sizes* have been missing from the picture altogether. We do not give an operational interpretation for approximate witness sizes, but approximate negative witness, along with their witness sizes, will appear in the analysis of reflection program algorithms of Section 3.4.

### Reflection program manipulations

In the last couple of sections we have given equal attention to positive and negative witnesses, errors, witness sizes et cetera. From this point on, however, we will focus on algorithms for positively $\lambda$-approximating reflection programs and ignore negatively approximating ones. The reason for that is two-fold. On the one hand, positively $\lambda$-approximating span programs (and their associated reflection programs) are all we will need in the following chapters. On the other hand, there exists a simple transformation that turns a negatively $\lambda$-approximating reflection program for a function $f$ into a positively $\lambda$-approximating span program for $\bar{f} = 1 - f$. We formally prove this in the following theorem.

**Theorem 43.** *Let $\mathcal{R} = (\mathcal{H}, \{\mathcal{H}(x)\}_{x\in X}, \mathcal{K}, |w_0\rangle)$ be a reflection program over $X \subseteq [q]^n$ that negatively $\lambda$-approximates a function $f$. Define the negation of $\mathcal{R}$ as $\overline{\mathcal{R}} = (\mathcal{H}', \{\mathcal{H}'(x)\}_{x\in X}, \mathcal{K}', |w_0'\rangle)$, where*

- $\mathcal{H}' = \mathcal{H}$,
- $\mathcal{H}'(x) = \mathcal{H}(x)^\perp$,

- $\mathcal{K}' = (\mathcal{K} \oplus span\{|w_0\rangle\})^\perp$,
- $|w_0'\rangle = |w_0\rangle$.

*Then, $\overline{\mathcal{R}}$ positively $\lambda$-approximates $\bar{f} = 1 - f$, for every $x \in f^{-1}(0)$, $w_+(x, \overline{\mathcal{R}}) = w_-(x, \mathcal{R})$ and $w_-(x, \overline{\mathcal{R}}) = w_+(x, \mathcal{R})$; and for every $x \in f^{-1}(1)$, $\tilde{w}_-(x, \overline{\mathcal{R}}) = \tilde{w}_+(x, \mathcal{R})$.*

*Proof.* Suppose that $x \in X$ is a positive instance of $\mathcal{R}$. Then, we find that $|w_0\rangle \in \mathcal{K} + \mathcal{H}(x)$, and hence that we can write $|w_0\rangle = |w_x\rangle + |k\rangle$, with $|w_x\rangle \in \mathcal{H}(x) = \mathcal{H}'(x)^\perp$ and $|k\rangle \in \mathcal{K}$. Reordering this we have $|w_x\rangle = |w_0\rangle - |k\rangle \in \mathcal{K} \oplus \text{span}\{|w_0\rangle\} = \mathcal{K}'^\perp$. It immediately follows that $\langle w_x | w_0\rangle = 1$ and $|w_x\rangle \in \mathcal{K}'^\perp \cap \mathcal{H}'(x)^\perp$ and hence we find that $|w_x\rangle$ is a negative witness for $\overline{\mathcal{R}}$. Since the manipulation is symmetric, we see that every negative witness for $\overline{\mathcal{R}}$ is a positive witness for $\mathcal{R}$. We conclude that $w_-(x, \overline{\mathcal{R}}) = w_+(x, \mathcal{R})$. For the other equality, we notice that the negation of the negation of $\mathcal{R}$ is $\mathcal{R}$ itself. Therefore, the same proof shows that $w_+(x, \overline{\mathcal{R}}) = w_-(x, \mathcal{R})$.

Now we show that in fact $\overline{\mathcal{R}}$ positively $\lambda$ approximates $\overline{f}$. By Definition 39, if $x \in f^{-1}(0)$, then $w_+(x, \mathcal{R}) = +\infty$. By the equivalence established before, this means that $x \in \overline{f}^{-1}(1) \Rightarrow w_-(x, \overline{\mathcal{R}}) = +\infty$. If, on the contrary, $x \in f^{-1}(1)$, then

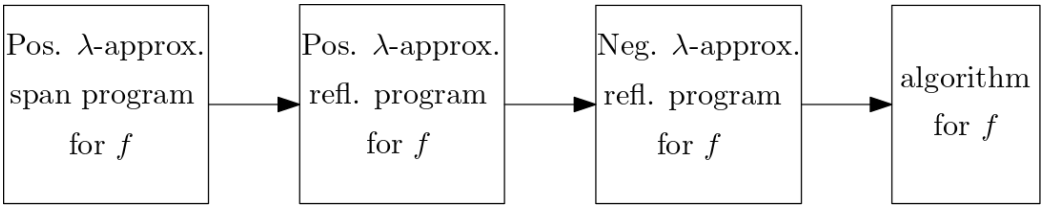
$$\begin{aligned} w_-(x, \mathcal{R}) \geq & \frac{1}{\lambda} W_-(f, \mathcal{R}) = \frac{1}{\lambda} \max_{x : f(x) = 0} w_-(x, \mathcal{R}) = \frac{1}{\lambda} \max_{x : \overline{f}(x) = 1} w_+(x, \overline{\mathcal{R}}) \\ = & \frac{1}{\lambda} W_+(\overline{f}, \overline{\mathcal{R}}). \end{aligned}$$

Since $w_-(x, \mathcal{R}) = w_+(x, \overline{\mathcal{R}})$, we conclude that $\overline{\mathcal{R}}$ positively $\lambda$-approximates $\overline{f}$. □

Notice we only claim that this negation transformation works for approximate reflection programs, not approximate span programs. In [Rei09] there is a procedure that negates exact span programs. We suspect that said transformation could be generalized to work with approximate span programs, but not without friction. The literature that deals with approximate span programs (mainly, [IJ19]) has a different approach. There, there is no need for span program negation since parallel analysis are provided for positive and negatively approximating span programs.

The issue here is that we are not guaranteed that if we take a span program $P$, compile it into a reflection program $\mathcal{R}$ and then negate it, the

result will still correspond to a span program. But, so what? The point of negation is that it allows us to give one algorithm that is useful to decide functions positively and negatively $\lambda$-approximated by $\mathcal{R}$. Our pipeline is rather simple:




### 3.3.4 Span programs and reflection programs

The attentive reader may have noticed that we have spent much more time and ink defining and analyzing reflection programs than we have with span programs. This is particularly striking since there will be almost no mention of reflection programs in the following chapters. The reason for this imbalance is simple. Everything we have shown for reflection programs directly applies to span programs except negation, which doesn't need to. That is the essence of the next lemma.

**Lemma 44.** *Let $P = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$ be a span program over $[q]^n$ and $\lambda \in [0, 1)$. Then the reflection program $\mathcal{R} = (\mathcal{H}, \{\mathcal{H}(x)\}_{x\in[q]^n}, \ker A, A^+|\tau\rangle)$ is such that for every $x \in [q]^n$:*

1. *$|w_x\rangle$ is a positive $\lambda$-approximate witness for $P$ if and only if it is also a positive $\lambda$-approximate witness for $\mathcal{R}$.*

2. *$\langle\omega_x|$ is a negative $\lambda$-approximate witness for $P$ if and only if $(\langle\omega_x|A)^\dagger$ is a negative $\lambda$-approximate witness for $\mathcal{R}$.*

*Proof.* The proof follows easily from the definitions and properties of witnesses in span programs and reflection programs. □

It is also possible to derive a span program from a reflection program, as long as the rule for $x \mapsto \mathcal{H}(x)$ has the direct sum property that $\mathcal{H}(x) = \bigoplus_{i\in[n]} \mathcal{H}_{i,x_i}$. The crux is in defining the map $A$ simply out of $|w_0\rangle$ and $\mathcal{K}$, but since we will not make use of this anywhere in this thesis we omit the details.

## 3.4 Algorithms for reflection programs

Section 3.3 dealt extensively with reflection programs, giving operational interpretations of witnesses, errors, and witness sizes. It also established the equivalence between positive and negatively approximating reflection programs, and the mapping from span to reflection programs.

It is now time to use all of the things we have defined and extensively discussed and build novel quantum algorithms out of them. We begin with an algorithm for reflection program decision. Later, in Section 3.4.2, we will modify slightly the algorithm for span program decision to obtain two algorithms capable of constructing a positive witness out of a span program.

### 3.4.1 Algorithms for reflection program decision

In this section we will describe an algorithm that decides a reflection program even when this is not normalized. The algorithm consists of phase estimation on the product of two reflections very similar to the span program unitary $U(x, \mathcal{R})$ of Section 3.2.3. In fact, the algorithm is a generalization of the span program algorithms from [Rei10; Chi21].

We will begin with some notation. Let $\mathcal{R} = (\mathcal{H}, \{\mathcal{H}(x)\}_{x\in X}, \mathcal{K}, |w_0\rangle)$ be a positively $\lambda$-approximating reflection program for a function $f : X \to \{0, 1\}$, and define its positive and negative witness complexities as

$$W_+ = W_+(f, \mathcal{R}) = \max_{x\in f^{-1}(1)} w_+(x, \mathcal{R}), \tag{3.15}$$

$$\widehat{W}_- = \widehat{W}_-(f, \mathcal{R}) = \max_{y\in f^{-1}(0)} \widetilde{w}_-\left(y, \mathcal{R}, e_-(x, \mathcal{R})\right), \tag{3.16}$$

where the positive witness sizes $w_+(x, \mathcal{R})$ are defined in Definition 31 and approximate negative witnesses are in Equation (3.14). We have chosen to use the wide hat notation to *remark* that this is the min. error approximate witness size. In the next chapter we will use a slightly different notion of approximate witness size, for which we reserve the wide tilde notation.[3]

In order to stress the difference between positive and negative instances in the analysis, we will refer to the former with the letter $x$ and to the latter with the letter $y$.

[3] It is rather unfortunate that the wide hat in [Jef20] denotes precisely what we will denote with a wide tilde. We do so to stay consistent with the notation in [CJO+20].

Let $|\hat{0}\rangle$ be a vector orthogonal to $\mathcal{H}$ and define the state:

$$|\Psi_0\rangle = \frac{1}{\sqrt{1+\frac{\||w_0\rangle\|^2}{W_+}}}\left(|\hat{0}\rangle + \frac{|w_0\rangle}{\sqrt{W_+}}\right) = \frac{1}{\sqrt{\mu_0}}\left(|\hat{0}\rangle + \frac{|w_0\rangle}{\sqrt{W_+}}\right).$$

Define also the projectors:

$$\Pi_x = \Pi_{\mathcal{H}(x)} + |\hat{0}\rangle\langle\hat{0}|, \tag{3.17}$$
$$\Lambda = \Pi_{\mathcal{K}} + |\Psi_0\rangle\langle\Psi_0|. \tag{3.18}$$

Our algorithm will use a variant of the reflection program unitary defined as:

$$U_x := (2\Pi_x - I)(2\Lambda - I). \tag{3.19}$$

We denote the orthogonal projector onto the $(e^{i\varphi})$-eigenspaces of $U_x$ with eigenphase $\varphi \in [-\theta, \theta]$ as $P_\theta^x$.

Observe that the state $|\Psi_0\rangle$ is independent of the input $x$, and so is the state $|\hat{0}\rangle$ and the projector $\Lambda$. Only the projector $\Pi_x$ depends on the input, and we will show later in Theorem 53 that the cost of the reflection $2\Pi_x - I$ is the same as the cost of $2\Pi_{\mathcal{H}(x)} - I$. For reflection programs that come from a span program, the cost of that last reflection is simply 2 queries to the oracle ([IJ19, Lemma 3.1]). For general reflection programs the query complexity will depend on the correspondence $x \mapsto \mathcal{H}(x)$, which is left to the user. We will go into more detail about the construction of these states and reflections in Section 4.3 for the particular case of span programs. Until then, we state the results in terms of calls to the unitary $U_x$.

**Theorem 45.** *Let $\mathcal{R}$ be a reflection program that positively $\lambda$-approximates a function $f$ with $\lambda \in [0,1)$. Then there exists an algorithm that evaluates $f$ with bounded error and makes $\mathcal{O}\left(\frac{\sqrt{W_+\widehat{W}_-}}{(1-\lambda)^{3/2}}\log\frac{1}{1-\lambda}\right)$ queries to $U_x$, where $W_+$, $\widehat{W}_-$ are defined in Eq. (3.15), and Eq. (3.16).*

*Proof.* The algorithm will run phase estimation with precision $\frac{C_1(\sqrt{1-\lambda})}{W_+\widehat{W}_-}$ on the unitary $U_x$ with initial state $|\hat{0}\rangle$, followed by $C_2(1-\lambda)^{-1}$ rounds of amplitude amplification on top of phase estimation for a total of $\widetilde{\mathcal{O}}\left(\frac{\sqrt{W_+\widehat{W}_-}}{(1-\lambda)^{3/2}}\right)$ applications of $U_x$, where $C_1, C_2$ are some constants.

For the analysis of the algorithm, let us start by assuming that $x \in f^{-1}(1)$, and let $|w_x\rangle$ be the exact minimal witness for $x$. Define the vectors:

$$|\Psi_x\rangle := \frac{1}{\sqrt{1+\frac{w_+(x)}{W_+}}}\left(|\hat{0}\rangle + \frac{1}{\sqrt{W_+}}|w_x\rangle\right) = \frac{1}{\sqrt{\mu}_x}\left(|\hat{0}\rangle + \frac{1}{\sqrt{W_+}}|w_x\rangle\right). \tag{3.20}$$

Observe now that $\Pi_x|\Psi_x\rangle = |\Psi_x\rangle$ because $|w_x\rangle \in \mathcal{H}(x)$. If we manage to show that $|\Psi_x\rangle$ is a 1-eigenvector of $\Lambda$ too, then we would have that it is in fact a 1-eigenvector of $U_x$. By definition, every positive witness is of the form $|w_x\rangle = |w_0\rangle + |w_x^\perp\rangle$, where $|w_x^\perp\rangle \in \mathcal{K}$. It follows that

$$|\Psi_x\rangle = \frac{1}{\sqrt{\mu}_x}\left(|\hat{0}\rangle + \frac{|w_0\rangle}{\sqrt{W_+}} + \frac{|w_x^\perp\rangle}{\sqrt{W_+}}\right),$$

and

$$\begin{aligned}
\Lambda|\Psi_x\rangle =& \Pi_{\mathcal{K}}|\Psi_x\rangle + |\Psi_0\rangle\langle\Psi_0|\Psi_x\rangle \\
=& \frac{1}{\sqrt{\mu}_x}\left(\frac{|w_x^\perp\rangle}{\sqrt{W_+}} + |\Psi_0\rangle\langle\Psi_0|\left(|\hat{0}\rangle + \frac{|w_0\rangle}{\sqrt{W_+}}\right)\right) \\
=& \frac{1}{\sqrt{\mu}_x}\left(\frac{|w_x^\perp\rangle}{\sqrt{W_+}} + |\Psi_0\rangle\langle\Psi_0|\sqrt{\mu_0}|\Psi_0\rangle\right) = \frac{1}{\sqrt{\mu}_x}\left(\frac{|w_x^\perp\rangle}{\sqrt{W_+}} + |\hat{0}\rangle + \frac{|w_0\rangle}{\sqrt{W_+}}\right) \\
=& |\Psi_x\rangle.
\end{aligned} \tag{3.21}$$

This proves that for every $x$ such that $f(x) = 1$, $P_0^x|\Psi_x\rangle = |\Psi_x\rangle$ because it is a 0-phase eigenvector of $U_x$. Since we chose $|w_x\rangle$ to be the optimal positive witness, it follows that $\|\langle\Psi_x|\hat{0}\rangle\|^2 = 1/\mu_x \geq \frac{1}{2}$. We conclude that for every $x \in X$ such that $f(x) = 1$,

$$\left\|P_0^x|\hat{0}\rangle\right\|^2 = \frac{1}{\mu_x} \geq \frac{1}{2}. \tag{3.22}$$

Notice how this does not depend on the precision we choose for phase estimation. That will be determined by the negative instances.

Speaking of which, let us focus now on the negative instances $y \in f^{-1}(0)$. We have already established that for positive instances, the amplitude of $|\hat{0}\rangle$ in the 0-phase eigenspace of $U_x$ is at least $1/2$. In order to have a functioning algorithm we will now show that for negative inputs, the state $|\hat{0}\rangle$ has small overlap with the eigenspaces of $U_y$ with eigenphases small enough to

be confused with the 0-phase space by phase estimation. The main technical ingredient will be the effective spectral gap Lemma 5, which we restate here for convenience.

**Lemma** (Effective spectral gap lemma)**.** *Let $|\phi\rangle$ be a unit vector such that $\Lambda|\phi\rangle = 0$, let $P_\theta^x$ be the projector onto the eigenvectors of $U_x = (2\Pi_x - I)(2\Lambda - I)$ with eigenvalues $e^{i\gamma}$ with $|\gamma| \leq \theta$ for some $\theta \geq 0$. Then $\|P_\theta \Pi_x |\phi\rangle\|^2 \leq \theta^2/4$.*

Let $y$ be such that $f(y) = 0$, and let $|\tilde{\omega}_y\rangle$ be an optimal min. error negative witness. Consider the state

$$|\phi_y\rangle = \frac{|\hat{0}\rangle - \sqrt{W_+}|\tilde{\omega}_y\rangle}{\sqrt{1 + W_+ \tilde{w}_-(y)}} = \frac{1}{\sqrt{\nu_y}} \left( |\hat{0}\rangle - \sqrt{W_+}|\tilde{\omega}_y\rangle \right). \tag{3.23}$$

Remember that we have defined $\Lambda$ as $\Lambda = \Pi_{\mathcal{K}} + |\Psi_0\rangle\langle\Psi_0|$. We will prove that $|\phi_y\rangle$ is in the kernel of both terms. For the second one, we have

$$\begin{aligned} \langle \phi_y | \Psi_0 \rangle =& \frac{1}{\sqrt{\mu_0 \nu_y}} \left( \langle \hat{0}| - \sqrt{W_+}\langle \tilde{\omega}_y| \right) \left( |\hat{0}\rangle + \frac{1}{\sqrt{W_+}}|w_0\rangle \right) \\ =& \frac{1}{\sqrt{\mu_0 \nu_y}} \left( 1 - \langle \tilde{\omega}_y | w_0 \rangle \right) = 0, \end{aligned} \tag{3.24}$$

where we have used that $|\tilde{\omega}_y\rangle$ is a negative witness in the last equality. This proves that $|\Psi_0\rangle\langle\Psi_0|\phi_y\rangle = 0$.

Remember, too, that the new state $|\hat{0}\rangle$ is perpendicular to $\mathcal{K}$, and by Definition 38, every approximate negative witness is in $\mathcal{K}^\perp$, so $\Pi_{\mathcal{K}}|\phi_y\rangle = 0$. Therefore, we conclude that $\Lambda|\phi_y\rangle = 0$.

Let us assume for now that no exact negative witness exists for $y$. We'll deal with those later. By Theorem 42 we have that $|w_y\rangle = \frac{\Pi_y|\tilde{\omega}_y\rangle}{e_-(y)}$ is an optimal exact positive witness. So we can define the state:

$$|\Psi_y\rangle = \frac{|\hat{0}\rangle + \frac{1}{\sqrt{W_+}}|w_y\rangle}{\sqrt{1 + \frac{w_+(y)}{W_+}}} = \frac{1}{\sqrt{\mu_y}} \left( |\hat{0}\rangle + \frac{1}{\sqrt{W_+}}|w_y\rangle \right). \tag{3.25}$$

From equations (3.23) and (3.25) it follows that we can rewrite $|\hat{0}\rangle$ and $|w_y\rangle$ as:

$$|\hat{0}\rangle = \sqrt{\nu_y}\Pi_y|\phi_y\rangle + \sqrt{W_+}\Pi_y|\tilde{\omega}_y\rangle \quad \text{and} \quad |w_y\rangle = \sqrt{W_+}\left(\sqrt{\mu_y}|\Psi_y\rangle - |\hat{0}\rangle\right),$$

and so $P_\theta^y|\hat{0}\rangle$ can be expressed as:

$$P_\theta^y|\hat{0}\rangle = \sqrt{\nu_y}P_\theta^y\Pi_y|\phi_y\rangle + W_+e_-(y)P_\theta^y\left(\sqrt{\mu_y}|\Psi_y\rangle - |\hat{0}\rangle\right).$$

Grouping terms and recalling that $e_-(y) = 1/w_+(y)$ we have:

$$\left(1+\frac{W_+}{w_+(y)}\right)P_\theta^y|\hat{0}\rangle = \sqrt{\nu_y}P_\theta^y\Pi_y|\phi_y\rangle + \frac{W_+}{w_+(y)}P_\theta^y\sqrt{\mu_y}|\Psi_y\rangle.$$

Luckily, $|\Psi_y\rangle$ is a 1-eigenvector of $U_y$ for the same reasons $|\Psi_x\rangle$ was one for $U_x$ when $x \in f^{-1}(1)$, and from the Effective Spectral Gap Lemma we have that $P_0^y\Pi_y|\phi_y\rangle = 0$, which means that the two vectors in the r.h.s of the previous equation are actually orthogonal. We arrive at:

$$\begin{aligned}\left\|P_\theta^y|\hat{0}\rangle\right\|^2 =& \frac{\nu_y}{\left(1+\frac{W_+}{w_+(y)}\right)^2}\left\|P_\theta^y\Pi_y|\phi_y\rangle\right\|^2 + \frac{\mu_y\frac{W_+^2}{w_+(y)^2}}{\left(1+\frac{W_+}{w_+(y)}\right)^2}\left\|P_\theta^y|\Psi_y\rangle\right\|^2 \\ =& \frac{\nu_y}{\left(1+\frac{W_+}{w_+(y)}\right)^2}\left\|P_\theta^y\Pi_y|\phi_y\rangle\right\|^2 + \frac{1}{\mu_y} \leq \left(1+W_+\widehat{W}_-\right)\frac{\theta^2}{4} + \frac{1}{\mu_y}.\end{aligned} \tag{3.26}$$

What about $y$ for which we have exact negative witnesses? As it turns out, that case is easier. If a negative witness $|\omega_y\rangle$ exists, then there is no exact positive witness, $|\phi_y\rangle$ can be defined using the exact negative witness but $|\Psi_y\rangle$ cannot be defined, and $\Pi_y|\omega_y\rangle = 0$. It follows that

$$|\hat{0}\rangle = \sqrt{\nu_y}\Pi_y|\phi_y\rangle,$$

and so, by the Effective Spectral Gap Lemma, we arrive at:

$$\left\|P_\theta^y|\hat{0}\rangle\right\|^2 = \nu_y\left\|P_\theta^y\Pi_y|\phi_y\rangle\right\|^2 \leq \left(1+W_+\widehat{W}_-\right)\frac{\theta^2}{4}.$$

Now, let $U_x = \sum_{j=1}^m e^{i\phi_j}|\psi_j\rangle\langle\psi_j|$ be an eigendecomposition of $U_x$, and let $|\hat{0}\rangle = \sum_{j=1}^m \alpha_j|\psi_j\rangle$ be a decomposition of $|\hat{0}\rangle$ in the eigenbasis of $U_x$. The phase estimation procedure of Theorem 9 on $U_x$ with precision $\theta$, accuracy $\varepsilon$ applied to $|\hat{0}\rangle$ produces a state $|\hat{0}'\rangle := PE(U_x,\theta,\varepsilon)|\hat{0}\rangle = \sum_{j=1}^m \alpha_j|\psi_j\rangle|w_j\rangle_P$ such that, if $\phi_j = 0$, $|w_j\rangle_P = |0\rangle_P$, and if $|\phi_j| \geq \theta$, then $|\langle 0|w_j\rangle_P|^2 \leq \varepsilon$.

For $x \in f^{-1}(1)$, we showed that $\|P_0^x|\hat{0}\rangle\|^2 = \sum_{j:\varphi_j=0} |\alpha|^2 \geq \frac{1}{2}$, and so

$$\begin{aligned}\left\||0\rangle\langle 0|_P|\hat{0}'\rangle\right\|^2 &= \left\|\sum_{j=1}^{m} \alpha_j |0\rangle\langle 0|_P|w_j\rangle_P|\psi_j\rangle\right\|^2 \\ &\geq \sum_{j:\varphi_j=0}^{m} |\alpha_j|^2 = \|P_0^x|\hat{0}\rangle\|^2 \geq \frac{1}{2} =: p_0. \end{aligned} \tag{3.27}$$

Meanwhile, for $y \in f^{-1}(0)$ we have:

$$\begin{aligned}\||0\rangle\langle 0|_P|\hat{0}'\rangle\|^2 &= \left\|\sum_{j:|\phi_j|\leq\theta}^{m} \alpha_j |0\rangle\langle 0|_P|w_j\rangle_P|\psi_j\rangle\right\|^2 + \left\|\sum_{j:|\phi_j|>\theta}^{m} \alpha_j |0\rangle\langle 0|_P|w_j\rangle_P|\psi_j\rangle\right\|^2 \\ &\leq \sum_{j:|\phi_j|\leq\theta}^{m} |\alpha_j|^2 + \sum_{j:|\phi_j|>\theta}^{m} |\alpha_j|^2 |\langle 0|w_j\rangle|^2 \leq \|P_\theta^y|\hat{0}\rangle\|^2 + \varepsilon \\ &\leq (1 + W_+\widehat{W}_-)\frac{\theta^2}{4} + \frac{1}{\mu_y} + \varepsilon. \end{aligned} \tag{3.28}$$

Remember that $\frac{1}{\mu_y} = 0$ if $y$ has an exact negative witness, and that $\frac{1}{\mu_y} \leq \frac{1}{1+\frac{1}{\lambda}}$ because $\mathcal{R}$ positively $\lambda$-approximates $f$. Therefore, for all $y \in f^{-1}(0)$, choosing a precision $\theta = \frac{1}{2}\sqrt{\frac{1-\lambda}{1+W_+\widehat{W}_-}}$ and accuracy $\varepsilon = \frac{1-\lambda}{16}$ we have:

$$\||0\rangle\langle 0|_P|\hat{0}'\rangle\|^2 \leq \frac{1-\lambda}{16} + \frac{1}{1+\frac{1}{\lambda}} + \frac{1-\lambda}{16} =: p_1. \tag{3.29}$$

A simple calculation yields $p_0 - p_1 \geq \frac{1-\lambda}{8}$, so by Corollary 12, we can distinguish these two cases with $\mathcal{O}\left(\frac{\sqrt{p_0}}{p_0-p_1}\right)$ calls to $PE(U_x, \theta, \varepsilon)$. Since each call to Phase Estimation uses $\mathcal{O}\left(\frac{1}{\theta}\log\frac{1}{\varepsilon}\right) = \mathcal{O}\left(\sqrt{\frac{W_+\widehat{W}_-}{1-\lambda}}\log\frac{1}{1-\lambda}\right)$ calls to $U_x$, we conclude that the $f$ can be decided with $\mathcal{O}\left(\frac{1}{(1-\lambda)^{3/2}}\sqrt{W_+\widehat{W}_-}\log\frac{1}{1-\lambda}\right)$ calls to $U_x$. □

Observe that it is not necessary that the state $|w_0\rangle$ in the reflection program be normalized. This construction bypasses completely the need for normalization in [IJ19]. We also note that the exponent 3/2 in the $\lambda$-dependent

factor of the algorithm complexity can be reduced to 1 if we know the true spectral gap of $U_x$. The flipside is that the complexity will depend on the spectral gap instead of the span program complexity.

According to the discussion that follows Theorem 42, it is enough to find an approximate negative witness with error $\epsilon = \lambda/W_+$ for every $x \in f^{-1}(0)$ to know for certain that $f$ is $\lambda$-approximated by $\mathcal{R}$. Theorem 45 requires us to find min. error witnesses — a strictly harder task — which can be too restrictive, as will be the case in Chapter 4. As a corollary to Theorem 45 we show how we can use $\epsilon$-approximate negative witnesses with error $\epsilon = \lambda/W_+$ instead of $\epsilon = e_-(x, \mathcal{R})$. The trade-off is that the range of the approximation factor is reduced to $\lambda \in [0, 1/2)$. We use a relaxed version of the approximate witness size defined as:

$$\widetilde{W}_- = \widetilde{W}_-(f, \mathcal{R}) = \max_{y \in f^{-1}(0)} \widetilde{w}_- \left( y, \mathcal{R}, \frac{\lambda}{W_+} \right). \tag{3.30}$$

We use the following corollary in Chapter 4.

**Corollary 46.** *Let $\mathcal{R}$ be a reflection program that positively $\lambda$-approximates a function $f$ with $\lambda \in [0, 1/2)$. Then there exists an algorithm that evaluates $f$ with bounded error and makes $\mathcal{O}\left( \frac{\sqrt{W_+ \widetilde{W}_-}}{(1-2\lambda)^{3/2}} \log \frac{1}{1-2\lambda} \right)$ queries to $U_x$, where $W_+$, $\widetilde{W}_-$ are defined in Eq. (3.15), and Eq. (3.30).*

*Proof.* The proof follows parallel to that of Theorem 45. Let $|\tilde{\omega}_y\rangle$ be an $\epsilon$-approximate negative witness for $y \in f^{-1}(1)$ with error $\epsilon \leq \lambda/W_+$. Define the state:

$$|\phi_y\rangle = \frac{|\hat{0}\rangle + \sqrt{W_+}|\tilde{\omega}_y\rangle}{\sqrt{1 + W_+ \tilde{w}_-(y)}} = \frac{1}{\sqrt{\nu_y}} \left( |\hat{0}\rangle + \sqrt{W_+}|\tilde{\omega}_y\rangle \right).$$

As before, $\Lambda|\phi_y\rangle = 0$ because $|\tilde{\omega}_y\rangle$ is a negative witness. Now, $|\hat{0}\rangle = \sqrt{\nu_y}\Pi_y|\phi_y\rangle + \sqrt{W_+}\Pi_y|\tilde{\omega}_y\rangle$, so we give the following upper bound on the norm of $P_\theta^y|\hat{0}\rangle$:

$$\begin{aligned} \left\| P_\theta^y|\hat{0}\rangle \right\|^2 &= \nu_y \left\| P_\theta^y \Pi_y |\phi_y\rangle \right\|^2 + W_+ \left\| P_\theta^y \Pi_y |\tilde{\omega}_y\rangle \right\|^2 \\ &\leq \nu_y \frac{\theta^2}{4} + W_+ \left\| P_\theta^y \Pi_y |\tilde{\omega}_y\rangle \right\|^2 \\ &\leq \left( 1 + W_+ \widetilde{W}_- \right) \frac{\theta^2}{4} + W_+ \frac{\lambda}{W_+}, \end{aligned} \tag{3.31}$$

where in the first inequality we have used Lemma 5, and in the second we have used that $|\tilde{\omega}_y\rangle$ is a $\frac{\lambda}{W_+}$-approximate negative witness. Following the proof of Theorem 45 we see that substituting this into Equation (3.28) we obtain that for any $y \in f^{-1}(0)$,

$$\| |0\rangle\langle 0|_P |\hat{0}'\rangle \|^2 \leq (1 + W_+ \widetilde{W}_-) \frac{\theta^2}{4} + \lambda + \varepsilon. \tag{3.32}$$

Hence, as long as $\lambda < 1/2$, we can choose precision $\theta = \frac{1}{2}\sqrt{\frac{1-2\lambda}{1+W_+\widetilde{W}_-}}$ and accuracy $\varepsilon = \frac{1-2\lambda}{8}$ to obtain:

$$\| |0\rangle\langle 0|_P |\hat{0}'\rangle \|^2 \leq \frac{1}{2} - \frac{1-2\lambda}{8} =: p_1.$$

Meanwhile, for $x \in f^{-1}(1)$, it is still true that

$$\left\| |0\rangle\langle 0|_P |\hat{0}'\rangle \right\|^2 \geq \frac{1}{2} =: p_0.$$

Using Corollary 12 we conclude that we can distinguish these two cases with bounded error using $\mathcal{O}\left(\frac{1}{(1-2\lambda)^{3/2}}\sqrt{W_+\widetilde{W}_-}\log\frac{1}{1-2\lambda}\right)$ calls to $U_x$. □

### 3.4.2 An algorithm for witness generation

A long time ago, in the ancient and obscure depths of Section 3.2.2, we referred to witnesses, positive and negative, as qualitatively analogous to certificates, partial assignments of $(x_1, \ldots, x_n)$ that are sufficient to decide $f$. The existence of witnesses certifies that the target has this or that geometric property, and their norm gives us an intuitive idea of how hard an instance is to decide (recently formalized in [ACK20]). Something that often confuses those who encounter span programs for the first time is that, for all the talk about witness and how they act as "certificates" of sorts, the algorithms for span programs do *not* try to find these witnesses. In fiction, and particularly in film, this is called a McGuffin. An artifact that moves the plot forward but does not play a role in the resolution. The staff of Ra.[4] The actual algorithm goes along different lines. If positive witnesses exist, they are at least *this* big, and so the span program unitary has *this* feature. If negative

[4]Famous McGuffin in "Raiders of the Lost Ark" - Steven Spielberg, 1981.

witnesses exist, they are at least *that* big, and so the span program unitary has *that* other feature.

Reflection programs, (and span programs as a special case) encode much more information than just the one bit output of the function $f$ they decide. In [IJ19], the authors give two algorithms that can estimate the witness size of an input. This is useful when the witness size is a quantity of interest, as is the case for the *st*-connectivity span program, see Chapter 5. But what if the witnesses themselves are objects of interest? Can we use the elements of a reflection program to design an algorithm that makes calls to an input-dependent unitary $U_x$ and outputs an optimal witness $|w_x\rangle$? This is precisely the problem of witness generation.

**Problem.** *Fix a reflection program* $\mathcal{R} = (\mathcal{H}, \{\mathcal{H}(x)\}_{x\in X}, \mathcal{K}, |w_0\rangle)$*, encoding a function* $f : X \subset [q]^n \to \{0,1\}$*. The* witness generation problem *is the following: given an input* $x \in f^{-1}(1)$*, find a quantum procedure that outputs a normalized version of the state*

$$|w_x\rangle = \operatorname{argmin}\{\||w\rangle\|^2 : \Pi_{\mathcal{K}^\perp}|w\rangle = |w_0\rangle, |w\rangle \in \mathcal{H}(x)\}.$$

We end this chapter by providing a novel algorithm for this problem. The algorithm is a modification of the algorithm for decision problems analyzed in the previous section.

As before, given a reflection program $\mathcal{R} = (\mathcal{H}, \{\mathcal{H}(x)\}_{x\in X}, \mathcal{K}, |w_0\rangle)$, let $|\hat{0}\rangle$ be a vector orthogonal to $\mathcal{H}$, $\alpha > 0$, and define the state

$$|\Psi_0^\alpha\rangle = \frac{1}{\sqrt{1 + \frac{\||w_0\rangle\|^2}{\alpha^2}}}\left(|\hat{0}\rangle + \frac{|w_0\rangle}{\alpha}\right) = \frac{1}{\sqrt{\mu}_0}\left(|\hat{0}\rangle + \frac{|w_0\rangle}{\alpha}\right). \tag{3.33}$$

Define also the projectors

$$\Pi_x = \Pi_{\mathcal{H}(x)} + |\hat{0}\rangle\langle\hat{0}|, \tag{3.34}$$
$$\Lambda^\alpha = \Pi_{\mathcal{K}} + |\Psi_0^\alpha\rangle\langle\Psi_0^\alpha|. \tag{3.35}$$

If the reflection program corresponds to a span program $P = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$, where $\mathcal{K} = \ker A$ and $|w_0\rangle = A^+|\tau\rangle$, then $\Lambda^\alpha = \Pi_{\ker(\tilde{A}^\alpha)}$, where

$$\tilde{A}^\alpha = \frac{1}{\alpha}|\tau\rangle\langle\hat{0}| - A.$$

In order to construct the positive witness $|w_x\rangle$, we will need an estimation of $\||w_x\rangle\|$ with multiplicative error $\epsilon = \mathcal{O}(1)$. This is the purpose of the following lemma.

**Lemma 47.** *Let $\mathcal{R}$ be a reflection program deciding a function $f$ and let $x$ be a positive input. Let $|w_x\rangle$ be an optimal positive witness for $x$ in $\mathcal{R}$, $w_+(x) = \||w_x\rangle\|^2$ be the exact positive witness size of $x$ for $\mathcal{R}$, and $\widetilde{w}_-(x)$ be the min. error negative witness size for $x$ in $\mathcal{R}$. Then there exists a procedure that with bounded error outputs a number $\alpha_*^2 \in \left(\frac{w_+(x)}{2}, 2w_+(x)\right)$ and makes $\mathcal{O}\left(\sqrt{w_+(x)\widetilde{w}_-(x)}\log(w_+(x))\log\log W_+\right)$ controlled calls to $U_{x,\alpha} = (2\Pi_x - I)(2\Lambda^\alpha - I)$, where $W_+$ is an upper bound for $w_+(x)$ and $\alpha > 0$ can change from one call to the next.*

The result of this lemma is rather similar to that of Theorem 29 from [IJ19]. The differences are that this lemma is proven for reflection programs and we restrict ourselves to error $\mathcal{O}(1)$, and that the complexity is slightly improved. In the interest of clarity, we will prove Lemma 47 after we prove Theorem 48.

**Theorem 48.** *Let $\mathcal{R}$ be a reflection program deciding a function $f$ and let $x$ be a positive input. Let $|w_x\rangle$ be an optimal positive witness for $x$ in $\mathcal{R}$, $w_+(x) = \||w_x\rangle\|^2$ be the exact positive witness size of $x$ for $\mathcal{R}$, and $\widetilde{w}_-(x)$ be the min. error negative witness size for $x$ in $\mathcal{R}$. Then there exists a procedure that succeeds with probability $\geq 2/9$ and for any $\varepsilon > 0$ prepares a state $|\tilde{w}\rangle$ such that $\left\||\tilde{w}\rangle - \frac{|w_x\rangle}{\||w_x\rangle\|}\right\|^2 = \mathcal{O}(\varepsilon)$ using $\widetilde{\mathcal{O}}\left(\sqrt{w_+(x)\widetilde{w}_-(x)}\right) + \mathcal{O}\left(\frac{\sqrt{w_+(x)\widetilde{w}_-(x)}}{\sqrt{\varepsilon}}\right)$ controlled calls to $U_{x,\alpha} = (2\Pi_x - I)(2\Lambda^\alpha - I)$, where $\alpha > 0$ can change from one call to the next.*

*Proof.* The proof follows from two observations on the proof of Theorem 45.

First, observe that the states $|\Psi_x\rangle$ are a combination of $|\hat{0}\rangle$ and the state we are interested in, $|w_x\rangle$, and these two can be separated because $|\hat{0}\rangle$ is perpendicular to $\mathcal{H}$. Hence, if we could approximate $|\Psi_x\rangle$, we could distill $|w_x\rangle$ from it.

Second, we will show that the post-measurement state after we perform $PE(U_{x,\alpha}, \theta, \epsilon)$ on $|\hat{0}\rangle$ and project onto $|0\rangle\langle 0|_P$ is close to $|\Psi_x\rangle$. The parameter $\alpha$ allows us to control both the probability of obtaining $|0\rangle_P$ in this measurement and the weight of $|w_x\rangle$ within $|\Psi_x\rangle$.

Let $x$ be such that $f(x) = 1$ and let $|\omega_x\rangle$ be the optimal min. error negative witness for $x$ in $\mathcal{R}$. With a slight abuse of notation, define the vector

$$|\phi_x\rangle = \frac{|\hat{0}\rangle - \alpha|\omega_x\rangle}{\sqrt{1 + \alpha^2 \widetilde{w}_-(x)}}, \tag{3.36}$$

where $\alpha$ is left implicit. It is no coincidence that we use the same notation as in Eq. (3.23). Indeed, if we chose $\alpha = \sqrt{W_+}$ we would recover the same states we use there. Now, following the same proof we used in Theorem 45, we can see that $\Lambda^\alpha|\phi_x\rangle = 0$. We will use this later to apply the effective spectral gap lemma to $|\phi_x\rangle$. How about the other projector? Observe that $\Pi_x|\hat{0}\rangle = |\hat{0}\rangle$, so

$$\Pi_x|\phi_x\rangle = \frac{|\hat{0}\rangle - \alpha\Pi_x|\omega_x\rangle}{\sqrt{1 + \alpha^2 \widetilde{w}_-(x)}}.$$

Again, we define the quantity $\nu_x = 1 + \alpha^2 \widetilde{w}_-(x)$ to simplify the expressions. Then, isolating $|\hat{0}\rangle$ from the equation above we obtain

$$|\hat{0}\rangle = \sqrt{\nu_x}\Pi_x|\phi_x\rangle + \alpha\Pi_x|\omega_x\rangle.$$

Recall that the last term of this equation has already appeared in Theorem 42 as $\Pi_x|\omega_x\rangle = \Pi_{\mathcal{H}(x)}|\omega_x\rangle = |w_x\rangle e_-(x) = \frac{|w_x\rangle}{w_+(x)}$, where $|w_x\rangle$ is an optimal exact positive witness for $x$. All in all, we have that $|\hat{0}\rangle$ can be expressed as

$$|\hat{0}\rangle = \sqrt{\nu_x}\Pi_x|\phi_x\rangle + \frac{\alpha}{w_+(x)}|w_x\rangle. \tag{3.37}$$

Now, let us define yet another state related to $|\hat{0}\rangle$ and the optimal positive witness $|w_x\rangle$,

$$|\Psi_x\rangle = \frac{|\hat{0}\rangle + \frac{1}{\alpha}|w_x\rangle}{\sqrt{1 + \frac{w_+(x)}{\alpha^2}}}. \tag{3.38}$$

As was the case for vectors $|\Psi_x\rangle$ in the proof of Theorem 45, this vector is in the 1-eigenspace of $U_{x,\alpha}$. We can rearrange terms to express $|w_x\rangle$ as

$$|w_x\rangle = \alpha\left(\sqrt{\mu_x}|\Psi_x\rangle - |\hat{0}\rangle\right), \tag{3.39}$$

where $\mu_x = 1 + \frac{w_+(x)}{\alpha^2}$. Putting together equations (3.37) and (3.39) we arrive at

$$\begin{aligned}|\hat{0}\rangle =& \frac{\sqrt{\nu_x}}{1+\frac{\alpha^2}{w_+(x)}}\Pi_x|\phi_x\rangle + \frac{\frac{\alpha^2}{w_+(x)}\sqrt{\mu_x}}{1+\frac{\alpha^2}{w_+(x)}}|\Psi_x\rangle \\ =& \frac{\sqrt{\nu_x}}{1+\frac{\alpha^2}{w_+(x)}}\Pi_x|\phi_x\rangle + \frac{1}{\sqrt{\mu_x}}|\Psi_x\rangle. \end{aligned} \tag{3.40}$$

So far, we have only rewritten $|\hat{0}\rangle$ in a way that is advantageous for us, but we have not done anything to this state. Remember that the goal is to process $|\hat{0}\rangle$ to extract $|w_x\rangle$ from all this. The first step in our processing will be to apply the Phase Estimation procedure of Theorem 9 with unitary $U_{x,\alpha}$ to the state $|\hat{0}\rangle$ with precision $\theta$ and accuracy $\varepsilon$. Then, we project onto the $|0\rangle_P$ state in the phase register created by the phase estimation.

More formally, let $U_{x,\alpha} = \sum_{j=1}^m e^{i\varphi_j}|\psi_j\rangle\langle\psi_j|$ be an eigendecomposition of $U_{x,\alpha}$, and let $|\hat{0}\rangle = \sum_{j=1}^m \beta_j|\psi_j\rangle$[5] be a decomposition of $|\hat{0}\rangle$ in the eigenbasis of $U_{x,\alpha}$. The phase estimation procedure of Theorem 9 on $U_{x,\alpha}$ with precision $\theta$, accuracy $\varepsilon$ applied to $|\hat{0}\rangle$ produces a state $|\hat{0}'\rangle := PE(U_{x,\alpha},\theta,\varepsilon)|\hat{0}\rangle = \sum_{j=1}^m \beta_j|\psi_j\rangle|w_j\rangle_P$ such that, if $\varphi_j = 0$, $|w_j\rangle_P = |0\rangle_P$, and if $|\varphi_j| \geq \theta$, then $|\langle 0|w_j\rangle_P|^2 \leq \varepsilon$. Therefore, we have two alternative decompositions of $|\hat{0}\rangle$:

$$\begin{aligned}|\hat{0}\rangle =& \sum_{j=1}^m \beta_j|\psi_j\rangle = \sum_{j:\varphi_j\neq 0}\beta_j|\psi_j\rangle + \sum_{j:\varphi_j=0}\beta_j|\psi_j\rangle \\ |\hat{0}\rangle =& \frac{\sqrt{\nu_x}}{1+\frac{\alpha^2}{w_+(x)}}\Pi_x|\Phi_x\rangle + \frac{1}{\sqrt{\mu_x}}|\Psi_x\rangle.\end{aligned}$$

Now, remember that $U_{x,\alpha}|\Psi_x\rangle = |\Psi_x\rangle$ and that $\|P_0^x\Pi_x|\Phi_x\rangle\| = 0$ (just apply the effective spectral gap with $\theta = 0$). This means that the two decompositions match term by term. That is

$$\sum_{j:\varphi_j\neq 0}\beta_j|\psi_j\rangle = \frac{\sqrt{\nu_x}}{1+\frac{\alpha^2}{w_+(x)}}\Pi_x|\Phi_x\rangle \quad \text{and} \quad \sum_{j:\varphi_j=0}\alpha_j|\psi_j\rangle = \frac{1}{\sqrt{\mu_x}}|\Psi_x\rangle.$$

[5] We have switched to betas for the amplitudes of the phase spaces here because $\alpha$ is now a free parameter.

Applying $PE(U_{x,\alpha}, \theta, \varepsilon)$ to $|\hat{0}\rangle$ produces a state $|\hat{0}'\rangle = PE(U_{x,\alpha}, \theta, \varepsilon)|\hat{0}\rangle$ of the form:

$$\begin{aligned}|\hat{0}'\rangle &= \sum_{j=1}^{m} \beta_j |\psi_j\rangle|w_j\rangle_P = \sum_{j:\varphi_j \neq 0} \beta_j |\psi_j\rangle|w_j\rangle_P + \sum_{j:\varphi_j = 0} \beta_j |\psi_j\rangle|0\rangle_P \\ &= \sum_{j:\varphi_j \neq 0} \beta_j |\psi_j\rangle|w_j\rangle_P + \frac{1}{\sqrt{\mu_x}}|\Psi_x\rangle|0\rangle_P.\end{aligned}$$

Here we have used that phase estimation correctly identifies 0-phases. We use the other property of the Phase Estimation Algorithm when we project into the $|0\rangle_P$ state in the phase register.

$$\begin{aligned}|0\rangle\langle 0|_P|\hat{0}'\rangle &= \left( \frac{1}{\sqrt{\mu_x}}|\Psi_x\rangle + \sum_{\substack{j:|\varphi_j| \leq \theta \\ \varphi_j \neq 0}}^{m} \beta_j |\psi_j\rangle\langle 0|w_j\rangle_P + \sum_{j:|\varphi_j| > \theta}^{m} \beta_j |\psi_j\rangle\langle 0|w_j\rangle_P \right) |0\rangle_P \\ &=: \left( \frac{1}{\sqrt{\mu_x}}|\Psi_x\rangle + |\eta\rangle + |\xi\rangle \right) |0\rangle_P. \qquad (3.41)\end{aligned}$$

Here, the state $|\xi\rangle$ has norm $\||\xi\rangle\|^2 \leq \varepsilon$ because we are performing phase estimation with precision $\theta$ and accuracy $\varepsilon$. We use the Effective Spectral Gap Lemma to bound the norm of $|\eta\rangle$ as:

$$\||\eta\rangle\|^2 \leq \sum_{\substack{j:|\varphi_j| \leq \theta \\ \varphi \neq 0}}^{m} |\beta_j|^2 = \left\| \frac{\sqrt{\nu_x}}{1 + \frac{\alpha^2}{w_+(x)}} P_\theta^x \Pi_x |\phi_x\rangle \right\|^2 \leq \left(1 + \alpha^2 \widetilde{w}_-(x)\right) \frac{\theta^2}{4}.$$

The probability that we obtain $|0\rangle_P$ when we measure the phase register is:

$$\frac{1}{\mu_x} \leq \left\| |0\rangle\langle 0|_P |\hat{0}'\rangle \right\|^2 = \left\| \frac{1}{\sqrt{\mu_x}}|\Psi_x\rangle + |\eta\rangle + |\xi\rangle \right\|^2 \leq \left(1 + \alpha^2 \widetilde{w}_-(x)\right) \frac{\theta^2}{4} + \frac{1}{\mu_x} + \varepsilon. \qquad (3.42)$$

At this point, we choose the precision of phase estimation to be $\theta = \sqrt{\frac{4\varepsilon}{1+\alpha^2 \widetilde{w}_-(x)}}$. Altogether, the probability of obtaining $|0\rangle_P$ is:

$$\frac{1}{\mu_x} \leq \left\| |0\rangle\langle 0|_P |\hat{0}'\rangle \right\|^2 \leq \frac{1}{\mu_x} + 2\varepsilon, \tag{3.43}$$

and the post-measurement state will be:

$$\frac{|0\rangle\langle 0|_P |\hat{0}'\rangle}{\left\| |0\rangle\langle 0|_P |\hat{0}'\rangle \right\|} = \frac{1}{\sqrt{\mu_x}\left\| |0\rangle\langle 0|_P |\hat{0}'\rangle \right\|} |\Psi_x\rangle|0\rangle_P + \frac{|\eta\rangle + |\xi\rangle}{\left\| |0\rangle\langle 0|_P |\hat{0}'\rangle \right\|} |0\rangle_P. \tag{3.44}$$

In particular, the amplitude on the state $|\Psi_x\rangle$ will be at least $\frac{1}{\sqrt{1+2\varepsilon\mu_x}}$ and the amplitude on the garbage state $|\eta\rangle + |\xi\rangle$ will be at most $\sqrt{\mu_x 2\varepsilon}$. We see now that our choice of $\alpha$ will affect us in two ways. First, it determines the probability of obtaining $|0\rangle_P$ if we measure the phase register after phase estimation. Second, it determines the "quality" of the output state if we obtain $|0\rangle_P$ if we measure the phase register after phase estimation.

Let $\mathcal{D}_\alpha$ denote the 2-step procedure consisting of:

1. Apply $PE(U_{x,\alpha}, \theta, \varepsilon)$ to $|\hat{0}\rangle$ with $\theta = \sqrt{\frac{4\varepsilon}{1+\alpha^2\widetilde{w}_-(x)}}$.

2. measure the phase register in the computational basis.

We say that $\mathcal{D}$ *succeeds* if the outcome of the measurement is $|0\rangle_P$. The probability of success and the post-measurement outcome is what we just computed.

The optimal choice would be to pick $\alpha = \sqrt{w_+(x)}$, because then the success probability would be $\sim 1/2$ and the output state would be $\varepsilon$ close to $|\Psi_x\rangle$. Since we do not know $\sqrt{w_+(x)}$ we will use Lemma 47 to obtain an estimate $\alpha_*^2 \in \left(\frac{w_+(x)}{2}, 2w_+(x)\right)$.

At last, we few, we merry few, are ready to finish the proof. Running $\mathcal{D}$ on $PE(U_{x,\alpha_*}, \theta, \varepsilon)$ with precision $\theta = \mathcal{O}\left(\sqrt{\frac{\varepsilon}{\alpha_*^2\widetilde{w}_-(x)}}\right) = \mathcal{O}\left(\sqrt{\frac{\varepsilon}{w_+(x)\widetilde{w}_-(x)}}\right)$ and accuracy $\varepsilon$ will succeed with probability $p_{\text{succ}} \geq 1/3$ (because $\alpha_*^2 \geq \frac{w_+(x)}{2}$), and produce a state $|\psi\rangle$ such that $\||\psi\rangle - |\Psi_x\rangle\|^2 \leq \mathcal{O}(\varepsilon)$.

Since $\alpha_*^2 \leq 2w_+(x)$, with a further probability of $1/3 - \varepsilon$, this state $|\psi\rangle$ can collapse onto a state $|\tilde{w}\rangle$ such that $\left\| |\tilde{w}\rangle - \frac{|w_x\rangle}{\||w_x\rangle\|} \right\|^2 \leq \mathcal{O}(\varepsilon)$ by projecting

onto the space perpendicular to $|\hat{0}\rangle$.

The total cost of producing $|\tilde{w}\rangle$ is the cost of running $\mathcal{D}_\alpha$ on $PE(U_{x,\alpha_*}, \theta, \varepsilon)$, which is $\mathcal{O}\left(\sqrt{\frac{w_+(x)\widetilde{w}_-(x)}{\varepsilon}}\right)$. □

In the last theorem, we used Lemma 47 to obtain an estimate $\alpha_*^2 \in \left(\frac{w_+(x)}{2}, 2w_+(x)\right)$. We now give the proof of that statement.

*Proof of Lemma 47.* The proof follows parallel to the proof of Theorem 48. Recall that we defined $\mathcal{D}_\alpha$ to be the 2-step procedure consisting of:

1. Apply $PE(U_{x,\alpha}, \theta, \varepsilon)$ to $|\hat{0}\rangle$ with $\theta = \sqrt{\frac{4\varepsilon}{1+\alpha^2\widetilde{w}_-(x)}}$.
2. measure the phase register in the computational basis.

We say that $\mathcal{D}$ *succeeds* if the outcome of the measurement is $|0\rangle_P$. The probability of success and the post-measurement outcome are in equations (3.43) and (3.44).

Thankfully, the success probability is extremely well behaved with regards to $\alpha$ and $\varepsilon$. Assume a constant but sufficiently small accuracy, say $\varepsilon = \frac{1}{200}$. Then, the success probability of $\mathcal{D}_\alpha$ is:

$$\frac{1}{1+\frac{w_+(x)}{\alpha^2}} \leq p_{\text{succ}}(\mathcal{D}_\alpha) \leq \frac{1}{1+\frac{w_+(x)}{\alpha^2}} + \frac{1}{100}.$$

Now, consider the three following ranges of $\alpha$.

$$\text{If } \alpha^2 \leq \frac{w_+(x)}{4}, \text{ then } p_{\text{succ}}(\mathcal{D}_\alpha) \in \left[0, \frac{1}{5} + \frac{1}{100}\right] = \left[0, \frac{21}{100}\right], \tag{3.45}$$

$$\text{If } \alpha^2 \in \left[\frac{w_+(x)}{4}, \frac{w_+(x)}{2}\right], \text{ then } p_{\text{succ}}(\mathcal{D}_\alpha) \in \left[\frac{19}{100}, \frac{1}{3} + \frac{1}{100}\right] \subseteq \left[\frac{19}{100}, \frac{35}{100}\right]. \tag{3.46}$$

$$\text{If } \alpha^2 \in \left[\frac{w_+(x)}{2}, w_+(x)\right], \text{ then } p_{\text{succ}}(\mathcal{D}_\alpha) \in \left[\frac{1}{3}, \frac{51}{100}\right] \subseteq \left[\frac{32}{100}, \frac{51}{100}\right]. \tag{3.47}$$

Let $\mathsf{Decide}(\mathcal{D}_\alpha)$ be an algorithm that with high probability returns 1 if $p_{\text{succ}}(\mathcal{D}_\alpha) \geq \frac{32}{100}$, returns 0 if $p_{\text{succ}}(\mathcal{D}_\alpha) \leq \frac{21}{100}$, and returns any bit if $p_{\text{succ}}(\mathcal{D}_\alpha) \in (\frac{21}{100}, \frac{32}{100})$. We will later construct a quantum algorithm that

does exactly this. Let us for now assume that the algorithm outputs a correct answer every time it is called.

Let $W_+$ be an upper bound for $w_+(x)$ for all $x \in f^{-1}(1)$.

**Algorithm 49.**

1. *Set* $\alpha = 1$.
2. *For* $i = 1, \ldots, \lfloor \log(W_+) \rfloor$*:*
   (a) *If* $\mathsf{Decide}(\mathcal{D}_\alpha) = 1$, **return** $2\alpha_i^2$.
   (b) *Else, set* $\alpha_{i+1}^2 = 2\alpha_i^2$.
3. **return** 1.

First, of all, notice that $w_+(x) \geq \| |w_0\rangle \|^2 = 1$, since we assume $|w_0\rangle$ to be normalized. This is not necessary, but it is convenient. Assume that in the first iteration, $\mathsf{Decide}(\mathcal{D}_\alpha)$ returns 0. Otherwise, we would already have $\alpha^2 \geq w_+(x)/4$ and so the output of the algorithm is correct. Notice that in the iteration $i = \lfloor \log(w_+(x)) \rfloor$ the algorithm will terminate with high probability since $\alpha_{\lfloor \log(w_+(x)) \rfloor} = 2^{\lfloor \log w_+(x) \rfloor} \in \left[ \frac{w_+(x)}{2}, w_+(x) \right]$, and so $\mathsf{Decide}(\mathcal{D}_\alpha)$ should output 1 by (3.47).

Let $i_*$ be the first time $\mathsf{Decide}(\mathcal{D}_\alpha)$ outputs 1. That can be because $p_{\text{succ}}(\mathcal{D}_{\alpha_{i_*}}) \geq \frac{32}{100}$ or because we got lucky and $p_{\text{succ}}(\mathcal{D}_{\alpha_{i_*}}) \in (\frac{21}{100}, \frac{32}{100})$. In either case, we know that $p_{\text{succ}}(\mathcal{D}_{\alpha_{i_*}}) > \frac{19}{100}$, and so $\alpha_{i_*}^2 > \frac{w_+(x)}{4}$ by (3.45).

Similarly, in the previous iteration the output of $\mathsf{Decide}(\mathcal{D}_{\alpha_{i_*-1}})$ was 0. That means that $p_{\text{succ}}(\mathcal{D}_{\alpha_{i_*-1}}) < \frac{32}{100}$, and so $\alpha_{i_*-1}^2 < \frac{w_+(x)}{2}$ by (3.47).

Since $\alpha_{i_*}^2 = 2\alpha_{i_*-1}^2$, we obtain

$$\alpha_{i_*^2}^2 \in \left( \frac{w_+(x)}{4}, w_+(x) \right). \tag{3.48}$$

Hence, the algorithm returns a value $\alpha_*^2 = 2\alpha_{i_*^2}^2$ in the range $\left( \frac{w_+(x)}{2}, 2w_+(x) \right)$ if every step succeeds. Let us now define $\mathsf{Decide}(\mathcal{D}_\alpha)$ exactly and show that this happens with high probability.

Consider the procedure $\mathcal{D}_\alpha$, with $\varepsilon = 1/200$, but instead of measuring in the last step, we apply the unitary that maps $|0\rangle_P \mapsto |0\rangle$, and maps all phases different from $|0\rangle_P$ to $|1\rangle$. This would create a state of the form

$$\sqrt{p(\alpha)}|0\rangle|\psi_{good}\rangle + \sqrt{1-p(\alpha)}|1\rangle|\psi_{bad}\rangle,$$

where $p(\alpha)$ is precisely $p_{\text{succ}}(\mathcal{D}_\alpha)$. Then, we can apply the Amplitude Discrimination Algorithm from Corollary 12 to distinguish the case $p_{\text{succ}}(\mathcal{D}_\alpha) \geq 32/100 = p_1$ from the case $p_{\text{succ}}(\mathcal{D}_\alpha) \leq 21/100 =: p_0$ using $\mathcal{O}\left(\frac{\sqrt{p_0}}{p_1-p_0}\right) = \mathcal{O}(1)$ calls to $PE(U_{x,\alpha}, \theta, \varepsilon)$. This algorithm succeeds with probability $3/4$, which is not enough, so we repeat it $\log\log(W_+^C)$ times for some constant $C$ and take the majority. We call this procedure $\mathsf{Decide}(\mathcal{D}_\alpha)$. The success probability of one run of $\mathsf{Decide}(\mathcal{D}_\alpha)$ is $1 - \frac{1}{C\log W_+}$. Since Algorithm 49 makes at most $\log w_+(x)$ calls to $\mathsf{Decide}(\mathcal{D}_\alpha)$, the total success probability is

$$\left(1 - \frac{1}{C\log W_+}\right)^{\log w_+(x)} = 1 - \Omega\left(\frac{\log w_+(x)}{\log W_+}\right). \tag{3.49}$$

Thus, the cost of obtaining an estimate $\alpha_*^2 \in \left(\frac{w_+(x)}{2}, 2w_+(x)\right)$ with bounded error is $\mathcal{O}(\log w_+(x))$ times the cost of $\mathsf{Decide}(\mathcal{D}_\alpha)$, which is $\log\log W_+$ times the cost of $PE\left(U_{x,\alpha}, \theta, \varepsilon = \frac{1}{200}\right)$ with $\theta = \sqrt{\frac{4\varepsilon}{1+\alpha^2\widetilde{w}_-(x)}}$ and $1 \leq \alpha^2 \leq w_+(x)$. We conclude that the total number of calls to $U_{x,\alpha}$ is

$$\mathcal{O}\left(\sqrt{w_+(x)\widetilde{w}_-(x)}\log(w_+(x))\log\log W_+\right).$$

□

The error dependence of Theorem 48 can be exponentially improved if we have a lower bound for the phase gap of $U_{x,\alpha}$. We have already discussed in Section 3.2.3 how Ito and Jeffery give an algorithm for span programs that estimates the witness size with query complexity $\widetilde{\mathcal{O}}\left(\sqrt{\frac{w_+(x)\max_x \widetilde{w}_-(x)}{\varepsilon^{3/2}}}\right)$ (see Theorem 29). Observe how this is not exactly the same complexity we achieve in Lemma 47. What we did not mention is that they also give an algorithm that estimates the witness sizes $w_\pm(x, P)$ with complexity $\widetilde{\mathcal{O}}\left(\frac{1}{\varepsilon}\frac{\sqrt{w_\pm(x)}}{\Delta(U(x,\mathcal{R}))}\right)$, where $\Delta(U(x, P))$ is the smallest non-zero phase of the span program unitary $U(x, P)$.

In the last theorem of the chapter, we modify the proof of Theorem 48 to give a second algorithm that approximately constructs the optimal positive witness. The algorithm makes use of the true phase gap of the span program

unitary to achieve an exponentially better dependency in the approximation error $\varepsilon$, but has different dependence on other parameters. Before that, we prove a lemma that relates the phase gaps of $U_{x,\alpha}$ and $U(x,\mathcal{R})$.

**Lemma 50.** *Let $\mathcal{R} = (\mathcal{H}, \{\mathcal{H}(x)\}_{x\in X}, \mathcal{K}, |w_0\rangle)$ be a reflection program, and let $x$ be a positive input for $\mathcal{R}$. Define $U(x,\mathcal{R}) = (2\Pi_{\mathcal{K}} - I)\,(2\Pi_{\mathcal{H}(x)} - I)$ and $U_{x,\alpha} = (2\Pi_x - I)(2\Lambda^\alpha - I)$. Then, for any $\alpha > 0$, $\Delta(U_{x,\alpha}) \geq \Delta(U(x,\mathcal{R}))$.*

*Proof.* In this lemma, we deal with two different unitaries which do not act on the same space. Indeed, $U(x,\mathcal{R})$ acts on the $\mathcal{H}$ space of $\mathcal{R}$, while $U_{x,\alpha}$ acts on a slightly larger space $\mathcal{H}' := \mathcal{H} \oplus \mathrm{span}\{|\hat{0}\rangle\}$. Therefore, we have to be very careful when using perpendicular spaces. For that reason, we reserve the overline and $\perp$ notations for perpendicularity in $\mathcal{H}'$ and $\mathcal{H}$ respectively. In equations, let $A \subseteq \mathcal{H}'$ be a subspace, then $\overline{\Pi}_A := I_{\mathcal{H}'} - \Pi_A$ and $\Pi_{A^\perp} = I_{\mathcal{H}} - \Pi_A$.

It is an immediate consequence of Jordan's Lemma proven in Corollary 2 that the phase gap $\Delta(U)$ of a unitary $U = (2\Pi_A - I)(2\Pi_B - I)$ is related to the smallest non-zero singular value of its discriminant $D = \Pi_A\Pi_B$, $\sigma_{\min}(D)$ through the equality

$$\sin\left(\frac{\Delta(-U)}{2}\right) = \sigma_{\min}(D). \tag{3.50}$$

Alternatively, we can write this last equation as $\sin\left(\frac{\Delta(U)}{2}\right) = \sigma_{\min}(\widetilde{D}')$, where $\widetilde{D}' = \overline{\Pi}_A\Pi_B$. Now, choosing $A$ and $B$ such that $\Lambda^\alpha = \Pi_A$, and $\Pi_x = \Pi_B$, we have that

$$\sin\left(\frac{\Delta(U_{x,\alpha})}{2}\right) = \sigma_{\min}\left(\widetilde{D}'\right),$$

where $\widetilde{D}' := \overline{\Lambda^\alpha}\Pi_x$. Let us look more closely at this operator $\widetilde{D}'$. Let $|\Psi_0^\alpha\rangle$ be the vector define in Equation (3.33).

$$\begin{aligned}\widetilde{D}' =& \overline{\Lambda}^\alpha\Pi_x = (I_{\mathcal{H}'} - \Lambda^\alpha)\,\Pi_x = \left(I_{\mathcal{H}'} - \Pi_{\mathcal{K}} - \Pi_{|\Psi_0^\alpha\rangle}\right)\Pi_x \\ =& \overline{\Pi}_{\mathcal{K}}\Pi_x - \Pi_{|\Psi_0^\alpha\rangle}\Pi_x = \left(I_{\mathcal{H}'} - \Pi_{|\Psi_0^\alpha\rangle}\right)\overline{\Pi}_{\mathcal{K}}\Pi_x = \overline{\Pi}_{|\Psi_0^\alpha\rangle}\overline{\Pi}_{\mathcal{K}}\Pi_x.\end{aligned} \tag{3.51}$$

In the last line we used that $|\hat{0}\rangle$ and $|w_0\rangle$ are orthogonal to $\mathcal{K}$, and so $|\Psi_0^\alpha\rangle \in \overline{\mathcal{K}}$. Let us denote the last product of projectors as $\widetilde{D} := \overline{\Pi}_{\mathcal{K}}\Pi_x$, and consider

the state

$$|\psi_x\rangle = \frac{|\hat{0}\rangle + \frac{1}{\alpha}|w_x\rangle}{\sqrt{1 + \frac{\||w_0\rangle\|^2}{\alpha}}},$$

where $|w_x\rangle$ is an optimal positive witness for $x$ in $\mathcal{R}$. Our goal now will be to relate the spectral gap of $\widetilde{D}$, i.e. its smallest non-zero singular value, with that of $\widetilde{D}'$. Observe that

$$\widetilde{D}|\psi_x\rangle = \overline{\Pi}_{\mathcal{K}}\Pi_x \frac{\left(|\hat{0}\rangle + \frac{|w_x\rangle}{\alpha}\right)}{\sqrt{\mu_0}} = \frac{|\hat{0}\rangle + \frac{\overline{\Pi}_{\mathcal{K}}|w_x\rangle}{\alpha}}{\sqrt{\mu_0}} \tag{3.52}$$

$$= \frac{|\hat{0}\rangle + \frac{|w_0\rangle}{\alpha}}{\sqrt{\mu_0}} = |\Psi_0^\alpha\rangle. \tag{3.53}$$

This equality relies on the fact that $|w_x\rangle$ is an exact positive witness, and so $|w_x\rangle \in \mathcal{H}(x)$ and $\overline{\Pi}_{\mathcal{K}}|w_x\rangle = \Pi_{\mathcal{K}^\perp}|w_x\rangle = |w_0\rangle$. Most importantly, it shows that $|\Psi_0^\alpha\rangle$ is in the image of $\widetilde{D}$.

This is almost identical to [IJ19, Theorem 3.11], so we follow the proof there.

Since $|\Psi_0^\alpha\rangle$ is in the image of $\widetilde{D}$, we can find an orthogonal basis of $\widetilde{D}$ of the form $\{|\phi_0\rangle = |\Psi_0^\alpha\rangle, |\phi_1\rangle, \ldots, |\phi_{r-1}\rangle\}$, and we can write $\widetilde{D}$ as $\widetilde{D} = \sum_{i=0}^{r-1}|\phi_i\rangle\langle v_i|$ for $|v_i\rangle = \widetilde{D}^\dagger|\phi_i\rangle$. Then, $\widetilde{D}' = \overline{\Pi}_{|\Psi_0^\alpha\rangle}\widetilde{D} = \sum_{i=1}^{r-1}|\phi_i\rangle\langle v_i|$, and so $\operatorname{col}\widetilde{D}' = \operatorname{span}\left\{|\phi\rangle \in \operatorname{col}\widetilde{D} : \langle\phi|\Psi_0^\alpha\rangle = 0\right\}$. Hence, the spectral gap of $\widetilde{D}'$ is

$$\sigma_{\min}(\widetilde{D}') = \min_{|u\rangle\in\operatorname{col}\widetilde{D}'} \frac{\left\|\langle u|\widetilde{D}'\right\|}{\||u\rangle\|} = \min_{|u\rangle\in\operatorname{col}\widetilde{D}:\langle u|\Psi_0^\alpha\rangle=0} \frac{\left\|\langle u|\overline{\Pi}_{|\Psi_0^\alpha\rangle}\widetilde{D}\right\|}{\||u\rangle\|}$$

$$= \min_{|u\rangle\in\operatorname{col}\widetilde{D}:\langle u|\Psi_0^\alpha\rangle=0} \frac{\left\|\langle u|\widetilde{D}\right\|}{\||u\rangle\|} \geq \min_{|u\rangle\in\operatorname{col}\widetilde{D}} \frac{\left\|\langle u|\widetilde{D}\right\|}{\||u\rangle\|} = \sigma_{\min}(\widetilde{D}).$$

Finally, we relate $\widetilde{D}$ back to $U(x, \mathcal{R})$. Notice that $\mathcal{H}'$ is only slightly bigger than $\mathcal{H}$, and that $\overline{\Pi}_{\mathcal{K}} = \Pi_{\mathcal{K}^\perp} + |\hat{0}\rangle\langle\hat{0}|$. Together with the fact that $\Pi_x$ is defined as $\Pi_x = \Pi_{\mathcal{H}(x)} + |\hat{0}\rangle\langle\hat{0}|$ we have that

$$\widetilde{D} = \Pi_{\mathcal{K}^\perp}\Pi_{\mathcal{H}(x)} + |\hat{0}\rangle\langle\hat{0}|,$$

hence $\sigma_{\min}(\widetilde{D}) = \sigma_{\min}(\Pi_{\mathcal{K}^\perp}\Pi_{\mathcal{H}(x)})$. From this and Equation (3.50) follows the result $\Delta(U_{x,\alpha}) \geq \Delta(U(x,\mathcal{R}))$. □

Without further ado, the algorithm.

**Theorem 51.** *Let $\mathcal{R}$ be a reflection program deciding a function $f$ and let $x$ be a positive input. Let $|w_x\rangle$ be an optimal positive witness for $x$ in $\mathcal{R}$, $w_+(x) = \||w_x\rangle\|^2$ be the exact positive witness size of $x$ for $\mathcal{R}$. Let $U(x,\mathcal{R}) = (2\Pi_{\mathcal{K}} - I)(2\Pi_{\mathcal{H}(x)} - I)$ and assume that its phase gap $\Delta(U(x,\mathcal{R}))$ is known. Then there exists a procedure that succeeds with probability $\geq 2/9$ and prepares a state $|\tilde{w}\rangle$ such that $\left\||\tilde{w}\rangle - \frac{|w_x\rangle}{\||w_x\rangle\|}\right\|^2 = \mathcal{O}(\varepsilon)$ using $\mathcal{O}\left(\frac{\log w_+(x) \log\log W_+}{\Delta(U(x,\mathcal{R}))}\right) + \mathcal{O}\left(\frac{\log \frac{1}{\varepsilon}}{\Delta(U(x,\mathcal{R}))}\right)$ controlled calls to $U_{x,\alpha}$, where $W_+$ is an upper bound on $w_+(x)$, and $\alpha > 0$ can change.*

*Proof.* The proof proceeds parallel to that of Theorem 48 up until Eq. (3.54). Nonetheless, let us remind the reader of the setup. We start with a unitary $U_{x,\alpha} = (2\Pi_x - I)(2\Lambda^\alpha - I)$, and a state $|\hat{0}\rangle$, and apply the procedure $\mathcal{D}$[6] that now performs $PE(U_{x,\alpha}, \theta, \varepsilon)$ with precision $\theta = \Delta(U(x,\mathcal{R}))$ and accuracy $\varepsilon$ to the state $|\hat{0}\rangle$, and then measures the phase register. If the procedure succeeds and we obtain $|0\rangle_P$, the unnormalized output is:

$$\begin{aligned}|0\rangle\langle 0|_P|\hat{0}'\rangle &= \left(\frac{1}{\sqrt{\mu_x}}|\Psi_x\rangle + \sum_{\substack{j:|\varphi_j|\leq\theta \\ \varphi_j\neq 0}}^{m} \beta_j|\psi_j\rangle\langle 0|w_j\rangle_P + \sum_{j:|\varphi_j|>\theta}^{m} \beta_j|\psi_j\rangle\langle 0|w_j\rangle_P\right)|0\rangle_P \\ &=: \left(\frac{1}{\sqrt{\mu_x}}|\Psi_x\rangle + |\eta\rangle + |\xi\rangle\right)|0\rangle_P. \qquad (3.54)\end{aligned}$$

The difference being that, this time, $|\eta\rangle = 0$ because by Lemma 50, $\Delta(U_{x,\alpha}) \geq \Delta(U(x,\mathcal{R}))$ and so $|\hat{0}\rangle$ is not supported in eigenspaces of $U_{x,\alpha}$ with non-zero eigenphase $\leq \theta = \Delta(U(x,\mathcal{R}))$. The state $|\xi\rangle$ still has norm $\||\xi\rangle\|^2 \leq \varepsilon$ by Theorem 9. We conclude that the success probability of $\mathcal{D}$ is

$$\frac{1}{\mu_x} \leq \left\||0\rangle\langle 0|_P|\hat{0}'\rangle\right\|^2 \leq \frac{1}{\mu_x} + \varepsilon, \qquad (3.55)$$

[6]Since the precision of phase estimation does not depend on $\alpha$, we dispense with it.

where $\mu_x = 1 + \frac{w_+(x)}{\alpha^2}$. The post-measurement state is

$$\frac{|0\rangle\langle 0|_P|\hat{0}'\rangle}{\left\||0\rangle\langle 0|_P|\hat{0}'\rangle\right\|} = \frac{1}{\sqrt{\mu_x}\left\||0\rangle\langle 0|_P|\hat{0}'\rangle\right\|}|\Psi_x\rangle + \frac{1}{\left\||0\rangle\langle 0|_P|\hat{0}'\rangle\right\|}|\xi\rangle. \tag{3.56}$$

Observe that the success probability is almost identical to that of Equation (3.43), and the post measurement state is similar to that of Equation (3.44). The only differences are that now we have reached this point doing phase estimation to precision $\Delta(U(x,\mathcal{R}))$ instead of $\sqrt{\frac{\varepsilon}{w_+(x)\widetilde{w}_-(x)}}$, and that the garbage state is slightly different.

Therefore, we can recycle the machinery detailed in Lemma 47, in particular Algorithm 49 and $\mathsf{Decide}(\mathcal{D})$. As was the case there, the optimal choice of $\alpha$ is $\alpha = \sqrt{w_+(x)}$, and an estimate $\alpha_* \in \left[\sqrt{w_+(x)/2}, \sqrt{2w_+(x)}\right]$ can be obtained with $\mathcal{O}(\log w_+(x))$ calls to $\mathsf{Decide}(\mathcal{D})$, which makes $\mathcal{O}(\log\log W_+)$ calls to phase estimation on $U_{x,\alpha}$ with precision $\theta = \mathcal{O}(\Delta(U(x,\mathcal{R})))$ and accuracy $\epsilon = 1/100$.

Once we have that estimate, we run $\mathcal{D}$, which will succeed with probability $\geq 1/3$ and generate a state $|\psi\rangle$ that can, with probability 2/3, be further projected onto a state $|w\rangle$ such that $\left\||w\rangle - \frac{|w_x\rangle}{\||w_x\rangle\|}\right\|^2 \leq \varepsilon$.

The total cost of these operations is $\mathcal{O}\left(\frac{\log w_+(x)\log\log W_+}{\Delta(U(x,\mathcal{R}))}\right) + \mathcal{O}\left(\frac{\log\frac{1}{\varepsilon}}{\Delta(U(x,\mathcal{R}))}\right)$. □

## 3.5 Discussion

This chapter takes the reader on a journey through the theory of span programs, from its first application to quantum computing to the newest understanding of the topic. We do not claim, however, that this is the only way other people think about span programs, or the only notation they use, far from it.

Still, we hope it has become clear, as the formulation has progressed, that span programs are a good match to bridge the gap between decision problems and quantum algorithms. They allow us to encode and understand these functions in geometrical and in linear algebraic terms. Moreover, the algorithms that we compile out of them are all cut from the same cloth of amplitude amplification, phase estimation and products of reflections. Considering that all functions admit a query optimal span program, it follows

that phase estimation, amplitude amplification, and products of reflections, arranged in some very regular and predictable way are sort of universal subroutines.

By this we mean that they suffice to construct query-optimal algorithms. The informal truism that all quantum query algorithms are clever combinations of the same three or four pieces is essentially, well, true. In the next chapter we will see if and when the truism extends to time complexity (it does, sometimes).

Reflection programs, our first innovation in this chapter, serve the same conceptual role as span programs. They bring a cleaner notation that simplifies the proofs of the algorithms in Section 3.4, but at the end of the day, the proofs could be adapted to the span program notation of [IJ19].

The real *raison d'être* of reflection programs, however, is that they give us greater insight of the inner workings of a span program. In particular, they make the geometrical and operational interpretations of Sections 3.3.1 and 3.3.2 possible. The original goal was to build an increased intuition that would allow us to solve Conjecture 40. The fact that we state it as a conjecture says it all about the author's success in that regard, but is not the end of the story. As we write these lines, our colleague and co-author Arjan Cornelissen claims to have found a way to decide a $\lambda$-approximating reflection program with $\widetilde{\mathcal{O}}\left(\sqrt{W_-\widetilde{W}_+}/(1-\lambda)\right)$ calls to $U_x$. To the best of the author's knowledge, who has been privy to that manuscript, that proof is correct.

The algorithm that we give in Section 3.4.1 decides a function $f$ that is $\lambda$-approximated by a reflection program for any $\lambda \in (0,1)$. The complexity of the algorithm, depending on the positive and min. error negative witness sizes. That means that in order to apply this algorithm, the user must bound the min. error witness size, i.e. find a min. error witness. This can be difficult. Still, we built the algorithm because we felt it was necessary to give an algorithm for approximate reflection programs, even if it is not much different from the current algorithms for approximate span programs. More importantly, the algorithms for witness generation are built on the back of this decision algorithm.

In the next chapter, we will construct a span program out of a query algorithm and show that it $\lambda$-approximates the same function the algorithm

decides. We will do this by finding $\epsilon$-approximate witnesses with $\epsilon \leq \frac{\lambda}{W_+}$. But we will *not* then go and find min. error witnesses. This means that we will need to use the algorithm in Corollary 46 instead of that in Theorem 45.

Alas, this is the current state of affairs. We can have an algorithm that decides a function $\lambda$-approximated by a reflection program $\mathcal{R}$ for every $\lambda \in [0, 1)$ but requires us to bound the min. error negative witness size of $f$ for $\mathcal{R}$. Or we can have an algorithm that decides a function $\lambda$-approximated by a reflection program $\mathcal{R}$ and only requires us to bound the $\epsilon$-approximate negative witness size of $f$ for $\mathcal{R}$ with $\epsilon = \lambda/W_+$, but works only for $\lambda \in [0, 1/2)$. It remains an open problem how to construct an algorithm that works for every $\lambda \in (0, 1)$ *and* whose complexity does not require the user to find min. error witnesses.

We closed this chapter with two algorithms for state generation that are of independent interest. In Chapter 5, we will discuss a possible application of state generation to finding short-*ish* paths. Another possible direction of future research is to understand if these algorithms can be used to reason about the state generation problem.

The connection between span programs and adversary bounds is only completely mapped for two-outcome functions. It is an open question whether our formalism can be used to better understand the generalized adversary bound and function evaluation through the non-binary span programs introduced in [BT20]. We have discussed how span programs correspond to dual solutions to the general adversary bound. Much less is known about span programs and dual solutions to the positive adversary bound (a weaker version of the former). It is a well established fact that this positive adversary bound is strictly weaker than the general bound in some cases. Span programs could be an alternative way to understand what goes wrong and when.

# Chapter 4

# Span programs and time complexity

## 4.1 Overview

This chapter is based on joint work with Arjan Cornelissen, Stacey Jeffery, and Maris Ozols [CJO+20]. We make progress in understanding the relationship between span programs and quantum time complexity by showing that for any decision problem, it is possible to design an almost time-optimal quantum algorithm (i.e., optimal up to polylogarithmic factors) using the span program framework. We do this by giving a construction that maps any quantum algorithm to a span program. The problem of mapping an arbitrary quantum algorithm to a span program has been considered previously. In [Rei09], Reichardt showed how to convert any quantum query algorithm with one-sided error to a span program whose complexity matches the algorithm's query complexity. This was extended to the standard case of (two-sided) bounded error quantum query algorithms in [Jef20].

We modify their construction to take any quantum algorithm with time complexity $T$ and query complexity $S$, and map it to a span program with complexity $\mathcal{O}(S)$, such that the unitary $U$ associated with the span program can be implemented in time $T/S$, up to polylog$(T)$ factors, meaning that the algorithm compiled from the span program has time complexity $\widetilde{\mathcal{O}}(T)$.

The major theoretical implication of this result is that for any decision problem, one can find a quantum algorithm that is optimal in not only space and query complexity, but also time complexity, using the span program

framework. Moreover, using our construction we prove that these three flavors of optimality can be achieved simultaneously. Thus, we can definitively say that span programs *are* quantum algorithms.

**Algorithms as inputs** This construction takes an algorithm and uses it to define the elements $\mathcal{H}, \mathcal{V}, A, |\tau\rangle$ that form a span program. We discuss what we mean by this in Section 4.2, where the access to an algorithm is defined through three distinct oracles. The span program algorithm then uses these oracles as subroutines to decide a function.

**Time complexity of implementing a reflection program** The algorithms in Section 3.4 for reflection programs have their complexities specified in terms of queries to unitaries $U_{x,\alpha}$. In Section 4.3, we give an account of the different subroutines necessary to implement the reflection program algorithms from Chapter 3. In this chapter we will only make use of the algorithm in Corollary 46, and only to evaluate span programs, which are a special case of reflection programs. Therefore, reflection programs will not feature in this chapter anywhere after Section 4.3.

**Implementing subspaces** As we said, we will define a construction mapping any algorithm to a span program. In the analysis of the time complexity of the span program's algorithm, we identify an input-dependent subspace of the state space which we are guaranteed to stay within throughout the execution of the span program algorithm. This allows us to drastically decrease the implementation cost of some of its subroutines. We refer to this subspace as the *implementing subspace*, and we believe that this technique can be used to analyze the time complexity of a wider variety of algorithms than those considered in this text.

**Span programs for bounded error algorithms** As we said before, the problem of mapping an arbitrary quantum algorithm to a span program has been considered previously in [Rei09] and extended in [Jef20], where an explicit mapping from algorithms to span programs was shown to map query complexity to span program complexity. We extend these results to time complexity in Sections 4.4, and 4.5, showing that a quantum algorithm with time complexity $T$ and query complexity $S$ can be mapped to a span program

that, if compiled back into an algorithm, can be implemented in time $\widetilde{\mathcal{O}}(T)$, and $\mathcal{O}(S)$ queries.

It is natural to ask if our result, and in particular our construction mapping quantum algorithms to span programs, is of practical relevance since normally quantum algorithms themselves are the end goal in designing span programs. One reason that it can be useful to convert a quantum algorithm into a span program is that span programs compose very nicely [Rei09] – more so than quantum algorithms. It can thus be desirable to convert several quantum algorithms to span programs, compose them, and then convert the result back to a quantum algorithm.

**Variable-time search** To illustrate this, we improve a result of Ambainis [Amb10] for variable-time quantum search in Section 4.6. Given $n$ bounded-error query algorithms evaluating Boolean functions $f_1, \ldots, f_n$ with costs $C_1, \ldots, C_n$, respectively, Ambainis provides a way to evaluate the function $f = \bigvee_{i=1}^n f_i$ with cost $\mathcal{O}(\sqrt{\sum_{i=1}^n C_i^2})$. We left the notion of *cost* purposefully ambiguous here, as Ambainis's construction allows for defining any notion of cost associated with providing uniform access to the algorithms, i.e., the ability to apply the gate that is executed at any given time step in any of the algorithms. The resulting algorithm depends on the notion of cost selected, and from Ambainis's construction, it is not apparent how one would obtain the claimed scaling in multiple notions of cost simultaneously. Moreover, Ambainis's construction assumes that all instance-independent gates, i.e., all operations that are not part of the original algorithms, have cost zero, which means that a proper analysis of the time complexity of the resulting algorithm evaluating $f$ is lacking.

Our result improves on Ambainis's result in the following manner. If the $n$ original algorithms have query complexity $S_1, \ldots, S_n$, time complexity $T_1, \ldots, T_n$, and we have efficient uniform access to them, then we can evaluate $f$ with bounded error with $\widetilde{\mathcal{O}}(\sqrt{\sum_{i=1}^n S_i^2})$ queries and $\widetilde{\mathcal{O}}(\sqrt{\sum_{i=1}^n T_i^2})$ gates. Moreover, the number of auxiliary qubits introduced is at most polylogarithmic in $T_{\max} = \max_{i\in[n]} T_i$ and $n$. Thus, we achieve the desired scaling in the query and time complexities simultaneously, while also counting all instance-independent gates in our analysis of the time complexity of the resulting algorithm.

We achieve this result by converting the original algorithms into span programs, which we subsequently compose using techniques from [Rei09]. We

turn the resulting composed span program back into an algorithm, reusing some ideas from [Amb10], and using our technique of implementing subspaces.

Perhaps the most interesting future direction suggested by our work is to find new algorithm composition results by turning algorithms into span programs, taking advantage of the relative ease of span program composition, and then converting the result back into an algorithm.

## 4.2 Accessing an algorithm as input

Throughout the rest of the chapter, we will consider algorithms that, among other things, take other algorithms as input. This section concerns how we model this through several oracles. The model is essentially a generalization of the one used in [Amb10].

Let $m \in \mathbb{N}$ and let $\mathcal{A} = \{\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(m)}\}$ be a set of quantum query algorithms. For every $j \in [m]$, let $T^{(j)}$ be the time complexity of $\mathcal{A}^{(j)}$, let $\mathcal{S}^{(j)} \subseteq [T^{(j)}]$ be the set of time steps at which $\mathcal{A}^{(j)}$ performs queries to the input, let $U_1^{(j)}, \ldots, U_{T^{(j)}}^{(j)}$ be the sequence of unitaries in $\mathcal{A}^{(j)}$, and suppose that $\mathcal{A}^{(j)}$ evaluates a function $f_j : X^{(j)} \subseteq \{0,1\}^{n^{(j)}} \to \{0,1\}$ with bounded error. For convenience we define $T_{\max} = \max_{j\in[m]} T^{(j)}$ and $n_{\max} = \max_{j\in[m]} n^{(j)}$, and we assume that all unitaries $U_t^{(j)}$ act on some space $\mathbb{C}^{[n_{\max}]\times\mathcal{W}}$, where the first register is large enough to hold the input bit label for any of the Boolean functions $f_j$.

We define three different oracles associated with $\mathcal{A}$. First, the *algorithm oracle*, sometimes referred to as **Select**, acts on $\mathbb{C}^{[m]\times[T_{\max}]\times[n_{\max}]\times\mathcal{W}}$ as

$$\forall j \in [m], t \in [T^{(j)}] \setminus \mathcal{S}^{(j)}, |\psi\rangle \in \mathbb{C}^{[n_{\max}]\times\mathcal{W}}, \qquad \mathcal{O}_{\mathcal{A}} : |j\rangle|t\rangle|\psi\rangle \mapsto |j\rangle|t\rangle U_t^{(j)}|\psi\rangle.$$

Second, the *query time step oracle*, which allows us to determine whether a given algorithm $\mathcal{A}^{(j)}$ performs a query at a given time step $t$, acts on $\mathbb{C}^{[m]\times[T_{\max}]}$ as

$$\forall j \in [m], t \in [T^{(j)}], \qquad \mathcal{O}_{\mathcal{S}} : |j\rangle|t\rangle \mapsto \begin{cases} -|j\rangle|t\rangle, & \text{if } t \in \mathcal{S}^{(j)}, \\ |j\rangle|t\rangle, & \text{otherwise.} \end{cases}$$

Finally, given a list of inputs $x = (x^{(1)}, \ldots, x^{(m)})$, where $x^{(j)} \in \{0,1\}^{n^{(j)}}$ is

the input to function $f_j$, the *input oracle* to $x$ acts on $\mathbb{C}^{[m]\times[n_{\max}]}$ as

$$\forall j \in [m],\ \mathcal{O}_x = \sum_{j=1}^{m} |j\rangle\langle j| \otimes \mathcal{O}_{x^{(j)}},\ \text{where } \forall i \in [n_j],\ \mathcal{O}_{x^{(j)}} : |i\rangle \mapsto (-1)^{x_i^{(j)}} |i\rangle.$$

On computational basis states that are not specified above, the behavior of the three oracles can be arbitrary.

By saying that we have *uniform access* to the set of algorithms $\mathcal{A}$, we mean that we have access to these three oracles $\mathcal{O}_{\mathcal{A}}$, $\mathcal{O}_{\mathcal{S}}$ and $\mathcal{O}_x$. Moreover, if the time complexity of implementing the oracles $\mathcal{O}_{\mathcal{A}}$ and $\mathcal{O}_{\mathcal{S}}$ is polylogarithmic in $T_{\max} = \max_{j\in[m]} T^{(j)}$ and $m$, then we say that we have *efficient uniform access* to $\mathcal{A}$.

Note that if $m = 1$, then the first register in all above oracles only contains one dimension and hence can be omitted. In that case, we drop all the superscripts and $\mathcal{O}_x$ reduces to the regular input oracle $\mathcal{O}_x$ that we defined in Equation (2.2).

These oracles fully capture the set $\mathcal{A}$ and provide an interface for the higher-level algorithms to execute the algorithms in $\mathcal{A}$ as subroutines. From a computer science point of view, one can think about these oracles as black boxes provided the user. To use our results for a particular set of algorithms $\mathcal{A}$, one has to provide implementations of these three oracles. The machinery we develop in the remainder of this text then takes care of the rest of the construction, and our analysis provides the number of calls made to these oracles, alongside with the number of extra gates used.

A natural question to ask is how difficult it is in general to implement these oracles. If the algorithms from $\mathcal{A}$ are very unstructured, then it is in general very time-consuming to implement these oracles. In that case, one could implement $\mathcal{O}_{\mathcal{A}}$ and $\mathcal{O}_{\mathcal{S}}$ by querying a quantum read-only random access memory (commonly referred to as QRAM) storing the algorithms $\mathcal{A}^{(j)}$ as lists of gates. A similar model, called *quantum random access stored-program machines* was recently formalized in [WY20].

However, quantum query algorithms that we encounter in practice can usually be described very succinctly, and we have some efficient constructive procedure to calculate what gate has to be applied in the $j$th algorithm at the $t$th time step and at what time steps the algorithms perform a query. These procedures can be used to implement the oracles $\mathcal{O}_{\mathcal{A}}$ and $\mathcal{O}_{\mathcal{S}}$ efficiently and provide us with efficient uniform access. For the query oracle, one can usually provide an efficient implementation of $\mathcal{O}_x$ as well if the individual

$\mathcal{O}_{x^{(j)}}$'s are similar to each other (think, for example, an oracle for the $AND$ of $st_i$-connectivity span programs on the same graph $G$ could be made out of the oracle for $G$). All of these constructions are always instance-dependent, though, and hence we cannot elaborate on them further without losing generality.

We conclude this section by remarking that this final argument is more generally applicable to oracular algorithms. The results about query complexity are in general most interesting and applicable in a setting where the oracles themselves can be substituted by efficient algorithms. The same goes for the uniform access model we consider here.

## 4.3 Time complexity of a span program algorithm

We now turn our attention to the time complexity of the algorithms for reflection program evaluation discussed in Theorem 45, and Corollary 46. We will only apply the contents of this section to span program algorithms (hence the title), but we prove the more general case of reflection programs for the sake of completeness. We have established already that the algorithm consists of phase estimation and amplitude amplification that calls on a particular unitary $U_x$. Since the number of calls to $U_x$ is known, the complexity of the algorithm (typically time complexity), which we will sometimes call the *cost*, will be determined by the complexity of $U_x$. In this section we break down this unitary into different subroutines dependent on $\mathcal{R}$ which we treat as black-boxes, and express the cost of the algorithm in terms of the number of calls we perform to those black-boxes, see Theorem 53. As such, this theorem is not a true analysis of the time complexity of a span program algorithm but a *meta*-analysis.[1]

Admittedly, the proof of the theorem amounts to little more than bookkeeping, but it is important for two reasons. First, it gives us an account of the resources to analyze once we settle on a particular reflection program; in other words, it makes reflection program algorithms modular and simplifies our lives. Second, it introduces the notion of *implementing subspaces*, which are of relevance beyond the setting of reflection programs.

[1] The indiscriminate use of greek prefixes is a risky business. In some countries it can get you a seat in parliament.

Before we analyze the time complexity, though, we first introduce the concept of an *implementing subspace*. This subspace depends on the particular input $x \in \{0,1\}^n$, and has the property that it is often much smaller than the ambient Hilbert space $\mathcal{H}$. Most importantly the state vector remains in this subspace throughout the execution of the reflection program algorithm. Therefore, all operations in the reflection program algorithm need only be defined in this subspace to ensure successful computation of the span program. We restrict ourselves to Boolean alphabets because it simplifies the proof here, but the statements also hold for non-Boolean alphabets.

**Definition 52** (Implementing subspace)**.** Let $\mathcal{R} = (\mathcal{H}, \{\mathcal{H}(x)\}_{x\in X}, \mathcal{K}, |w_0\rangle)$ be a reflection program that positively $\lambda$-approximates a function $f : X \subseteq \{0,1\}^n \to \{0,1\}$ with $\lambda \in [0,1)$. Let $x \in X$ and let $\mathcal{H}_x$ be a subspace of $\mathcal{H}$ such that:

1. $\Pi_{\ker(A)}\mathcal{H}_x \subseteq \mathcal{H}_x$.
2. $\Pi_{\mathcal{H}(x)}\mathcal{H}_x \subseteq \mathcal{H}_x$.
3. $|0\rangle \in \mathcal{H}_x$, where $|0\rangle$ is the all-zeros computational basis state.
4. $|w_0\rangle \in \mathcal{H}_x$.

Then we refer to $\mathcal{H}_x$ as an *implementing subspace of* $\mathcal{R}$ *for* $x$.

For any $x \in X$, a valid implementing subspace $\mathcal{H}_x$ of $\mathcal{R}$ for $x$ is $\mathcal{H}$ itself. In that case we can always implement $2|0\rangle\langle 0| - I_{\mathcal{H}}$ in complexity $\mathcal{O}(\log \dim \mathcal{H})$, by simply checking that every qubit is in the state $|0\rangle$. However, for algorithms with large space complexity, such as the element distinctness algorithm [Amb07], this is very costly, especially if we have to do it many times. In some cases, as in our main theorem in Section 4.4, we can show that the span program give there has an implementing subspace in which implementing $2|0\rangle\langle 0| - I$ is easy, thus circumventing an undesired $\log \dim \mathcal{H}$ overhead in the time complexity of the span program algorithm.

The notion of an implementing subspace is not exclusive to reflection or span program algorithms. Indeed, any algorithm with high space complexity would run into the same problem if it contains a reflection around any state (i.e., a one-dimensional subspace), even a computational basis state. This includes most quantum walk based algorithms. However, even algorithms with low space complexity could benefit from this technique.

Now, we can state the main result of this section.

**Theorem 53.** *Fix $\lambda \in [0, 1/2)$. Suppose $\mathcal{R} = (\mathcal{H}, \{\mathcal{H}(x)\}_{x\in X}, \mathcal{K}, |w_0\rangle)$ is a reflection program that positively $\lambda$-approximates a function $f : X \subseteq \{0,1\}^n \to \{0,1\}$. For all $x \in X$, let $\mathcal{H}_x$ be an implementing subspace for $\mathcal{R}$. Suppose that we have access to the following subroutines and their controlled versions:*

1. *A subroutine $\mathcal{R}_{\mathcal{K}}$ that acts on $\mathcal{H}_x$ as $2\Pi_{\mathcal{K}} - I$.*
2. *A subroutine $\mathcal{C}_{|w_0\rangle}$ that leaves $\mathcal{H}_x$ invariant and maps $|0\rangle$ to $|w_0\rangle / \||w_0\rangle\|$.*
3. *A subroutine $\mathcal{R}_{\mathcal{H}(x)}$ that acts on $\mathcal{H}_x$ as $2\Pi_{\mathcal{H}(x)} - I$.*
4. *A subroutine $\mathcal{R}_{|0\rangle}$ that acts on $\mathcal{H}_x$ as $2|0\rangle\langle 0| - I$.*

*Then we can decide $f$ with bounded error using a number of calls to the previous subroutines of order $\mathcal{O}\left(\frac{\sqrt{W_+\widetilde{W}_-}}{(1-2\lambda)^{3/2}} \log \frac{1}{1-2\lambda}\right)$, where $W_+$, $\widetilde{W}_-$ are defined in Eq. (3.15), and Eq. (3.30). The number of extra gates and auxiliary qubits used is $\mathcal{O}\left(\text{polylog}(W_+\widetilde{W}_-, \frac{1}{1-\lambda})\right)$. Finally, it suffices to merely use upper bounds on $W_+$, $\widetilde{W}_-$ and $\lambda$, if one substitutes these upper bounds in the relevant complexities.*

The purpose of Theorem 53 is to enumerate the fundamental instance-dependent operations that have to be given by the user to compile a particular reflection program algorithm. In other words, if one wants to compile a time-efficient algorithm from a reflection program, it suffices to give time-efficient implementations of the four subroutines listed in Theorem 53.

The remainder of this section is dedicated to proving Theorem 53. The proof follows easily from four lemmas which we prove first, followed by the proof of the theorem itself. Due to the structure of the algorithm, we need to extend the space $\mathcal{H}$ to $\mathcal{H}' = \mathcal{H} \oplus \text{span}\{|\hat{0}\rangle\}$. We also assume in the remainder of the section that we can reflect through the state $|\hat{0}\rangle$ in $\mathcal{O}(1)$ gates and with only $\mathcal{O}(1)$ auxiliary qubits. One way to implement this is to make the state space of the system equal to $\mathcal{H} \otimes \mathbb{C}^2$, identifying $\mathcal{H}$ with $\mathcal{H} \otimes |0\rangle$ and define $|\hat{0}\rangle = |0, 1\rangle$. Now, we leave it to the reader to check that the unitary $I_{\mathcal{H}} \otimes (2|1\rangle\langle 1| - I_2)$ acts as $(2|\hat{0}\rangle\langle\hat{0}| - I_{\mathcal{H}'})$ on $\mathcal{H}'$. Moreover, these unitaries can be implemented with $\mathcal{O}(1)$ gates and extra qubits.

The first lemma deals with the preparation of states of a certain kind.

**Lemma 54.** *Let $\alpha_0, \alpha_1, \in \mathbb{C}$ be such that $|\alpha_0|^2 + |\alpha_1|^2 = 1$, and let $|w_0\rangle$ be the minimal witness for the reflection program $\mathcal{R} = (\mathcal{H}, \{\mathcal{H}(x)\}_{x\in X}, \mathcal{K}, |w_0\rangle)$.*

*For all $x \in X$, let $\mathcal{H}_x$ be an implementing subspace. We define*

$$|\eta\rangle = \alpha_0 \frac{|w_0\rangle}{\||w_0\rangle\|} + \alpha_1 |\hat{0}\rangle.$$

*Assume that we have access to controlled versions of the following subroutines:*

1. *A subroutine $\mathcal{C}_{|w_0\rangle}$ that leaves $\mathcal{H}_x$ invariant and maps $|0\rangle$ to $|w_0\rangle / \||w_0\rangle\|$.*
2. *A subroutine $\mathcal{R}_{|0\rangle}$ that acts on $\mathcal{H}_x$ as $2|0\rangle\langle 0| - I$.*

*Let $\mathcal{H}'_x = \mathcal{H}_x \oplus \mathrm{span}\{|\hat{0}\rangle\}$. Then we can implement a circuit $\mathcal{C}_{|\eta\rangle}$ that leaves $\mathcal{H}'_x$ invariant and maps $|0\rangle$ to $|\eta\rangle$, with one call to $\mathcal{C}_{|w_0\rangle}$, two calls to $\mathcal{R}_{|0\rangle}$, and $\mathcal{O}(1)$ extra gates and auxiliary qubits.*

*Proof.* Recall that we can encode $|\hat{0}\rangle$ as $|\hat{0}\rangle = |0,1\rangle$, and identify every $|h\rangle \in \mathcal{H}$ with $|h\rangle \otimes |0\rangle$. Our mapping $\mathcal{C}_{|\eta\rangle}$ is supposed to map $|0\rangle \in \mathcal{H} \subseteq \mathcal{H}'$ to $|\eta\rangle \in \mathcal{H}'$, so it is supposed to implement $|0,0\rangle \mapsto |\eta\rangle$.

First of all, we check if the first register is in state $|0\rangle$ by preparing an auxiliary qubit in the state $|+\rangle$, and then controlled on this auxiliary qubit calling the routine $\mathcal{R}_{|0\rangle}$. If the first register was in the state $|0\rangle$, then we remain in $|+\rangle$, and if not we get a $|-\rangle$ in this qubit. Using a single Hadamard gate, we can now store in the auxiliary qubit whether the first register is in the $|0\rangle$-state.

Next, controlled on the first register being in the $|0\rangle$-state, we apply the mapping $|0\rangle \mapsto \alpha_0|0\rangle + \alpha_1|1\rangle$ to the second register. This can be implemented using $\mathcal{O}(1)$ gates, namely by implementing a controlled rotation in the plane $\mathrm{span}\{|0\rangle, |1\rangle\}$.

Now, we uncompute the first part of our computation, i.e., we uncompute the auxiliary qubit that stored whether the first register was in state $|0\rangle$. This again takes one controlled call to $\mathcal{R}_{|0\rangle}$ and $\mathcal{O}(1)$ extra gates. Observe that the total mapping has now only modified the second register when the first register was in the state $|0\rangle$. But as $|0\rangle \otimes \mathbb{C}^{\{0,1\}} = \mathrm{span}\{|0,0\rangle, |\hat{0}\rangle\} \subseteq \mathcal{H}'_x$, the mapping that we have implemented up to now leaves $\mathcal{H}'_x$ invariant.

Finally, controlled on the second register being in the state $|0\rangle$, we call the circuit $\mathcal{C}_{|w_0\rangle}$ on the first register. Checking whether the second register is in state $|0\rangle$ can be done in $\mathcal{O}(1)$ gates, and this takes one controlled call to $\mathcal{C}_{|w_0\rangle}$. Moreover, as $\mathcal{C}_{|w_0\rangle}$ leaves $\mathcal{H}_x$ invariant, we find that $\mathcal{C}_{|w_0\rangle} \otimes |0\rangle\langle 0|$ also leaves $\mathcal{H}_x \otimes |0\rangle \subseteq \mathcal{H}'_x$ invariant. This completes the proof. □

The following lemma constructs the reflection around $\Lambda = \Pi_{\mathcal{K}} + |\Psi_0\rangle\langle\Psi_0|$, where $|\Psi_0\rangle = \frac{1}{\sqrt{1+\frac{\||w_0\rangle\|^2}{W_+}}}\left(|\hat{0}\rangle + \frac{|w_0\rangle}{\sqrt{W_+}}\right)$, using the ability to reflect around $\mathcal{K}$, $|0\rangle$ and generate $|w_0\rangle$.

**Lemma 55.** *Let $\mathcal{R} = (\mathcal{H}, \{\mathcal{H}(x)\}_{x\in X}, \mathcal{K}, |w_0\rangle)$ be a reflection program, and for all $x \in X$, let $\mathcal{H}_x$ be an implementing subspace. Suppose that we have access to the following subroutines and their controlled versions:*

1. *A subroutine $\mathcal{R}_{\mathcal{K}}$ that acts on $\mathcal{H}_x$ as $2\Pi_{\mathcal{K}} - I$.*
2. *A subroutine $\mathcal{C}_{|w_0\rangle}$ that leaves $\mathcal{H}_x$ invariant and implements the mapping $|0\rangle \mapsto |w_0\rangle / \||w_0\rangle\|$.*
3. *A subroutine $\mathcal{R}_{|0\rangle}$ that acts on $\mathcal{H}_x$ as $2|0\rangle\langle 0| - I$.*

*Let $\mathcal{H}'_x = \mathcal{H}_x \oplus \mathrm{span}\{|\hat{0}\rangle\}$. Then we can implement the circuit $\mathcal{R}_\Lambda$ that acts on $\mathcal{H}'_x$ as $2\Lambda - I$, using $\mathcal{O}(1)$ controlled calls to the subroutines, extra gates and auxiliary qubits.*

*Proof.* First, recall that since $|\Psi_0\rangle$ is orthogonal to $\mathcal{K}$, and $\Lambda = \Pi_{\mathcal{K}} + |\Psi_0\rangle\langle\Psi_0|$, we can implement the reflection through $\Lambda$ up to a global phase as a product of the reflection through $\mathcal{K}$ on the one hand, and $|\Psi_0\rangle$ on the other.

Recall that we identify $\mathcal{H}$ with $\mathcal{H} \otimes |0\rangle$, and $|\hat{0}\rangle$ with $|0, 1\rangle$. Thus, in order to implement the reflection around $\mathcal{K}$ on $\mathcal{H}'_x$, we apply $\mathcal{R}_{\mathcal{K}}$ on the first register, controlled on the second register being in the state $|0\rangle$, and we add a minus if the second register is in the state $|1\rangle$. I.e., we apply the operation $\mathcal{R}_{\mathcal{K}} \otimes |0\rangle\langle 0| - I_{\mathcal{H}} \otimes |1\rangle\langle 1|$. As $\mathcal{R}_{\mathcal{K}}$ leaves $\mathcal{H}_x$ invariant, we easily check that this operation leaves $\mathcal{H}'_x$ invariant. Moreover, we can recognize whether the second register is in state $|0\rangle$ using $\mathcal{O}(1)$ gates, so implementing this operation takes only $\mathcal{O}(1)$ gates and one call to $\mathcal{R}_{\mathcal{K}}$.

Moreover, recall from Lemma 54 that we can implement the mapping $\mathcal{C} = \mathcal{C}_{|\Psi_0\rangle}$ with $\mathcal{O}(1)$ calls to the subroutines $\mathcal{C}_{|w_0\rangle}$ and $\mathcal{R}_{|0\rangle}$, extra gates, and auxiliary qubits. Moreover, observe that $\mathcal{R}_{|0\rangle} \otimes |0\rangle\langle 0| - I_{\mathcal{H}} \otimes |1\rangle\langle 1|$ implements $2|0\rangle\langle 0| - I$ on $\mathcal{H}'_x$. As

$$2|\Psi_0\rangle\langle\Psi_0| - I = \mathcal{C}\,(2|0\rangle\langle 0| - I)\,\mathcal{C}^\dagger,$$

we can reflect through the state $|\Psi_0\rangle$ with $\mathcal{O}(1)$ calls to the subroutines, extra gates and auxiliary qubits.

Thus, implementing the operations $2|\Psi_0\rangle\langle\Psi_0| - I$ and $2\Pi_{\mathcal{K}} - I$ consecutively allows for implementing the reflection around $\Lambda$. As both individual

reflections leave $\mathcal{H}'_x$ invariant, so does their product, and the total number of calls to the subroutines, extra gates and auxiliary qubits are all $\mathcal{O}(1)$. Note that for the controlled implementation of the reflection through $\Lambda$, we need to add an extra $Z$-gate to the control qubit to account for the global phase we neglected here. This completes the proof. □

Now that we know how to implement the reflection around $\Lambda$, we proceed with analyzing the cost of reflecting around $\mathcal{H}'(x) = \mathcal{H}(x) \oplus \mathrm{span}\{|\hat{0}\rangle\}$. This is the objective of the following lemma.

**Lemma 56.** *Let $\mathcal{R}$ be a reflection program, and for all $x \in X$, let $\mathcal{H}_x$ be an implementing subspace. Suppose that we have controlled access to a subroutine $\mathcal{R}_{\mathcal{H}(x)}$ that on $\mathcal{H}_x$ acts as $2\Pi_{\mathcal{H}(x)} - I$. Then we can implement a circuit $\mathcal{R}_{\mathcal{H}'(x)}$ that on $\mathcal{H}'_x = \mathcal{H}_x \oplus \mathrm{span}\{|\hat{0}\rangle\}$ acts as $2\Pi_{\mathcal{H}'(x)} - I$, with one controlled call to $\mathcal{R}_{\mathcal{H}(x)}$ and $\mathcal{O}(1)$ extra qubits and gates.*

*Proof.* From Theorem 45 we find that $\Pi_x = \Pi_{\mathcal{H}(x)} + |\hat{0}\rangle\langle\hat{0}|$ i.e. $\Pi_x$ is the projector into $\mathcal{H}'(x) = \mathcal{H}(x) \oplus \mathrm{span}\{|\hat{0}\rangle\}$. Since $|\hat{0}\rangle$ is orthogonal to $\mathcal{H}(x)$, the reflection through $\mathcal{H}'(x)$ up to a global phase is merely the product of the reflections through $\mathcal{H}(x)$ and $\mathrm{span}\{|\hat{0}\rangle\}$. Furthermore, the controlled implementation of $2\Pi_x - I$ has to have another $Z$-gate on the control qubit to account for the global phase that we neglect here.

Since we identify $|\hat{0}\rangle$ with $|0,1\rangle$, we can implement the reflection through $\mathrm{span}\{|\hat{0}\rangle\}$ in $\mathcal{H}'_x$ in time $\mathcal{O}(1)$, by simply implementing the operation $I_{\mathcal{H}} \otimes (2|1\rangle\langle 1| - I)$ in $\mathcal{O}(1)$ gates.

Similarly, we can apply the reflection through $\mathcal{H}(x)$ on $\mathcal{H}'_x$ with one call to $\mathcal{R}_{\mathcal{H}(x)}$ controlled on the second register being $|0\rangle$. This can be done with $\mathcal{O}(1)$ extra gates and auxiliary qubits, and one controlled call to $\mathcal{R}_{\mathcal{H}(x)}$, completing the proof. □

Now we are ready to give the proof of the main theorem of this section.

*Proof of Theorem 53.* Remember that the algorithm in Corollary 46 performs phase estimation and amplitude amplification on $U_x = (2\Pi_x - I)(2\Lambda - I)$.

By Lemmas 56 and 55, we can implement circuits that perform these two reflections on $\mathcal{H}'_x$ with $\mathcal{O}(1)$ calls to the subroutines, extra gates and auxiliary qubits. Thus, we conclude that we can implement $U_x$ with essentially the

same cost, and remark that we can thus also implement a controlled-$U$ operation, where we have to add another $Z$-gate to the control qubit to account for the global phase in $U_x$.

Finally, recall that the total number of calls to controlled-$U$, and hence to the subroutines, in the algorithm in Corollary 46 is of order

$$\mathcal{O}\left(\frac{\sqrt{W_+\widetilde{W}_-}}{(1-2\lambda)^{3/2}}\log\frac{1}{1-2\lambda}\right).$$

Moreover, as the algorithm compiled from the reflection program implements phase estimation up to precision $\Theta = \frac{1-2\lambda}{\sqrt{W_+\widetilde{W}_-}}$ with error probability at most $\varepsilon = \mathcal{O}(1-2\lambda)$, and amplitude estimation up to precision $\Theta' = \frac{1}{1-2\lambda}$, the number of extra gates and auxiliary qubits used by these algorithms are of order

$$\mathcal{O}\left(\operatorname{polylog}\left(\frac{1}{\Theta'},\frac{1}{\Theta}\log\frac{1}{\varepsilon}\right)\right) = \mathcal{O}\left(\operatorname{polylog}\left(\frac{\sqrt{W_+\widetilde{W}_-}}{(1-2\lambda)^{3/2}}\right)\right).$$

As a last remark we observe that if we only know upper bounds to $W_+$, $\widetilde{W}_-$ and $\lambda$, we are merely running the phase estimation and amplitude estimation routines with a better accuracy than strictly necessary, which does not impact negatively on the success probability of the algorithm. This completes the proof. □

Recall that a span program $P = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$ is a particular kind of reflection program where the assignment $x \mapsto \mathcal{H}(x)$ is predetermined by the structure of $\mathcal{H}$, $\mathcal{K} = \ker A$, and $|w_0\rangle = A^+|\tau\rangle$. In the remainder of this chapter we will use Theorem 53 only in the context of span programs.

## 4.4 From algorithms to span programs

Let $\mathcal{A}$ be a clean quantum algorithm that evaluates a function $f : X \subseteq \{0,1\}^n \to \{0,1\}$ with error probability $0 \leq \varepsilon < 1/2$, as in Definition 6. Based on this algorithm, one can construct a span program that approximates the same function and whose complexity is equal to the query complexity of $\mathcal{A}$, up to a multiplicative constant. This construction was first introduced by

Reichardt [Rei09] in the case where the algorithm has one-sided error, and extended to the case of bounded (two-sided) error in [Jef20].

Our contribution is to extend this construction so that it not only preserves the query complexity of $\mathcal{A}$ but also the time complexity. Starting with a quantum algorithm $\mathcal{A}$ whose query complexity is $S$ and time complexity is $T$, we construct a corresponding span program $P_{\mathcal{A}}$ that accounts for individual gates of $\mathcal{A}$. If the span program is compiled back to a quantum algorithm, the resulting algorithm still solves the same problem, its query complexity is $\widetilde{\mathcal{O}}(S)$ and its time complexity remains $\mathcal{O}(T)$. This requires modifications to the span program construction, but more importantly, an additional highly non-trivial analysis of the time complexity of the span program implementation.

### 4.4.1 The span program of an algorithm

Recall from Section 2.2.1 that we can assume without loss of generality that there are no two consecutive queries in the algorithm $\mathcal{A}$, and that the first and last unitaries are not queries. We label the time steps where the algorithm queries the input by

$$\mathcal{S} = \{q_1, \ldots, q_S\} \subseteq [T], \tag{4.1}$$

where $T$ is the total time complexity and $S$ denotes the total number of queries. For convenience, we also define $q_0 = 0$, $q_{S+1} = T+1$. We denote the $\ell$-th block of contiguous non-query time steps by $\mathcal{B}_\ell := \{q_{\ell-1}+1, \ldots, q_\ell - 1\} = \{t : q_{\ell-1} < t < q_\ell\} \subseteq [T]$, with $\ell \in [S+1]$. See Figure 4.1 for an overview of this notation.

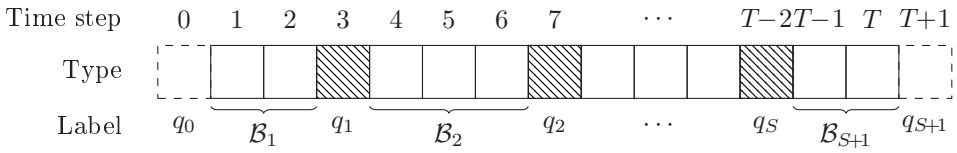



Figure 4.1: Synopsis of the notation. The cells denote time steps of the algorithm $\mathcal{A}$ where time progresses to the right. They are indexed from 1 to $T$. The hatched cells denote time steps in which a query to the input $x$ is performed. In all other time steps $t$ a unitary $U_t$ independent of $x$ is applied.

Recall that $\mathcal{W}$ is a finite set that labels the basis of the workspace of $\mathcal{A}$.

for all $i \in [n], b \in \{0,1\}$, we define the following spaces:

$$\begin{aligned} \mathcal{H}_{i,b} &= \operatorname{span}\{|t,b,i,j\rangle : t+1 \in \mathcal{S}, j \in \mathcal{W}\}, \\ \mathcal{H}_{\text{true}} &= \operatorname{span}\{|t,0,i,j\rangle : t+1 \in [T+1] \setminus \mathcal{S}, i \in [n], j \in \mathcal{W}\}, \\ \mathcal{H}_{\text{false}} &= \{0\}. \end{aligned} \tag{4.2}$$

As usual, the spaces $\mathcal{H}(x)$ and $\mathcal{H}$ are defined from these as:

$$\forall x \in \{0,1\}^n, \qquad \mathcal{H}(x) = \left( \bigoplus_{i=1}^{n} \mathcal{H}_{i,x_i} \right) \oplus \mathcal{H}_{\text{true}}, \qquad \text{and} \tag{4.3}$$

$$\mathcal{H} = \left( \bigoplus_{\substack{i \in [n] \\ b \in \{0,1\}}} \mathcal{H}_{i,b} \right) \oplus \mathcal{H}_{\text{true}} \oplus \mathcal{H}_{\text{false}}. \tag{4.4}$$

For better intuition, we provide a graphical depiction of $\mathcal{H}$, $\mathcal{H}_{\text{true}}$, $\mathcal{H}(x)$ and $\mathcal{H}_{i,b}$ in Figure 4.2.

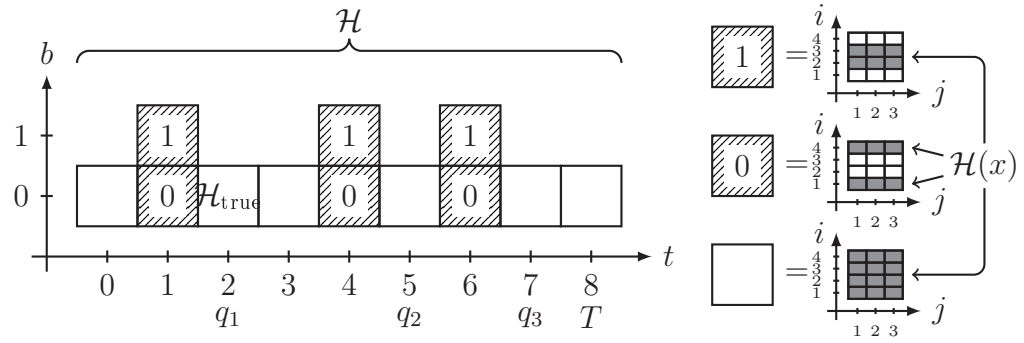



Figure 4.2: Graphical depiction of the relevant spaces when $T = 8$, $\mathcal{S} = \{2,5,7\}$, $n = 4$, $|\mathcal{W}| = 3$ and $x = 0110$. The total space $\mathcal{H}$ is a direct sum of all blocks on the left, where the block at position $(t,b) \in [T]_0 \times \{0,1\}$ denotes the subspace spanned by all computational basis states of the form $|t,b,\cdot,\cdot\rangle$. Every block is of one of three types, white, 0 or 1, shown on the right. The subspace $\mathcal{H}_{\text{true}}$ is the direct sum of all white blocks. Each block further decomposes as a direct sum over computational basis states $|i,j\rangle$, $i \in [n]$, $j \in \mathcal{W}$. The gray cells of all blocks together span the space $\mathcal{H}(x)$. Finally, for a given $i \in [n]$, the subspaces $\mathcal{H}_{i,0}$ and $\mathcal{H}_{i,1}$ consist of the $i$-th row within all 0 and 1 blocks, respectively.

Let $[T]_0 := \{0, \ldots, T\}$. We define the target space $\mathcal{V}$ and the target vector $|\tau\rangle \in \mathcal{V}$ as follows:

$$\mathcal{V} = \text{span}\{|t, i, j\rangle : t \in [T]_0, i \in [n], j \in \mathcal{W}\}, \; |\tau\rangle = |0\rangle|\Psi_0\rangle - |T\rangle|\Psi_T\rangle, \quad (4.5)$$

where $|\Psi_0\rangle$ is the initial state of $\mathcal{A}$ (see Eq. (2.3)) and $|\Psi_T\rangle$ is the final accepting state (see Definition 6).

Recall that $S$ denotes the total number of queries and $\varepsilon$ is the error probability of $\mathcal{A}$. Let

$$a = \sqrt{\frac{\varepsilon}{2S+1}} \qquad \text{and} \qquad M = \max_{\ell \in [S+1]} \sqrt{|\mathcal{B}_\ell|}, \quad (4.6)$$

where $\mathcal{B}_\ell \subseteq [T]$ is the $\ell$-th contiguous block of non-query gates (see Figure 4.1). By Definition 6 and Lemma 7, we can assume that $M \leq \sqrt{3T/S}$. For all computational basis vectors $|t, b, i, j\rangle$ in $\mathcal{H}$, we define the action of the span program operator $A \in \mathcal{L}(\mathcal{H}, \mathcal{V})$ as follows:

$$A|t, b, i, j\rangle = \begin{cases} a|T, i, j\rangle & \text{if } t = T, \\ M(|t, i, j\rangle - |t+1\rangle U_{t+1}|i, j\rangle) & \text{if } \exists \ell \in [S+1] : t+1 \in \mathcal{B}_\ell, \\ |t, i, j\rangle - (-1)^b |t+1, i, j\rangle & \text{if } \exists \ell \in [S] : t+1 = q_\ell. \end{cases} \quad (4.7)$$

The weights $a$ and $M$ are the main difference between our construction and that of [Jef20], and will enable the implementation of the span program algorithm described in Section 4.5 to be both time and query efficient. The unitary $U_{t+1}$ is the $(t+1)$-th unitary of algorithm $\mathcal{A}$ as defined in Section 2.2.1.

**Definition 57** (Span program of an algorithm)**.** The *span program of a quantum algorithm* $\mathcal{A}$ is $P_\mathcal{A} = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$, where $\mathcal{H}$ is defined in Eq. (4.2) and (4.4), $\mathcal{V}$ and $|\tau\rangle$ in Eq. (4.5), and $A$ in Eq. (4.7).

We spend the remainder of this section proving various properties of span programs of this type. We start by analyzing the positive and negative witness sizes $W_+(P)$ and $W_-(P)$, and the approximation factor $\lambda$.

**Theorem 58.** *Let $\mathcal{A}$ be a clean quantum algorithm for $f$ with error probability $0 \leq \varepsilon < 1/5$, making $S$ queries, and let $P_\mathcal{A}$ be the span program for $\mathcal{A}$ from Definition 57. Then $P_\mathcal{A}$ positively $5\varepsilon$-approximates $f$ with complexities $W_+(P_\mathcal{A}) = \mathcal{O}(S)$ and $\widetilde{W}_-(P_\mathcal{A}) = \mathcal{O}(S)$.*

Theorem 58 follows directly from Lemma 59, and Lemma 60 below. The proofs are similar to those of [Jef20], which are themselves similar to [Rei09], with the difference that the operator $A$ of our span program now has slightly modified weights, see Eq. (4.7).

**Lemma 59.** *Let $\mathcal{A}$ be a clean quantum algorithm with query complexity $S$, time complexity $T$ and error probability $0 \leq \varepsilon < 1/2$. Let $P_{\mathcal{A}}$ be the span program for $\mathcal{A}$ from Definition 57. Then,*

$$W_+(P_{\mathcal{A}}) \leq 3(2S+1) = \mathcal{O}(S).$$

*Proof.* Let $\mathcal{Z} = [n] \times \mathcal{W}$, so the state space of the algorithm $\mathcal{A}$ is $\mathbb{C}^{\mathcal{Z}}$. Recall from Eq. (2.3) that $|\Psi_t(x)\rangle \in \mathbb{C}^{\mathcal{Z}}$ denotes the state of $\mathcal{A}$ on input $x$ at time $t$, i.e., immediately after the application of $U_t$. We will construct a positive witness for every positive input $x \in f^{-1}(1)$ and upper bound its norm.

Keeping Eq. (4.7) in mind, for every $t \in [T]_0$ we define

$$|\widehat{\Psi}_t(x)\rangle = \begin{cases} \frac{1}{a}|0\rangle|\Psi_T(x)\rangle & \text{if } t = T, \\ \frac{1}{M}|0\rangle|\Psi_t(x)\rangle & \text{if } \exists \ell \in [S+1] : t+1 \in \mathcal{B}_\ell, \\ L_x|\Psi_t(x)\rangle & \text{if } \exists \ell \in [S] : t+1 = q_\ell, \end{cases}$$

where $L_x \in \mathcal{L}(\mathbb{C}^{\mathcal{Z}}, \mathbb{C}^2 \otimes \mathbb{C}^{\mathcal{Z}})$ is defined on the computational basis vectors as follows:

$$\forall i \in [n], j \in \mathcal{W}, \qquad L_x|i,j\rangle = |x_i, i, j\rangle.$$

For all $t \in [T]_0$, we easily verify that $|t\rangle|\widehat{\Psi}_t(x)\rangle \in \mathcal{H}(x)$ by referring to Eq. (4.2) and Eq. (4.4). Next, we define

$$|w_x\rangle = \sum_{t=0}^{T-1} |t\rangle|\widehat{\Psi}_t(x)\rangle + \frac{1}{a}|T\rangle|0\rangle \left(|\Psi_T(x)\rangle - |\Psi_T\rangle\right),$$

where $|\Psi_T\rangle$ is the final accepting state from Definition 6. Since $|T, 0, z\rangle \in \mathcal{H}(x)$ for all $z \in \mathcal{Z}$, we find by linearity that $|w_x\rangle \in \mathcal{H}(x)$. By splitting the time steps into query and non-query steps we find that

$$|w_x\rangle = \frac{1}{a}|T\rangle|0\rangle \left(|\Psi_T(x)\rangle - |\Psi_T\rangle\right) + \sum_{\ell=1}^{S+1} \sum_{t=q_{\ell-1}}^{q_\ell - 2} |t\rangle \frac{1}{M}|0\rangle|\Psi_t(x)\rangle$$
$$+ \sum_{\ell=1}^{S} |q_\ell - 1\rangle L_x|\Psi_{q_\ell - 1}(x)\rangle.$$

Applying $A$ we get

$$\begin{aligned}
A|w_x\rangle &= |T\rangle|\Psi_T(x)\rangle - |T\rangle|\Psi_T\rangle \\
&\quad + \sum_{\ell=1}^{S+1} \sum_{t=q_{\ell-1}}^{q_\ell-2} M \left[ |t\rangle \frac{1}{M} |\Psi_t(x)\rangle - |t+1\rangle \frac{1}{M} U_{t+1} |\Psi_t(x)\rangle \right] \\
&\quad + \sum_{\ell=1}^{S} \left[ |q_\ell - 1\rangle |\Psi_{q_\ell-1}(x)\rangle - |q_\ell\rangle \mathcal{O}_x |\Psi_{q_\ell-1}(x)\rangle \right] \\
&= |T\rangle|\Psi_T(x)\rangle - |T\rangle|\Psi_T\rangle + \sum_{\ell=1}^{S+1} \sum_{t=q_{\ell-1}}^{q_\ell-2} \left[ |t\rangle|\Psi_t(x)\rangle - |t+1\rangle|\Psi_{t+1}(x)\rangle \right] \\
&\quad + \sum_{\ell=1}^{S} \left[ |q_\ell - 1\rangle |\Psi_{q_\ell-1}(x)\rangle - |q_\ell\rangle |\Psi_{q_\ell}(x)\rangle \right] \\
&= |T\rangle|\Psi_T(x)\rangle - |T\rangle|\Psi_T\rangle + \sum_{t=0}^{T-1} \left[ |t\rangle|\Psi_t(x)\rangle - |t+1\rangle|\Psi_{t+1}(x)\rangle \right] \\
&= |0\rangle|\Psi_0\rangle - |T\rangle|\Psi_T\rangle = |\tau\rangle,
\end{aligned}$$

where most terms cancel since the final sum is telescopic. In particular, we find that $|w_x\rangle$ is indeed a positive witness for $x$. We can use its size to bound the size of the minimum positive witness for $x$:

$$\begin{aligned}
w_+(x, P_{\mathcal{A}}) &= \min\{ \||w\rangle\|^2 : |w\rangle \in \mathcal{H}(x), A|w\rangle = |\tau\rangle \} \\
&\leq \||w_x\rangle\|^2 = \sum_{t=0}^{T-1} \left\| |\widehat{\Psi}_t(x)\rangle \right\|^2 + \frac{\||\Psi_T(x)\rangle - |\Psi_T\rangle\|^2}{a^2} \\
&= \sum_{\ell=1}^{S} \||\Psi_{q_\ell-1}(x)\rangle\|^2 + \sum_{\ell=1}^{S+1} \sum_{t=q_{\ell-1}}^{q_\ell-2} \frac{1}{M^2} \||\Psi_t(x)\rangle\|^2 + \frac{\||\Psi_T(x)\rangle - |\Psi_T\rangle\|^2}{a^2} \\
&\leq S + (S+1) + \frac{1}{a^2} \cdot 2\varepsilon \leq 2S + 1 + \frac{2S+1}{\varepsilon} \cdot 2\varepsilon = 3(2S+1),
\end{aligned}$$

where we used $M^2 \geq |\mathcal{B}_\ell| = q_\ell - q_{\ell-1} - 1$ from Eq. (4.6) to bound the second term. To bound the third term, we used $a = \sqrt{\varepsilon/(2S+1)}$ from Eq. (4.6) and the inequality

$$\||\Psi_T(x)\rangle - |\Psi_T\rangle\|^2 = 2(1 - \mathrm{Re}\langle\Psi_T(x)|\Psi_T\rangle) = 2(1 - p_1(x)) \leq 2\varepsilon$$

which holds for any $x \in f^{-1}(1)$ (see Lemma 7 and Definition 6). Thus,

$$W_+(P_\mathcal{A}) = \max_{x \in f^{-1}(1)} w_+(x, P_\mathcal{A}) \leq 3(2S+1).$$

□

**Lemma 60.** *Let $\mathcal{A}$ be a clean quantum algorithm with query complexity $S$, time complexity $T$ and error probability $0 \leq \varepsilon < \frac{1}{5}$. Let $P_\mathcal{A}$ be the span program for $\mathcal{A}$ from Definition 57. Then, for all $x \in f^{-1}(0)$, there exists an approximate negative witness $|\widetilde{\omega}_x\rangle$ such that $\left\|\langle\widetilde{\omega}_x|A\Pi_{\mathcal{H}(x)}\right\|^2 \leq 5\varepsilon/(3(2S+1))$ and $\|\langle\widetilde{\omega}_x|A\|^2 \leq 2(4S+1)$. Thus:*

1. *$P_\mathcal{A}$ positively $\lambda$-approximates $f$ for $\lambda = 5\varepsilon$.*
2. *The approximate negative witness complexity of $P_\mathcal{A}$ is $\widetilde{W}_-(P_\mathcal{A}) = \mathcal{O}(S)$.*

*Proof.* Given a negative input $x$, we define an approximate negative witness and bound the negative error and minimum approximate negative witness size using this witness. To that end, let $x \in f^{-1}(0)$. Define

$$\langle\widetilde{\omega}_x| = \frac{1}{1 - \langle\Psi_T(x)|\Psi_T\rangle} \sum_{t=0}^{T} \langle t|\langle\Psi_t(x)|.$$

Note that this is well-defined as $x$ is a negative instance, hence, by Definition 6 and Lemma 7 we have $|\langle\Psi_T(x)|\Psi_T\rangle| \leq \varepsilon < 1$. Recalling from Eq. (4.5) that $|\tau\rangle = |0\rangle|\Psi_0\rangle - |T\rangle|\Psi_T\rangle$, observe that

$$\langle\widetilde{\omega}_x|\tau\rangle = \frac{\langle\Psi_0|\Psi_0\rangle - \langle\Psi_T(x)|\Psi_T\rangle}{1 - \langle\Psi_T(x)|\Psi_T\rangle} = 1. \tag{4.8}$$

Next, let $|t, b, i, j\rangle$ be a computational basis vector in $\mathcal{H}(x)$ and let $A$ be the operator defined in Eq. (4.7). If $t+1 = q_\ell$ for some $\ell \in [S]$, then $b = x_i$ and

$$\begin{aligned}
\langle\widetilde{\omega}_x|A|t, b, i, j\rangle &= \langle\widetilde{\omega}_x|\big[|t, i, j\rangle - (-1)^{x_i}|t+1, i, j\rangle\big] \\
&= \frac{1}{1 - \langle\Psi_T(x)|\Psi_T\rangle}\big[\langle\Psi_t(x)|i, j\rangle - (-1)^{x_i}\langle\Psi_{t+1}(x)|i, j\rangle\big] \\
&= \frac{1}{1 - \langle\Psi_T(x)|\Psi_T\rangle}\left[\langle\Psi_t(x)|i, j\rangle - \langle\Psi_t(x)|\mathcal{O}_x^\dagger(-1)^{x_i}|i, j\rangle\right] = 0.
\end{aligned}$$

On the other hand, if $t+1 \in \mathcal{B}_\ell$ for some $\ell \in [S+1]$, then $b=0$ and

$$\begin{aligned}\langle\widetilde{\omega}_x|A|t,b,i,j\rangle &= M\langle\widetilde{\omega}_x|\big[|t,i,j\rangle - |t+1\rangle U_{t+1}|i,j\rangle\big] \\ &= \frac{M}{1-\langle\Psi_T(x)|\Psi_T\rangle}\big[\langle\Psi_t(x)|i,j\rangle - \langle\Psi_{t+1}(x)|U_{t+1}|i,j\rangle\big] \\ &= \frac{M}{1-\langle\Psi_T(x)|\Psi_T\rangle}\left[\langle\Psi_t(x)| - \langle\Psi_t(x)|U_{t+1}^\dagger U_{t+1}\right]|i,j\rangle = 0.\end{aligned}$$

Finally, if $t=T$, then $\langle\widetilde{\omega}_x|A|T,0,i,j\rangle = a\langle\widetilde{\omega}_x|T,i,j\rangle = \frac{a\langle\Psi_T(x)|i,j\rangle}{1-\langle\Psi_T(x)|\Psi_T\rangle}$, where $a$ is defined in Eq. (4.6). This might not evaluate to 0, potentially contributing to the negative witness error of $\langle\widetilde{\omega}_x|$ for $x$. Using $\langle\Psi_T(x)|\Psi_T\rangle = p_1(x) \le \varepsilon < 1/5$ (see Definition 6 and Lemma 7) we find that

$$\begin{aligned}\left\|\langle\widetilde{\omega}_x|A\Pi_{\mathcal{H}(x)}\right\|^2 &= \sum_{i\in[n],j\in\mathcal{W}} |\langle\widetilde{\omega}_x|A|T,0,i,j\rangle|^2 = \frac{a^2\,\||\Psi_T(x)\rangle\|^2}{|1-\langle\Psi_T(x)|\Psi_T\rangle|^2} \\ &= \frac{a^2}{(1-p_1(x))^2} \le \frac{a^2}{(1-\varepsilon)^2} \le \frac{\varepsilon}{(2S+1)\left(1-\frac{1}{5}\right)^2} \\ &< \frac{\frac{25}{15}\varepsilon}{2S+1} = \frac{5\varepsilon}{3(2S+1)} \le \frac{5\varepsilon}{W_+(P)},\end{aligned}$$

where in the last inequality we used Lemma 59. We find that $P$ positively $\lambda$-approximates $f$ with $\lambda = 5\varepsilon$, completing the proof of the first claim. To prove the second claim, recall from Eq. (4.8) that $\langle\widetilde{\omega}_x|\tau\rangle = 1$. Hence, for any $x \in f^{-1}(0)$, using that $\mathcal{Z} = [n]\times\mathcal{W}$ we have:

$$\begin{aligned}&\widetilde{w}_-(x,P) = \min_{|\widetilde{\omega}\rangle\in\mathcal{V}}\left\{\|\langle\widetilde{\omega}|A\|^2 : \langle\widetilde{\omega}|\tau\rangle = 1, \left\|\langle\widetilde{\omega}|A\Pi_{\mathcal{H}(x)}\right\|^2 \le \frac{\lambda}{W_+(P)}\right\} \\ &\le \|\langle\widetilde{\omega}_x|A\|^2 = \sum_{\substack{\ell\in[S],z\in\mathcal{Z}\\ b\in\{0,1\}}} \left|\langle\widetilde{\omega}_x|\big[|q_\ell-1,z\rangle - (-1)^b|q_\ell,z\rangle\big]\right|^2 + \sum_{z\in\mathcal{Z}}|\langle\widetilde{\omega}_x|a|T,z\rangle|^2 \\ &+ \sum_{\substack{\ell\in[S+1],z\in\mathcal{Z}\\ t\in\{q_{\ell-1},\ldots,q_\ell-2\}}} \left|\langle\widetilde{\omega}_x|M\big(|t,z\rangle - |t+1\rangle U_{t+1}|z\rangle\big)\right|^2\end{aligned}$$

$$\leq \frac{1}{|1-\langle\Psi_T(x)|\Psi_T\rangle|^2}\left[\sum_{\ell\in[S],z\in\mathcal{Z}} 2\,|\langle\Psi_{q_\ell-1}(x)|z\rangle|^2 + 2\,|\langle\Psi_{q_\ell}(x)|z\rangle|^2\right.$$

$$\left. + \sum_{\substack{\ell\in[S+1],z\in\mathcal{Z}\\ t\in\{q_{\ell-1},\ldots,q_\ell-2\}}} M^2\,|\{\langle\Psi_t(x)| - \langle\Psi_{t+1}(x)|U_{t+1}\}\,|z\rangle|^2 + \sum_{z\in\mathcal{Z}} a^2\,|\langle\Psi_T(x)|z\rangle|^2\right]$$

$$= \frac{1}{|1-\langle\Psi_T(x)|\Psi_T\rangle|^2}\left[\sum_{\ell\in[S]} 2\,\|\langle\Psi_{q_\ell-1}(x)|\|^2 + 2\,\|\langle\Psi_{q_\ell}(x)|\|^2 + a^2\,\|\langle\Psi_T(x)|\|^2\right]$$

$$= \frac{4S+a^2}{|1-\langle\Psi_T(x)|\Psi_T\rangle|^2} \leq \frac{4S+a^2}{(1-\varepsilon)^2} \leq \frac{4S+a^2}{\left(1-\frac{1}{5}\right)^2}$$

$$\leq \left[4S + \frac{\varepsilon}{2S+1}\right]\cdot\frac{25}{16} \leq 2(4S+1) = \mathcal{O}(S).$$

□

Together, Lemma 59 and Lemma 60 prove Theorem 58, which in turn implies an upper bound on the query complexity of implementing the span program $P_{\mathcal{A}}$.

### Witness anatomy of the span program of an algorithm

We conclude this section by characterizing the kernel of the span program operator $A$ in Lemma 61, and subsequently finding the minimal witness size in Lemma 62. These will prove relevant in the analysis of the time complexity of the algorithm compiled from $P_{\mathcal{A}}$, to which we turn our attention in Section 4.5.

The span program map $A$ written in Eq. (4.7) is a complicated object, obscured by the parameters $a$ and $M$ and with three different "regimes". Before we completely characterize it in Lemma 61, let us study a simplified version of $A$ which will give us a better grasp of the problem at hand.

With $\mathcal{H}$ defined in Eq. (4.4), define a stripped down version of $A$ in Eq. (4.7) acting on $|tbz\rangle \in \mathcal{H}$ as

$$A|t,b,z\rangle = \begin{cases} |t,z\rangle - |t+1\rangle U_{t+1}|z\rangle & \text{if } t+1 \notin \mathcal{S} \\ |t,z\rangle - |t+1\rangle(-1)^b|z\rangle & \text{if } t+1 \in \mathcal{S}. \end{cases} \tag{4.9}$$

**No queries** The first case we will study is that of an algorithm without queries, i.e. $\mathcal{S} = \emptyset$. Let $|h\rangle = \sum_{t\in[T-1]_0} |t\rangle|0\rangle|\varphi_t\rangle$ for some unnormalized states $|\varphi_t\rangle$. Then

$$\begin{aligned} A|h\rangle &= \sum_{t\in[T-1]_0} \left(|t\rangle|\varphi_t\rangle - |t+1\rangle U_{t+1}|\varphi_t\rangle\right) \\ &= |0\rangle|\varphi_0\rangle + \sum_{t\in[T-1]} |t\rangle \left(|\varphi_t\rangle - U_t|\varphi_{t-1}\rangle\right) - |T\rangle|\varphi_T\rangle. \end{aligned}$$

We can cancel most terms of the sum by defining $|\varphi_t\rangle = U_t|\varphi_{t-1}\rangle$, but the first and last term can only cancel if $|\varphi_0\rangle = |\varphi_T\rangle = 0$. It is very easy to see that for this $A$, $\ker(A) = 0$. That is, the best we can hope for in terms of cancellations when we have no queries is a telescopic sum where the first and last term survive.

**One query** When we have exactly one query, the dimension of $\ker(A)$ is still 0 but we have much to learn from trying to find $|h\rangle$ such that $A|h\rangle = 0$. Let us assume that we have one query at time step $t = q$. And let the vector $|h\rangle$ be defined as

$$|h\rangle = \sum_{t<q-1} |t\rangle|0\rangle|\varphi_t\rangle + \sum_{b\in\{0,1\}} |q-1\rangle|b\rangle|\varphi_{q-1}^{(b)}\rangle + \sum_{t>q-1} |t\rangle|0\rangle|\varphi_t\rangle.$$

Now, choosing $|\varphi_t\rangle = U_t|\varphi_{t-1}\rangle$ we have that $A|h\rangle$ is going to create two telescopic sums with a term $A\left(\sum_{b\in\{0,1\}}|q-1\rangle|b\rangle|\varphi_{q-1}^{(b)}\rangle\right)$ in between. More importantly, we can choose $|\varphi_{q-1}^{(0)}\rangle$ in such a way that we cancel either the endpoint of the first telescopic sum or the first term of the second telescopic sum. Regardless of what we do, we are left with at least one endpoint of one telescopic sum standing. We conclude that $\ker(A) = 0$. Nevertheless, we have learned that a query allows us to cancel an endpoint to a telescopic sum.

**Two queries** What happens if we go to two queries? As a toy model let us assume that the algorithm has exactly $T$ time steps, the first and last being queries. That is, let us assume that $\mathcal{A} = \mathcal{O}_x, U_2, \ldots, U_{T-1}, \mathcal{O}_x$. For every $|\varphi\rangle$

in the workspace, let us define the vector

$$|h\rangle = |0\rangle\frac{1}{2}\left(|0\rangle - |1\rangle\right)|\varphi\rangle + \sum_{t\in[T-2]} |t\rangle|0\rangle|\varphi_t\rangle + |T-1\rangle\frac{1}{2}\left(|0\rangle + |1\rangle\right)|\varphi_{T-1}\rangle, \tag{4.10}$$

where we define $|\varphi_t\rangle = U_t|\varphi_{t-1}\rangle$ for $t \in [T-1]$ and $|\varphi_0\rangle = |\varphi\rangle$. We claim that this vector is in the kernel of $A$. Indeed,

$$\begin{aligned} A|h\rangle &= |0\rangle\frac{1}{2}\left(|\varphi\rangle - |\varphi\rangle\right) - |1\rangle\frac{1}{2}\left(|\varphi\rangle - (-1)|\varphi\rangle\right) + A\left(\sum_{t\in[T-2]} |t\rangle|\varphi_t\rangle\right) \\ &\quad + |T-1\rangle\frac{1}{2}\left(|\varphi_{T-1}\rangle + |\varphi_{T-1}\rangle\right) - |T\rangle\frac{1}{2}\left(|\varphi_{T-1}\rangle + (-1)|\varphi_{T-1}\rangle\right) \\ &= 0. \end{aligned}$$

In fact, it is straightforward to prove that all vectors in the kernel of A are of this form. What allows us to cancel all terms is that $A$ creates a telescopic sum in the block between the queries whose starting point is annihilated by the $t = 0$ term of $|h\rangle$ and whose endpoint is annihilated by the $t = T-1$ term in $|h\rangle$. If an algorithm has more than one block (more than 2 queries), the same analysis can be done for each separate block. What we have found is that for every block between queries and every vector $|\varphi\rangle \in \mathcal{W}$ we have exactly one vector in $\ker(A)$ with the same form as that of equation (4.10). We are now finally ready to tackle the original problem of understanding $\ker(A)$ in its full complexity.

Before we do that, we will define notation to denote the product of a subsequence of unitaries of $\mathcal{A}$. Let $t_1 \leq t_2$, then we define

$$U_{t_2;t_1} := U_{t_2}U_{t_2-1}\cdots U_{t_1}, \quad \text{and} \quad U^\dagger_{t_1;t_2} := U^\dagger_{t_1}U^\dagger_{t_1+1}\cdots U^\dagger_{t_2}. \tag{4.11}$$

If $t_2 < t_1$, then we define $U_{t_2;t_1} := I$, and $U^\dagger_{t_1;t_2}$.

**Lemma 61.** *Let $\mathcal{A}$ be a clean quantum query algorithm with error probability $0 \leq \varepsilon < 1$. Let $P_{\mathcal{A}} = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$ be the span program for $\mathcal{A}$. Let $\mathcal{Z} = [n] \times \mathcal{W}$. For $\ell \in \{2, \ldots, S\}$, we define the linear map $\Phi_\ell$ from $\mathbb{C}^{\mathcal{Z}}$ to $\mathcal{H}$ as*

$$\Phi_\ell|\psi\rangle = |q_{\ell-1}-1\rangle\frac{|-\rangle}{\sqrt{2}}|\psi\rangle + \frac{1}{M}\sum_{t=q_{\ell-1}}^{q_\ell-2} |t\rangle|0\rangle U_{t;q_{\ell-1}+1}|\psi\rangle \tag{4.12}$$

$$+|q_\ell - 1\rangle\frac{|+\rangle}{\sqrt{2}}U_{q_\ell-1;q_{\ell-1}+1}|\psi\rangle, \tag{4.13}$$

*where* $|\pm\rangle = (|0\rangle \pm |1\rangle)/\sqrt{2}$ *and* $M$ *was defined in Eq.* (4.6)*. We also define the linear map* $\Phi_{S+1}$ *from* $\mathbb{C}^{\mathcal{Z}}$ *to* $\mathcal{H}$ *as*

$$\Phi_{S+1}|\psi\rangle = |q_S - 1\rangle\frac{|-\rangle}{\sqrt{2}}|\psi\rangle + \frac{1}{M}\sum_{t=q_S}^{T-1}|t\rangle|0\rangle U_{t;q_S+1}|\psi\rangle + \frac{1}{a}|T\rangle|0\rangle U_{T;q_S+1}|\psi\rangle. \tag{4.14}$$

*Then*

$$\ker(A) = \bigoplus_{\ell=2}^{S+1}\Phi_\ell\left(\mathbb{C}^{\mathcal{Z}}\right).$$

*Proof.* By direct calculation we can check that all vectors in the image of the linear maps $\Phi_\ell$ are elements in the kernel of $A$. Thus, it remains to show that any vector in the kernel of $A$ can be written as a linear combination of vectors in the image of the $\Phi_\ell$'s. To that end, let $|\Psi\rangle \in \ker(A) \subseteq \mathcal{H}$. We first of all split this state in several disjointly supported parts, i.e.,

$$\begin{aligned}|\Psi\rangle = {} & \frac{1}{M}\sum_{t=0}^{q_1-2}|t\rangle|0\rangle|\psi_{1,t}\rangle + |q_1-1\rangle\frac{|+\rangle}{\sqrt{2}}|\psi_{1,q_1-1}\rangle \\ & + \sum_{\ell=2}^{S}\left(|q_{\ell-1}-1\rangle\frac{|-\rangle}{\sqrt{2}}|\psi_{\ell,q_{\ell-1}-1}\rangle + \frac{1}{M}\sum_{t=q_{\ell-1}}^{q_\ell-2}|t\rangle|0\rangle|\psi_{\ell,t}\rangle + |q_\ell-1\rangle\frac{|+\rangle}{\sqrt{2}}|\psi_{\ell,q_\ell-1}\rangle\right) \\ & + |q_S-1\rangle\frac{|-\rangle}{\sqrt{2}}|\psi_{S+1,q_S-1}\rangle + \frac{1}{M}\sum_{t=q_S}^{T-1}|t\rangle|0\rangle|\psi_{S+1,t}\rangle + \frac{1}{a}|T\rangle|0\rangle|\psi_{S+1,T}\rangle,\end{aligned}$$

where all the amplitudes are absorbed in the unnormalized $|\psi_{\ell,t}\rangle$-vectors. Now, we apply $A$ to this vector to obtain

$$\begin{aligned}A|\Psi\rangle = {} & \sum_{t=0}^{q_1-2}\left(|t\rangle|\psi_{1,t}\rangle - |t+1\rangle U_{t+1}|\psi_{1,t}\rangle\right) + |q_1-1\rangle|\psi_{1,q_1-1}\rangle \\ & + \sum_{\ell=2}^{S}\left(-|q_{\ell-1}\rangle|\psi_{\ell,q_{\ell-1}-1}\rangle + \sum_{t=q_{\ell-1}}^{q_\ell-2}\left(|t\rangle|\psi_{\ell,t}\rangle - |t+1\rangle U_{t+1}|\psi_{\ell,t}\rangle\right) + |q_\ell-1\rangle|\psi_{\ell,q_\ell-1}\rangle\right) \\ & - |q_S\rangle|\psi_{S+1,q_S-1}\rangle + \sum_{t=q_S}^{T-1}\left(|t\rangle|\psi_{S+1,t}\rangle - |t+1\rangle U_{t+1}|\psi_{S+1,t}\rangle\right) + |T\rangle|\psi_{S+1,T}\rangle\end{aligned}$$

$$
\begin{aligned}
&= |0\rangle|\psi_{1,0}\rangle + \sum_{t=1}^{q_1-1} |t\rangle(|\psi_{1,t}\rangle - U_t|\psi_{1,t-1}\rangle) \\
&+ \sum_{\ell=2}^{S} \left( |q_{\ell-1}\rangle(|\psi_\ell, q_{\ell-1}\rangle - |\psi_{\ell,q_{\ell-1}-1}\rangle) + \sum_{t=q_{\ell-1}+1}^{q_\ell-1} |t\rangle(|\psi_{\ell,t}\rangle - U_t|\psi_{\ell,t-1}\rangle) \right) \\
&+ |q_S\rangle(|\psi_{S+1,q_S}\rangle - |\psi_{S+1,q_S-1}\rangle) + \sum_{t=q_S+1}^{T} |t\rangle(|\psi_{S+1,t}\rangle - U_t|\psi_{S+1,t-1}\rangle).
\end{aligned}
$$

As $|\Psi\rangle \in \ker(A)$, the above expression has to equal 0. We learn by inspection that this happens if and only if the following conditions are satisfied:

$$
\begin{aligned}
|\psi_{1,0}\rangle &= 0 \\
|\psi_{1,t}\rangle &= U_t|\psi_{1,t-1}\rangle \quad \forall t \in \{1,\dots,q_1-1\}, \\
|\psi_{\ell,q_{\ell-1}}\rangle &= |\psi_{\ell,q_{\ell-1}-1}\rangle \quad \forall \ell \in \{2,\dots,S+1\}, \\
|\psi_{\ell,t}\rangle &= U_t|\psi_{\ell,t-1}\rangle \quad \forall \ell \in \{1,\dots,S\}, t \in \{q_{\ell-1}+1,\dots,q_\ell-1\}, \\
|\psi_{S+1,t}\rangle &= U_t|\psi_{S+1,t-1}\rangle \quad \forall t \in \{q_S+1,\dots,T\}.
\end{aligned}
$$

Using the abbreviation $|\psi_\ell\rangle = |\psi_{\ell,q_{\ell-1}-1}\rangle$ for $\ell \in \{2,\dots,S+1\}$, these conditions simplify to:

$$
\begin{aligned}
|\psi_{1,t}\rangle &= 0 \quad \forall t \in \{0,\dots,q_1-1\}, \\
|\psi_{\ell,q_{\ell-1}}\rangle &= |\psi_{\ell,q_{\ell-1}-1}\rangle = |\psi_\ell\rangle \quad \forall \ell \in \{2,\dots,S+1\}, \\
|\psi_{\ell,t}\rangle &= U_{t;q_{\ell-1}+1}|\psi_{\ell,q_{\ell-1}}\rangle \quad \forall \ell \in [S], t \in \{q_{\ell-1}+1,\dots,q_\ell-1\}, \\
&= U_{t;q_{\ell-1}+1}|\psi_\ell\rangle \\
|\psi_{S+1,t}\rangle &= U_{t;q_S+1}|\psi_{S+1,q_S}\rangle \quad \forall t \in \{q_S+1,\dots,T\}, \\
&= U_{t;q_S+1}|\psi_{S+1}\rangle.
\end{aligned}
$$

Using these constraints, we can rewrite $|\Psi\rangle$ as

$$
\begin{aligned}
|\Psi\rangle = \sum_{\ell=2}^{S} \Bigg( &|q_{\ell-1}-1\rangle\frac{|-\rangle}{\sqrt{2}}|\psi_\ell\rangle + \frac{1}{M}\sum_{t=q_{\ell-1}}^{q_\ell-2} |t\rangle|0\rangle U_{t;q_{\ell-1}+1}|\psi_\ell\rangle \\
&+|q_\ell-1\rangle\frac{|+\rangle}{\sqrt{2}}U_{q_\ell-1;q_{\ell-1}+1}|\psi_\ell\rangle\Bigg) + |q_S-1\rangle\frac{|-\rangle}{\sqrt{2}}|\psi_{S+1}\rangle
\end{aligned}
$$

$$+\frac{1}{M}\sum_{t=q_S}^{T-1}|t\rangle|0\rangle U_{t;q_S+1}|\psi_{S+1}\rangle+\frac{1}{a}|T\rangle|0\rangle U_{T;q_S+1}|\psi_{S+1}\rangle$$
$$=\sum_{\ell=2}^{S+1}\Phi_\ell(|\psi_\ell\rangle),$$

□

Finally, we give a closed form for the minimal positive witness $|w_0\rangle = A^+|\tau\rangle$. Remember that the two traits that completely characterize this vector are that $A|w_0\rangle = |\tau\rangle$ and that $|w_0\rangle \perp \ker(A)$.

**Lemma 62.** *Let $\mathcal{A}$ be a clean quantum query algorithm with error probability $0 \le \varepsilon < 1$. Let $P_{\mathcal{A}}$ be the span program for $\mathcal{A}$ from Definition 57. Then the minimal witness $|w_0\rangle = A^+|\tau\rangle$ is*

$$|w_0\rangle = \frac{1}{M}\sum_{t=0}^{q_1-2}|t\rangle|0\rangle U_{t;1}|\Psi_0\rangle + |q_1-1\rangle\left(\frac{1}{2}|0\rangle+\frac{1}{2}|1\rangle\right)U_{q_1-1;1}|\Psi_0\rangle$$
$$+\frac{1}{Ca^2+1}\left[|q_S-1\rangle\left(\frac{1}{2}|0\rangle-\frac{1}{2}|1\rangle\right)U^\dagger_{q_S+1;T}|\Psi_T\rangle\right.$$
$$\left.+\frac{1}{M}\sum_{t=q_S}^{T-1}|t\rangle|0\rangle U^\dagger_{t+1;T}|\Psi_T\rangle\right]-\frac{Ca}{Ca^2+1}|T\rangle|0\rangle|\Psi_T\rangle,$$

*where $C = \frac{T-q_S}{M^2}+\frac{1}{2}$ and $a$ and $M$ are defined in Eq. (4.6). In addition, the squared norm of $|w_0\rangle$ is $N = \frac{q_1-1}{M^2}+\frac{1}{2}+\frac{C}{Ca^2+1}$.*

*Proof.* We first prove that $|w_0\rangle$ is orthogonal to all vectors in the kernel of $A$. By Lemma 61, it suffices to take $|\psi\rangle \in \mathbb{C}^{[n]\times\mathcal{W}}$ arbitrarily and check that for all $\ell \in \{2,\dots,S+1\}$, $\langle\psi|\Phi_\ell^\dagger|w_0\rangle = 0$. Observe that $|w_0\rangle$ does not have support in states with time $t \in \{q_1,\dots q_S-2\}$, hence, for all $\ell \in \{3,\dots,S-1\}$, we easily obtain that the vectors $|w_0\rangle$ and $\Phi_\ell|\psi\rangle$ are orthogonal. For $\ell = 2$, we find that the supports only overlap at $t = q_1-1$ with the term $|q_1-1\rangle\frac{|-\rangle}{\sqrt{2}}|\psi\rangle$ of $\Phi_1|\psi\rangle$, so

$$\langle\psi|\Phi_1^\dagger|w_0\rangle = \frac{1}{4}\langle\psi|U_{q_1-1;1}|\Psi_0\rangle - \frac{1}{4}\langle\psi|U_{q_1-1;1}|\Psi_0\rangle = 0.$$

A similar computation shows that $|w_0\rangle$ and $\Phi_S|\psi\rangle$ are orthogonal, as their supports only overlap at $t = q_S-1$ with the term $|q_S-1\rangle\frac{|+\rangle}{\sqrt{2}}U_{q_S-1}\dots U_{q_{S-1}+1}|\psi\rangle$

of $\Phi_S|\psi\rangle$. Finally,

$$\begin{aligned}\langle\psi|\Phi_{S+1}^\dagger|w_0\rangle &= \frac{1}{Ca^2+1}\left[\frac{1}{4}\langle\psi|U_{q_S+1;T}^\dagger|\Psi_T\rangle + \frac{1}{4}\langle\psi|U_{q_S+1;T}^\dagger|\Psi_T\rangle\right.\\ &\quad\left.+\frac{1}{M^2}\sum_{t=q_S}^{T-1}\langle\psi|U_{q_S+1;T}^\dagger|\Psi_T\rangle\right] - \frac{C}{Ca^2+1}\langle\psi|U_{q_S+1;T}^\dagger|\Psi_T\rangle\\ &= \left(\frac{1}{Ca^2+1}\cdot\left[\frac{1}{2}+\frac{T-q_S}{M^2}\right] - \frac{C}{Ca^2+1}\right)\langle\psi|U_{q_S+1;T}^\dagger|\Psi_T\rangle\\ &= \left[\frac{C}{Ca^2+1} - \frac{C}{Ca^2+1}\right]\langle\psi|U_{q_S+1;T}^\dagger|\Psi_T\rangle = 0.\end{aligned}$$

Thus $|w_0\rangle$ is orthogonal to the kernel of $A$. It is also mapped to $|\tau\rangle$ by $A$, as

$$\begin{aligned}A|w_0\rangle &= \sum_{t=0}^{q_1-2}\left(|t\rangle U_{t;1}|\Psi_0\rangle - |t+1\rangle U_{t+1;1}|\Psi_0\rangle\right)\\ &\quad+\left(\frac{1}{2}+\frac{1}{2}\right)|q_1-1\rangle U_{q_1-1;1}|\Psi_0\rangle - \left(\frac{1}{2}-\frac{1}{2}\right)|q_1\rangle U_{q_1-1;1}|\Psi_0\rangle\\ &\quad+\frac{1}{Ca^2+1}\left[\left(\frac{1}{2}-\frac{1}{2}\right)|q_S-1\rangle U_{q_S+1;T}^\dagger|\Psi_T\rangle\right.\\ &\quad-\left(\frac{1}{2}+\frac{1}{2}\right)|q_S\rangle U_{q_S+1;T}^\dagger|\Psi_T\rangle\\ &\quad\left.+\sum_{t=q_S}^{T-1}\left(|t\rangle U_{t+1;T}^\dagger|\Psi_T\rangle - |t+1\rangle U_{t+2;T}^\dagger|\Psi_T\rangle\right)\right] - \frac{Ca^2}{Ca^2+1}|T\rangle|\Psi_T\rangle\\ &= |0\rangle|\Psi_0\rangle - \frac{1}{Ca^2+1}|T\rangle|\Psi_T\rangle - \frac{Ca^2}{Ca^2+1}|T\rangle|\Psi_T\rangle = |\tau\rangle.\end{aligned}$$

This proves that $|w_0\rangle$ is the minimal witness of $A$. We conclude the proof by computing its squared norm $N = \||w_0\rangle\|^2$,

$$\begin{aligned}N &= \sum_{t=0}^{q_1-2}\frac{1}{M^2} + \frac{1}{4} + \frac{1}{4} + \frac{1}{(Ca^2+1)^2}\left[\frac{1}{4}+\frac{1}{4}+\sum_{t=q_S}^{T-1}\frac{1}{M^2}\right] + \frac{C^2a^2}{(Ca^2+1)^2}\\ &= \frac{q_1-1}{M^2} + \frac{1}{2} + \frac{C}{(Ca^2+1)^2} + \frac{C^2a^2}{(Ca^2+1)^2} = \frac{q_1-1}{M^2} + \frac{1}{2} + \frac{C}{Ca^2+1}.\end{aligned}$$

□

## 4.5 Time complexity of the algorithm

Now we analyze the time complexity of implementing $P_{\mathcal{A}}$. Remember that in Section 4.3 we broke down the span program algorithm into its essential subroutines, namely, the reflections around $\ker(A)$, $\mathcal{H}(x)$, $|0\rangle$ and the circuit that generates $|w_0\rangle$. The bulk of this section (and bear with us, it is a hefty one) will be devoted to constructing said subroutines and characterizing their time complexity. All this will then be combined to prove the following theorem.

**Theorem 63.** *Let $\mathcal{A}$ be a clean quantum query algorithm that acts on $k$ qubits, has query complexity $S$, time complexity $T$, and evaluates a function $f : X \subseteq \{0,1\}^n \to \{0,1\}$ with bounded error as in Definition 6. Let $P_{\mathcal{A}}$ be the span program for this algorithm, as in Definition 57. Then we can implement the algorithm compiled from $P_{\mathcal{A}}$ with:*

1. *$\mathcal{O}(S)$ calls to $\mathcal{O}_x$.*
2. *$\mathcal{O}(T)$ calls to $\mathcal{O}_{\mathcal{A}}$ and $\mathcal{O}_{\mathcal{S}}$, as defined in Section 4.2.*
3. *$\mathcal{O}(T\text{polylog}(T))$ additional gates.*
4. *$\mathcal{O}(\text{polylog}(T))$ auxiliary qubits.*

As we just said, the proof leans on the structure of Theorem 53. We first define a suitable implementing subspace in Section 4.5.1, and then continue with providing efficient implementations of the four subroutines that are required to use Theorem 53. In fact, we provide generalizations of all four subroutines (the reflections around $\ker(A)$, $\mathcal{H}(x)$, $|0\rangle$ and the circuit $\mathcal{C}_{|w_0\rangle} : |0\rangle \mapsto |w_0\rangle$) to the case where we have several algorithms and we want to run each subroutine in coherent superposition for all algorithms. We call this *concurrent access*, and define it formally as:

**Definition 64.** Let $\mathcal{C}^{(1)}, \ldots, \mathcal{C}^{(n)}$ be quantum subroutines, all acting on the same Hilbert space $\mathcal{H}$. We say that a subroutine $\mathcal{C}$ provides *concurrent access* to $\{\mathcal{C}^{(j)}\}_{j=1}^n$ if it performs the following action on $\mathbb{C}^{[n]} \otimes \mathcal{H}$:

$$\mathcal{C} = \sum_{j=1}^{n} |j\rangle\langle j| \otimes \mathcal{C}^{(j)}.$$

This is not strictly necessary for the theorem above, but will be a necessity in the final section of the chapter, where we make an algorithm that computes the OR of several functions out of algorithms computing each one.

### 4.5.1 Implementing subspace

The necessity for the introduction of implementing subspaces stems from the need to reflect around the state $|0\rangle|0\rangle|\Psi_0\rangle$. Doing this in principle is very simple because we can assume that $|\Psi_0\rangle = |\psi_0\rangle|0\rangle$ where $|\psi_0\rangle \in \mathbb{C}^{\otimes(\log(n)+\log(|\mathcal{W}|))}$. However, in general, the time complexity of this reflection is $\Theta(\log(n)+\log(|\mathcal{W}|))$, since it is necessary to check that every qubit is in the $|0\rangle$ state.[2] That would make such a reflection very costly for algorithms with high space complexity, such as the element distinctness algorithm of [Amb07]. Specifically, if the time complexity of $\mathcal{A}$ is polynomially related to the number of qubits used in $\mathcal{A}$, then we find that $\log(n) + \log(|\mathcal{W}|) = \Theta(\text{poly}(T))$. In this section, we explain how we circumvent this polynomial dependence using implementing subspaces.

Our construction relies on the fact that at intermediate steps the state of an algorithm compiled from $P_{\mathcal{A}}$ is not completely arbitrary but is guaranteed to live within an implementing subspace. In other words, of all states in the workspace of $\mathcal{A}$, any given run will only visit a simple, one-dimensional path in the workspace. Moreover, all such states are labeled by the time register. So, given a time step $t \in [T]_0$ and an input $x \in X$, we can deduce what the corresponding state in algorithm $\mathcal{A}$ must be at that time step, and hence we can deduce the state in the last register of $\mathcal{H}$.

For the implementing subspace that we will describe below, we will show that for every $x \in X$, when $t = 0$, the algorithm compiled from $P_{\mathcal{A}}$ only visits superpositions of states of the form $|t, b, i, j, a\rangle = |0, 0, \psi_0, a\rangle$, where $|\psi_0\rangle|0\rangle = |\Psi_0\rangle$ is the initial state of the algorithm and the last index $a = 0, 1$ represents the single qubit answer register. Therefore, In order to reflect around the all-zeroes state, we need not check all registers. It is enough to check that the time and the answer registers are at zero. Furthermore, while the implementing subspace depends on the input, this property is independent of it, and so we do not need to know the exact implementing subspace to run the algorithm, just that it exists.

Unfortunately, we are not able to provide an exact implementing sub-

[2] This detail has been neglected in previous work. For example, efficient implementation of such a reflection is not discussed in [Amb10]. While this is not inconsistent with the stated results, since the result only claims to count oracle calls to $\mathcal{O}_x$ and $\mathcal{O}_{\mathcal{A}}$, this is not true of subsequent work that uses the results of [Amb10]. We suspect that an argument like ours could also be made in previous work, but feel it is sufficiently non-trivial that it should not be taken for granted.

space. Instead, we will use an *approximate implementing subspace*, i.e., we define a subspace $\mathcal{H}_x \subseteq \mathcal{H}$ and we prove that all operations map states in $\mathcal{H}_x$ to states that have high overlap with $\mathcal{H}_x$. The way we handle the propagation of errors is similar to standard approximation arguments: if the overlap with $\mathcal{H}_x^\perp$ after one approximate operation is at most $\delta$, then the combined error after $N$ such approximation operations is at most $\mathcal{O}(N\delta)$. Hence, if we make sure that $\delta < o(1/N)$, then the total cumulative "lost amplitude" is $o(1)$, and the influence on the final success probability of the algorithm is at most $o(1)$ as well.

Now, we work towards the formal definition of $\mathcal{H}_x$ for a clean quantum algorithm $\mathcal{A}$ as defined in Def. 6. Remember that in a clean quantum algorithm, the last bit of the workspace is the answer register, and the initial state is $|\Psi_0\rangle = |\psi_0, 0\rangle$. For each $x \in X$ and $t \in \{0, \ldots, T-1\}$, we define the following vectors:

$$|\Psi_t^{(0)}(x)\rangle = U_t \cdots U_1 |\psi_0, 0\rangle = U_{t;1}|\psi_0, 0\rangle,$$
$$|\Psi_t^{(1)}(x)\rangle = U_t \cdots U_1 |\psi_0, 1\rangle = U_{t;1}|\psi_0, 1\rangle,$$

which we name the *t-step push-forwards* of $|\Psi_0\rangle = |\psi_0, 0\rangle$ and $|\Psi_T\rangle = |\psi_0, 1\rangle$ respectively. These are just fancy, shorthand names for the result of running the $t$ steps of the algorithm on $|\psi_0, 0\rangle$ or $|\psi_0, 1\rangle$.

Alternatively, we could run the algorithm in reverse using $|\psi_0, 0\rangle$ or $|\psi_0, 1\rangle$ as initial states. Thus, we define the *t-step pullbacks* of $|\psi_0, 0\rangle$ and $|\psi_0, 1\rangle$ as

$$|\widetilde{\Psi}_t^{(0)}(x)\rangle = U_{t+1}^\dagger \cdots U_T^\dagger |\psi_0, 0\rangle = U_{t+1;T}^\dagger |\psi_0, 0\rangle,$$
$$|\widetilde{\Psi}_t^{(1)}(x)\rangle = U_{t+1}^\dagger \cdots U_T^\dagger |\psi_0, 1\rangle = U_{t+1;T}^\dagger |\psi_0, 1\rangle.$$

The reason we bother defining these pull-backs and push-forwards is that they are part of the minimal witness $|w_0\rangle$ and need to be considered in order to build an efficient reflection around the state $|0\rangle|\psi_0, 0\rangle$. Moreover, these vectors are intimately related by the fact that $\mathcal{A}$ is a clean quantum algorithm. Indeed we have

**Claim 2.** For any clean quantum algorithm $\mathcal{A}$, all $x \in \{0,1\}^n$, and all $t \in [T]_0$:

$$\langle \Psi_t^{(0)}(x) | \widetilde{\Psi}_t^{(1)}(x) \rangle = \langle \Psi_t^{(1)}(x) | \widetilde{\Psi}_t^{(0)}(x) \rangle = p_1(x),$$
$$\langle \Psi_t^{(0)}(x) | \widetilde{\Psi}_t^{(0)}(x) \rangle = \langle \Psi_t^{(1)}(x) | \widetilde{\Psi}_t^{(1)}(x) \rangle = p_0(x),$$
$$\langle \Psi_t^{(0)}(x) | \Psi_t^{(1)}(x) \rangle = \langle \widetilde{\Psi}_t^{(0)}(x) | \widetilde{\Psi}_t^{(1)}(x) \rangle = 0.$$

*Proof.* The first statement follows from Definition 6. Since $|\Psi_T\rangle = |\psi_0, 1\rangle$

$$p_1(x) = \langle \Psi_T^{(0)}(x)|\Psi_T\rangle = \langle \Psi_t^{(0)}(x)|U_{t+1;T}^\dagger|\psi_0, 1\rangle = \langle \Psi_t^{(0)}(x)|\widetilde{\Psi}_t^{(1)}(x)\rangle.$$

The proofs of the other statements are very similar and are left to the reader. □

Now, we want to define an implementing subspace $\mathcal{H}_x$ such that it is left invariant by $\Pi_{\ker(A)}$ and $\Pi_{\mathcal{H}(x)}$, and contains $|0\rangle|\psi_0, 0\rangle$, and $|w_0\rangle$. Therefore, our implementing subspace had better respect the structure of $|w_0\rangle$ and the vectors of $\ker(A)$ that we have studied before. It is reasonable, looking at those vectors, to treat each block separately. Therefore, we propose the following spaces. All the arguments of the vector sums below should be enclosed in a span{}, but we omit those for ease of notation. For the first block $\ell = 1$, we define

$$\mathcal{H}_{x,1}^{(b)} = \bigoplus_{t=0}^{q_1-2} |t\rangle|0\rangle|\Psi_t^{(b)}(x)\rangle \oplus |q_1 - 1\rangle|+\rangle|\Psi_{q_1-1}^{(b)}(x)\rangle. \tag{4.15}$$

Observe that this contains the initial vector $|0, 0\rangle|\psi_0, 0\rangle$ if $b = 0$.

For $\ell = S + 1$, we define

$$\mathcal{H}_{x,S+1}^{(b)} = |q_S - 1\rangle|-\rangle|\widetilde{\Psi}_{q_S}^{(f(x)\oplus b)}(x)\rangle \oplus \bigoplus_{t=q_S}^{T} |t\rangle|0\rangle|\widetilde{\Psi}_t^{(f(x)\oplus b)}(x)\rangle. \tag{4.16}$$

Observe that this contains the target state of the algorithm with the correct time label $|T\rangle|0\rangle|\Psi_T\rangle = |T, 0\rangle|\psi_0, 1\rangle$ if $b = 1$.

For the other blocks, i.e. $\ell = 2, \ldots, S$, let $\theta = \arccos p_{f(x)}(x)$. For all $b \in \{0, 1\}$, we define the ($b$-)implementing subspace of block $\ell$ as

$$\begin{aligned}\mathcal{H}_{x,\ell}^{(b)} = \Bigg( & |q_{\ell-1} - 1\rangle|-\rangle \left[ \frac{\sin\frac{(\ell-1)\theta}{S}}{\sin\theta} |\widetilde{\Psi}_{q_{\ell-1}}^{(f(x)+b)}(x)\rangle + \frac{\sin\left(\theta - \frac{(\ell-1)\theta}{S}\right)}{\sin\theta} |\Psi_{q_{\ell-1}}^{(b)}(x)\rangle \right] \\ & \oplus \bigoplus_{t=q_{\ell-1}}^{q_\ell-2} |t\rangle|0\rangle \left[ \frac{\sin\frac{(\ell-1)\theta}{S}}{\sin\theta} |\widetilde{\Psi}_t^{(f(x)+b)}(x)\rangle + \frac{\sin\left(\theta - \frac{(\ell-1)\theta}{S}\right)}{\sin\theta} |\Psi_t^{(b)}(x)\rangle \right] \\ & \oplus |q_\ell - 1\rangle|+\rangle \left[ \frac{\sin\frac{(\ell-1)\theta}{S}}{\sin\theta} |\widetilde{\Psi}_{q_\ell-1}^{f(x)+b}(x)\rangle + \frac{\sin\left(\theta - \frac{(\ell-1)\theta}{S}\right)}{\sin\theta} |\Psi_{q_\ell-1}^{(b)}(x)\rangle \right] \Bigg).\end{aligned}$$

Alright, there is a lot to unpack here. Let's see what we have done. First observe that the first two registers are consistent with the form of the vectors in $\ker(A)$, that is, the edges of the block have a $|+\rangle$ or $|-\rangle$ in the second register, while the non-query time-steps have a $|0\rangle$ in the second register. In the third register we have a superposition of pull-backs and push-forwards that depends on the index of the block $\ell$ and are always consistent with the value in the time register. As $\ell$ grows, the vector in the implementing subspace approaches the pullbacks. Finally, in the last block, the vectors in the implementing subspace are merely pullbacks from $|\psi_0, b\rangle$. The reasons behind this choice are rather technical and all relate to the need to satisfy all four constraints of an implementing subspace at the same time, while still making it easy to reflect around the state $|t\rangle|0\rangle|\psi_0, 0\rangle$.

Finally, we define the implementing subspace as:

**Definition 65** (Implementing subspace for $\mathcal{A}$)**.** Let $\mathcal{A}$ be a clean quantum query algorithm that decides a function $f : X \to \{0, 1\}$, let $P_{\mathcal{A}}$ be the span program for this algorithm, as in Definition 57. Then we define the implementing subspace of $\mathcal{A}$ as:

$$\mathcal{H}_x = \begin{cases} \bigoplus_{\ell=1}^{S+1} \mathcal{H}_{x,\ell}^{(0)}, & \text{if } f(x) = 1, \\ \bigoplus_{\ell=1}^{S+1} \mathcal{H}_{x,\ell}^{(0)} \oplus \mathcal{H}_{x,\ell}^{(1)}, & \text{if } f(x) = 0. \end{cases}$$

Now we prove that this is an approximate implementing subspace.

**Lemma 66.** *For all $x \in X$, we have*

1. $\Pi_{\ker(A)} \mathcal{H}_x \subseteq \mathcal{H}_x$,
2. $|w_0\rangle \in \mathcal{H}_x$,
3. $|0\rangle|0\rangle|\psi_0, 0\rangle \in \mathcal{H}_x$,
4. *For all $|h\rangle \in \mathcal{H}_x$, we have* $\|\Pi_{\mathcal{H}_x^\perp}(2\Pi_{\mathcal{H}(x)} - I)|h\rangle\| \leq \mathcal{O}\left(\frac{\sqrt{\varepsilon}}{S}\right)$.

*Proof.* First of all, observe that the state $|0\rangle|0\rangle|\psi_0, 0\rangle$ is in $\mathcal{H}_x^{(0)}$. We also easily obtain that the initial state $|w_0\rangle$ from Lemma 62 is in $\mathcal{H}_{x,1}^{(0)} \oplus \mathcal{H}_{x,S+1}^{(f(x)\oplus 1)}$ by inspection.

Next, we show that the projection onto the kernel of $A$ also leaves the implementing subspace invariant. To that end, we observe that by Lemma 61,

$\ker(A) = \bigoplus_{\ell=2}^{S+1} \mathrm{Im}(\Phi_\ell)$ for some maps $\Phi_\ell$ defined in the lemma. Se can write the projection onto the kernel as:

$$\Pi_{\ker(A)} = \sum_{\ell=2}^{S} \frac{\Phi_\ell \Phi_\ell^\dagger}{1 + \frac{|\mathcal{B}_\ell|}{M^2}} + \frac{\Phi_{S+1}\Phi_{S+1}^\dagger}{\frac{1}{2} + \frac{|\mathcal{B}_{S+1}|}{M^2} + \frac{1}{a^2}}.$$

We will show that each term in the summation leaves $\mathcal{H}_x$ invariant. The last term follows using the exact same proof strategy. Hence, let $\ell \in \{2, \ldots, S\}$, $b \in \{0, 1\}$ and $|h\rangle \in \mathcal{H}_{x,\ell}^{(b)}$. We can disregard any part of $|h\rangle$ that is not supported on the image of $\Phi_\ell$, as it will be mapped to 0 by $\Phi_\ell^\dagger$. Hence, we assume $|h\rangle$ to be of the form

$$\begin{aligned}
|h\rangle = \alpha_{q_{\ell-1}-1}|q_{\ell-1}-1\rangle|-\rangle &\left[\frac{\sin\frac{(\ell-1)\theta}{S}}{\sin\theta}|\widetilde{\Psi}_{q_{\ell-1}}^{(f(x)+b)}(x)\rangle + \frac{\sin\left(\theta - \frac{(\ell-1)\theta}{S}\right)}{\sin\theta}|\Psi_{q_{\ell-1}}^{(b)}(x)\rangle\right] \\
+ \sum_{t=q_{\ell-1}}^{q_\ell-2} \alpha_t|t\rangle|0\rangle &\left[\frac{\sin\frac{(\ell-1)\theta}{S}}{\sin\theta}|\widetilde{\Psi}_t^{(f(x)+b)}(x)\rangle + \frac{\sin\left(\theta - \frac{(\ell-1)\theta}{S}\right)}{\sin\theta}|\Psi_t^{(b)}(x)\rangle\right] \\
+ \alpha_{q_\ell-1}|q_\ell-1\rangle|+\rangle &\left[\frac{\sin\frac{(\ell-1)\theta}{S}}{\sin\theta}|\widetilde{\Psi}_{q_\ell-1}^{f(x)+b}(x)\rangle + \frac{\sin\left(\theta - \frac{(\ell-1)\theta}{S}\right)}{\sin\theta}|\Psi_{q_\ell-1}^{(b)}(x)\rangle\right].
\end{aligned}$$

In this state, the last part is determined by the fact that $|h\rangle \in \mathcal{H}_{x,\ell}^{(b)}$. The spaces $\mathcal{H}_{x,\ell}^{(b)}$ have been defined so as to fit $\mathrm{Im}(\Phi_\ell)$ perfectly, and so this does not restrict the form of $h$ further. The free parameters are the weights $\alpha_i$. We observe that $\Phi_\ell^\dagger$ maps this to

$$\begin{aligned}
&\left[\frac{1}{\sqrt{2}}\alpha_{q_{\ell-1}-1} + \frac{1}{M}\sum_{t=q_\ell}^{q_\ell-2}\alpha_t + \frac{1}{\sqrt{2}}\alpha_{q_\ell-1}\right] \cdot \\
&\left[\frac{\sin\frac{(\ell-1)\theta}{S}}{\sin\theta}|\widetilde{\Psi}_{q_{\ell-1}}^{(f(x)+b)}(x)\rangle + \frac{\sin\left(\theta - \frac{(\ell-1)\theta}{S}\right)}{\sin\theta}|\Psi_{q_{\ell-1}}^{(b)}(x)\rangle\right].
\end{aligned}$$

As the image of $\Phi_\ell$ applied to the above expression is contained in $\mathcal{H}_x^{(b)}$, we deduce that $\left(2\Pi_{\ker(A)} - I\right)|h\rangle \in \mathcal{H}_{x,\ell}^{(b)}$. By linearity, it then follows that $\left(2\Pi_{\ker(A)} - I\right)|h\rangle \in \mathcal{H}_x$ for any $|h\rangle \in \mathcal{H}_x$.

Finally, we argue how reflecting through $\mathcal{H}(x)$ leaves the implementing subspace approximately invariant. Observe that for all blocks, $2\Pi_{\mathcal{H}(x)} - I$ acts as the identify on all states with time-step $t \neq q_\ell - 1, q_{\ell-1} - 1$. That leaves us with states of the form $|q_{\ell_1} - 1\rangle|-\rangle|\cdot\rangle$ and $|q_\ell - 1\rangle|+\rangle|\cdot\rangle$.Let us deal with the second case first. Let $|\Psi^{(b)}_{q_\ell - 1}(x)\rangle = \sum_{i\in[n], j\in\mathcal{W}} \alpha_{i,j}|i,j\rangle$ and write $|+\rangle = (|x_i\rangle + |\overline{x}_i\rangle)/\sqrt{2}$. Then

$$\begin{aligned}\left(2\Pi_{\mathcal{H}(x)} - I\right)|q_\ell - 1\rangle|+\rangle|\Psi^{(b)}_{q_\ell-1}(x)\rangle &= \left(2\Pi_{\mathcal{H}(x)} - I\right)|q_\ell - 1\rangle|+\rangle \sum_{\substack{i\in[n],\\ j\in\mathcal{W}}} \alpha_{i,j}|i,j\rangle \\ &= |q_\ell - 1\rangle \sum_{\substack{i\in[n],\\ j\in\mathcal{W}}} \frac{(|x_i\rangle - |\overline{x}_i\rangle)}{\sqrt{2}}\alpha_{i,j}|i,j\rangle = |q_\ell - 1\rangle \sum_{\substack{i\in[n],\\ j\in\mathcal{W}}} (-1)^{x_i}|-\rangle\alpha_{i,j}|i,j\rangle \\ &= |q_\ell - 1\rangle|-\rangle(\mathcal{O}_x \otimes I)|\Psi^{(b)}_{q_\ell-1}(x)\rangle = |q_\ell - 1\rangle|-\rangle|\Psi^{(b)}_{q_\ell}(x)\rangle.\end{aligned}$$

Since $\mathcal{O}_x = \mathcal{O}_x^\dagger$, following the exact same reasoning we have

$$\left(2\Pi_{\mathcal{H}(x)} - I\right)|q_\ell - 1\rangle|+\rangle|\widetilde{\Psi}^{(b)}_{q_\ell-1}(x)\rangle = |q_\ell - 1\rangle|-\rangle|\widetilde{\Psi}^{(b)}_{q_\ell}(x)\rangle.$$

The case where $|h\rangle = |q_{\ell_1} - 1\rangle|-\rangle|\cdot\rangle$ follows immediately from the proof because the reflection $\left(2\Pi_{\mathcal{H}(x)} - I\right)$ is its own inverse.

A problem arises here because states of the form $|q_\ell - 1\rangle|+\rangle|\cdot\rangle$ are in $\mathcal{H}^{(b)}_{x,\ell}$, but states of the form $|q_\ell - 1\rangle|-\rangle|\cdot\rangle$ are in the next block's implementing subspace $\mathcal{H}^{(b)}_{x,\ell+1}$. This would be alright except that consecutive blocks put slightly different weights on the push-forwards and the pull-backs. Therefore, or task now is to show that the reflection around $\mathcal{H}(x)$ takes a vector in $\mathcal{H}^{(b)}_{x,\ell}$ and maps it to a vector close to $\mathcal{H}^{(b)}_{x,\ell+1}$. Without loss of generality, we restrict our attention to vectors of the form

$$|h\rangle = |q_\ell - 1\rangle|+\rangle\left[\frac{\sin\frac{(\ell-1)\theta}{S}}{\sin\theta}|\widetilde{\Psi}^{(f(x)+b)}_{q_\ell-1}(x)\rangle + \frac{\sin\left(\theta - \frac{(\ell-1)\theta}{S}\right)}{\sin\theta}|\Psi^{(b)}_{q_\ell-1}(x)\rangle\right]$$

with $\ell \in [S]$ and $b \in \{0,1\}$, or the corresponding form with $|-\rangle$ and $\ell \in \{2,\ldots,S+1\}$. It follows that

$$\begin{aligned}&\left(2\Pi_{\mathcal{H}(x)} - I\right)|h\rangle \\ &\quad = |q_\ell - 1\rangle|-\rangle\left[\frac{\sin\frac{(\ell-1)\theta}{S}}{\sin\theta}|\widetilde{\Psi}^{(f(x)+b)}_{q_\ell}(x)\rangle + \frac{\sin\left(\theta - \frac{(\ell-1)\theta}{S}\right)}{\sin\theta}|\Psi^{(b)}_{q_\ell}(x)\rangle\right]\end{aligned}$$

This vector is not in $\mathcal{H}^{(b)}_{x,\ell+1}$, but the following vector $|h'\rangle$ is:

$$|h'\rangle = |q_\ell - 1\rangle|-\rangle \left[ \frac{\sin\frac{(\ell)\theta}{S}}{\sin\theta} |\widetilde{\Psi}^{(f(x)+b)}_{q_\ell}(x)\rangle + \frac{\sin\left(\theta - \frac{(\ell)\theta}{S}\right)}{\sin\theta} |\Psi^{(b)}_{q_\ell}(x)\rangle \right].$$

It follows that

$$\left\| \Pi_{H_x^\perp}(2\Pi_{\mathcal{H}(x)} - I)|h\rangle \right\| \leq \left\| (2\Pi_{\mathcal{H}(x)} - I)|h\rangle - |h'\rangle \right\|.$$

It is this last norm what we shall bound. See that the last registers are both in a superposition of two states that are not orthogonal. For the purposes of taking the norm, we would like to express these vectors as sums of two orthogonal vectors. Define

$$|\psi^\perp\rangle = \frac{|\widetilde{\Psi}^{(f(x)+b)}_{q_\ell}(x)\rangle - \cos\theta|\Psi^{(b)}_{q_\ell}(x)\rangle}{\sin\theta}.$$

It is imminent that $|\psi^\perp\rangle$ and $|\Psi^{(b)}_{q_\ell}(x)\rangle$ are orthogonal, as

$$\langle\Psi_{q_\ell}(x)|\psi^\perp\rangle = \frac{p_{f(x)}(x)}{\sin\theta} - \frac{\cos\theta}{\sin\theta} = 0,$$

where we recall that we defined $\theta = \arccos p_{f(x)}(x)$. Moreover, we find that

$$\begin{aligned}
&\frac{\sin(a\theta)}{\sin\theta}|\widetilde{\Psi}^{(f(x)+b)}_{q_\ell}(x)\rangle + \frac{\sin(\theta - a\theta)}{\sin\theta}|\Psi^{(b)}_{q_\ell}(x)\rangle \\
&= \sin(a\theta)|\psi^\perp\rangle + \left[\frac{\sin(\theta - a\theta)}{\sin\theta} + \frac{\sin(a\theta)}{\sin\theta}\cos\theta\right]|\Psi^{(b)}_{q_\ell}(x)\rangle \\
&= \sin(a\theta)|\psi^\perp\rangle + \cos(a\theta)|\Psi^{(b)}_{q_\ell}(x)\rangle.
\end{aligned}$$

Using this relation with $a = \frac{\ell-1}{S}$, and $a = \frac{\ell}{S}$ we can now compute the norm

of $\Pi_{H_x^\perp}(2\Pi_{\mathcal{H}(x)} - I)|h\rangle$,

$$\begin{aligned}
&\left\|\Pi_{H_x^\perp}(2\Pi_{\mathcal{H}(x)} - I)|h\rangle\right\|^2 \leq \left\|(2\Pi_{\mathcal{H}(x)} - I)|h\rangle - |h'\rangle\right\|^2 \\
&\leq \left\| \frac{\sin\frac{\ell\theta}{S} - \sin\frac{(\ell-1)\theta}{S}}{\sin\theta} |\widetilde{\Psi}_{q_\ell}^{(f(x)+b)}(x)\rangle + \frac{\sin\left(\theta - \frac{\ell\theta}{S}\right) - \sin\left(\theta - \frac{(\ell-1)\theta}{S}\right)}{\sin\theta} |\Psi_{q_\ell}^{(b)}(x)\rangle \right\|^2 \\
&= \left\| \left(\sin\frac{\ell\theta}{S} - \sin\frac{(\ell-1)\theta}{S}\right)|\psi^\perp\rangle + \left(\cos\frac{\ell\theta}{S} - \cos\frac{(\ell-1)\theta}{S}\right)|\Psi_{q_\ell}^{(b)}(x)\rangle \right\|^2 \\
&= \left(\sin\frac{\ell\theta}{S} - \sin\frac{(\ell-1)\theta}{S}\right)^2 + \left(\cos\frac{\ell\theta}{S} - \cos\frac{(\ell-1)\theta}{S}\right)^2 \leq 2\left(\frac{\theta}{S}\right)^2 \\
&= \frac{2\arccos^2 p_{f(x)}(x)}{S^2} \leq \frac{2\arccos^2(1-\varepsilon)}{S^2} \leq \frac{2}{S^2}\left(2\varepsilon + \mathcal{O}(\varepsilon^2)\right) = \mathcal{O}\left(\frac{\varepsilon}{S^2}\right),
\end{aligned}$$

where in the fourth line we used that sin and cos are both 1-Lipschitz, and so the Mean Value Theorem applies. The same bound can be obtained when $|h\rangle$ is of the slightly different form with $|-\rangle$. Hence, we conclude that

$$\begin{aligned}
\sup_{|h\rangle\in\mathcal{H}_x} \left\|\Pi_{\mathcal{H}_x^\perp}\left(2\Pi_{\mathcal{H}(x)} - I\right)|h\rangle\right\| &\leq \max_{b\in\{0,1\}} \sup_{|h\rangle\in\mathcal{H}_x^{(b)}} \left\|\left(I - \Pi_{\mathcal{H}_x^{(b)}}\right)\left(2\Pi_{\mathcal{H}(x)} - I\right)|h\rangle\right\| \\
&= \mathcal{O}\left(\frac{\sqrt{\varepsilon}}{S}\right).
\end{aligned}$$

□

### 4.5.2 Reflection around $|0\rangle$

The implementing subspace is a rather complicated object, and defining it and proving its fitness has been quite the hurdle. Now we shall reap the benefits of all that effort with the implementation of the first of the subroutines of Theorem 53, the reflection around a simple computational state $|0\rangle|0\rangle|\Psi_0\rangle \in \mathcal{H}$ restricted to $\mathcal{H}_x$. Let us see how to easily implement this unitary.

**Lemma 67.** *Let $\mathcal{A}$ be a clean quantum query algorithm with query complexity $S$, and time complexity $T$. Let $P_\mathcal{A} = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$ be the span program for this algorithm, as in Definition 57. Let $|0\rangle := |0\rangle|0\rangle|\psi_0, 0\rangle$ We can implement a map $G$ that, when restricted to $\mathcal{H}_x$, acts as $\mathcal{R}_{|0\rangle} = (2|0\rangle\langle 0| - I)$ with $\mathcal{O}(1)$ auxiliary qubits and $\mathcal{O}\left(\text{polylog}(T)\right)$ gates.*

*Proof.* As in Definition 6, we assume the basis states of the workspace are labeled by $\mathcal{W} = \mathcal{W}' \times \{0,1\}$, so the last qubit is the answer register. The basis states of the overall space $\mathcal{H}$ are $|t\rangle|b\rangle|i,j,a\rangle$, where $i \in [n]$, $j \in \mathcal{W}'$, and $a \in \{0,1\}$ is the content of the answer register. The map $G$ reflects around states with $t = 0$, and $a = 0$:

$$G|t\rangle|b\rangle|i,j,a\rangle = \begin{cases} |t\rangle|b\rangle|i,j,a\rangle & \text{if } t = a = 0, \\ -|t\rangle|b\rangle|i,j,a\rangle & \text{else.} \end{cases}$$

To implement this reflection, we simply compute a bit $b_{\text{flag}}$ in a new register such that $b_{\text{flag}} = 0$ if and only if $t = 0$, and $a = 0$. Then we can apply a $Z$-gate on this register, and then uncompute. Since $t \in [T]_0$, $a \in \{0,1\}$, this can be done in time $\mathcal{O}(\log T)$.

By definition, the only vectors in $\mathcal{H}_x$, with a time register $t = 0$ are $|0\rangle|0\rangle|\psi_0, 0\rangle$, and, if $f(x) = 0$, $|0\rangle|0\rangle|\psi_0, 1\rangle$. It follows that $G$ acts as $2|0\rangle\langle 0| - I$ on $\mathcal{H}_x$. □

### 4.5.3 Implementation of $2\Pi_{\mathcal{H}(x)} - I$

The second subroutine we deal with is the reflection around $\mathcal{H}(x)$. We give a simple construction that perfectly implements this reflection on the whole of $\mathcal{H}$. The is made very simple given the structure of $\mathcal{H}$ as all we need to do to know if a state $|t,b,i,j\rangle$ is in $\mathcal{H}(x)$ is check if $t+1 \in \mathcal{S}$ using a query to $\mathcal{O}_{\mathcal{S}}$ and, if so, check if $b = x_i$, which we do with a single query to the oracle $\mathcal{O}_x$. We give a detailed construction that also extends to the concurrent case (see Definition 64) in the lemma below.

**Lemma 68.** *Let $\mathcal{A}$ be a clean quantum query algorithm with query complexity $S$, time complexity $T$, and error probability $\varepsilon$. Let $P_{\mathcal{A}} = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$ be the span program for this algorithm, as in Definition 57. Then the reflection $2\Pi_{\mathcal{H}(x)} - I$ can be implemented with $\mathcal{O}(1)$ calls to $\mathcal{O}_x$ and $\mathcal{O}_{\mathcal{S}}$ and auxiliary qubits, and $\mathcal{O}(\text{polylog}(T))$ extra gates.*

*Similarly, let $\{\mathcal{A}^{(j)}\}_{j=1}^n$ be a set of clean quantum query algorithms. For all $j \in [n]$, let $S_j$ and $T_j$ denote the query and time complexity of $\mathcal{A}^{(j)}$. Let $P^{(j)}$ be the span program of $\mathcal{A}^{(j)}$. Then we can implement concurrent access (see Def. 64) to $\{2\Pi_{\mathcal{H}^{(j)}(x^{(j)})} - I\}_{j=1}^n$ with $\mathcal{O}(1)$ calls to $\mathcal{O}_x$ and $\mathcal{O}_{\mathcal{S}}$ and auxiliary qubits, and $\mathcal{O}(\text{polylog}(T_{\max}))$ extra gates.*

*Proof.* First, we consider the case where we just have one algorithm, $\mathcal{A}$. For all $x \in \{0,1\}^n$, recall that

$$\mathcal{H}(x) = \bigoplus_{i=1}^{n} \mathcal{H}_{i,x_i} \oplus \mathcal{H}_{\text{true}} = \text{span}\{|t, x_i, i, j\rangle : t+1 \in \mathcal{S}, i \in [n], j \in \mathcal{W}\}$$
$$\oplus \text{span}\{|t, 0, i, j\rangle : t+1 \in [T+1] \setminus \mathcal{S}, i \in [n], j \in \mathcal{W}\}.$$

From this, and the definition of $\mathcal{H}$, it readily follows that the orthogonal complement of $\mathcal{H}(x)$ is given by

$$\mathcal{H}(x)^{\perp} = \text{span}\{|t, 1-x_i, i, j\rangle : t+1 \in \mathcal{S}, i \in [n], j \in \mathcal{W}\}.$$

In order to reflect around $\mathcal{H}(x)$, all we have to do is put a minus phase if we are in $\mathcal{H}(x)^{\perp}$. To that end, call the oracle $\mathcal{O}_{\mathcal{S}}$ once to distinguish whether the time step in the first register is a state $|t\rangle$ for which $t+1 \in \mathcal{S}$ (e.g. by first increasing the first register, performing the call and then decreasing again). Store this bit in an auxiliary flag register. Next, conditioned on the flag qubit being 1, perform one query to $\mathcal{O}_x$ to get a phase $(-1)^{x_i}$. Finally, apply $-Z$ to the second register, also controlled on the flag qubit, where $Z$ is the Pauli-$Z$ gate. Then if the second register is in the state $|1-x_i\rangle$, the overall phase will be $-(-1)^{x_i+1-x_i} = (-1)$, and if it is in the state $|x_i\rangle$, the overall phase will be $(+1)$, as desired. Finally, we need to uncompute the flag qubit, which again takes one call to $\mathcal{O}_{\mathcal{S}}$. All the other operations can be implemented in a number of elementary gates that is polylogarithmic in $T$.

If we instead have multiple algorithms $\{\mathcal{A}^{(j)}\}_{j=1}^{n}$, then all that changes is the size of the time register. It is now of size $T_{\max} = \max_{j\in[n]} T_j$, and hence the arithmetic operations on it now require $\mathcal{O}(\text{polylog}(T_{\max}))$ gates. This completes the proof. □

### 4.5.4 Implementation of $2\Pi_{\ker(A)} - I$

In this section, we prove Lemma 69, i.e., we provide an implementation of the routine that reflects around the kernel of the span program operator $A$ as defined in Eq. (4.7). In addition, we also elaborate on how one would obtain concurrent access to these routines when considering multiple such span program operators, because we need that in the proof of Theorem 78. The result is summarized in the following lemma.

**Lemma 69.** *Let $\mathcal{A}$ be a clean quantum query algorithm with query complexity $S$ and time complexity $T$. Let $P_{\mathcal{A}} = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$ be the span program for this algorithm, as in Definition 57. Then, the reflection $2\Pi_{\ker(A)} - I$ can be implemented to error $\delta > 0$ with $\mathcal{O}(T/S)$ calls to $\mathcal{O}_{\mathcal{A}}$ and $\mathcal{O}_{\mathcal{S}}$, $\mathcal{O}(\text{polylog}(T))$ auxiliary qubits and a number of extra gates that satisfies*

$$\mathcal{O}\left(\frac{T}{S}\text{polylog}\left(T, \frac{1}{\delta}\right)\right).$$

*Similarly, let $\{\mathcal{A}^{(j)}\}_{j=1}^{n}$ be a set of clean quantum query algorithms. For all $j \in [n]$, let $S_j$ and $T_j$ be the query and time complexity of $\mathcal{A}^{(j)}$, respectively. Let $P^{(j)} = (\mathcal{H}^{(j)}, \mathcal{V}^{(j)}, A^{(j)}, |\tau^{(j)}\rangle)$ be the span program of $\mathcal{A}^{(j)}$. We can provide concurrent access to $\{2\Pi_{\ker(A^{(j)})} - I\}_{j=1}^{n}$ up to precision $\delta > 0$ with $\mathcal{O}(\max_{j\in[n]} T_j/S_j)$ calls to $\mathcal{O}_{\mathcal{A}}$ and $\mathcal{O}_{\mathcal{S}}$, $\mathcal{O}(\text{polylog}(T_{\max}))$ auxiliary qubits and a number of extra gates that satisfies*

$$\mathcal{O}\left(\max_{j\in[n]} \frac{T_j}{S_j}\text{polylog}\left(T_{\max}, \frac{1}{\delta}\right)\right).$$

The main idea of the proof is to use the characterization of the kernel of $A$ given in Lemma 61, and to map this space isometrically to another space around which we can reflect more easily. The formal proof of Lemma 69 is given at the end of this section.

First, we implement a subroutine that we will use throughout this section.

**Definition 70.** Let $\mathcal{A}$ be a clean quantum algorithm with time complexity $T$ and query complexity $S$ and let $P_{\mathcal{A}} = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$ be its span program. For all $\alpha \in [0, 1]$ and $t \in [T-1]_0$, the non-trivial action of the maps $\mathcal{S}_{t,\alpha}$ is described as follows. For $|\psi\rangle \in \mathbb{C}^{[n]\times\mathcal{W}}$, if $t+1 \in \mathcal{S}$,

$$\mathcal{S}_{t,\alpha} : \begin{cases} |t\rangle|-\rangle|\psi\rangle & \mapsto \alpha|t\rangle|-\rangle|\psi\rangle + \sqrt{1-\alpha^2}|t+1\rangle|0\rangle|\psi\rangle \\ |t+1\rangle|0\rangle|\psi\rangle & \mapsto -\sqrt{1-\alpha^2}|t\rangle|-\rangle|\psi\rangle + \alpha|t+1\rangle|0\rangle|\psi\rangle \end{cases} \tag{4.17}$$

If $t+2 \in \mathcal{S}$,

$$\mathcal{S}_{t,\alpha} : \begin{cases} |t\rangle|0\rangle|\psi\rangle & \mapsto \alpha|t\rangle|0\rangle|\psi\rangle + \sqrt{1-\alpha^2}|t+1\rangle|+\rangle U_{t+1}|\psi\rangle \\ |t+1\rangle|+\rangle|\psi\rangle & \mapsto -\sqrt{1-\alpha^2}|t\rangle|0\rangle U_{t+1}^{\dagger}|\psi\rangle + \alpha|t+1\rangle|+\rangle|\psi\rangle \end{cases} \tag{4.18}$$

Otherwise,

$$\mathcal{S}_{t,\alpha} : \begin{cases} |t\rangle|0\rangle|\psi\rangle & \mapsto \alpha|t\rangle|0\rangle|\psi\rangle + \sqrt{1-\alpha^2}|t+1\rangle|0\rangle U_{t+1}|\psi\rangle \\ |t+1\rangle|0\rangle|\psi\rangle & \mapsto -\sqrt{1-\alpha^2}|t\rangle|0\rangle U_{t+1}^{\dagger}|\psi\rangle + \alpha|t+1\rangle|0\rangle|\psi\rangle. \end{cases} \tag{4.19}$$

In all three cases we have $\mathcal{S}_{t,\alpha} : |t'\rangle|\phi\rangle \mapsto |t'\rangle|\phi\rangle$ for $t' \notin \{t, t+1\}$. We refer to these maps as *splitting maps*.

Note that for all choices of $t$ and $\alpha$, $\mathcal{S}_{t,\alpha}$ leaves $\mathcal{H}_x$ invariant. We leave this to the reader to check. In the lemma below, we elaborate on how we can implement this splitting map efficiently.

**Lemma 71.** *Let $\mathcal{A}$ be a clean quantum algorithm with time complexity $T$ and query complexity $S$, and let $P_{\mathcal{A}} = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$ be its span program. Let $t \in [T-1]_0$ and $\alpha \in [0,1]$. We can implement $\mathcal{S}_{t,a}$ with two controlled calls to (the inverse of) $\mathcal{O}_{\mathcal{A}}$ and $\mathcal{O}(\mathrm{polylog}(T))$ additional gates.*

*Furthermore, if we have a binary description of $t$ and $\alpha$ in auxiliary registers, where the description of $\alpha$ is $\delta$-precise, we can implement $\mathcal{S}_{t,\alpha}$ up to error $\delta > 0$ with two controlled calls to (the inverse of) $\mathcal{O}_{\mathcal{A}}$ and $\mathcal{O}(\mathrm{polylog}(T, 1/\delta))$ additional gates.*

*Proof.* First of all, we check which of the three cases in Definition 70 applies. This we can do with one call to $\mathcal{O}_{\mathcal{S}}$ and polylogarithmically many extra gates in $T$. Each of these three cases we treat separately and consecutively. We only give the explicit description of the last case here, as the others come down to the same circuit with some minor adjustments.

We implement the bottom mapping in Def. 70 in three steps.

1. First, controlled on the first register being in time $t+1$, we call the inverse of $\mathcal{O}_{\mathcal{A}}$. This will map $|t\rangle|0\rangle|\psi\rangle$ to itself, and it will map $|t+1\rangle|0\rangle|\psi\rangle$ to $|t+1\rangle|0\rangle U_{t+1}^{\dagger}|\psi\rangle$. This takes 1 call to $\mathcal{O}_{\mathcal{A}}$, and $\mathcal{O}(\mathrm{polylog}(T))$ other gates.

2. Next, we apply the following mapping to the first register:

   $$|t\rangle \mapsto \alpha|t\rangle + \sqrt{1-\alpha^2}|t+1\rangle \qquad \text{and} \qquad |t+1\rangle \mapsto -\sqrt{1-\alpha^2}|t\rangle + \alpha|t+1\rangle$$

   Since this is a two-level rotation, we can implement it with $\mathcal{O}(\mathrm{polylog}(T))$ single qubit gates and CNOTs.

3. Finally, we apply $\mathcal{O}_{\mathcal{A}}$, controlled on the first register being in time $t+1$. Just as in step 1, this takes 1 call to $\mathcal{O}_{\mathcal{A}}$ and $\mathcal{O}(\mathrm{polylog}(T))$ additional gates.

One can easily check that this implements the third mapping in Def. 70.

Furthermore, if we have a binary description of $t$ and $\alpha$ stored in an extra register, we can implement the desired mapping in a similar number of gates. While we cannot hardcode $t$ in steps 1 and 3, we can control on its value. Similarly, in step 2, as $\alpha$ is not hardcoded, we have to substitute the rotation with $\mathcal{O}(\text{polylog}(1/\delta))$ rotations controlled on the qubits storing $\alpha$. All of the necessary computations are efficiently implementable classically, and hence only add (additive) polylogarithmic overhead in the error parameter to the time complexity. □

Next, we define what we call the left and right block oracles, $\mathcal{O}_L$ and $\mathcal{O}_R$. These will be necessary further on because our construction requires us to be able to tell, given a state with time label $t$, what are the closest query time-steps to its left and to its right, and, if $t+1$ is a query, whether we should count the state as pertaining to $\mathcal{H}_{x,\ell}$ or $\mathcal{H}_{x,\ell+1}$. Remember that all the way back at the beginning of Section 4.4.1, we denoted the blocks of contiguous non-query time steps by $\mathcal{B}_\ell = \{q_{\ell-1}-1, \ldots, q_\ell - 1\}$. Query time steps act as the separators between contiguous blocks. When $t$ is the index of a non-query, it belongs to a unique query block, and so the left-endpoint of its query block, $q_{\ell-1}$, is uniquely defined, as is the right-endpoint, $q_\ell$. Thus, we can define operations $\mathcal{O}_L$ and $\mathcal{O}_R$ that, for any such $t$, compute these values, or, rather, for technical reasons, given $|t\rangle$ such that $t+1 \in \mathcal{B}_\ell$, $\mathcal{O}_L$ and $\mathcal{O}_R$ return $q_{\ell-1}-1$ and $q_\ell - 1$ respectively.

When $t+1$ is the index of a query, there is ambiguity, because it is contiguous to two blocks – it is the left-endpoint of one, and the right-endpoint of another. We use an auxiliary qubit to resolve this ambiguity: for a state $|t\rangle|+\rangle$, we interpret $t+1 = q_\ell$ as the right-endpoint of a block, so $\mathcal{O}_L$ and $\mathcal{O}_R$ return $q_{\ell-1}-1$ and $q_\ell - 1$ respectively; and for a state $|t\rangle|-\rangle$, we interpret $t+1 = q_\ell$ as the left-endpoint of a block, so $\mathcal{O}_L$ and $\mathcal{O}_R$ return $q_\ell - 1$ and $q_{\ell+1} - 1$ respectively. In other words, we reinterpret blocks as starting with a query and finishing immediately before the next one. Notice how this is consistent with the form of the vectors in $\mathcal{H}_x$ and $\ker(A)$. The precise actions of $\mathcal{O}_L$ and $\mathcal{O}_R$ are defined as follows.

**Definition 72.** Let $\mathcal{A}$ be a clean quantum algorithm as in Definition 6,and let $P_\mathcal{A} = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$ be its span program. We define the left and right block oracles as unitaries on $\mathcal{H} \otimes \mathbb{C}^{\{-1,\ldots,T\}}$, acting as

$$\mathcal{O}_L : \begin{cases} |t\rangle|0\rangle|0\rangle & \mapsto & |t\rangle|0\rangle|q_{\ell-1}-1\rangle, & \text{if } t+1 \in \mathcal{B}_\ell, \\ |t\rangle|+\rangle|0\rangle & \mapsto & |t\rangle|+\rangle|q_{\ell-1}-1\rangle, & \text{if } t+1 = q_\ell, \\ |t\rangle|-\rangle|0\rangle & \mapsto & |t\rangle|-\rangle|q_{\ell-1}-1\rangle, & \text{if } t+1 = q_{\ell-1}, \end{cases}$$

and

$$\mathcal{O}_R : \begin{cases} |t\rangle|0\rangle|0\rangle & \mapsto & |t\rangle|0\rangle|q_\ell - 1\rangle, & \text{if } t+1 \in \mathcal{B}_\ell, \\ |t\rangle|+\rangle|0\rangle & \mapsto & |t\rangle|+\rangle|q_\ell - 1\rangle, & \text{if } t+1 = q_\ell, \\ |t\rangle|-\rangle|0\rangle & \mapsto & |t\rangle|-\rangle|q_\ell - 1\rangle, & \text{if } t+1 = q_{\ell-1}. \end{cases}$$

Now, we show how to implement these block oracles efficiently.

**Lemma 73.** *Let $\mathcal{A}$ be a clean quantum algorithm with query complexity $S$ and time complexity $T$, and let $P_{\mathcal{A}} = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$ be its span program. Then we can implement $\mathcal{O}_L$ and $\mathcal{O}_R$ with $\mathcal{O}(T/S)$ queries to $\mathcal{O}_{\mathcal{S}}$, $\mathcal{O}(\text{polylog}(T))$ ancillary qubits and a number of additional gates that scales as*

$$\mathcal{O}\left(\frac{T}{S}\text{polylog}(T)\right).$$

*Similarly, suppose $\{\mathcal{A}^{(j)}\}_{j=1}^n$ is a set of clean quantum algorithms. Let $j \in [n]$ and let $S_j$ and $T_j$ be the query and time complexity of $\mathcal{A}^{(j)}$, respectively. Similarly, let $\mathcal{O}_L^{(j)}$ and $\mathcal{O}_R^{(j)}$ be the left and right block oracles of $\mathcal{A}^{(j)}$, respectively. We can provide concurrent access to $\{\mathcal{O}_L^{(j)}\}_{j=1}^n$ and $\{\mathcal{O}_R^{(j)}\}_{j=1}^n$ with $\mathcal{O}(\max_{j\in[n]} T_j/S_j)$ queries to $\mathcal{O}_{\mathcal{S}}$, $\mathcal{O}(\text{polylog}(T_{\max}))$ ancillary qubits and a number of additional gates that scales as*

$$\mathcal{O}\left(\max_{j\in[n]} \frac{T_j}{S_j}\text{polylog}(T_{\max})\right).$$

*Proof.* We first focus on the case where we have just one algorithm and leave the case where we have multiple algorithms for the final paragraph. We only show how to implement $\mathcal{O}_L$, as the implementation of $\mathcal{O}_R$ is similar. First of all, we check if $t+1 \in \mathcal{S}$ using one call to $\mathcal{O}_{\mathcal{S}}$ and $\mathcal{O}(\text{polylog}(T))$ other gates and store the result in an auxiliary qubit. If this flag qubit is $|1\rangle$, we apply a Hadamard to the second register. If the second register is now 1, we copy the time $t$ to the last register. Observe that if the input was $|t\rangle|-\rangle|0\rangle$, where $t+1 = q_{\ell-1}$ then we are done (up to reapplying the Hadamard to the second register and uncomputing the flag qubit). The only interesting case now is when the second bit is 0, so we apply all the following operations controlled on this bit being 0.

In this last case, what we would like to do is write in a new register the index of the last query before our timestep $t$. For that purpose we initialize a new counter register having $\lceil \log(3T/S) \rceil + 1$ qubits, in the state $|0\rangle$, and iteratively decrement the time register until we reach a time step

that is one less than a query time step, and after that the counter register is incremented. This means that after these iterations, we have the correct query time step stored in the time register, while the counter will contain a function of $t$, $q_{\ell-1}$, and $\lfloor 3T/S \rfloor$. This task can be done by repeating the following operation $\lfloor 3T/S \rfloor$ times. First, we check whether the time register is one less than a query time step and if it is we increment the counter register. This can be done using 2 queries to $\mathcal{O}_{\mathcal{S}}$ and a number of extra gates that is polylogarithmic in $T$. After that, we decrement the time register controlled on the counter being in the $|0\rangle$ state. This also takes a number of gates that is polylogarithmic in $T$. We can now copy the time into the last register, and then uncompute all of these iterations, returning the time register to the state $|t\rangle$ and the counter to the state $|0\rangle$.

At last, we undo the computations we did in the beginning, i.e., we apply the controlled Hadamard again and reset the flag that indicated whether $t+1 \in \mathcal{S}$ using one more query to $\mathcal{O}_{\mathcal{S}}$ and $\mathcal{O}(\mathrm{polylog}(T))$ extra gates. We easily check that the total cost of this construction matches the claim in the statement of the lemma.

Finally, in order to provide concurrent access to $\{\mathcal{O}_L^{(j)}\}_{j=1}^n$, we can simply run the loop in the second paragraph for $\max_{j\in[n]} \lfloor 3T_j/S_j \rfloor$ iterations. The size of the time register now has to be $T_{\max}$ and so the arithmetic operations on this register take a number of gates that is polylogarithmic in $T_{\max}$. This completes the proof. □

Next, we define a mapping that generates the vectors of the kernel of $A$.

**Definition 74.** Let $\mathcal{A}$ be a clean quantum algorithm with time complexity $T$ and query complexity $S$, and let $P_{\mathcal{A}} = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$ be its span program. Define $\mathcal{C}$ as a unitary on $\mathcal{H}$, which, for all $\ell \in \{2, \ldots, S+1\}$ and $|\psi\rangle \in \mathbb{C}^{[n]\times\mathcal{W}}$, acts as

$$\mathcal{C} : |q_{\ell-1} - 1\rangle|-\rangle|\psi\rangle \mapsto \frac{\Phi_\ell|\psi\rangle}{\|\Phi_\ell|\psi\rangle\|},$$

and otherwise, $\mathcal{C}$ acts arbitrarily but leaves $\mathcal{H}_x$ invariant.

These maps, or rather, their inverses, are the key to the reflection around $\ker(A)$. Indeed, $\mathcal{C}^\dagger$ maps every element in $\ker(A)$ onto a state of the form $|q_{\ell-1} - 1\rangle|-\rangle|\psi\rangle$, so if we can implement this map and reflect around these simpler states, we can reflect around $\ker(A)$. Let us now see how to implement $\mathcal{C}$.

**Lemma 75.** *Let $\mathcal{A}$ be a clean quantum algorithm with time complexity $T$ and query complexity $S$, and let $P_{\mathcal{A}} = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$ be its span program. We can implement a mapping $\mathcal{C}$ that satisfies the conditions in Definition 74, up to error $\delta > 0$ in operator norm with $\mathcal{O}(T/S)$ queries to $\mathcal{O}_{\mathcal{A}}$ and $\mathcal{O}_{\mathcal{S}}$, $\mathcal{O}(\mathrm{polylog}(T))$ ancillary qubits, and with a number of additional gates that scales as*

$$\mathcal{O}\left(\frac{T}{S}\mathrm{polylog}\left(T, \frac{1}{\delta}\right)\right).$$

*Similarly, let $\{\mathcal{A}^{(j)}\}_{j=1}^{n}$ be a set of clean quantum algorithms. For all $j \in [n]$, let $S_j$ and $T_j$ denote the query and time complexity of $\mathcal{A}^{(j)}$, respectively. Let $\mathcal{C}^{(j)}$ be the routine defined in Def. 74 for $\mathcal{A}^{(j)}$. We can provide concurrent access to $\{\mathcal{C}^{(j)}\}_{j=1}^{n}$ up to error $\delta > 0$ in operator norm with $\mathcal{O}(\max_{j\in[n]} T_j/S_j)$ queries to $\mathcal{O}_{\mathcal{A}}$ and $\mathcal{O}_{\mathcal{S}}$, with $\mathcal{O}(\mathrm{polylog}(T_{\max}))$ ancillary qubits, and with a number of additional gates that scales as*

$$\mathcal{O}\left(\max_{j\in[n]} \frac{T_j}{S_j}\mathrm{polylog}\left(T_{\max}, \frac{1}{\delta}\right)\right).$$

*Proof.* While the behavior of $\mathcal{C}$ is only fully specified on states with $t$ such that $t + 1 \in \mathcal{S}$ in the first register, more generally, we must ensure that $\mathcal{C}$ leaves $\mathcal{H}_x$ invariant, which leads to a more involved construction. We first consider the case where we have just one algorithm $\mathcal{A}$, and leave the case where we have multiple algorithms for the final paragraph of this proof.

First of all, we call $\mathcal{O}_L$ and $\mathcal{O}_R$ and store the results in some auxiliary registers. According to Lemma 73, this takes $\mathcal{O}(T/S)$ calls to $\mathcal{O}_{\mathcal{S}}$ and $\mathcal{O}(T/S \cdot \mathrm{polylog}(T))$ additional gates.

Next, we distinguish between three cases. First, if the application of $\mathcal{O}_L$ amounts to $|-1\rangle$ in the last register, then necessarily $t$ belongs to the block before the first query, in which case we simply do nothing, i.e., we act as the identity. Second, if the result of $\mathcal{O}_R$ is $T$, then we started out in a state in which the time step $t$ was higher than the last query, which requires separate treatment. The third case is when neither of these happened. These cases can be distinguished with $\mathcal{O}(\mathrm{polylog}(T))$ gates, and they can be handled separately and consecutively. We only explain how we handle the final case, as the first one is trivial and the other is similar to the third.

Hence, we assume that the auxiliary registers are in the states $|q_{\ell-1} - 1\rangle$ and $|q_\ell - 1\rangle$ for some $\ell \in \{2, \ldots, S\}$. Now, we repeat the following procedure $\lfloor 3T/S \rceil$ times. In the $i$th iteration, where $i = 0, 1, \ldots, \lfloor 3T/S \rfloor$, we perform the following steps.

1. First, we initialize an auxiliary qubit and set it to $|1\rangle$ if and only if $q_{\ell-1} + i < q_\ell$. This takes time $\mathcal{O}(\text{polylog}(T))$. Steps 2 – 4 we do controlled on the auxiliary qubit being $|1\rangle$.
2. From the auxiliary registers, we calculate
$$\alpha_i = \begin{cases} \frac{1}{\sqrt{2}\cdot\sqrt{1+\frac{|\mathcal{B}_\ell|}{M^2}}}, & \text{if } i = 0, \\ \frac{1}{M\cdot\sqrt{\frac{1}{2}+\frac{|\mathcal{B}_\ell|-(i-1)}{M^2}}}, & \text{otherwise,} \end{cases}$$
and store it in a binary representation in another auxiliary register. This calculation can be done up to precision $\Theta(\delta S/T)$ in a number of gates polylogarithmic in $T$ and $1/\delta$ using standard classical methods.
3. Next, we apply the splitting map $\mathcal{S}_{q_{\ell-1}+i-1,\alpha_i}$, where $q_{\ell-1} + i - 1$ and $\alpha_i$ are stored in separate registers, with $\alpha_i$ up to error $\Theta(\delta S/T)$, which by Lemma 71 incurs 2 controlled calls to (the inverse of) $\mathcal{O}_\mathcal{A}$ and a number of extra gates that is polylogarithmic in $T$ and $1/\delta$.
4. We uncompute the parameter $\alpha_i$ from step 2.
5. We uncompute the check that $q_{\ell-1} + i < q_\ell$.

These steps have the effect of applying
$$\mathcal{S}_{q_\ell-2,\alpha_{|\mathcal{B}_\ell|}}\mathcal{S}_{q_\ell-3,\alpha_{|\mathcal{B}_\ell|-1}}\cdots\mathcal{S}_{q_{\ell-1},\alpha_1}\mathcal{S}_{q_{\ell-1}-1,\alpha_0},$$
where each factor is implemented up to error $\Theta(S\delta/T)$. As there are at most $\mathcal{O}(T/S)$ factors, the total error is at most $\delta$.

First of all, remember that by construction, $\mathcal{S}_{t,\alpha}$ leaves $\mathcal{H}_x$ and invariant. Moreover, observe that the only values of $t$ for which we execute $\mathcal{S}_{t,\alpha}$ are the values $\{q_{\ell-1} - 1, \ldots, q_\ell - 2\}$. By Definition 70, for any $\alpha$:

- $\mathcal{S}_{q_{\ell-1}-1,\alpha}$ only acts non-trivially on $\text{span}\{|q_{\ell-1}-1\rangle|-\rangle, |q_{\ell-1}\rangle|0\rangle\}\otimes\mathbb{C}^{[n]\times\mathcal{W}}$, which it also leaves invariant (since $q_{\ell-1} \in \mathcal{S}$);
- $\mathcal{S}_{q_\ell-2,\alpha}$ only acts non-trivially on $\text{span}\{|q_\ell-2\rangle|0\rangle, |q_\ell-1\rangle|+\rangle\}\otimes\mathbb{C}^{[n]\times\mathcal{W}}$, which it also leaves invariant (since $q_\ell \in \mathcal{S}$);
- for all $t \in \{q_{\ell-1}, \ldots, q_\ell-3\}$, $\mathcal{S}_{t,\alpha}$ only acts non-trivially on $\text{span}\{|t\rangle|0\rangle, |t+1\rangle|0\rangle\}\otimes\mathbb{C}^{[n]\times\mathcal{W}}$, which it also leaves invariant (since $t+1, t+2 \notin \mathcal{S}$).

This means that we only act non-trivially on the vectors $|q_{\ell-1} - 1\rangle|-\rangle|\psi\rangle$, $|q_\ell - 1\rangle|+\rangle|\psi\rangle$ and $|t\rangle|0\rangle|\psi\rangle$ where $t+1 \in \mathcal{B}_\ell$ and $|\psi\rangle \in \mathbb{C}^{[n]\times\mathcal{W}}$, and we leave invariant the space
$$(\text{span}\{|q_{\ell-1} - 1\rangle|-\rangle\} \oplus \text{span}\{|t\rangle|0\rangle : t+1 \in \mathcal{B}_\ell\} \oplus \text{span}\{|q_\ell - 1\rangle|+\rangle\})\otimes\mathbb{C}^{[n]\times\mathcal{W}}. \tag{4.20}$$

In particular, we leave invariant the subspaces $\mathcal{H}_{x,\ell}^{(b)}$ for $b = 0, 1$. This implies that the time register always contains a value $t$ such that $t + 1$ is within the query block bounded by $q_{\ell-1}$ from the left, and $q_\ell$ from the right, or a value. As we shall see shortly, we have generated a superposition of vectors with time step in the $\mathcal{B}_\ell$ plus a vector with time step $q_{\ell-1}$ that has a $|-\rangle$ in the second register and a vector in the time step $q_\ell$ having $|+\rangle$ in the second register. The auxiliary qubits generated by $\mathcal{O}_L$ and $\mathcal{O}_R$ containing the boundaries of the current block are no longer necessary so we uncompute them by simply calling their inverses of $\mathcal{O}_L$ and $\mathcal{O}_R$.

Clearly this mapping leaves $\mathcal{H}_x$ invariant since $\mathcal{S}_{t,\alpha}$ leaves $\mathcal{H}_x$ invariant, and all other operations do not matter as they are uncomputed.

Moreover, we claim that this circuit implements a mapping $\mathcal{C}$ that satisfies the conditions from Definition 74. As we are considering the case that the block $\mathcal{B}_\ell$ is neither the first nor the last block, suppose that we start with the state $|q_\ell - 1\rangle|-\rangle|\psi\rangle$, for some $|\psi\rangle \in \mathbb{C}^{[n]\times\mathcal{W}}$. Now, in the first iteration $(i = 0)$ we apply $\mathcal{S}_{q_{\ell-1}-1,\alpha_0}$, to arrive at the state

$$\frac{1}{\sqrt{1+\frac{|\mathcal{B}_\ell|}{M^2}}}\left[\frac{1}{\sqrt{2}}|q_\ell - 1\rangle|-\rangle|\psi\rangle + \sqrt{\frac{1}{2}+\frac{|\mathcal{B}_\ell|}{M^2}}|q_\ell\rangle|0\rangle|\psi\rangle\right].$$

We easily check by induction that after the $i$th iteration with $1 \leq i < q_\ell - q_{\ell-1}$, we are in the state

$$\frac{1}{\sqrt{1+\frac{|\mathcal{B}_\ell|}{M^2}}}\left[\begin{array}{l}\frac{1}{\sqrt{2}}|q_{\ell-1} - 1\rangle|-\rangle|\psi\rangle + \frac{1}{M}\sum_{t=q_{\ell-1}}^{q_{\ell-1}+i-1}|t\rangle|0\rangle U_{t;q_{\ell-1}+1}|\psi\rangle \\ +\sqrt{\frac{1}{2}+\frac{|\mathcal{B}_\ell|-i}{M^2}}|q_{\ell-1}+i\rangle|0\rangle U_{q_{\ell-1}+i;q_{\ell-1}+1}|\psi\rangle\end{array}\right],$$

which implies that after the iteration where $i = |\mathcal{B}_\ell|$, we are in the desired state. Whenever $i \geq q_\ell - q_{\ell-1}$, we don't do anything due to the condition that is checked in step 1. Hence, this circuit indeed implements a mapping $\mathcal{C}$ that satisfies the conditions outlined in Definition 74.

We observe that there are $\mathcal{O}(T/S)$ iterations, each of which uses $\mathcal{O}(1)$ calls to $\mathcal{O}_\mathcal{A}$ and $\mathcal{O}(\text{polylog}(T, 1/\delta))$ extra gates. In addition we do $\mathcal{O}(T/S)$ calls to $\mathcal{O}_\mathcal{S}$ and $\mathcal{O}(T/S \cdot \text{polylog}(T))$ extra gates when we call $\mathcal{O}_L$ and $\mathcal{O}_R$ and their inverses.

In order to implement concurrent access to $\{\mathcal{C}^{(j)}\}_{j=1}^n$, we can run the loop a total of $\max_{j\in[n]}\lfloor 3T_j/S_j\rfloor$ iterations. The time register now has to be of size

$T_{\max} = \max_{j\in[n]} T_j$, and hence the arithmetic operations on this register now take $\mathcal{O}(\mathrm{polylog}(T_{\max}))$ gates. We can now calculate the coefficients $\alpha_i$ with precision $\mathcal{O}(\delta \min_{j\in[n]} S_j/T_j)$. Now all the maps $\mathcal{C}^{(j)}$ are implemented up to precision $\delta$, which implies that the concurrent access is also implemented up to precision $\delta$. This completes the proof. □

We have finally all the coupons necessary to prove Lemma 69, let us now redeem them.

*Proof of Lemma 69.* We first focus on the case where we have just one algorithm $\mathcal{A}$. Using the characterization of the kernel of $A$ in Lemma 61, we see that

$$\mathcal{C} : \underbrace{\mathrm{span}\{|q_{\ell-1} - 1\rangle|-\rangle : \ell \in [S+1] \setminus \{1\}\}}_{X} \otimes \mathbb{C}^{[n]\times\mathcal{W}} \mapsto \ker(A)$$

isometrically. Hence, we obtain that

$$2\Pi_{\ker(A)} - I = \mathcal{C}\left[(2\Pi_X - I) \otimes I\right]\mathcal{C}^\dagger.$$

As we can implement the reflection around $X$ using $\mathcal{O}(\log(T))$ extra gates and a single controlled query to $\mathcal{O}_{\mathcal{S}}$, the cost of reflecting around $\ker(A)$ essentially becomes twice the cost of implementing $\mathcal{C}$, which is given in Lemma 75.

If we have multiple algorithms $\mathcal{A}^{(j)}$, we can use the exact same idea, but now we should use concurrent access to the $\mathcal{C}^{(j)}$'s and a concurrent reflection around the spaces $X^{(j)}$'s. The cost of implementing concurrent access to $\{\mathcal{C}^{(j)}\}_{j=1}^n$ is analyzed in Lemma 75, and the concurrent reflection around the $X^{(j)}$'s can be implemented with $\mathcal{O}(\mathrm{polylog}(T_{\max}))$ gates and one controlled call to $\mathcal{O}_{\mathcal{S}}$. This completes the proof. □

As a final remark, we would like to point out that this is not the only possible construction of the reflection around the kernel of $A$. One could alternatively employ a more general method of constructing a block-encoding of $A$ and using phase estimation to separate the vectors in the kernel of $A$ from those that are orthogonal to it. Implementing this construction carefully yields the same time and query complexity, but requires a spectral analysis of $A$, which is possible but turns out to be quite involved.

### 4.5.5 Construction of $|w_0\rangle$

The goal of this section is to prove Lemma 76, i.e., we provide an implementation of the circuit that constructs the minimal positive witness that is analytically calculated in Lemma 62. Additionally, we also elaborate on how one would do this concurrently, because we need this is in the proof of Theorem 78.

**Lemma 76.** *Let $\mathcal{A}$ be a clean quantum query algorithm with query complexity $S$, time complexity $T$, and error probability $\varepsilon$. Let $P_{\mathcal{A}} = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$ be the span program for this algorithm, as in Definition 57. We can implement a unitary $\mathcal{C}_{|w_0\rangle}$ that maps the all-zeroes state $|0\rangle|0\rangle|\Psi_0\rangle$ to $|w_0\rangle / \||w_0\rangle\|$ up to error $\delta > 0$ in the operator norm with $\mathcal{O}(T/S)$ calls to $\mathcal{O}_{\mathcal{A}}$ and $\mathcal{O}_{\mathcal{S}}$, $\mathcal{O}(1)$ auxiliary qubits and a number of gates that satisfies*

$$\mathcal{O}\left(\frac{T}{S}\text{polylog}\left(T, \frac{1}{\delta}\right)\right).$$

*Similarly, let $\{\mathcal{A}^{(j)}\}_{j=1}^n$ be a set of clean quantum query algorithms. For all $j \in [n]$, let $S_j$ and $T_j$ denote the query and time complexity of $\mathcal{A}^{(j)}$, respectively, and let $\varepsilon_j$ be the error probability. Let $P^{(j)}$ be the span program of $\mathcal{A}^{(j)}$, and let $|w_0^{(j)}\rangle$ be the minimal positive witness of $P^{(j)}$. Then we can implement concurrent access $\mathcal{C}$ to $\{\mathcal{C}_{|w_0^{(j)}\rangle}\}_{j=1}^n$ up to error $\delta > 0$ in operator norm with $\mathcal{O}(T/S)$ calls to $\mathcal{O}_{\mathcal{A}}$ and $\mathcal{O}_{\mathcal{S}}$, $\mathcal{O}(1)$ auxiliary qubits and a number of gates that satisfies*

$$\mathcal{O}\left(\max_{j\in[n]} \frac{T_j}{S_j}\text{polylog}\left(T_{\max}, \frac{1}{\delta}\right)\right).$$

We first name the three parts that form $|w_0\rangle$. From the form of $|w_0\rangle$ in Lemma 62 we have

$$\frac{|w_0\rangle}{\||w_0\rangle\|} = \frac{|\psi\rangle}{\sqrt{N}} + \frac{|\chi\rangle}{\sqrt{N}} + \frac{|\phi\rangle}{\sqrt{N}},$$

where

$$
\begin{aligned}
|\psi\rangle &= \frac{1}{M}\sum_{t=0}^{q_1-2}|t\rangle|0\rangle U_{t;1}|\Psi_0\rangle + |q_1-1\rangle\left(\frac{1}{2}|0\rangle+\frac{1}{2}|1\rangle\right)U_{q_1-1;1}|\Psi_0\rangle,\\
|\chi\rangle &= \frac{1}{Ca^2+1}\left[|q_S-1\rangle\left(\frac{1}{2}|0\rangle-\frac{1}{2}|1\rangle\right)U^\dagger_{q_S+1;T}|\Psi_T\rangle\right.\\
&\quad\left.+\frac{1}{M}\sum_{t=q_S+1}^{T-1}|t\rangle|0\rangle U^\dagger_{t+1;T}|\Psi_T\rangle\right],\\
|\phi\rangle &= -\frac{Ca}{Ca^2+1}|T\rangle|0\rangle|\Psi_T\rangle,
\end{aligned}
$$

and $C = \frac{T-q_S}{M^2}+\frac{1}{2}$ and $a$ and $M$ are defined in Eq. (4.6). We can easily calculate the norms of the respective vectors, which results in

$$
\||\psi\rangle\|^2 = \frac{q_1-1}{M^2}+\frac{1}{2}, \qquad \||\chi\rangle\|^2 = \frac{C}{(Ca^2+1)^2}, \text{ and } \||\phi\rangle\|^2 = \frac{C^2a^2}{(Ca^2+1)^2}.
$$

Of these three states, observe that the first two have a structure similar to the vectors in $\ker(A)$, while the third one is very simple. By assumption on the state $|\Psi_T\rangle$, the state $|\phi\rangle/\||\phi\rangle\|$ can be generated in $\mathcal{O}(1)$ gates. For the generation of the other two we will reuse the techniques in the previous section for mapping simple states to states in $\ker(A)$. This is the focus of the following lemma.

**Lemma 77.** *We can implement routines $\mathcal{C}_{|\psi\rangle}$ and $\mathcal{C}_{|\chi\rangle}$ that map $|0\rangle|0\rangle|\Psi_0\rangle$ to $|\psi\rangle/\||\psi\rangle\|$ and $|T-1\rangle|0\rangle U_T^\dagger|\Psi_T\rangle$ to $|\chi\rangle/\||\chi\rangle\|$ respectively up to error $\delta>0$ in operator norm, with $\mathcal{O}(T/S)$ calls to $\mathcal{O}_\mathcal{A}$, $\mathcal{O}(1)$ auxiliary qubits and a number of gates that satisfies*

$$
\mathcal{O}\left(\frac{T}{S}\text{polylog}\left(T,\frac{1}{\delta}\right)\right).
$$

*Moreover, both circuits leave $\mathcal{H}_x$ invariant, and their inverses leave all the states orthogonal to $|\psi\rangle$ (resp. $|\chi\rangle$) invariant.*

*Similarly, we can provide concurrent access to $\{\mathcal{C}^{(j)}_{|\psi\rangle}\}_{j=1}^n$ and $\{\mathcal{C}^{(j)}_{|\chi\rangle}\}_{j=1}^n$ up to precision $\delta>0$ with a number of calls to $\mathcal{O}_\mathcal{A}$ and $\mathcal{O}_\mathcal{S}$ of $\mathcal{O}(\max_{j\in[n]}T_j/S_j)$, $\mathcal{O}(1)$ auxiliary qubits and a number of extra gates that scales as*

$$
\mathcal{O}\left(\max_{j\in[n]}\frac{T_j}{S_j}\text{polylog}\left(T_{\max},\frac{1}{\delta}\right)\right).
$$

*Proof.* Note that $|\psi\rangle$ and $|\chi\rangle$ are very similar to the states in the kernel of $A$ that span the image of the $\Phi_\ell$'s. Therefore, $\mathcal{C}_{|\psi\rangle}$ and $\mathcal{C}_{|\chi\rangle}$ are very similar to the circuit $\mathcal{C}$ defined in Definition 74. We can use the exact same techniques as we used in implementing $\mathcal{C}$ in Lemma 75 to implement $\mathcal{C}_{|\psi\rangle}$ and $\mathcal{C}_{|\chi\rangle}$. The cost of implementing these routines carries over from the proof of Lemma 75. This completes the proof. □

*Proof of Lemma 76.* We first restrict ourselves to the case where we just have one algorithm, $\mathcal{A}$, and we postpone the treatment of the case where we have a set of algorithms to the final paragraph of this proof.

First of all, for $\alpha = \frac{\||\psi\rangle\|}{\||w_0\rangle\|}$, we implement a circuit $\mathcal{C}_1$ whose action is

$$\mathcal{C}_1 : \begin{cases} |0\rangle|0\rangle|\Psi_0\rangle & \mapsto \quad \alpha|0\rangle|0\rangle|\Psi_0\rangle + \sqrt{1-\alpha^2}|T\rangle|0\rangle|\Psi_T\rangle \\ |T\rangle|0\rangle|\Psi_T\rangle & \mapsto \quad -\sqrt{1-\alpha^2}|0\rangle|0\rangle|\Psi_0\rangle + \alpha|T\rangle|0\rangle|\Psi_T\rangle \end{cases}$$

This circuit can be implemented in a similar way as we implemented the splitting map in Lemma 71. Conditioned on the first register being in state $|T\rangle$, one first applies the map $|\Psi_T\rangle = |\psi_0, 1\rangle \mapsto |\Psi_0\rangle = |\psi_0, 0\rangle$ to the last register, which amounts to applying a controlled $X$ operation targeted to the answer register. Then, one applies a rotation on a two-dimensional subspace of the state space of the first register spanned by $\text{span}\{|0\rangle, |T\rangle\}$, which can be implemented with $\mathcal{O}(\text{polylog}(T))$ gates as it is a register on $\log(T)$ qubits. Finally, one applies the mapping $|\Psi_0\rangle \mapsto |\Psi_T\rangle$, again controlled on the first register being in state $|T\rangle$. Counting the auxiliary qubits and gates reveals that we can do this with $\mathcal{O}(1)$ auxiliary qubits and $\mathcal{O}(\text{polylog}(T))$ gates.

Next, one applies the mapping $\mathcal{S}^\dagger_{T-1,\beta}$, with $\beta = \||\chi\rangle\| \,/\, \||\chi\rangle + |\phi\rangle\|$. The combined mapping now acts as

$$\mathcal{S}^\dagger_{T-1,\beta}\mathcal{C}_1 : |0\rangle|0\rangle|\Psi_0\rangle \mapsto \frac{\||\psi\rangle\|}{\sqrt{N}}|0\rangle|0\rangle|\Psi_0\rangle + \frac{\||\chi\rangle\|}{\sqrt{N}}|T-1\rangle|0\rangle U_T^\dagger|\Psi_T\rangle \tag{4.21}$$

$$+ \frac{\||\phi\rangle\|}{\sqrt{N}}|T\rangle|0\rangle|\Psi_T\rangle. \tag{4.22}$$

Thus, all that is left is mapping the first term to $|\psi\rangle$ and the second to $|\chi\rangle$ using the circuits that we already have for them, meaning that

$$\mathcal{C}_{|w_0\rangle} = \mathcal{C}_{|\psi\rangle}\mathcal{C}_{|\chi\rangle}\mathcal{S}^\dagger_{T_1,\beta}\mathcal{C}_1 : |0\rangle|0\rangle|\Psi_0\rangle \mapsto \frac{|w_0\rangle}{\sqrt{N}}.$$

Moreover, all the four subroutines can be implemented with $\mathcal{O}(T/S)$ calls to $\mathcal{O}_{\mathcal{A}}$, polylogarithmically in $T$ many auxiliary qubits, and a number of gates that satisfies

$$\mathcal{O}\left(\frac{T}{S}\text{polylog}(T)\right).$$

If we have multiple algorithms $\{\mathcal{A}^{(j)}\}_{j=1}^{n}$, we simply run the concurrent versions of $\mathcal{C}_{|\psi\rangle}$ and $\mathcal{C}_{|\chi\rangle}$, and we supply a concurrent version of $\mathcal{C}_1$ which we build using the same techniques as in Lemma 75. With this, we can successfully implement concurrent access $\mathcal{C}$ to $\{\mathcal{C}_{|w_0^{(j)}\rangle}\}_{j=1}^{n}$ with the desired complexities. Since all the individual $\mathcal{C}_{|w_0^{(j)}\rangle}$'s are implemented up to error $\delta$, so is their concurrent access routine. □

### 4.5.6 Proof of Theorem 63

Having proven how to construct circuits that approximate the subroutines of Theorem 53, we are ready to prove Theorem 63, which we restate below for convenience.

**Theorem.** *Let $\mathcal{A}$ be a clean quantum query algorithm that acts on $k$ qubits, has query complexity $S$, time complexity $T$, and evaluates a function $f : X \subseteq \{0,1\}^n \to \{0,1\}$ with bounded error. Let $P_{\mathcal{A}}$ be the span program for this algorithm, as in Definition 57. Then we can implement the algorithm compiled from $P_{\mathcal{A}}$ with:*

1. *$\mathcal{O}(S)$ calls to $\mathcal{O}_x$.*
2. *$\mathcal{O}(T)$ calls to $\mathcal{O}_{\mathcal{A}}$ and $\mathcal{O}_{\mathcal{S}}$, as defined in Section 4.2.*
3. *$\mathcal{O}(T\text{polylog}(T))$ additional gates.*
4. *$\mathcal{O}(\text{polylog}(T))$ auxiliary qubits.*

*Proof.* From Theorem 58, we know that $P_{\mathcal{A}}$ positively $5\varepsilon$-approximates $f$ with complexity $C(P_{\mathcal{A}}) = \mathcal{O}(S)$. We can assume that the approximation factor $5\varepsilon$ is bounded away from 1 (i.e. $1 - 5\varepsilon = \mathcal{O}(1)$) because $\mathcal{A}$ is a clean quantum algorithm and so $\varepsilon \leq 1/3$. Otherwise, we can decrease the error of $\mathcal{A}$ using standard error-reduction techniques while only incurring a multiplicative cost to the overall time complexity. Hence, we deduce from Theorem 53 that we can implement the algorithm compiled from $P_{\mathcal{A}}$ with a number of calls to the subroutines $\mathcal{R}_{\ker(A)}$, $\mathcal{C}_{|w_0\rangle}$, $\mathcal{R}_{\mathcal{H}(x)}$ and $\mathcal{R}_{|0\rangle}$ that goes like

$$\mathcal{O}\left(\frac{C(P_{\mathcal{A}})}{(1-\lambda)^{3/2}}\log\frac{1}{1-\lambda}\right) = \mathcal{O}(S).$$

For $\mathcal{R}_{\ker(A)}$ and $\mathcal{C}_{|w_0\rangle}$ we choose error parameter $\delta = \Theta(S^{-1})$, which implies that $\log(1/\delta) = \mathcal{O}(\log(S))$. Given their respective query, space and time complexities in Lemmas 69, 76, 68, and 67, we can implement the span program algorithm with $\mathcal{O}(S)$ calls to $\mathcal{O}_x$, $\mathcal{O}(T)$ calls $\mathcal{O}_{\mathcal{A}}$ and $\mathcal{O}_{\mathcal{S}}$, $\mathcal{O}(\text{polylog}(T))$ extra qubits, and $\mathcal{O}\left(T\text{polylog}(T)\right)$ extra gates.

We proceed by analyzing the error introduced by our approximate implementation of the subroutines. First, an error is introduced due to the reflection $\mathcal{R}_{\mathcal{H}(x)}$ not leaving $\mathcal{H}_x$ exactly invariant. Observe that whenever we call $\mathcal{R}_{\mathcal{H}(x)}$, we are moving a part of the state outside of $\mathcal{H}_x$ that has amplitude at most $2\sqrt{2\varepsilon}/S$. So, there exists a state in $\mathcal{H}_x$ that is $2\sqrt{2\varepsilon}/S$-close to the state that we used in the analysis of the algorithm in 53, and we map to a state that is in turn $2\sqrt{2\varepsilon}/S$-close to this state. Hence, the total error introduced per call of $\mathcal{R}_{\mathcal{H}(x)}$ is $4\sqrt{2\varepsilon}/S$. Thus, the total error introduced is $\mathcal{O}(\sqrt{\varepsilon}) = \mathcal{O}(1)$.

Additional error is introduced by the approximate implementations of $\mathcal{R}_{|w_0\rangle}$ and $\mathcal{R}_{\ker(A)}$ in the step where we approximate the amplitudes of particular superpositions. Both are implemented up to error $\delta = \mathcal{O}(S^{-1})$ in the operator norm, which means that the total cumulative error is at most $\mathcal{O}(S \cdot \delta) = \mathcal{O}(1)$ as well.

All things included, running the algorithm of Theorem 53 with the subroutines of Lemmas 69, 76, 68, and 67 instead of the exact subroutines produces a state that is $\mathcal{O}(1)$ away from the ideal final state of the algorithm in the 2-norm. This might not be enough to guarantee that the total probability of error stays below 1/3. In that case, we choose an error $\delta = \mathcal{O}(S^{-1})$ with sufficiently small constant and use standard error-reduction techniques on the algorithm $\mathcal{A}$ for a constant number of rounds if necessary to make sure that he probability of erroneously deciding $f$ is $\varepsilon' \leq 1/3$. □

## 4.6 Application to variable-time search

One reason for converting quantum algorithms to span programs is that span programs compose very nicely (see [Rei09] for a number of examples). We illustrate this by describing a construction that, given $n$ span programs for $n$ functions $\{f_j : \{0,1\}^{m_j} \to \{0,1\}\}_{j=1}^n$, outputs a span program for the logical OR of their output: $f(x^{(1)}, \ldots, x^{(n)}) = \bigvee_{j=1}^n f_j(x^{(j)})$. In short, we show that given query-, time- and space-efficient quantum implementations for each $f_j$, the resulting span program can also be implemented query-, time- and

space-efficiently. The full theorem statement is provided below. Note that throughout this section for the sake of simplicity we write $f_j$ as functions on $\{0,1\}^{m_j}$ even though the results also hold for partial Boolean functions with arbitrary domains $X_j \subseteq \{0,1\}^{m_j}$.

**Theorem 78** (Variable-time quantum search)**.** *Let $\mathcal{A} = \{\mathcal{A}^{(j)}\}_{j=1}^n$ be a finite set of quantum algorithms, where $\mathcal{A}^{(j)}$ acts on $k_j \leq k_{\max}$ qubits and decides $f_j : \{0,1\}^{m_j} \to \{0,1\}$ with bounded error with query complexity $S_j$ and time complexity $T_j \leq T_{\max}$. Suppose that we have uniform access to the algorithms in $\mathcal{A}$ through the oracles $\mathcal{O}_\mathcal{A}$, $\mathcal{O}_\mathcal{S}$ and $\mathcal{O}_x$, as elaborated upon in Section 4.2, and that the $S_j$'s and $T_j$'s are known. Then we can implement a quantum algorithm that decides $f = \bigvee_{j=1}^n f_j$ with bounded error, with the following properties:*

1. *The number of calls to $\mathcal{O}_x$ is $\mathcal{O}\left(\sqrt{\sum_{j=1}^n S_j^2} \cdot \log\left(\sum_{j=1}^n S_j^2\right)\right)$.*
2. *The number of calls to $\mathcal{O}_\mathcal{A}$ and $\mathcal{O}_\mathcal{S}$ is $\mathcal{O}\left(\sqrt{\sum_{j=1}^n T_j^2} \cdot \log\left(\sum_{j=1}^n S_j^2\right)\right)$.*
3. *The number of extra gates is $\mathcal{O}\left(\sqrt{\sum_{j=1}^n T_j^2} \cdot \operatorname{polylog}(T_{\max}, n)\right)$.*
4. *The number of auxiliary qubits is $\mathcal{O}\left(\operatorname{polylog}(T_{\max}, n) + k_{\max}^{o(1)}\right)$.*

*If we additionally require that the error probabilities of the $\mathcal{A}^{(j)}$'s are all $o(1/\sum_{j=1}^n S_j^2)$, then the $\log(\sum_{j=1}^n S_j^2)$ factors and the $k_{\max}^{o(1)}$ term can be dropped. We can also drop the term $k_{\max}^{o(1)}$ if $T_j = k_j^{1+\Omega(1)}$ for all $j \in [n]$.*

A similar result was reached by Ambainis in [Amb10]. Let us discuss how our result compares to that of Ambainis.

First, we assume the uniform access model described in Section 4.2. This is a slight generalization of the model considered by Ambainis, as explained in [Amb10, Appendix A], because we differentiate between query and non-query time steps in the algorithms $\mathcal{A}^{(j)}$, whereas Ambainis does not. Therefore, Ambainis only considers the algorithm oracle $\mathcal{O}_\mathcal{A}$ and includes the queries to $x$ as part of this oracle, whereas we also assume to have explicit access to the oracles $\mathcal{O}_\mathcal{S}$ and $\mathcal{O}_x$.

One can obtain some of our results using Ambainis's construction and subsequently converting the resulting algorithm back to our setting. For instance, if one counts every query in the original algorithms as having unit cost, then Ambainis's construction yields an algorithm that evaluates $f =$

$\bigvee_{j=1}^n f_j$ with $\mathcal{O}(\sqrt{\sum_{j=1}^n S_j^2})$ queries to $\mathcal{O}_x$. This is a logarithmic factor better than our result, but the number of calls to $\mathcal{O}_\mathcal{A}$ and $\mathcal{O}_\mathcal{S}$ is unclear, and the time and space complexities are not analyzed.

Alternatively, if one assigns a unit cost to every gate in the original algorithms, then the algorithm that follows from Ambainis's construction performs $\mathcal{O}(\sqrt{\sum_{j=1}^n T_j^2})$ calls to $\mathcal{O}_\mathcal{A}$, $\mathcal{O}_\mathcal{S}$ and $\mathcal{O}_x$. Similarly as before, this is a logarithmic factor better in the scaling of the query complexity to $\mathcal{O}_\mathcal{A}$, but worse in the query complexity to $\mathcal{O}_x$ and again the time and space complexities are not analyzed.

Our improvement over Ambainis's work consists of the following elements. First, we show that one can attain both desired scalings in the number of calls to $\mathcal{O}_x$, $\mathcal{O}_\mathcal{S}$ and $\mathcal{O}_\mathcal{A}$ simultaneously, up to a single logarithmic factor. Second, our construction is also efficient with respect to the time and space complexities, as we show that we only suffer from polylogarithmic overhead in the number of extra gates and auxiliary qubits.

There are, however, some aspects to Ambainis's work that we did not reproduce. Ambainis proved a version of his theorem for the search problem: find $j$ such that $f_j(x^{(j)}) = 1$, whereas we only consider a decision version. By a standard reduction from the search version to the decision version, we also recover the analogous search result, but with an extra factor of $\log(n)$ overhead in the query and time complexities.

Ambainis also gives a result for the case where the costs of the original algorithms are unknown. It would be interesting to figure out whether our results can be similarly modified in the case where we do not know $\{T_j\}_{j=1}^n$ and/or $\{S_j\}_{j=1}^n$, but it is not immediately clear to us how one would go about this. We leave this for future research.

From Theorem 78 we easily deduce that if we have efficient uniform access to a set of algorithms, i.e., the oracles $\mathcal{O}_\mathcal{A}$ and $\mathcal{O}_\mathcal{S}$ can be implemented in time logarithmic in $T_{\max}$ and $n$, then the algorithm compiled from $P$ has query complexity $\widetilde{\mathcal{O}}(\sqrt{\sum_{j=1}^n S_j^2})$ and time complexity $\widetilde{\mathcal{O}}(\sqrt{\sum_{j=1}^n T_j^2})$.

The remainder of this section is dedicated to proving Theorem 78. In Section 4.6.1 we describe how we can merge $n$ span programs $P^{(1)}, \ldots, P^{(n)}$ evaluating functions $f_1, \ldots, f_n$, respectively, into one span program $P$ evaluating the OR of these functions, $f_1 \vee \cdots \vee f_n$. Subsequently, in Section 4.6.2, we relate the implementation of the algorithm compiled from $P$ to the implementation of the algorithms compiled from the individual $P^{(j)}$'s. Finally,

in Section 4.6.3, we specialize the span programs $P^{(j)}$ to be span programs of algorithms as defined in Section 4.4.1, and we relate the implementation of the required subroutines to the constructions in Section 4.5, completing the proof of Theorem 78.

### 4.6.1 The OR of span programs

Fix $\lambda \in (0, 1/n)$. For $j \in [n]$, let $P^{(j)} = (\mathcal{H}^{(j)}, \mathcal{V}^{(j)}, A^{(j)}, |\tau^{(j)}\rangle)$ be a span program on $\{0,1\}^{m_j}$ that positively $\lambda$-approximates $f_j : \{0,1\}^{m_j} \to \{0,1\}$.[3] Let $W_+^{(j)}$ and $W_-^{(j)}$ be some upper bounds on $W_+(P^{(j)})$ and $\widetilde{W}_-(P^{(j)})$ respectively, and assume that every $x \in f_j^{-1}(0)$ has an approximate negative witness $|\tilde{\omega}^{(j)}\rangle \in \mathcal{V}^{(j)}$ with $\left\|\langle\tilde{\omega}^{(j)}|A^{(j)}\Pi_{\mathcal{H}^{(j)}(x)}\right\|^2 \leq \lambda/W_+^{(j)}$ and $\left\|\langle\tilde{\omega}^{(j)}|A^{(j)}\right\|^2 \leq W_-^{(j)}$. Let $C_j = \sqrt{W_+^{(j)} W_-^{(j)}}$.

Assume, by applying an appropriate basis change, that $|\tau^{(j)}\rangle = |0\rangle$ for every $j \in [n]$. For each $j$, extend $|\tau^{(j)}\rangle = |0\rangle$ to an orthonormal basis $\{|0\rangle, |j,1\rangle, \ldots, |j, \dim(\mathcal{V}^{(j)}) - 1\rangle\}$ for $\mathcal{V}^{(j)}$ so that, aside from the single overlapping dimension $|0\rangle$, the subspaces $\mathcal{V}^{(j)}$ are orthogonal to one another. Let $\overline{\mathcal{V}}^{(j)} = \mathrm{span}\{|j,1\rangle, \ldots, |j, \dim(\mathcal{V}^{(j)}) - 1\rangle\}$, so that $\mathcal{V}^{(j)} = \mathrm{span}\{|0\rangle\} \oplus \overline{\mathcal{V}}^{(j)}$.

Let $f : \{0,1\}^{m_1+\cdots+m_n} \to \{0,1\}$ be the function defined by $f(x^{(1)}, \ldots, x^{(n)}) = \bigvee_{j=1}^n f_j(x^{(j)})$. We can define a span program $P$ on $\{0,1\}^{m_1+\cdots+m_n}$ that decides $f$ as follows:

$$\forall j \in [n], \ell \in [m_j], b \in \{0,1\}, \quad \mathcal{H}_{j,\ell,b} = \mathrm{span}\{|j\rangle\} \otimes \mathcal{H}_{\ell,b}^{(j)},$$

$$\mathcal{H}_{\mathrm{true}} = \bigoplus_{j=1}^n \mathcal{H}_{\mathrm{true}}^{(j)}, \quad \mathcal{H}_{\mathrm{false}} = \mathrm{span}\{|0,0\rangle\}$$

$$\mathcal{V} = \mathrm{span}\{|0\rangle\} \oplus \bigoplus_{j=1}^n \overline{\mathcal{V}}^{(j)}, \quad A = \sum_{j=1}^n \sqrt{W_+^{(j)}}\langle j| \otimes A^{(j)}, \quad |\tau\rangle = |0\rangle. \quad (4.23)$$

Above, we are indexing into an input $x \in \{0,1\}^{m_1+\cdots+m_n}$ by using a pair of indices, $j \in [n]$ and $\ell \in [m_j]$, in the obvious way. From this definition of $P$,

[3] We require $\lambda$ to be quite small here. One way to achieve this from an arbitrary span program is to convert it to an algorithm, reduce the error to $\mathcal{O}(1/n)$ at the expense of a $\mathcal{O}(\log n)$ multiplicative factor, and then convert that back to a span program using the construction in Section 4.4. Furthermore, we can just as well use partial functions here, but we don't for notational simplicity.

we get:

$$\mathcal{H}(x) = \bigoplus_{j\in[n]} \operatorname{span}\{|j\rangle\} \otimes \mathcal{H}^{(j)}(x^{(j)}),$$
$$\text{where} \quad \forall j \in [n], \quad \mathcal{H}^{(j)}(x^{(j)}) = \bigoplus_{\ell\in[m_j]} \mathcal{H}^{(j)}_{\ell,x^{(j)}_\ell}. \tag{4.24}$$

**Definition 79.** Let $\{P^{(j)}\}_{j=1}^n$ be a set of span programs, where $P^{(j)} = (\mathcal{H}^{(j)}, \mathcal{V}^{(j)}, A^{(j)}, |\tau^{(j)}\rangle)$. Then we let $P = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$ be the *OR span program* of these span programs, where $\mathcal{H}$, $\mathcal{V}$, $A$ and $|\tau\rangle$ are defined in equations (4.23), (4.24).

We proceed by proving various properties of the newly-defined OR span program. First, we prove that it indeed evaluates $f$ in the following theorem.

**Theorem 80.** *The span program $P$ positively $n\lambda$-approximates $f$ with complexity $C(P) \le \sqrt{\sum_{j=1}^n C_j^2}$.*

The proof will follow from Lemmas 81 and 82. First, we show that if $f(x) = 1$, $P$ accepts $x$, and give an upper bound on the positive witness complexity.

**Lemma 81.** *If $f(x) = 1$, then the span program $P$ accepts $x$, with positive witness complexity $w_+(x) \le 1$. Thus $W_+(P) \le 1$.*

*Proof.* If $f(x) = 1$, then there exists $j \in [n]$ such that $f_j(x^{(j)}) = 1$, so let $|w^{(j)}\rangle \in \mathcal{H}^{(j)}(x^{(j)})$ be a positive witness for $x^{(j)}$ in $P^{(j)}$ with $\left\||w^{(j)}\rangle\right\|^2 \le W_+^{(j)}$. Then let $|w\rangle = \frac{1}{\sqrt{W_+^{(j)}}}|j\rangle|w^{(j)}\rangle \in \mathcal{V}$. Then $A|w\rangle = A^{(j)}|w^{(j)}\rangle = |0\rangle$. Furthermore, since $\mathcal{H}(x) = \bigoplus_{j=1}^n \operatorname{span}\{|j\rangle\} \otimes \mathcal{H}^{(j)}(x^{(j)})$ by Eq. (4.24), and $|w^{(j)}\rangle \in \mathcal{H}^{(j)}(x^{(j)})$, we have $|w\rangle \in \mathcal{H}(x)$, so $|w\rangle$ is a positive witness for $x$. Since $\left\||w^{(j)}\rangle\right\|^2 \le W_+^{(j)}$, $|w\rangle$ has complexity $\||w\rangle\|^2 \le 1$. □

We complete the proof of Theorem 80 by exhibiting approximate negative witnesses.

**Lemma 82.** *If $f(x) = 0$, then there is an approximate negative witness $|\widetilde{\omega}\rangle$ with $\left\|\langle\widetilde{\omega}|A\Pi_{\mathcal{H}(x)}\right\|^2 \le n\lambda/W_+(P)$ and $\|\langle\widetilde{\omega}|A\|^2 \le \sum_{j=1}^n C_j^2$, so $P$ positively $n\lambda$-approximates $f$, and $\widetilde{W}_-(P) \le \sum_{j=1}^n C_j^2$.*

*Proof.* If $f(x) = 0$, then it must be the case that for all $j \in [n]$, $f_j(x^{(j)}) = 0$, so for each $j$, let $|\widetilde{\omega}^{(j)}\rangle$ be an approximate negative witness for $x^{(j)}$ in $P^{(j)}$ with $\left\|\langle\widetilde{\omega}^{(j)}|A^{(j)}\Pi_{\mathcal{H}^{(j)}(x^{(j)})}\right\|^2 \leq \lambda/W_+^{(j)}$, and $\left\|\langle\widetilde{\omega}^{(j)}|A^{(j)}\right\|^2 \leq W_-^{(j)}$. For each $j$, we can write $\langle\widetilde{\omega}^{(j)}| = \langle 0| + \langle\overline{\omega}^{(j)}|$ for some $|\overline{\omega}^{(j)}\rangle \in \overline{\mathcal{V}}^{(j)}$. We define $\langle\widetilde{\omega}| = \langle 0| + \sum_{j=1}^n \langle\overline{\omega}^{(j)}|$. Then $\langle\widetilde{\omega}|\tau\rangle = \langle\widetilde{\omega}|0\rangle = 1$. Furthermore, for each $j$, since the column space of $A^{(j)}$ is in $\mathcal{V}^{(j)} = \mathrm{span}\{|0\rangle\} \oplus \overline{\mathcal{V}}^{(j)}$, we have $\langle\widetilde{\omega}|A^{(j)} = (\langle 0| + \langle\overline{\omega}^{(j)}|)A^{(j)} = \langle\widetilde{\omega}^{(j)}|A^{(j)}$. Since $\mathcal{H}(x) = \bigoplus_{j=1}^n \mathrm{span}\{|j\rangle\} \otimes \mathcal{H}^{(j)}(x^{(j)})$ by Eq. (4.24),

$$\begin{aligned}\left\|\langle\widetilde{\omega}|A\Pi_{\mathcal{H}(x)}\right\|^2 &= \left\|\sum_{j=1}^n \sqrt{W_+^{(j)}}\langle j| \otimes (\langle\widetilde{\omega}^{(j)}|A^{(j)}\Pi_{\mathcal{H}^{(j)}(x^{(j)})})\right\|^2 \\ &\leq \sum_{j=1}^n W_+^{(j)}\frac{\lambda}{W_+^{(j)}} = n\lambda,\end{aligned}$$

so $P$ positively $n\lambda$-approximates $f$. Finally, we conclude $\widetilde{W}_-(P) \leq \sum_{j=1}^n C_j^2$ by observing:

$$\|\langle\widetilde{\omega}|A\|^2 = \sum_{j=1}^n W_+^{(j)}\left\|\langle\widetilde{\omega}^{(j)}|A^{(j)}\right\|^2 \leq \sum_{j=1}^n W_+^{(j)}W_-^{(j)} = \sum_{j=1}^n C_j^2. \qquad \square$$

We conclude this section by characterizing the minimal positive witness $|w_0\rangle$ and the kernel of $A$ in the following two lemmas.

**Lemma 83.** *The minimal positive witness of $P$ is given by*

$$|w_0\rangle = \frac{1}{\|\alpha\|^2}\sum_{j=1}^n \alpha_j|j\rangle \otimes \frac{|w_0^{(j)}\rangle}{\left\||w_0^{(j)}\rangle\right\|}, \qquad \textit{where} \qquad \alpha_j = \frac{\sqrt{W_+^{(j)}}}{\left\||w_0^{(j)}\rangle\right\|}$$

*and $|w_0^{(j)}\rangle$ are the minimal witnesses of $P^{(j)}$. Moreover, the minimal witness size is $N = 1/\|\alpha\|^2$.*

*Proof.* Observe that for every choice of $\beta_j$'s that sum to 1, we have

$$A\left[\sum_{j=1}^n \frac{\beta_j}{\sqrt{W_+^{(j)}}}|j\rangle \otimes |w_0^{(j)}\rangle\right] = \sum_{j=1}^n \beta_j A^{(j)}|w_0^{(j)}\rangle = \sum_{j=1}^n \beta_j|\tau\rangle = |\tau\rangle,$$

and that the minimal positive witness has to be of this form. Moreover, for all such choices of $\beta_j$, we have

$$\|\alpha\| \cdot \left\| \sum_{j=1}^{n} \frac{\beta_j}{\sqrt{W_+^{(j)}}} |j\rangle \otimes |w_0^{(j)}\rangle \right\| = \sqrt{\sum_{k=1}^{n} \alpha_k^2} \cdot \sqrt{\sum_{j=1}^{n} \frac{|\beta_j|^2 \left\| |w_0^{(j)}\rangle \right\|^2}{W_+^{(j)}}}$$
$$\geq \sum_{j=1}^{n} \alpha_j \cdot \frac{\beta_j}{\alpha_j} = \sum_{j=1}^{n} \beta_j = 1,$$

where we used the Cauchy-Schwarz inequality. Thus, we find that for all choices of $\beta_j$,

$$\left\| \sum_{j=1}^{n} \frac{\beta_j}{\sqrt{W_+^{(j)}}} |j\rangle \otimes |w_0^{(j)}\rangle \right\|^2 \geq \frac{1}{\|\alpha\|^2},$$

and the tightness of this inequality is shown by picking $\beta_j = \alpha_j^2 / \|\alpha\|^2$. Thus, the minimal witness is

$$|w_0\rangle = \frac{1}{\|\alpha\|^2} \sum_{j=1}^{n} \alpha_j |j\rangle \otimes \frac{|w_0^{(j)}\rangle}{\left\| |w_0^{(j)}\rangle \right\|},$$

completing the proof. □

The kernel of $A$ is not just the union of kernels of each $A^{(j)}$ because just as we can combine the minimal witnesses $|w_0^{(j)}\rangle$ to map to $|0\rangle$, we can make a combination that maps to 0. The following lemma characterizes such combinations of individual minimal witnesses and finds that they are orthogonal to the minimal witness for $P$.

**Lemma 84.** *Let* $K = \text{span}\{|j\rangle|w_0^{(j)}\rangle : j \in [n]\} \cap \text{span}\{|w_0\rangle\}^{\perp}$. *The kernel of* $A$ *is given by*

$$\ker(A) = \text{span}\{|0,0\rangle\} \oplus K \oplus \bigoplus_{j=1}^{n} \text{span}\{|j\rangle\} \otimes \ker(A^{(j)})$$

*Proof.* First, observe that $\bigoplus_{j=1}^{n} \text{span}\{|j\rangle\} \otimes \ker(A^{(j)}) \subseteq \ker(A)$, since for any $|h_j\rangle \in \ker(A^{(j)})$, $A|j\rangle|h_j\rangle = \sqrt{W_+^{(j)}} A^{(j)}|h_j\rangle = 0$. Similarly, observe that $|0,0\rangle$ vanishes under $A$, so it is part of the kernel of $A$ as well.

Thus, suppose $|h\rangle = \sum_{j=1}^{n} |j\rangle|h_j\rangle \in \mathcal{H}$ is in $\ker(A)$, and for all $j \in [n]$, $|h_j\rangle \in \mathrm{row}(A^{(j)})$. For all $k \in [n]$, we have

$$0 = \Pi_{\overline{\mathcal{V}}_k} \left[ A \sum_{j=1}^{n} |j\rangle|h_j\rangle \right] = \Pi_{\overline{\mathcal{V}}_k} \left[ \sum_{j=1}^{n} \sqrt{W_+^{(j)}} A^{(j)} |h_j\rangle \right] = \Pi_{\overline{\mathcal{V}}_k} A^{(k)} |h_k\rangle,$$

where we use the fact that $\Pi_{\overline{\mathcal{V}}_k} A^{(j)} = 0$ whenever $j \neq k$. Thus, we have that for all $k \in [n]$, $A^{(k)}|h_k\rangle \in \mathrm{span}\{|0\rangle\}$. Since we also have that $|h_k\rangle \in \mathrm{row}(A^{(k)})$, it must be the case that $|h_k\rangle \in \mathrm{span}\{|w_0^{(k)}\rangle\}$, so let $|h_k\rangle = \beta_k |w_0^{(k)}\rangle$. Then, we have:

$$\begin{aligned} 0 &= \sum_{j=1}^{n} \beta_j A^{(j)} |w_0^{(j)}\rangle = \sum_{j=1}^{n} \beta_j \sqrt{W_+^{(j)}} |0\rangle = \left[ \sum_{j=1}^{n} \beta_j \alpha_j \cdot \frac{\langle w_0^{(j)} | w_0^{(j)} \rangle}{\left\| |w_0^{(j)}\rangle \right\|} \right] |0\rangle \\ &= N \langle w_0 | h \rangle |0\rangle \end{aligned}$$

Hence, $|h\rangle = \sum_{j=1}^{n} \alpha_j |j\rangle |w_0^{(j)}\rangle \in \ker A$ if it is orthogonal to $|w_0\rangle$, which completes the proof. □

### 4.6.2 Implementation of the OR span program

Now that we have formally defined the OR span program in Definition 79, we proceed by analyzing the implementation cost of the algorithm compiled from it. To that end, we first of all assume that all of the spaces $\mathcal{H}^{(j)}$ correspond to $m$ qubits, i.e., $\mathcal{H}^{(j)} = \mathbb{C}^{2^m}$ for all $j \in [n]$. This is not much of a restriction, as we can always simply pad the smaller $\mathcal{H}^{(j)}$'s with extra qubits that we don't touch until our space is as big as the largest state space of the individual span programs.

The main idea of this section will be to use Theorem 53, and give implementations to the required subroutines in terms of the individual span programs $P^{(j)}$. This sometimes requires running several subroutines associated with the individual $P^{(j)}$'s in superposition. This is what we defined at the beginning of Section 4.5 as concurrent access. We repeat the definition here for convenience

**Definition** (Concurrent access)**.** Let $\mathcal{C}^{(1)}, \ldots, \mathcal{C}^{(n)}$ be quantum subroutines, all acting on the same Hilbert space $\mathcal{H}$. We say that a subroutine $\mathcal{C}$ provides

*concurrent access* to $\{\mathcal{C}^{(j)}\}_{j=1}^n$ if it performs the following action on $\mathbb{C}^{[n]} \otimes \mathcal{H}$:

$$\mathcal{C} = \sum_{j=1}^{n} |j\rangle\langle j| \otimes \mathcal{C}^{(j)}.$$

Next, we present the main theorem relating the cost of implementing the span program compiled from $P$ to the cost of implementing the subroutines that are associated with the individual $P^{(j)}$'s.

**Lemma 85.** *Let $\lambda \in [0, 1/n)$ and let $\{P^{(j)}\}_{j=1}^n$ be a set of span programs positively $\lambda$-approximating some functions $f_j$. For all $j \in [n]$, let $|w_0^{(j)}\rangle$ be a minimal positive witness for $P^{(j)}$ and let $W_+^{(j)} \geq W_+(P^{(j)})$ and $W_-^{(j)} \geq \widetilde{W}_-(P^{(j)})$ be upper bounds on the positive and negative complexities. Furthermore, for each $j \in [n]$ and $x^{(j)} \in \{0,1\}^{m_j}$, let $\mathcal{H}_{x^{(j)}}$ be an implementing subspace of $P^{(j)}$ for $x^{(j)}$. Let $P$ be the span program described in Eq. (4.23) and suppose that we have concurrent access to the following four sets of subroutines (as defined in Def. 64):*

1. *A circuit $\mathcal{R}_A$, providing concurrent access to the subroutines $\{\mathcal{R}_{\ker(A^{(j)})}\}_{j=1}^n$, where $\mathcal{R}_{\ker(A^{(j)})}$ acts on $\mathcal{H}_{x^{(j)}}$ as $2\Pi_{\ker(A^{(j)})} - I$.*
2. *A circuit $\mathcal{C}$, providing concurrent access to the subroutines $\{\mathcal{C}_{|w_0^{(j)}\rangle}\}_{j=1}^n$, where $\mathcal{C}_{|w_0^{(j)}\rangle}$ leaves $\mathcal{H}_{x^{(j)}}$ invariant and maps $|0\rangle$ to $|w_0^{(j)}\rangle/\||w_0^{(j)}\rangle\|$.*
3. *A circuit $\mathcal{R}_{\mathcal{H}}$, providing concurrent access to the subroutines $\{\mathcal{R}_{\mathcal{H}(x^{(j)})}\}_{j=1}^n$, where $\mathcal{R}_{\mathcal{H}(x^{(j)})}$ acts on $\mathcal{H}_{x^{(j)}}$ as $2\Pi_{\mathcal{H}(x^{(j)})} - I$.*
4. *A circuit $\mathcal{R}_0$, providing concurrent access to the subroutines $\{\mathcal{R}_{|0\rangle}^{(j)}\}_{j=1}^n$, where $\mathcal{R}_{|0\rangle}^{(j)}$ acts on $\mathcal{H}_{x^{(j)}}$ as $2|0\rangle\langle 0| - I$.*
5. *A circuit $\mathcal{C}_\alpha$ that prepares the superposition*

$$\mathcal{C}_\alpha : |0\rangle \mapsto \frac{1}{\|\alpha\|} \sum_{j=1}^{n} \alpha_j |j\rangle \qquad \text{where} \qquad \alpha_j = \frac{\sqrt{W_+^{(j)}}}{\left\||w_0^{(j)}\rangle\right\|}.$$

*Then, we can implement the span program algorithm for $P$ with a number of calls to the aforementioned circuits that satisfies*

$$\mathcal{O}\left(\frac{\sqrt{\sum_{j=1}^n C_j^2}}{(1-n\lambda)^{3/2}} \log \frac{1}{1-n\lambda}\right) \qquad \text{where} \qquad C_j = \sqrt{W_+^{(j)} W_-^{(j)}},$$

*and a number of extra gates and auxiliary qubits that satisfies*

$$\mathcal{O}\left(\text{polylog}\left(\sqrt{\sum_{j=1}^{n} C_j^2}, 1/(1-n\lambda), n\right)\right).$$

*Proof.* We apply Theorem 53 to $P$. As $P$ is positively $\lambda'$-approximating with $\lambda' = n\lambda < 1$, the first requirement is satisfied.

Next, we define the implementing subspace that we use. We take $\mathcal{H}_x$ to be

$$\mathcal{H}_x = \text{span}\{|0,0\rangle\} \oplus \bigoplus_{j=1}^{n} \text{span}\{|j\rangle\} \otimes \mathcal{H}_{x^{(j)}},$$

i.e., we have one orthogonal direction that contains all scalar multiples of the all-zeros state, and all the implementing subspaces associated with the individual $P^{(j)}$'s labeled by $j$. We refer to the first and second registers as the *label register* and *data register*, respectively.

Now, we turn our attention to the implementation of the four subroutines listed in Theorem 53. First, we implement the reflection through the $|0\rangle$-state, $\mathcal{R}_{|0\rangle}$. Observe that the all-zeros state in $\mathcal{H}_x$ is the state $|0,0\rangle$. But the only state in $\mathcal{H}_x$ that has zero in the label register is exactly the all-zeros state. Hence, we can simply reflect through $|0\rangle$ on the first register, which has only $\mathcal{O}(\log(n))$ qubits. Thus, we can implement $\mathcal{R}_{|0\rangle}$ in $\mathcal{O}(\log(n))$ gates.

Next, we turn our attention to the implementation of $\mathcal{C}_{|w_0\rangle}$. From Lemma 83 we find that

$$\frac{|w_0\rangle}{\||w_0\rangle\|} = \frac{1}{\|\alpha\|} \sum_{j=1}^{n} \alpha_j |j\rangle \otimes \frac{|w_0^{(j)}\rangle}{\left\||w_0^{(j)}\rangle\right\|}.$$

This allows for defining $\mathcal{C}_{|w_0\rangle}$ as the following procedure.

1. First, we prepare an auxiliary qubit in the state $|1\rangle$ whenever the data register is in the state $|0\rangle$, and $|0\rangle$ otherwise. This requires one controlled call to $\mathcal{R}_0$ together with $\mathcal{O}(1)$ auxiliary gates.
2. Next, conditioned on this auxiliary qubit, we apply $\mathcal{C}_\alpha$ to the label register.
3. Now, we uncompute the auxiliary qubit with the gates from step 1 applied in reverse. This uncomputation succeeds with certainty as the all-zeros states in all the $\mathcal{H}_{x^{(j)}}$'s are the same, and hence permuting the labels effectively permutes between different all-zeros states in the $\mathcal{H}_{x^{(j)}}$'s.

4. Finally, we call $\mathcal{C}$.

We observe that the first three steps perform some unitary on all the $n+1$ states that have the all-zeros state in the data register. As all these states are part of $\mathcal{H}_x$, they leave $\mathcal{H}_x$ invariant. Similarly, the fourth step leaves $\mathcal{H}_x$ invariant as all of the individual subroutines that make up $\mathcal{C}$ leave their respective implementing subspace $\mathcal{H}_{x^{(j)}}$ invariant. Hence, the entire procedure $\mathcal{C}_{|w_0\rangle}$ leaves $\mathcal{H}_x$ invariant.

Furthermore, observe that if we start in the state $|0,0\rangle$, the mapping that is implemented is the following

$$|0,0\rangle \overset{\text{steps } 1-3}{\mapsto} \frac{1}{\|\alpha\|}\sum_{j=1}^{n}\alpha_j|j,0\rangle \overset{\mathcal{C}}{\mapsto} \frac{1}{\|\alpha\|}\sum_{j=1}^{n}\alpha_j|j\rangle\otimes\frac{|w_0^{(j)}\rangle}{\left\||w_0^{(j)}\rangle\right\|}.$$

Thus, we conclude that we can implement $\mathcal{C}_{|w_0\rangle}$ using $\mathcal{O}(1)$ calls to $\mathcal{R}_0$, $\mathcal{C}_\alpha$ and $\mathcal{C}$ and $\mathcal{O}(1)$ extra gates and auxiliary qubits.

We proceed by providing an implementation of the reflection through the kernel of $A$. To that end, remember from Lemma 84 that

$$\ker(A) = \operatorname{span}\{|0,0\rangle\} \oplus \left[\operatorname{span}\{|w_0\rangle\}^\perp \cap \underbrace{\operatorname{span}\{|j\rangle|w_0^{(j)}\rangle : j\in[n]\}}_{W_0}\right] \bigoplus_{j=1}^{n} \operatorname{span}\{|j\rangle\}\otimes\ker(A^{(j)}).$$

As $\operatorname{span}\{|w_0\rangle\}\subseteq W_0$, we observe that

$$2\Pi_{\ker(A)} - I = (2|0,0\rangle\langle 0,0| - I)\left(\frac{2|w_0\rangle\langle w_0|}{\||w_0\rangle\|^2} - I\right)(2\Pi_{W_0} - I) \cdot \left(\sum_{j=1}^{n}|j\rangle\langle j|\otimes\left(2\Pi_{\ker(A^{(j)})} - I\right)\right).$$

The first factor is simply $\mathcal{R}_{|0\rangle}$ on $\mathcal{H}_x$. Similarly, the last factor is exactly the action of the concurrent reflection $\mathcal{R}_A$ on $\mathcal{H}_x$. The second factor can easily be implemented by the sequence $\mathcal{C}_{|w_0\rangle}\mathcal{R}_{|0\rangle}\mathcal{C}_{|w_0\rangle}^\dagger$. So, it remains to implement

the third factor, which we can achieve by observing that on $\mathcal{H}_x$ we have

$$2\Pi_{W_0} - I = \sum_{j=1}^{n} |j\rangle\langle j| \otimes \left( \frac{2|w_0^{(j)}\rangle\langle w_0^{(j)}|}{\left\||w_0^{(j)}\rangle\right\|^2} - I \right) = \sum_{j=1}^{n} |j\rangle\langle j| \otimes \left( \mathcal{C}_{|w_0^{(j)}\rangle} \mathcal{R}_{|0\rangle}^{(j)} \mathcal{C}_{|w_0^{(j)}\rangle}^{\dagger} \right)$$
$$= \mathcal{C}\mathcal{R}_0\mathcal{C}^{\dagger}.$$

Thus, we have

$$\mathcal{R}_{\ker(A)} = \mathcal{R}_{|0\rangle}\mathcal{C}_{|w_0\rangle}\mathcal{R}_{|0\rangle}\mathcal{C}_{|w_0\rangle}^{\dagger}\mathcal{C}\mathcal{R}_0\mathcal{C}^{\dagger}\mathcal{R}_A.$$

As all the individual factors leave $\mathcal{H}_x$ invariant, so does their product. Hence, we can implement $\mathcal{R}_{\ker(A)}$ with $\mathcal{O}(1)$ calls to the subroutines mentioned in the statement of the lemma.

It remains to implement the routine $\mathcal{R}_{\mathcal{H}(x)}$. To that end, observe that

$$2\Pi_{\mathcal{H}(x)} - I = \sum_{j=1}^{n} |j\rangle\langle j| \otimes \left(2\Pi_{\mathcal{H}^{(j)}(x^{(j)})} - I\right),$$

which implies that we can simply implement the reflection through $\mathcal{H}(x)$ with one call to $\mathcal{R}_{\mathcal{H}}$.

We have implemented all routines in the statement of Theorem 53 with $\mathcal{O}(1)$ calls to the routines listed in the statement of this lemma. That means that the total number of calls to these routines is equal up to constants to the expression in Theorem 53, which reduces to

$$\mathcal{O}\left( \frac{\sqrt{W_+(P)\widetilde{W}_-(P)}}{(1-n\lambda)^{3/2}} \log \frac{1}{1-2n\lambda} \right) = \mathcal{O}\left( \frac{\sqrt{\sum_{j=1}^{n} C_j^2}}{(1-n\lambda)^{3/2}} \log \frac{1}{1-n\lambda} \right).$$

Moreover, it follows directly from the statement of Theorem 53 that the total number of extra gates is $\mathcal{O}(\text{polylog}(\sqrt{\sum_{j=1}^{n} C_j^2}, 1/(1-n\lambda)))$. This completes the proof. □

### 4.6.3 Implementation of variable time quantum search

In this section, we prove Theorem 78. The core idea is to first convert the algorithms into span programs using the construction from Section 4.4, next merge them into an OR span program as in Definition 79, and finally convert that back into a quantum algorithm using Lemma 85.

There is one caveat though. If we naively use the span programs of the algorithms $\mathcal{A}^{(j)}$ from Definition 57 with the upper bounds on the positive witness sizes that follow from Lemma 59, then we might end up with a minimal witness $|w_0\rangle$ with completely arbitrary coefficients $\alpha_j$ (see Lemma 83), making it too time-consuming to implement $\mathcal{C}_\alpha$. We circumvent this using a technique that was already present in Ambainis's original paper [Amb10], which we name the *binning technique*. The next two lemmas formalize this idea.

**Lemma 86.** *Let $0 < \gamma_{\min} = \gamma_1 \leq \cdots \leq \gamma_n = \gamma_{\max}$. Then, we can efficiently find a sequence of integers $0 = j_0 \leq \cdots \leq j_k = n$ such that $k \leq \lceil \log(\gamma_{\max}/\gamma_{\min}) \rceil \cdot \lceil \log(n) \rceil$ and the following two properties hold:*

1. *For all $\ell \in [k]$, $j_\ell - j_{\ell-1}$ is a power of 2.*
2. *For all $\ell \in [k]$ and $j \in [j_{\ell-1}+1, j_\ell]$,*

$$\frac{\gamma_{j_\ell}}{2} \leq \gamma_j \leq \gamma_{j_\ell}.$$

*Proof.* We let $k' = \lceil \log(\gamma_{\max}/\gamma_{\min}) \rceil$. Then, with every $j \in [n]$, we associate the unique integer $m_j$ such that $\gamma_j \in [\gamma_{\min} \cdot 2^{m_j - 1}, \gamma_{\min} \cdot 2^{m_j})$. As

$$\gamma_{\min} \cdot 2^0 \leq \gamma_{\min} \leq \gamma_j \leq \gamma_{\max} = \gamma_{\min} \cdot 2^{\log(\gamma_{\max}/\gamma_{\min})} \leq \gamma_{\min} \cdot 2^{k'},$$

we find that $m_j \leq k'$, and hence $m_j \in [k']$. Moreover, as the $\gamma_j$'s are non-decreasing, so are the $m_j$'s. Now, for every $\ell \in [k']$, we define $j'_\ell = \max\{j \in [n] : m_j = \ell\}$ and we let $j'_0 = 0$. We find that for all $\ell \in [k']$ and $j \in [j'_{\ell-1}+1, j'_\ell]$,

$$\frac{\gamma_{j'_\ell}}{2} \leq \frac{\gamma_{\min} \cdot 2^\ell}{2} = \gamma_{\min} \cdot 2^{\ell-1} \leq \gamma_j \leq \gamma_{j'_\ell},$$

and hence the second condition is verified.

Now, for every $\ell \in [k']$, we write $j'_\ell - j'_{\ell-1}$ in terms of its binary expansion, i.e., we write

$$j'_\ell - j'_{\ell-1} = 2^{p_{\ell,1}} + \cdots + 2^{p_{\ell,k_\ell}},$$

where $p_{\ell,1} > \cdots > p_{\ell,k_\ell}$. As $j'_\ell - j'_{\ell-1} \leq n$, we have that $k_\ell \leq \lceil \log(n) \rceil$. Finally, we let

$$(j_\ell)_{\ell=1}^k = (j'_0, j'_0 + 2^{p_{1,1}}, \ldots, j'_0 + 2^{p_{1,1}} + \cdots + 2^{p_{1,k_1-1}}, j'_1, \ldots, j'_{k'})$$

The difference between two consecutive terms is always a power of two by construction, and the length satisfies

$$k = \sum_{\ell=1}^{k'} k_\ell \leq k' \cdot \lceil \log(n) \rceil \leq \lceil \log(\gamma_{\max}/\gamma_{\min}) \rceil \cdot \lceil \log(n) \rceil,$$

completing the proof. □

The above lemma is nothing more than a statement about how we can put a sequence of positive reals into several bins. We now use it to modify the upper bounds $W_+^{(j)}$, so as to make the cost of implementing $\mathcal{C}_\alpha$ scale more favorably.

**Lemma 87.** *Let $\mathcal{A} = \{\mathcal{A}^{(j)}\}_{j=1}^n$ be a finite set of quantum algorithms, where $\mathcal{A}^{(j)}$ has query complexity $1 \leq S_j \leq S_{\max}$. Let $P^{(j)}$ be the span program of $\mathcal{A}^{(j)}$. Then, we can define positive reals $\{W_+^{(j)}\}_{j=1}^n$ such that $W_+(P^{(j)}) \leq W_+^{(j)} \leq 12(2S_j + 1)$, and a sequence of integers $0 = j_0 \leq \cdots \leq j_k = n$ with $k \leq \lceil \frac{1}{2} \log(6S_{\max}) \rceil \cdot \lceil \log(n) \rceil$, such that for every $\ell \in [k]$, $j_\ell - j_{\ell-1}$ is a power of 2 and for every $j \in [j_{\ell-1} + 1, j_\ell]$,*

$$\alpha_j = \frac{\sqrt{W_+^{(j)}}}{\left\| |w_0^{(j)}\rangle \right\|} = \frac{\sqrt{W_+^{(j_\ell)}}}{\left\| |w_0^{(j_\ell)}\rangle \right\|} = \alpha_{j_\ell}.$$

*With this choice of upper bounds $\{W_+^{(j)}\}_{j=1}^n$, we can implement the circuit $\mathcal{C}_\alpha$, as defined in Lemma 85, with $\mathcal{O}(\log(S_{\max}) \log^2(n))$ gates and $\mathcal{O}(\log(n))$ auxiliary qubits.*

*Proof.* For all $j \in [n]$, let

$$\gamma_j = \frac{\sqrt{3(2S_j + 1)}}{\left\| |w_0^{(j)}\rangle \right\|}.$$

Assume without loss of generality that the algorithms $\mathcal{A}^{(j)}$ are ordered such that $0 < \gamma_{\min} = \gamma_1 \leq \cdots \leq \gamma_n = \gamma_{\max}$. From Lemma 62 we deduce that $1/\sqrt{2} \leq \left\| |w_0^{(j)}\rangle \right\| \leq \sqrt{3}$, and hence

$$\frac{\gamma_{\max}}{\gamma_{\min}} \leq \sqrt{\frac{3(2S_{\max} + 1)}{9}} \cdot \sqrt{6} \leq \sqrt{6S_{\max}}.$$

According to Lemma 86, we can now find a sequence $0 = j_1 \leq \cdots \leq j_k = n$ with $k \leq \lceil \frac{1}{2}\log(6S_{\max})\rceil \cdot \lceil \log(n)\rceil$, such that for all $\ell \in [k]$ we have that $j_\ell - j_{\ell-1}$ is a power of two and for all $j \in [j_{\ell-1}+1, j_\ell]$, we have

$$\frac{\gamma_{j_\ell}}{2} \leq \gamma_j \leq \gamma_{j_\ell}.$$

Given such a $j$, we define $W_+^{(j)} = \gamma_{j_\ell}^2 \cdot \left\| |w_0^{(j)}\rangle \right\|^2$. Then we find

$$W_+^{(j)} = \gamma_{j_\ell}^2 \cdot \left\| |w_0^{(j)}\rangle \right\|^2 \leq 4\gamma_j^2 \cdot \left\| |w_0^{(j)}\rangle \right\|^2 = 12(2S_j + 1),$$

and according to Lemma 59,

$$W_+(P^{(j)}) \leq 3(2S_j+1) = \gamma_j^2 \left\| |w_0^{(j)}\rangle \right\|^2 \leq \gamma_{j_\ell}^2 \cdot \left\| |w_0^{(j)}\rangle \right\|^2 = W_+^{(j)}.$$

Moreover, we have for all such $j$ that

$$\alpha_j = \frac{\sqrt{W_+^{(j)}}}{\left\| |w_0^{(j)}\rangle \right\|} = \gamma_{j_\ell} = \frac{\sqrt{W_+^{(j_\ell)}}}{\left\| |w_0^{(j_\ell)}\rangle \right\|} = \alpha_{j_\ell}.$$

Thus, it remains to show that we can implement $\mathcal{C}_\alpha$ in $\mathcal{O}(\log(S_{\max})\log(n))$ gates. To that end, we first of all define the mapping $\mathsf{S}$ that acts as the identity on $|0\rangle|0\rangle|0\rangle$ and that given a $j \in [j_{\ell-1}+1, j_\ell]$ implements

$$\mathsf{S} : |j\rangle|0\rangle|0\rangle \mapsto |0\rangle|\ell\rangle|j - j_{\ell-1} - 1\rangle,$$

where the registers are of size $\lceil \log(n+1)\rceil$, $\lceil \log(k+1)\rceil$ and $\lceil \log(n+1)\rceil$, respectively. Moreover, as the values of the $j_\ell$'s are known beforehand, this can be implemented with $\mathcal{O}(k)$ arithmetic circuits that all have $\mathcal{O}(\log(n))$ gates, so the number of gates needed to implement $\mathcal{S}$ is $\mathcal{O}(\log(S_{\max})\log^2(n))$.

We define the subspace

$$\mathcal{X} = \mathrm{span}\{|0\rangle|0\rangle|0\rangle\} \oplus \mathrm{span}\{|0\rangle|\ell\rangle|j - j_{\ell-1} - 1\rangle : j \in [j_{\ell-1}+1, j_\ell]\}.$$

Observe that $\mathsf{S}$ maps any state $|j\rangle|0\rangle|0\rangle$ into $\mathcal{X}$, and moreover that $\mathsf{S}^\dagger$ will set the final two registers to $|0\rangle$ when it is applied to any state in $\mathcal{X}$. Hence, as long as we stay in $\mathcal{X}$, we can always uncompute the final two registers.

Now, we implement $\mathcal{C}_\alpha$ as follows, where we treat the final two registers as ancilla registers.

1. We apply $\mathsf{S}$. This maps our state into $\mathcal{X}$, and will leave $|0\rangle|0\rangle|0\rangle$ unaltered.
2. Controlled on the last register being in the state $|0\rangle$, we apply on the second register the map

$$\mathcal{C} : |0\rangle \mapsto \frac{1}{\|\alpha\|} \sum_{\ell=1}^{k} \alpha_\ell \sqrt{j_\ell - j_{\ell-1}} |\ell\rangle.$$

   This leaves $\mathcal{X}$ invariant as $|0\rangle|\ell\rangle|0\rangle \in \mathcal{X}$ for every $\ell \in [k]_0$. As this is a map on $\mathcal{O}(\log(k))$ qubits, it can be implemented with a number of gates of order $\mathcal{O}(k) = \mathcal{O}(\log(S_{\max}) \log(n))$.
3. Next, controlled on the second register being in state $|\ell\rangle$, we perform the map $I \otimes H^{\otimes \log(j_\ell - j_{\ell-1})}$ to the final register. This only affects the basis states in $\mathcal{X}$ and implements

$$|0\rangle|\ell\rangle|0\rangle \mapsto |0\rangle|\ell\rangle \frac{1}{\sqrt{j_\ell - j_{\ell-1}}} \sum_{m=0}^{j_\ell - j_{\ell-1}-1} |m\rangle.$$

   This circuit can be built using $k$ times at most $\log(n)$ controlled $H$, plus some arithmetic circuits on $\log(k)$ qubits to set the controls. Hence, the number of gates needed to implement this step is $\mathcal{O}(k \log(n) + \log(k)) = \mathcal{O}(\log(S_{\max}) \log^2(n))$.
4. We implement $\mathsf{S}^\dagger$. Since steps 2 and 3 left $\mathcal{X}$ invariant, we can now uncompute the final two registers.

The total time complexity of $\mathcal{C}_\alpha$ hence now becomes the sum of the time complexities of the above steps, which is $\mathcal{O}(\log(S_{\max}) \log^2(n))$, and it maps

$$\begin{aligned} |0\rangle|0\rangle|0\rangle &\mapsto |0\rangle \frac{1}{\|\alpha\|} \sum_{\ell=1}^{k} \alpha_\ell \sqrt{j_\ell - j_{\ell-1}} |\ell\rangle|0\rangle \mapsto |0\rangle \frac{1}{\|\alpha\|} \sum_{\ell=1}^{k} \alpha_\ell |\ell\rangle \sum_{m=0}^{j_\ell - j_{\ell-1}-1} |m\rangle \\ &\mapsto \frac{1}{\|\alpha\|} \sum_{j=1}^{n} \alpha_\ell |j\rangle|0\rangle|0\rangle. \end{aligned}$$

This completes the proof. □

Now that we can implement $\mathcal{C}_\alpha$ in a number of gates that scales poly-logarithmically in both $S_{\max}$ and $n$, we turn our attention to the proof of Theorem 78.

*Proof of Theorem 78.* First of all, we consider the case where the $n$ algorithms $\{\mathcal{A}^{(j)}\}_{j=1}^n$ are clean quantum algorithms with error probabilities satisfying $\varepsilon_j < 1/(80n)$ and $\varepsilon_j = o(1/\sum_{j=1}^n S_j^2)$. In the final paragraph of this proof, we will lift this restriction.

We modify the algorithms $\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(n)}$ slightly. Similar to the proof of Lemma 8, we insert a sequence $\mathcal{O}_x, I, \mathcal{O}_x$ into all $\mathcal{A}^{(j)}$'s at a spacing $B$ defined by

$$B = \left\lceil \sqrt{\frac{\sum_{j=1}^n T_j^2}{\sum_{j=0}^n S_j^2}} \right\rceil.$$

We denote the algorithm that we obtain after this modification by $\overline{\mathcal{A}}^{(j)}$, and its query and time complexity by $\overline{S}_j$ and $\overline{T}_j$, respectively. Using a similar analysis as in the proof of Lemma 8, we obtain

$$\overline{S}_j = \Theta\left(S_j + \frac{T_j}{B}\right), \qquad \overline{T}_j = \Theta(T_j), \qquad \text{and} \qquad \frac{\overline{T}_j}{\overline{S}_j} = \mathcal{O}(B).$$

Next, we turn these algorithms $\{\overline{\mathcal{A}}^{(j)}\}_{j=1}^n$ into span programs $\{P^{(j)}\}_{j=1}^n$ using Definition 57. According to Lemma 87, we can define the upper bounds $\{W_+^{(j)}\}_{j=1}^n$ such that

$$W_+(P^{(j)}) \leq W_+^{(j)} \leq 12(2\overline{S}_j + 1) = \mathcal{O}(\overline{S}_j),$$

and such that we can implement $\mathcal{C}_\alpha$ in a number of gates and auxiliary qubits that scales polylogarithmically in $S_{\max}$ and $n$. In addition, for all $j \in [n]$ we can take $W_-^{(j)} = \mathcal{O}(\overline{S}_j)$ by virtue of Lemma 60, which implies that $C_j = \mathcal{O}(\overline{S}_j)$. From Lemma 60 we have a negative witness $|\tilde{\omega}\rangle$ that satisfies

$$\left\| \langle \tilde{\omega} | A^{(j)} \Pi_{\mathcal{H}^{(j)}(x)} \right\|^2 \leq \frac{5\varepsilon_j}{3(2\overline{S}_j + 1)} \leq \frac{20\varepsilon_j}{W_+^{(j)}},$$

which implies that all $P^{(j)}$'s are positive $\lambda$-approximating with $\lambda \leq 20\varepsilon_j < 1/(4n)$.

We have now shown that we satisfy the requirements for constructing the OR span program $P$, as defined in Definition 79. According to Theorem 80,

the complexity of this span program is now upper bounded by

$$\begin{aligned} C(P) &\leq \sqrt{\sum_{j=1}^{n} C_j^2} = \mathcal{O}\left(\sqrt{\sum_{j=1}^{n} \overline{S}_j^2}\right) = \mathcal{O}\left(\sqrt{\sum_{j=1}^{n}\left[S_j^2 + \frac{T_j^2}{B^2}\right]}\right) \\ &= \mathcal{O}\left(\sum_{j=1}^{n} S_j^2 + \frac{1}{B^2}\sum_{j=1}^{n} T_j^2\right) = \mathcal{O}\left(\sum_{j=1}^{n} S_j^2 + \frac{\sum_{j=1}^{n} S_j^2}{\sum_{j=1}^{n} T_j^2} \cdot \sum_{j=1}^{n} T_j^2\right) \\ &= \mathcal{O}\left(\sum_{j=1}^{n} S_j^2\right). \end{aligned}$$

According to Lemma 85, implementing the algorithm compiled from $P$ takes a number of calls to the subroutines $\mathcal{R}_{\mathcal{A}}$, $\mathcal{C}$, $\mathcal{R}_{\mathcal{H}}$ and $\mathcal{R}_0$ that satisfies

$$\mathcal{O}\left(\frac{\sqrt{\sum_{j=1}^{n} C_j^2}}{(1-n\lambda)^{3/2}} \cdot \log\frac{1}{1-n\lambda}\right) = \mathcal{O}\left(\sqrt{\sum_{j=1}^{n} S_j^2}\right),$$

and a number of extra gates and auxiliary qubits that satisfies

$$\mathcal{O}\left(\text{polylog}\left(\sqrt{\sum_{j=1}^{n} C_j^2}, \frac{1}{1-2n\lambda}\right)\right) = \mathcal{O}\left(\text{polylog}(S_{\max}, n)\right).$$

According to Lemmas 69, 76, and 68, we can construct $\mathcal{R}_{\mathcal{A}}$, $\mathcal{C}$ and $\mathcal{R}_{\mathcal{H}}$ with $\mathcal{O}(1)$ calls to $\mathcal{O}_x$, a number of calls to $\mathcal{O}_{\mathcal{A}}$, $\mathcal{O}_{\mathcal{S}}$ that satisfies

$$\mathcal{O}\left(\max_{j\in[n]} \frac{\overline{T}_j}{\overline{S}_j}\right) = \mathcal{O}\left(B\right),$$

a number of auxiliary gates that satisfies

$$\mathcal{O}\left(\max_{j\in[n]} \frac{\overline{T}_j}{\overline{S}_j} \cdot \text{polylog}(\overline{T}_j)\right) = \mathcal{O}\left(B \cdot \text{polylog}(T_{\max})\right),$$

and a number of auxiliary qubits that is polylogarithmic in $T_{\max}$. If we ensure that the answer register is located on the same qubit for all the algorithms $\mathcal{A}^{(j)}$'s, we can implement $\mathcal{R}_0$ with $\mathcal{O}(\text{polylog}(T_{\max}))$ gates. This implies that the total number of calls to $\mathcal{O}_{\mathcal{A}}$ and $\mathcal{O}_{\mathcal{S}}$ is

$$\mathcal{O}\left(\sqrt{\sum_{j=1}^{n} S_j^2} \cdot B\right) = \mathcal{O}\left(\sqrt{\sum_{j=1}^{n} S_j^2} \cdot \sqrt{\frac{\sum_{j=1}^{n} T_j^2}{\sum_{j=1}^{n} S_j^2}}\right) = \mathcal{O}\left(\sqrt{\sum_{j=1}^{n} T_j^2}\right),$$

and the total number of auxiliary gates is

$$\mathcal{O}\left(\sqrt{\sum_{j=1}^{n} S_j^2}\cdot B\cdot \mathrm{polylog}(T_{\max},n)\right)=\mathcal{O}\left(\sqrt{\sum_{j=1}^{n} T_j^2}\cdot \mathrm{polylog}(T_{\max},n)\right).$$

This completes the proof of the claimed complexities.

It remains to check that the success probability of our algorithm compiled from $P$ is sufficiently high. We have $\mathcal{O}(\sqrt{\varepsilon_j})$-precise implementations of $\mathcal{R}_{\ker(A^{(j)})}$ and $\mathcal{R}^{(j)}_{|0\rangle}$ w.r.t. operator norm. Thus, our resulting implementations of $\mathcal{R}_{\ker(A)}$ and $\mathcal{R}_0$ are accurate in the operator norm up to error

$$\max_{j\in[n]} 4\sqrt{2\varepsilon_j}=o\left(\frac{1}{\sqrt{\sum_{j=1}^{n} S_j^2}}\right).$$

Similarly, the subroutines $\mathcal{C}_{|w_0^{(j)}\rangle}$ only approximately stay within $\mathcal{H}_{x^{(j)}}$. Thus,

$$\sup_{\substack{|h\rangle\in\mathcal{H}_x\\ \||h\rangle\|=1}} \left\|\Pi_{\mathcal{H}_x^{\perp}}\mathcal{C}_{|w_0\rangle}|h\rangle\right\| \le \max_{j\in[n]} 2\sqrt{2\varepsilon_j}=o\left(\frac{1}{\sqrt{\sum_{j=1}^{n} S_j^2}}\right).$$

As we call these two subroutines a total of $\sqrt{\sum_{j=1}^{n} S_j^2}$ times, these errors influence the final success probability at most by $o(1)$, using a similar argument as in the proof of Theorem 63. Thus, our implementation of the algorithm compiled from $P$ succeeds with bounded error.

Finally, we remove the restriction that we imposed on the algorithms $\{\mathcal{A}^{(j)}\}_{j=1}^{n}$ at the beginning of this proof. We can always reduce the error probability of our algorithms to $o(1/\sum_{j=1}^{n} S_j^2)$ using standard techniques. This conversion incurs a multiplicative factor of $\mathcal{O}(\log(\sum_{j=1}^{n} S_j^2))$ in the query and time complexities, and in the worst case an additive term of $k_{\max}^{o(1)}$ in the number of auxiliary qubits. Accounting for them in the relevant complexities completes the proof. □

## 4.7 Discussion and outlook

In this chapter, we reach two main results. First, we prove in Section 4.5.6 that every quantum query algorithm can be converted into a span program

and back into a quantum algorithm while keeping the query and time complexity unaffected up to polylogarithmic factors. This implies that span programs fully capture both query and time complexity up to polylogarithmic factors, which strengthens the motivation for considering span programs as an important formalism from which to derive quantum algorithms.

Our second result, in Section 4.6, is an improvement on Ambainis's variable time search result – we can obtain a better-than-Grover speed-up in both query and time complexity simultaneously, where the query complexity is measured in the number of calls to $\mathcal{O}_x$ providing access to the input $x$ and the time complexity is measured in the number of calls to $O_{\mathcal{A}}$, and $\mathcal{O}_{\mathcal{S}}$ providing access to the descriptions of the algorithms, plus the other auxiliary gates, which are not counted in Ambainis's paper. Our construction goes via a composition of span programs. Even though the analysis of the time complexity of the algorithm compiled from this composed span program is quite involved, the actual composition is rather simple. This exemplifies the power of the span program framework.

This section leaves several open ends for further research as well. First, we do not re-derive all of the results that Ambainis obtains in his work. For instance, we do not consider the case where the query and time complexities of the original algorithms are not known in advance, so it would be interesting to investigate whether we could match Ambainis's result in this setting as well. This would probably require somewhat modifying the input model that we describe in Section 4.2.

Similarly, we handle the decision version of the search problem whereas Ambainis handles the full search version. It would be interesting to see if one can recover the full search algorithm as well. One possible direction would be to investigate whether one could use span programs with non-binary outputs for that, as described for instance in [BT20].

The most interesting direction of further research that we foresee, though, is whether the relative ease with which span programs can be composed can be exploited to obtain more composition results. The variable time search result composes a set of arbitrary functions with the OR function and obtains a better-than-Grover speed-up in the query and time complexity of the resulting algorithm. A natural next step would be to investigate if similar types of speed-ups can be obtained when one composes some arbitrary functions with threshold functions.

# Part III

# The one where we discuss applications of span programs

# Chapter 5

# Span programs for graph problems

## 5.1 Overview

At the end of the introductory section of Chapter 3 we promised the reader a three-course meal out of the *st*-connectivity span program, and by Turing, we shall give it to you.

While span program algorithms are universal for quantum query algorithms, it can also be fruitful to analyse the unitaries used in these algorithms in ways that are different from how they appear in the standard span program algorithm. For example, Ref. [IJ19] presents an algorithm to estimate span program witness sizes, and we have already presented algorithms for witness generation in Section 3.4. We take a similar approach in this chapter, deriving new algorithms based on unitaries from the span program algorithm for *st*-connectivity, deriving new span programs for graph connectivity based on the *st*-connectivity span program, and applying the algorithm for witness generation to the *st*-connectivity span program. The chapter contains the results of [JJK+18], as well as an application of the witness generation algorithm in Section 3.4.2 to *st*-path finding, and a new span program for graph connectivity. This is joint work with Stacey Jeffery and Shelby Kimmel.

**Span program for *st*-connectivity** In Section 5.2 we begin with describing in detail a span program for the *st*-connectivity problem, slightly generalizing that of [BR12]. In a nutshell, in the *st*-connectivity problem a

parent graph $G$ and two connected vertices $s$ and $t$ are fixed, and one is asked to decide whether $s$ and $t$ are still connected in a subgraph $G(x)$ of $G$, with $x$ in some domain $X \subseteq \{0,1\}^N$ for some $N \in \mathbb{N}$. In [BR12], the authors gave a tight characterization of the positive witness size as the effective resistance. Diverse upper bounds on the negative witness size have followed since [JK17; Bel12b], but no tight, general characterization was known.

**Characterization of the negative witness** In Section 5.3, we bring the story to its conclusion by showing that the negative witness size of the $st$-connectivity span program is exactly characterized by the *effective capacitance* of the input graph (Theorem 90). At a high-level, this well-studied electrical network quantity is a measure of the charge that the network could store between the component containing $s$ and the component containing $t$ if put under a unit potential difference. The more, shorter paths between these two components in the graph $G \setminus G(x)$, the greater the capacitance. This characterization tells us that quantum algorithms can quickly decide $st$-connectivity on graphs that are promised to have either small effective resistance or small effective capacitance. In Section 5.3.1, we apply this characterization to give a new quantum algorithm for estimating the capacitance of an input graph $G(x)$.

**An algorithm for graph connectivity** Next, we use this tighter analysis of the negative witness to analyse a new algorithm for graph connectivity. This problem was first studied in the context of quantum algorithms by Dürr, Høyer, Heiligman and Mhalla [DHH+06], who gave an optimal $\widetilde{\mathcal{O}}(n^{3/2})$ upper bound on the time complexity. A span-program-based quantum algorithm with optimal query complexity was later presented by Ārinš [Āri16], whose algorithm also uses only $\mathcal{O}(\log n)$ space.

Since a graph is connected if and only if every pair of vertices $\{u, v\}$ are connected, we propose an algorithm that uses the technique of [NT95; JK17] to convert the conjunction of $\binom{n}{2}$ $st$-connectivity span programs into a single $st$-connectivity span program: take $n(n-1)/2$ copies of $G(x)$, one for each pair of distinct vertices $\{u, v\}$ with $u < v$, and call $u$ the source and $v$ the sink of this graph. Connect these graphs in series, in any order, by identifying the sink of one to the source of the next. Call the source of the first graph $s$, and the sink of the last graph $t$. See Fig. 5.1 for an example when $G$ is a triangle. In this way we have created a graph (which we denote $\mathcal{G}(x)$)

that is $st$-connected if and only if $G(x)$ is connected. In other words, for any $x \in \{0,1\}^{E(G)}$, $\text{CONN}_G(x) = st\text{-CONN}_{\mathcal{G}}(x)$.

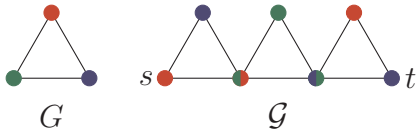



Figure 5.1: The graph $\mathcal{G}$ is $st$-connected if and only of $G$ is connected.

By analysing the effective resistance and capacitance of $\mathcal{G}$, we show that when $G$ has no multi-edges, $\text{CONN}_G$ can be solved in query complexity $\mathcal{O}(n\sqrt{R/\kappa})$ (Theorem 98), where $R$ is an upper bound on the average resistance if $G(x)$ is connected and $\kappa$ is a lower bound on the number of components if $G(x)$ is not connected. For the case when $G$ has multi-edges, we get an upper bound of $\mathcal{O}(n^{3/4}\sqrt{Rd_{\max}(G)}/\kappa^{1/4})$ on the query complexity, where $d_{\max}(G)$ is the maximum degree of any vertex in the graph.

In the worst case, when $R = n$ and $\kappa = 2$, our algorithm achieves the optimal query upper bound of $\widetilde{\mathcal{O}}(n^{3/2})$ when the parent graph has no multi-edges. Like the algorithm of Ref. [Āri16], our algorithm uses only $\mathcal{O}(\log n)$ space. It is also the first connectivity algorithm that applies to the case where $G$ is not necessarily the complete graph, although the other algorithms can likely be adapted to the more general case.

The algorithm of Āriņš can be seen as similar to ours, except that rather than connecting copies of $G(x)$ for each $\{u, v\}$ pair, his algorithm only considers pairs $\{1, v\}$ for $v \neq 1$. In contrast, our algorithm is symmetric in the vertex set, which makes a detailed analysis more natural.

**Spectral algorithms for graph connectivity** In Section 5.5, we present an alternative approach to deciding graph connectivity. It is based on phase estimation of a particular unitary that is also used in the $st$-connectivity span program, but applied to a different initial state.

We first show that the quantum query complexity of deciding $\text{CONN}_{G,X}$ is $\mathcal{O}(\sqrt{nd_{\max}(G)/(\kappa\lambda)})$, when we're promised that if $G(x)$ is connected, the second smallest eigenvalue of the Laplacian of $G(x)$, $\lambda_2(G(x))$, is at least $\lambda$, and otherwise, $G(x)$ has at least $\kappa > 1$ connected components (Corollary 104). Unlike the previous algorithm, this one is not optimal in the worst case, but it can be better than our first algorithm under some conditions. We compare

the two and discuss what these conditions are in Section 5.8. In addition to calls to the span program unitary, this algorithm requires an initial state of a particular form, and while this state is independent of the input, it may generally not be time efficient to produce such a state. In Sections 5.5.1 and 5.5.2 we analyse the time complexity of our algorithm in two contexts, one that works for every graph $G$ but may be significantly suboptimal and one specialized to Cayley graphs.

**Spectral algorithm for estimating the algebraic connectivity** In Section 5.5.3 we give an algorithm to estimate the *algebraic connectivity* of $G(x)$, $\lambda_2(G(x))$, when $G$ is a complete graph. The algebraic connectivity is closely related to the inverse of the mixing time, which is known to be small for many interesting families of graphs such as expander graphs. We give a protocol in Theorem 120 that with probability at least $2/3$ outputs an estimate of $\lambda_2(G(x))$ up to multiplicative error $\varepsilon$ in time complexity $\widetilde{\mathcal{O}}\left(\frac{1}{\varepsilon}\frac{n}{\sqrt{\lambda_2(G(x))}}\right)$.

**A span program for Connectivity without surgery** In Section 5.6 we modify the target of the *st*-connectivity span program to obtain a span program $P_s$ for $\text{CONN}_{G,X}$ for every vertex $s$ and derive a quantum algorithm that decides $\text{CONN}_{G,X}$ with bounded error and with query complexity $\mathcal{O}\left(\sqrt{RC\frac{n^2}{(n-n_{\max})^2}}\right)$, where $R$ is a known upper bound on $R_{\text{avg}}(G(x))$ for all connected $G(x)$, and for all disconnected $G(x)$, $C$ is an upper bound for the largest out-degree of any component of $G(x)$ and $n_{\max}$ upper bounds the size of the largest component of $G(x)$. Unlike our first algorithm for graph connectivity, this span program does not require us to concatenate copies of $G$ in a long chain, it is not symmetric with respect to the vertices, and unlike that span program, the span program algorithm associated to $P_s$ is not worst-case optimal. However, this span program has the advantage that its query and time complexities are easy to analyse and become well-behaved if there is not a component that dominates in size all the others when $G(x)$ is not connected, allowing it to outperform our other two algorithms in some cases (see Section 5.8).

**Application of witness generation to shortest-path finding** In Chapter 3 we gave two algorithms that generate an $\varepsilon$-close approximation of the

optimal positive witness for any span program. In Section 5.7, we apply the first witness generation algorithm to the *st*-connectivity span program of a particular graph encoding the function $\mathrm{OR}_{\mathrm{m}} \circ \mathrm{AND}_{\mathrm{d}}$. We use the witness generated to obtain an *st*-path in any connected subgraph $G(x)$ using $\mathcal{O}(\sqrt{md})$ queries and logarithmic space, which is quadratically better than the classical $\Theta(md)$ query lower bound and outperforms the path-finding algorithm in [DHH+06] in both query and space complexity. Our proof-of-concept algorithm is, we hope, a first step towards a quantum algorithm for path finding with small (logarithmic) space, which remains an open problem.

Finally, in Section 5.8 we discuss our results, compare our algorithms and talk about the open problems that emerge from them.

## 5.2 A span program for *st*-connectivity

An important example of a span program is the one for *st*-connectivity, first introduced in [KW93], and used in [BR12] to give a new quantum algorithm for *st*-connectivity.

We state this decision problem and span program below, somewhat generalized to include weighted (multi-)graphs, and to allow the input to be specified as a subgraph of some parent graph $G$ that is not necessarily the complete graph. Unless stated otherwise, we assume graphs can have more than one edge between two vertices. We assume $G$ has some implicit weighting function $c$. Weighted graphs, also known as *networks*, are described in Section 2.3.1.

We allow a string $x \in \{0,1\}^N$ to specify a subgraph $G(x)$ of $G$ in the fairly general way described in Section 2.3.1, *Subgraphs*. In particular, for $i \in [N]$, let $\overrightarrow{E}_{i,1} \subseteq \overrightarrow{E}(G)$ denote the set of (directed) edges associated with the literal $x_i$, and $\overrightarrow{E}_{i,0}$ the set of edges associated with the literal $\overline{x_i}$. Note that if $(u,v,\ell) \in \overrightarrow{E}_{i,b}$ then we must also have $(v,u,\ell) \in \overrightarrow{E}_{i,b}$, since $G(x)$ is an undirected graph. Subgraphs of a weighted graph are also weighted graphs.

For a parent graph $G$ and a family of subgraphs $G(x)$, $x \in X = \{0,1\}^N$, let $s$, $t$ denote two connected vertices in $G$. The *st*-connectivity problem is the function $f : X \to \{0,1\}$ defined as $f(x) = 1$ iff $s$ and $t$ are connected in $G(x)$. We denote this problem as $st\text{-}\mathrm{CONN}_{G,X}$.

Then we refer to the following span program as $P_G$:

**Definition 88** (*st*-connectivity span program)**.** Let $G = (V(G), E(G))$ be an undirected weighted multigraph and $x \in \{0,1\}^N$ specify a subgraph $G(x)$ as above. For all $i \in [N], b \in \{0,1\}$ we define the spaces

$$\mathcal{H}_{i,b} = \text{span}\{|u,v,\ell\rangle : (u,v,\ell) \in \overrightarrow{E}_{i,b}\}; \quad \mathcal{V} = \text{span}\{|v\rangle : v \in V(G)\} \tag{5.1}$$

And we define the span program map and target as

$$A = \sum_{(u,v,\ell)\in\overrightarrow{E}(G)} \sqrt{c(u,v,\ell)}(|u\rangle - |v\rangle)\langle u,v,\ell|; \quad |\tau\rangle = |s\rangle - |t\rangle. \tag{5.2}$$

We call $P_G = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$ the span program for *st*-connectivity over $G$.

Let us now provide some intuition for the inner workings of $P_G$. Pick any particular input $x$, assume $G(x)$ is connected and let $(s = u_1, \ldots, u_d = t)$ denote a path in $G(x)$ from $s$ to $t$ of length $d$. Assume, for simplicity, that all weights are 1. Then, the following state is a positive witness.

$$|w\rangle = \sum_{i=1}^{d-1} |u_i, u_{i+1}\rangle \tag{5.3}$$

That is because:

$$A|w\rangle = \sum_{i=1}^{d-1} (|u_i\rangle - |u_{i+1}\rangle) = |u_1\rangle - |u_d\rangle = |s\rangle - |t\rangle.$$

We conclude that every *st*-path in $G(x)$ is a witness for $|\tau\rangle$. In fact, it is possible to see that *any* witness is a linear combination of *st*-paths, possibly plus a few cycles, more on that later. This characterization, achieved through the correspondence between witnesses and *st*-flows, is the object of the next lemma. The proof follows closely that of Appendix *A* in [JK17], albeit with a little modification that will facilitate the discussion later.

**Lemma 89** ([JK17])**.** *Let $P_G$ be the st-connectivity span program from Definition 88. Then for any $x \in \{0,1\}^N$, $w_+(x, P_G) = \frac{1}{2}R_{s,t}(G(x))$.*

*Proof.* Let us assume that $s$ and $t$ are connected in $G(x)$ and let $\theta$ be a unit *st*-flow on $G(x)$ as in Definition 13. Define the state

$$|w\rangle = \sum_{(u,v,\ell)\in\overrightarrow{E}(G(x))} \frac{1}{2}\frac{\theta(u,v,\ell)}{\sqrt{c(u,v,\ell)}}|u,v,\ell\rangle. \tag{5.4}$$

Then, the state is obviously in $\mathcal{H}(x)$ and the image of $|w\rangle$ under $A$ is:

$$A|w\rangle = \frac{1}{2} \sum_{(u,v,\ell)\in\overrightarrow{E}(G(x))} \theta(u,v,\ell)(|u\rangle - |v\rangle). \tag{5.5}$$

Since $\theta$ is an $st$-flow, $\theta(u,v,\ell) = -\theta(v,u,\ell)$, so

$$A|w\rangle = \sum_{(u,v,\ell)\in\overrightarrow{E}(G(x))} \theta(u,v,\ell)|u\rangle. \tag{5.6}$$

Now, let $|u\rangle \in V(G) \setminus \{s,t\}$. From the definition of an $st$-flow we have:

$$\langle u|A|w\rangle = \sum_{v:(u,v,\ell)\in\overrightarrow{E}(G(x))} \theta(u,v,\ell) = 0.$$

But also,

$$\langle s|A|w\rangle = \sum_{v:(s,v,\ell)\in\overrightarrow{E}(G(x))} \theta(s,v,\ell) = 1,$$
$$\langle t|A|w\rangle = \sum_{v:(t,v,\ell)\in\overrightarrow{E}(G(x))} \theta(t,v,\ell) = -1.$$

Thus, we conclude that $A|w\rangle = |s\rangle - |t\rangle$, hence $|w\rangle$ is a positive witness for $x$, and the witness size of $|w\rangle$ is an upper bound on the positive witness size of $P_G$ for $x$. That is,

$$w_+(x,P_G) \leq \sum_{(u,v,\ell)\in\overrightarrow{E}(G(x))} \frac{1}{4}\frac{\theta^2(u,v,\ell)}{c(u,v,\ell)} = \frac{1}{2}\mathcal{J}_{\overrightarrow{E}(G(x))}(\theta). \tag{5.7}$$

In particular, if we take $\theta$ to be the minimal energy $st$-flow on $G(x)$, then we have $w_+(x,P_G) \leq R_{s,t}(G(x))/2$.

Now let us prove the opposite inequality. We shall see that given an optimal positive witness $|w_x\rangle$ we can define a unit $st$-flow whose energy is twice the witness size. Let $\theta : \overrightarrow{E}(G) \to \mathbb{R}$ be defined as

$$\forall (u,v,\ell) \in \overrightarrow{E}(G), \quad \theta(u,v,\ell) := 2\sqrt{c(u,v,\ell)}\langle u,v,\ell|w_x\rangle.$$

Since the vectors $\{|u,v,\ell\rangle : (u,v,\ell) \in \overrightarrow{E}(G)\}$ form a basis of $\mathcal{H}$, we have:

$$w_+(x,P_G) = \||w_x\rangle\|^2 = \sum_{(u,v,\ell)\in\overrightarrow{E}(G(x))} |\langle u,v,\ell|w_x\rangle|^2 = \sum_{(u,v,\ell)\in\overrightarrow{E}(G(x))} \frac{1}{4}\frac{\theta^2(u,v,\ell)}{c(u,v,\ell)}$$
$$= \frac{1}{2}\mathcal{J}_{\overrightarrow{E}(G(x))}(\theta) \geq \frac{1}{2}R_{s,t}(G(x)).$$

Here we have used that $|w_x\rangle$ is a positive witness to limit the sum to edges over $G(x)$. All that is left is proving that $\theta$ is a unit $st$-flow over $G(x)$. Observe that

$$|w_x\rangle = \sum_{(u,v,\ell)\in\overrightarrow{E}(G(x))} \frac{1}{2}\frac{\theta(u,v,\ell)}{\sqrt{c(u,v,\ell)}}|u,v,\ell\rangle,$$

and so $A|w_x\rangle = |\tau\rangle$ can be written as:

$$|\tau\rangle = A|w_x\rangle = \frac{1}{2}\sum_{(u,v,\ell)\in\overrightarrow{E}(G(x))} \frac{\theta(u,v,\ell)}{\sqrt{c(u,v,\ell)}}|u\rangle - \frac{\theta(u,v,\ell)}{\sqrt{c(u,v,\ell)}}|v\rangle$$
$$= \frac{1}{2}\sum_{(u,v,\ell)\in\overrightarrow{E}(G(x))} \frac{\theta(u,v,\ell)-\theta(v,u,\ell)}{\sqrt{c(u,v,\ell)}}|u\rangle. \tag{5.8}$$

Next, we prove that $\theta(u,v,\ell) = -\theta(v,u,\ell)$. Consider the positive witness $|w_x\rangle$ as written in the basis of edges of $G$, $|w_x\rangle = \sum\langle u,v,\ell|w_x\rangle|u,v,\ell\rangle$, then $|\widetilde{w}_x\rangle = -\sum\langle v,u,\ell|w_x\rangle|u,v,\ell\rangle$ is too a positive witness, and with the same norm as $|w_x\rangle$ because $A|u,v,\ell\rangle = -A|v,u,\ell\rangle$. Therefore $\frac{|w_x\rangle+|\widetilde{w}_x\rangle}{2}$ is a positive witness as well. By the triangle inequality,

$$\left\|\frac{|w_x\rangle+|\widetilde{w}_x\rangle}{2}\right\| \leq \frac{1}{2}\||w_x\rangle\| + \frac{1}{2}\||\widetilde{w}_x\rangle\| = \sqrt{w_+(x,P_G)},$$

with equality if and only if $|w_x\rangle$ and $|\widetilde{w}_x\rangle$ are parallel. By the optimality of $|w_x\rangle$ we conclude that that must be the case, which implies that $\langle u,v,\ell|w_x\rangle = -\langle v,u,\ell|w_x\rangle$ and $\theta(u,v,\ell) = -\theta(v,u,\ell)$ for all $(u,v,\ell) \in \overrightarrow{E}(G)$.

Coming back to Eq. (5.8), from the anti-symmetry of $\theta$ we get:

$$A|w_x\rangle = \sum_{(u,v,\ell)\in\overrightarrow{E}(G(x))} \theta(u,v,\ell)|u\rangle,$$

and so taking the inner product with $|s\rangle$, $|t\rangle$ and $|u\rangle$ for $u \in V(G)\setminus\{s,t\}$ we obtain:

$$\begin{aligned}\langle u|A|w\rangle &= \sum_{v,\ell:(u,v,\ell)\in\overrightarrow{E}(G(x))} \theta(u,v,\ell) = 0,\\ \langle s|A|w\rangle &= \sum_{v,\ell:(s,v,\ell)\in\overrightarrow{E}(G(x))} \theta(s,v,\ell) = 1,\\ \langle t|A|w\rangle &= \sum_{v,\ell:(t,v,\ell)\in\overrightarrow{E}(G(x))} \theta(t,v,\ell) = -1.\end{aligned}$$

We conclude that $\theta$ is a unit $st$-flow, and $\frac{R_{s,t}(G(x))}{2} \leq \frac{\mathcal{J}_{\overrightarrow{E}(G)}(\theta)}{2} = w_+(x, P_G)$. □

Jeffery and Kimmel prove in [JK17] a little bit more than this because they give an $st$-flow for any witness, not necessarily optimal. Observe that we could apply our arguments not only to the optimal positive witness for any given $x$ but to the minimal witness. Remember that the minimal witness is defined in Section 3.2.2 as the smallest $|w\rangle \in \mathcal{H}$ such that $A|w\rangle = |\tau\rangle$, a vector which we denoted as $|w_0\rangle = A^+|\tau\rangle$.

Therefore, Lemma 89 actually characterizes $|w_0\rangle$ as:

$$|w_0\rangle = \sum_{(u,v,\ell)\in\overrightarrow{E}(G)} \frac{1}{2}\frac{\theta(u,v,\ell)}{\sqrt{c(u,v,\ell)}}|u,v,\ell\rangle, \tag{5.9}$$

where $\theta$ is the minimal energy $st$-flow in $\overrightarrow{E}(G)$, and $\||w_0\rangle\|^2 = \frac{1}{2}R_{s,t}(G)$.

Now, let $|\varphi\rangle \in \mathcal{H}$ be such that $A|\varphi\rangle = 0$, that is, $|\varphi\rangle \in \ker(A)$. Define $\theta : \overrightarrow{E}(G) \to \mathbb{R}$ as $\theta(u,v,\ell) = \sqrt{c(u,v,\ell)}\,(\langle u,v,\ell|\varphi\rangle - \langle v,u,\ell|\varphi\rangle)$. Notice that we immediately obtain $\theta(u,v,\ell) = -\theta(v,u,\ell)$. Then

$$\begin{aligned}A|\varphi\rangle &= \sum_{(u,v,\ell)\in\overrightarrow{E}(G)} \sqrt{c(u,v,\ell)}\,(|u\rangle\langle u,v,\ell|\varphi\rangle - |v\rangle\langle u,v,\ell|\varphi\rangle)\\ &= \sum_{u\in V(G)} \sum_{v,\ell:(u,v,\ell)\in\overrightarrow{E}(G)} \sqrt{c(u,v,\ell)}\,(|u\rangle\langle u,v,\ell|\varphi\rangle - |u\rangle\langle v,u,\ell|\varphi\rangle)\\ &= \sum_{u\in V(G)} \sum_{v,\ell:(u,v,\ell)\in\overrightarrow{E}(G)} \theta(u,v,\ell)|u\rangle = 0.\end{aligned} \tag{5.10}$$

We conclude that for every $u \in V(G)$ we have

$$\sum_{v,\ell:(u,v,\ell)\in\vec{E}(G)} \theta(u,v,\ell) = 0.$$

This is what is known as a *circulation* over $G$, which is like a flow but without source or sink. It is easy to verify that every circulation is a convex combination of cycles on $G$, and that every circulation defines a vector in $\ker(A)$. It is just as easy to verify that every cycle in $G$ is a vector in $\ker(A)$, just as every $st$ path is an $st$-witness. Hence, we have characterized $\ker(A)$ as the cycle space of $G$, i.e. the space spanned by the cycles in $G$. All that is left is to characterize the negative witnesses. That is the purpose of the next section.

## 5.3 Effective capacitance and $st$-connectivity

In this section, we will prove the following theorem:

**Theorem 90.** *Let $P_G$ be the span program in Definition (88) and $C_{s,t}(G(x))$ be the capacitance between $s$ and $t$ in $G(x)$ from Definition 18. Then, for any $x \in \{0,1\}^N$, $w_-(x, P_G) = 2C_{s,t}(G(x))$.*

Previously, the negative witness size of $P_G$ was characterized by the size of a cut [RŠ12] or, in planar graphs, the effective resistance of a graph related to the planar dual of $G(x)$ [JK17].

We will prove Theorem 90 shortly, but first, we mention the following corollary:

**Corollary 91.** *Let $G$ be a multigraph with $s,t \in V(G)$. Then for* any *choice of (non-negative, real-valued) implicit weight function, the bounded error quantum query complexity of evaluating $st$-$\mathrm{CONN}_{G,X}$ is*

$$O\left(\sqrt{\max_{\substack{x\in X \\ st\text{-}\mathrm{CONN}_{G,X}(x)=1}} R_{s,t}(G(x)) \times \max_{\substack{x\in X \\ st\text{-}\mathrm{CONN}_{G,X}(x)=0}} C_{s,t}(G(x))}\right). \tag{5.11}$$

*Proof.* The positive and negative witness sizes are exactly characterized in Theorem 90 and Lemma 89. The result follows from Theorem 45. □

We emphasize that Corollary 91 holds for $R_{s,t}$ and $C_{s,t}$ defined with respect to any weight function, some of which may give a significantly better complexity for solving this problem. We are now ready to prove Theorem 90, the main result of this section.

*Proof of Theorem 90.* First, we prove that any unit *st*-potential on $G(x)$ as defined in Definition 16 can be transformed into a negative witness for $x$ in $P_G$ with witness size equal to twice the unit potential energy of that potential. This shows that $w_-(x, P_G) \leq 2C_{s,t}(G(x))$.

Given a unit *st*-potential $\mathsf{V} : V(G) \to \mathbb{R}$ on $G(x)$, we consider $\langle \omega_{\mathsf{V}}| = \sum_{v \in V(G)} \mathsf{V}(v)\langle v|$. Then because $\mathsf{V}(s) = 1$ and $\mathsf{V}(t) = 0$, we have $\langle \omega_{\mathsf{V}}|\tau\rangle = 1$. Secondly,

$$\begin{aligned}\langle \omega_{\mathsf{V}}|A\Pi_{\mathcal{H}(x)} &= \sum_{u' \in V(G)} \mathsf{V}(u')\langle u'| \sum_{(u,v,\ell) \in \overrightarrow{E}(G(x))} \sqrt{c(u,v,\ell)}(|u\rangle - |v\rangle)\langle u, v, \ell| \\ &= \sum_{(u,v,\ell) \in \overrightarrow{E}(G(x))} \sqrt{c(u,v,\ell)}(\mathsf{V}(u) - \mathsf{V}(v))\langle u, v, \ell| = 0, \qquad (5.12)\end{aligned}$$

where we've used the definition of unit *st*-potential, which states that $\mathsf{V}(u) - \mathsf{V}(v) = 0$ when $(u, v, \ell) \in E(G(x))$. Thus $\langle \omega_{\mathsf{V}}|$ is a valid negative witness for input $x$.

We have

$$\begin{aligned}w_-(x, P_G) &\leq \min_{\mathsf{V}} \|\langle \omega_{\mathsf{V}}|A\|^2 \\ &= \min_{\mathsf{V}} \left\| \sum_{(u,v,\ell) \in \overrightarrow{E}(G)} \sqrt{c(u,v,\ell)}(\mathsf{V}(u) - \mathsf{V}(v))\langle u, v, \ell| \right\|^2 \\ &= 2 \min_{\mathsf{V}} \sum_{(u,v,\ell) \in E(G)} (\mathsf{V}(u) - \mathsf{V}(v))^2 c(u,v,\ell) = 2C_{s,t}(G(x)), \qquad (5.13)\end{aligned}$$

where the minimization is over unit *st*-potentials on $G(x)$.

Next, we show that any negative witness $\langle \omega|$ for $P_G$ on input $x$ can be transformed into a unit *st*-potential $\mathsf{V}_\omega$ on $G(x)$, with negative witness size equal to twice the unit potential energy of $\mathsf{V}_\omega$. This shows that $w_-(x, P_G) \geq 2C_{s,t}(G(x))$.

Let $\langle \omega|$ be a negative witness for input $x$, define the potential $\mathsf{V}_\omega(v) := \langle \omega|(|v\rangle - |t\rangle)$ for $v \in V(G)$. Then $\mathsf{V}_\omega(s) = \langle \omega|(|s\rangle - |t\rangle) = \langle \omega|\tau\rangle = 1$, and

$\mathsf{V}_\omega(t) = \langle\omega|(|t\rangle - |t\rangle) = 0$. Also for $(u, v, \ell) \in E(G(x))$, we have

$$\begin{aligned}\mathsf{V}_\omega(u) - \mathsf{V}_\omega(v) &= \langle\omega|(|u\rangle - |t\rangle) - \langle\omega|(|v\rangle - |t\rangle) = \langle\omega|(|u\rangle - |v\rangle) \\ &= \langle\omega|A|(u, v, \ell)\rangle = \langle\omega|A\Pi_{H(x)}|(u, v, \ell)\rangle = 0, \end{aligned} \tag{5.14}$$

because $\langle\omega|A\Pi_{H(x)} = 0$. Thus, $\mathsf{V}_\omega$ is an $st$-unit potential for $G(x)$.

Then

$$\begin{aligned} w_-(x, P_G) =& \min_{\langle\omega|} \|\langle\omega|A\|^2 = \min_{\langle\omega|} \sum_{(u,v,\ell)\in\overrightarrow{E}(G)} (\langle\omega|(|u\rangle - |v\rangle))^2 c(u, v, \ell) \\ =& 2 \min_{\langle\omega|} \sum_{(u,v,\ell)\in E(G)} (\mathcal{V}_\omega(u) - \mathcal{V}_\omega(v))^2 c(u, v, \ell) \geq 2C_{s,t}(G(x)), \end{aligned} \tag{5.15}$$

where the minimization is over negative witnesses. Since we have both $w_-(x, P_G) \geq 2C_{s,t}(G(x))$ and $w_-(x, P_G) \leq 2C_{s,t}(G(x))$, we conclude that $w_-(x, P_G) = 2C_{s,t}(G(x))$. □

### 5.3.1 Estimating the capacitance of a circuit

The second use of this characterization of $w_-(x, P_G)$ will be an algorithm for estimating the capacitance of a graph. The algorithm is a particular case of an algorithm described in [IJ19] for arbitrary span programs, adapted for the $st$-connectivity span program. The general algorithm is based on the direct connection between the minimal witness $|w_0\rangle$, the 0-phase eigenspace of $U(x, P_G)$ and the negative witness size that we discussed in Section 3.3.2. We restate here the statement for convenience.

**Theorem 92** ([IJ19])**.** *Fix $X \subseteq [q]^N$ and $f : X \to \mathbb{R}_{\geq 0}$. Let $P = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$ be a span program on $[q]^N$ such that for all $x \in X$, $f(x) = w_-(x, P)$ and define $\widehat{W}_+ = \widehat{W}_+(P) = \max_{x\in X} \widetilde{w}_+(x, P, e_-(x, P))$. Then there exists a quantum algorithm that estimates $f$ to accuracy $\varepsilon$ and uses $\widetilde{\mathcal{O}}\left(\frac{1}{\varepsilon^{3/2}}\sqrt{w_-(x)\widehat{W}_+}\right)$ calls to $U(x, P) = (2\Pi_{\ker(A)} - I)(2\Pi_{\mathcal{H}(x)} - I)$ and elementary gates.*

By Theorem 90, $w_-(x, P_G) = 2C_{s,t}(G(x))$, so we can apply Theorem 92 to estimate $C_{s,t}(G(x))$. By Theorem 92, the complexity of doing this depends on $C_{s,t}(G(x))$ and $\widehat{W}_+(P_G) = \max_x \tilde{w}_+(x, P_G)$. We will prove the following theorem:

**Theorem 93.** *Let $P_G$ be the st-connectivity span program from Definition 88. Then, we have $\widehat{W}_+(P_G) = \mathcal{O}(\max_p J_{E(G)}(p))$, where the maximum runs over all unit st-flows $p$ that are paths from $s$ to $t$ and $J$ is the energy of the flow from Definition 14.*

Note that when the weights are all 1, $\max_p J_{E(G)}(p)$ is just the length of the longest self-avoiding $st$-path in $G$. Combining Theorem 92, Theorem 90, and Theorem 93, we have:

**Corollary 94.** *Given a network $(G, c)$, with $s, t \in V(G)$ and access to an oracle $\mathcal{O}_x$, the bounded error quantum query complexity of estimating $C_{s,t}(G(x))$ to accuracy $\varepsilon$ is $\widetilde{\mathcal{O}}(\varepsilon^{-3/2}\sqrt{C_{s,t}(G(x)) \max_p J_{E(G)}(p)})$ where the maximum runs over all unit st-flows $p$ that are paths from $s$ to $t$.*

Similarly, we can give a time upper bound for this problem:

**Corollary 95.** *Let $\mathsf{U}$ be the cost of implementing the map*

$$|u\rangle|0\rangle \mapsto \sum_{v,\ell:(u,v,\ell)\in\vec{E}(G)} \sqrt{\frac{c(u,v,\ell)}{d_G(u)}}|u,v,\ell\rangle.$$

*Then the quantum time complexity of estimating $C_{s,t}(G(x))$ to accuracy $\epsilon$ is*

$$\widetilde{\mathcal{O}}\left(\frac{1}{\epsilon^{3/2}}\frac{1}{\delta(G)^{1/2}}\sqrt{C_{s,t}(G(x)) \max_p J_{E(G)}(p)}\mathsf{U}\right),$$

*where $\delta(G)$ is the spectral gap of the symmetric Laplacian $L_G^{\mathrm{sym}}$ defined in Section 2.3.2.*

*Proof.* The algorithm in Theorem 92 requires a number of calls to a unitary $U(x, P_G)$ of order $\widetilde{\mathcal{O}}(\epsilon^{-3/2}\sqrt{C_{s,t}(G(x)) \max_p J_{E(G)}(p)})$ and a similar number of other elementary operations ([IJ19]). By [JK17] (generalizing [BR12]), for any $G$, $U(x, P_G)$ can be implemented in cost $\frac{1}{\sqrt{\delta(G)}}\mathsf{U}$. □

To prove Theorem 93, we first relate unit $st$-flows on $G$ to approximate positive witnesses. Intuitively, an approximate positive witness is an $st$-flow on $G$ that has energy as small as possible on edges in $E(G) \setminus E(G(x))$. Thus, we can upper bound the approximate positive witness size by the highest possible energy of any $st$-flow on $G$, which is always achieved by a flow that is an $st$-path.

**Claim 3.** Let $P_G$ be the span program of Definition 88. Then the positive error of $x$ in $P_G$ is

$$e_+(x, P_G) = \frac{1}{2} \min_\theta \{J_{E(G)\setminus E(G(x))}(\theta)\}, \tag{5.16}$$

where $\theta$ runs over unit $st$-flows on $G$. The min. error approximate positive witness size is

$$\tilde{w}_+(x, P_G) = \frac{1}{2} \min_\theta J_{E(G)}(\theta) \tag{5.17}$$

where $\theta$ runs over unit $st$-flows on $G$ such that $J_{E(G)\setminus E(G(x))}(\theta) = e_+(x, P_G)$.

*Proof.* The proof follows from the definition of min. error approximate positive witness size and the fact, proven in [JK17, Lemma 11], that every positive witness $|w\rangle$ corresponds to a unit $st$-flow $\theta$ over $E(G)$ such that $\||w\rangle\|^2 = \frac{1}{2} J_{E(G)}(\theta)$. It follows then that the negative error is half the energy of some flow through $E(G) \setminus E(G(x))$ and the min. error positive witness size is half the energy of a flow over $G$ that minimizes its energy over $E(G) \setminus E(G(x))$. □

We are now ready to prove Theorem 93.

*Proof of Theorem 93.* From Claim 3, we have that the min. error approximate positive witness corresponds to a particular unit $st$-flow over $G$. Since the unit $st$-flow over $G$ with maximum energy is a path from $s$ to $t$, — because splitting the flow reduces the energy — the result follows. □

## 5.4 Graph connectivity

Let $\text{CONN}_{G,X}$ be the problem of deciding, given $x \in X$, whether $G(x)$ is connected. That is:

$$\text{CONN}_{G,X} = \bigwedge_{\{u,v\}:u,v\in V(G)} uv\text{-CONN}_{G,X}. \tag{5.18}$$

Using the technique of converting logical AND into $st$-connectivity problems in series [NT95; JK17], we note that the above problem is equivalent to $n(n-1)/2$ $st$-connectivity problems in series, one for each pair of distinct vertices in $V(G)$. (The approach in Ref. [Āri16] is similar, but only looks

at $n-1$ instances — the pairs $s$ and $v$ for each $v \in V(G)$. Our approach is symmetrized over the vertices and thus makes a tight analysis simpler.)

More precisely, we define a graph $\mathcal{G}$ such that:

$$\begin{aligned} V(\mathcal{G}) &= V(G) \times \{\{u,v\} : u \neq v \in V(G)\} \\ E(\mathcal{G}) &= E(G) \times \{\{u,v\} : u \neq v \in V(G)\} \end{aligned} \tag{5.19}$$

where $\times$ denotes the Cartesian product, and $\{u,v\}$ is an extra label denoting that that edge or vertex is in the $\{u,v\}^{\text{th}}$ copy of the graph $G$ present as a subgraph in $\mathcal{G}$. Choose any labeling of the vertices from 1 to $n$ (with slight abuse of notation, we use $u$ both for the original vertex name and the label). We then label the vertex $(1,\{1,2\})$ as $s$ and the vertex $(n,\{n-1,n\})$ as $t$. Next identify vertices $(v,\{u,v\})$ and $(u,\{u,v+1\})$ if $u<v$ and $v<n$, and identify vertices $(v,\{u,v\})$ and $(u+1,\{u+1,u+2\})$ if $v=n$ and $u<n-1$. See Fig. 5.2 for an example of this construction.

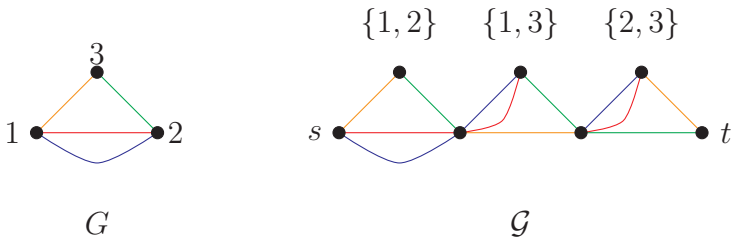



Figure 5.2: Example of how $\mathcal{G}$ is formed from a graph $G$. We have labeled the subgraphs of $\mathcal{G}$ according to the $st$-connectivity problems the subgraphs represent.

Finally, we define $\mathcal{G}(x)$ to be the subgraph of $\mathcal{G}$ with edges

$$E(\mathcal{G}(x)) = E(G(x)) \times \{\{u,v\} : u \neq v \in V(G)\}. \tag{5.20}$$

We can see that any $st$-path in $\mathcal{G}(x)$ must go through each of the copies of $G(x)$, meaning it must include, for each $\{u,v\}$, a $uv$-path through the copy of $G(x)$ labeled $\{u,v\}$. Thus, there is an $st$-path in $\mathcal{G}(x)$ if and only if $G(x)$ is connected.

We consider the span program $P_{\mathcal{G}}$, where $c(e)=1$ for all $e \in E(\mathcal{G})$. We will use $P_{\mathcal{G}}$ to solve $st$-connectivity on $\mathcal{G}(x)$. To analyse the resulting algorithm, we need to upper bound the negative and positive witness sizes $w_-(x,P_{\mathcal{G}}) = 2C_{s,t}(\mathcal{G}(x))$ and $w_+(x,P_{\mathcal{G}}) = \frac{1}{2}R_{s,t}(\mathcal{G}(x))$.

**Lemma 96.** *For any $x$ such that $G(x)$ is connected, let $R_{\text{avg}}(G(x))$ be the average resistance from Eq. (2.12). Then $w_+(x, P_{\mathcal{G}}) = \frac{n(n-1)}{2} R_{\text{avg}}(G(x))$.*

*Proof.* Using the rule that resistances in series add from Proposition 19, we have:

$$R_{s,t}(\mathcal{G}(x)) = \frac{1}{2} \sum_{u,v \in V(G)} R_{u,v}(G(x)) = n(n-1) R_{\text{avg}}(G(x)). \tag{5.21}$$

This is equal to $2w_+(x, P_{\mathcal{G}})$. □

Now we bound $C_{s,t}(\mathcal{G}(x))$, to prove the following:

**Lemma 97.** *Fix $\kappa > 1$, and suppose $G(x)$ has $\kappa$ connected components. Then if $G$ is a subgraph of a complete graph (that is, $G$ has at most one edge between any pair of vertices), we have $w_-(x, P_{\mathcal{G}}) = \mathcal{O}(1/\kappa)$. Otherwise, we have $w_-(x, P_{\mathcal{G}}) = \mathcal{O}(d_{\max}(G)/\sqrt{n\kappa})$.*

*Proof.* Using the rule for capacitors in series from Proposition 19, and accounting for double counting pairs of vertices, we have

$$\frac{1}{C_{s,t}(\mathcal{G}(x))} = \frac{1}{2} \sum_{\substack{s',t' \in V(G) \\ s' \neq t'}} \frac{1}{C_{s',t'}(G(x))}. \tag{5.22}$$

To put an upper bound on $C_{s,t}(\mathcal{G}(x))$, we can put an upper bound on each term $C_{s',t'}(G(x))$. To upper bound $C_{s',t'}(G(x))$, consider the following unit $s't'$-potential for $G(x)$. Set $\mathsf{V}(v) = 1$ for all $v$ in the same connected component as $s'$ in $G(x)$, set $\mathsf{V}(v) = 0$ for all $v$ in the same connected component as $t'$ in $G(x)$, and set $\mathsf{V}(v) = \nu$ for all other vertices in $G$. We now find the minimum unit potential energy of this $s't'$-potential (minimizing over $\nu$). This will be an upper bound on $C_{s',t'}(G(x))$ by Definition 18, since it is not necessarily the optimal choice to set all vertices not connected to $s'$ or $t'$ to have the same unit potential value.

We use Proposition 19 and the fact that our choice of unit $s't'$-potential effectively creates a graph with three vertices: one vertex corresponds to the connected component of $G(x)$ containing $s'$ (let $n_{s'}$ be the number of vertices in this component), one vertex corresponds to the connected component of $G(x)$ containing $t'$ (let $n_{t'}$ be the number of vertices in this component), and one vertex corresponds to all the other vertices in the graph (let $a$ be a

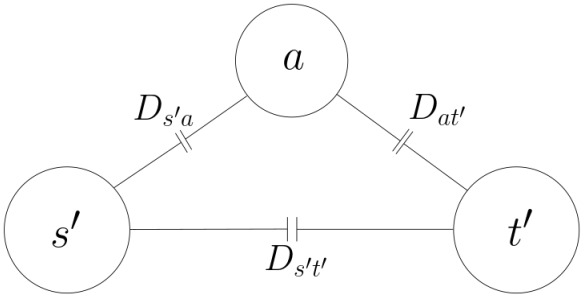



Figure 5.3: Our choice of unit $s't'$-potential effectively creates a graph with three vertices.

vertex in these components, and $n_a = n - n_{s'} - n_{t'}$ be the remaining number of vertices). Since these components are disconnected in $G(x)$ but connected in $G$, for any pair of vertices $u$ and $v$, let $D_u$ be the number of edges of $G$ coming out of the component of $G(x)$ containing $u$, and let $D_{uv}$ be the number of edges in $G \setminus G(x)$ between the components containing $u$ and $v$. Since all the weights have been chosen to be $c(e) = 1$, $D_{uv}$ is exactly the capacitance of the subgraph of $G(x)$ that contains only the component of $u$ and the component of $v$. See Figure 5.4 for a sketch of what is going on. Using the rules for calculating capacitance in series and parallel in Proposition 19, we have

$$C_{s',t'}(G(x)) \leq D_{s't'} + \left( \frac{1}{D_{s'} - D_{s't'}} + \frac{1}{D_{t'} - D_{s't'}} \right)^{-1} = \frac{D_{s'} D_{t'} - D_{s't'}^2}{D_{s'} + D_{t'} - 2D_{s't'}}. \tag{5.23}$$

Here, the inequality comes from the fact that our unit potential need not be optimal. Let us denote $s'$ and $t'$ being disconnected in $G(x)$ by $s' \not\sim t'$. Using Eq. (5.22), we have

$$\frac{1}{C_{s,t}(\mathcal{G}(x))} \geq \frac{1}{2} \sum_{\substack{s',t' \in V(G) \\ s' \not\sim t'}} \frac{D_{s'} + D_{t'} - 2D_{s't'}}{D_{s'} D_{t'} - D_{s't'}^2}. \tag{5.24}$$

Now the expression on the right-hand side of Eq. (5.24) depends only on which connected components $s'$ and $t'$ are in, so instead of summing over the vertices of $G$, we can instead sum over the $\kappa$ connected components of $G(x)$. Let $n_i$ be the number of vertices in the $i^{\text{th}}$ connected component. Then,

continuing from Eq. (5.24) we have:

$$\begin{aligned}\frac{1}{C_{s,t}(\mathcal{G}(x))} &\geq \frac{1}{2}\sum_{i,j\in[\kappa]:i\neq j} n_i n_j \frac{D_i + D_j - 2D_{ij}}{D_i D_j - D_{ij}^2} \\ &= \frac{1}{2}\sum_{i,j\in[\kappa]:i\neq j} n_i n_j \frac{(D_i - D_{ij}) + (D_j - D_{ij})}{D_i D_j - D_{ij}^2} \\ &= \frac{1}{2}\left(2\sum_{i,j\in[\kappa]:i\neq j} n_i n_j \frac{(D_i - D_{ij})}{D_i D_j - D_{ij}^2}\right) = \sum_{i,j\in[\kappa]:i\neq j} n_i n_j \frac{D_j - D_{ij}}{D_i D_j - D_{ij}^2}.\end{aligned} \tag{5.25}$$

First, consider the case when $G$ is a complete graph. In that case, $D_i - D_{ij}$, the number of edges leaving component $i$ and going to a component other than $i$ or $j$, is exactly $n_i(n - n_i - n_j)$, whereas the number of edges leaving component $i$ is $D_i = n_i(n - n_i)$. Finally, $D_{ij} = n_i n_j$ is the number of edges going from the $i^{\text{th}}$ component to the $j^{\text{th}}$ component. Thus, continuing from Eq. (5.25), we have:

$$\begin{aligned}\frac{1}{C_{s,t}(\mathcal{G}(x))} &\geq \sum_{i,j\in[\kappa]:i\neq j} n_i n_j \frac{n_i(n - n_i - n_j)}{n_i(n - n_i)n_j(n - n_j) - n_i^2 n_j^2} \\ &= \sum_{i,j\in[\kappa]:i\neq j} \frac{n_i(n - n_i - n_j)}{(n - n_i)(n - n_j) - n_i n_j} = \sum_{i,j\in[\kappa]:i\neq j} \frac{n_i(n - n_i - n_j)}{n^2 - n n_i - n n_j} \\ &= \sum_{i,j\in[\kappa]:i\neq j} \frac{n_i}{n} = \kappa - 1.\end{aligned} \tag{5.26}$$

Note that this upper bound on $C_{s,t}(\mathcal{G}(x))$ applies to *any* subgraph of a complete graph, since adding edges can only increase the capacitance. Thus, we have completed the first part of the proof.

We now continue with the more general case, where $G$ can have multi-edges. Let $d = d_{\max}(G)$. Continuing from Eq. (5.25), and using the fact that

for any component, we have:

$$\begin{aligned}
\frac{1}{C_{s,t}(\mathcal{G}(x))} &\geq \frac{1}{2}\sum_{i,j\in[\kappa]:i\neq j} n_i n_j \frac{D_i + D_j - 2D_{ij}}{D_i D_j - D_{ij}^2} \\
&= \frac{1}{2}\sum_{i,j\in[\kappa]:i\neq j} n_i n_j \frac{(\sqrt{D_i} - \sqrt{D_j})^2 + 2\sqrt{D_i D_j} - 2D_{ij}}{D_i D_j - D_{ij}^2} \\
&\geq \sum_{i,j\in[\kappa]:i\neq j} n_i n_j \frac{\sqrt{D_i D_j} - D_{ij}}{D_i D_j - D_{ij}^2} \geq \sum_{i,j\in[\kappa]:i\neq j} n_i n_j \frac{\sqrt{D_i D_j - D_{ij}^2}}{D_i D_j - D_{ij}^2} \\
&\geq \sum_{i,j\in[\kappa]:i\neq j} n_i n_j \frac{1}{\sqrt{D_i D_j}} \geq \sum_{i,j\in[\kappa]:i\neq j} \frac{\sqrt{n_i n_j}}{d} \geq \frac{1}{d}\sqrt{\sum_{i\in[\kappa]}\sum_{j\neq i} n_i n_j} \\
&= \frac{1}{d}\sqrt{\sum_{i\in[\kappa]} n_i(n - n_i)}. \qquad (5.27)
\end{aligned}$$

Above we used the fact that for any component, $D_i \leq d n_i$. The sum $\sum_{i\in[\kappa]} n_i^2$ is maximized when the $n_i$ are as far as possible from uniform. In this case, we have $n_i \geq 1$ for all $i$, so $\sum_{i\in[\kappa]} n_i^2 \leq (\kappa - 1) + (n - (\kappa - 1))^2$. Thus, continuing, we have

$$\begin{aligned}
\frac{1}{C_{s,t}(\mathcal{G}(x))} &\geq \frac{1}{d}\sqrt{n\sum_{i\in[\kappa]} n_i - \sum_{i\in[\kappa]} n_i^2} \\
&\geq \frac{1}{d}\sqrt{n^2 - (\kappa - 1) - n^2 - (\kappa - 1)^2 + 2n(\kappa - 1)} \\
&= \frac{1}{d}\sqrt{(2n - \kappa)(\kappa - 1)} \geq \frac{1}{d}\sqrt{n\kappa}. \qquad (5.28)
\end{aligned}$$

The result follows by Theorem 90, which says that $w_-(x, P_\mathcal{G}) = 2C_{s,t}(\mathcal{G}(x))$. □

Combining Lemmas 96 and 97 and Theorem 45, we have the following:

**Theorem 98.** *For any family of graphs $G$ such that $G$ has no multiedges, and $X \subseteq \{0,1\}^{E(G)}$ such that for all $x \in X$, if $G(x)$ is connected, $R_{\text{avg}}(G(x)) \leq R$, and if $G(x)$ is not connected, it has at least $\kappa$ components, the bounded error quantum query complexity of $\text{CONN}_{G,X}$ is $\mathcal{O}\left(n\sqrt{R/\kappa}\right)$.*

*For any family of connected graphs $G$ and $X \subseteq \{0,1\}^{E(G)}$ such that for all $x \in X$, if $G(x)$ is connected, $R_{\text{avg}}(G(x)) \leq R$, and if $G(x)$ is not connected, it has at least $\kappa$ components, the bounded error quantum query complexity of $\text{CONN}_{G,X}$ is $\mathcal{O}\left(n^{3/4}\sqrt{Rd_{\max}(G)}/\kappa^{1/4}\right)$.*

Similarly, we can show:

**Corollary 99.** *Let $\mathsf{U}$ be the cost of implementing the map*

$$|u\rangle|0\rangle \mapsto \sum_{v,\ell:(u,v,\ell)\in\vec{E}(G)} \frac{1}{\sqrt{d_G(u)}}|u,v,\ell\rangle.$$

*If $G$ has no multiedge, the quantum time complexity of $\text{CONN}_{G,X}$ is*

$$\mathcal{O}(n\sqrt{R/(\delta(\mathcal{G})\kappa)}\mathsf{U}).$$

*For any family of connected graphs $G$ and $X \subseteq \{0,1\}^{E(G)}$ such that for all $x \in X$, if $G(x)$ is connected, $R_{\text{avg}}(G(x)) \leq R$, and if $G(x)$ is not connected, it has at least $\kappa$ components, the quantum time complexity of $\text{CONN}_{G,X}$ is $\mathcal{O}\left(n^{3/4}\sqrt{Rd_{\max}(G)}\kappa^{-1/4}\delta(\mathcal{G})^{-1/2}\mathsf{U}\right)$.*

*Proof.* By the previous theorem, the span program algorithm of Theorem 45 makes $\mathcal{O}\left(n^{3/4}\sqrt{Rd_{\max}(G)}/\kappa^{1/4}\right)$ calls to a unitary $U(x, P_{\mathcal{G}})$. By [JK17] (generalizing [BR12]), for any $G$, $U(x, P_{\mathcal{G}})$ can be implemented in cost $\mathsf{U}/\sqrt{\delta(\mathcal{G})}$. □

## 5.5 Spectral algorithm for deciding connectivity

In this section, we will give alternative quantum algorithms for deciding connectivity. We begin by relating the span program for *st*-connectivity of Definition 88 to the Laplacian of a graph defined in Section 2.3.2. Equipped with this understanding, we present an algorithmic template, outlined in Algorithm 102, that produces spectral algorithms for graph connectivity. The template requires the instantiation of an input-independent initial state whose construction depends greatly on the parent graph $G$, in particular its structure and Laplacian spectrum. We want to remark that, by design,

these algorithms will work better the more we know about the spectrum of $L_G$ and the more well-behaved this is. Hence, the moniker *spectral*. Let us demonstrate what we mean by this.

Let $P_G = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$ be the span program for $st$-connectivity defined in Eq. (88). Note that only $|\tau\rangle$ depends on $s$ and $t$. our construction will make use of all parts of $P_G$ except $|\tau\rangle$. We let $A(x) = A\Pi_{\mathcal{H}(x)}$ and consider the product of $A(x)$ and $A(x)^T$.

$$\begin{aligned} A(x)A(x)^T &= \sum_{(u,v,\ell)\in\vec{E}(G(x))} c(u,v,\ell)(|u\rangle\langle u| - |u\rangle\langle v| - |v\rangle\langle u| + |v\rangle\langle v|) \\ &= \sum_{u\in[n]} 2d_G(u)|u\rangle\langle u| - 2\mathcal{A}_{G(x)} = 2(\mathcal{D}_{G(x)} - \mathcal{A}_{G(x)}) \\ &= 2L_{G(x)}, \end{aligned} \tag{5.29}$$

where, $L_{G(x)}$ is the Laplacian of $G(x)$ (see Section 2.3.2). By a similar computation to the one above, we have $AA^T = 2L_G$, where $G$ is the parent graph, upon which $A$ depends. Recall that for any $G$, the eigenvalues of $L_G$ lie in $[0, d_{\max}]$, with $|\mu\rangle = \frac{1}{\sqrt{n}}\sum_v |v\rangle$ as a 0-eigenvalue. In our case, since $G$ is assumed to be connected, $|\mu\rangle$ is the only 0-eigenvector of $L_G$, so $\mathrm{row}(L_G)$ is the orthogonal complement of $|\mu\rangle$. For any $x$, $L_{G(x)}$ also has $|\mu\rangle$ as a 0-eigenvalue, and if $G(x)$ is connected, this is the only 0-eigenvalue. In general, the dimension of the 0-eigenspace of $L_{G(x)}$ is the number of components of $G(x)$. Thus, Eq. (5.29) implies the following:

1. The multiset of nonzero eigenvalues of $L_G$ are exactly half of the squared singular values of $A$, and in particular, since no eigenvalue of $L_G$ can be larger than the maximum degree of $G$, $\sigma_{\max}(A) \leq \sqrt{2d_{\max}(G)}$.

2. The multiset of nonzero eigenvalues of $L_{G(x)}$ are exactly half the squared singular values of $A(x)$, and in particular, if $G(x)$ is connected, then $\sigma_{\min}(A(x)) = \sqrt{2\lambda_2(G(x))}$, where $\lambda_2(G(x))$ is the second smallest eigenvalue of $L_{G(x)}$, which is non-zero if and only if $G(x)$ is connected.

3. The support of $L_G$ is $\mathrm{col}(A)$, which is the orthogonal complement of the uniform vector $|\mu\rangle = \frac{1}{\sqrt{n}}\sum_v |v\rangle$.

For a particular span program $P$, and input $x$, we define the associated unitary

$$U(x,P) = (2\Pi_{\ker(A)} - I)(2\Pi_{\mathcal{H}(x)} - I). \tag{5.30}$$

This unitary is a close relative to the one used in Theorem 45 to decide a span program. Our algorithm will be based on the following connection between the connectivity of $G(x)$ and the presence of a 0-phase eigenvector of $U(x, P)$ in $\mathrm{row}(A) = \ker(A)^\perp$. On the one hand, we have:

**Lemma 100.** *$G(x)$ has $\kappa > 1$ components if and only if $\dim(\mathrm{row}(A) \cap \mathcal{H}(x)^\perp) = \kappa - 1$, i.e. there exists a $(\kappa-1)$-dimensional subspace of $\mathrm{row}(A)$ that is fixed by $U(x, P)$.*

*Proof.* ($\Rightarrow$). Let $G(x)$ be a subgraph of $G$ with $\kappa$ components. It is a well known fact that the number of components of a graph is equal to the number of 0-eigenvectors of its Laplacian, and that the uniform vector $|\mu\rangle$ is always a zero eigenvector of $L_{G(x)}$. Let $|\xi_i\rangle$, $i \in [\kappa - 1]$ be an orthonormal set of 0-eigenvectors of $L_{G(x)}$ orthogonal to $|\mu\rangle$. Since the eigenvectors of $L_{G(x)}$ are the left-singular vectors of $A(x)$ it follows that $\langle\xi_i|A(x) = \langle\xi_i|A\Pi_{\mathcal{H}(x)} = 0$. Define the vectors

$$|\psi_i\rangle := A^\dagger|\xi_i\rangle.$$

We have just seen that $\Pi_{\mathcal{H}(x)}|\psi_i\rangle = 0$, and since all $|\xi_i\rangle$ are orthogonal to $|\mu\rangle$, it follows that for all $i \in [\kappa - 1]$, $|\xi_i\rangle \in \mathrm{col}(A)$ because the columnspace of $A$ is $\mathrm{span}\{|\mu\rangle^\perp\}$ since $G$ is connected. Finally, we observe that the set $\{A^\dagger|\xi_i\rangle\}$ must be linearly independent because $A^\dagger$ is linear and full rank in its rowspace, which is $\mathrm{row}(A^\dagger) = \mathrm{col}(A)$. We conclude that the dimension of $\mathrm{row}(A) \cap \mathcal{H}(x)^\perp$ is at least $\kappa - 1$.

($\Leftarrow$). Let $\{|\psi_i\rangle : i \in [\kappa - 1]\}$ be an orthonormal basis of $\mathrm{row}(A) \cap \mathcal{H}(x)^\perp$. Consider the Laplacian of $G(x)$, $A(x)A^T(x) = 2L_{G(x)}$. Then, for $i \in [\kappa - 1]$ the vectors

$$|\xi_i\rangle := (A^+)^\dagger|\psi_i\rangle$$

are 0-eigenvectors of $L_{G(x)}$. Indeed, since $|\psi_i\rangle \in \mathcal{H}(x)^\perp \cap \mathrm{row}(A)$ we have

$$\langle\xi_i|L_{G(x)} = \langle\psi_i|A^+A\Pi_{\mathcal{H}(x)}A^T(x) = \langle\psi_i|\Pi_{\mathrm{row}(A)}\Pi_{\mathcal{H}(x)}A^T(x) = 0.$$

And since all the $|\psi_i\rangle$ are orthonormal vectors in the rowspace of $(A^+)^\dagger$ it follows that the $|\xi_i\rangle$ are $\kappa - 1$ linearly independent vectors of $\ker(L_{G(x)})$. In addition, the uniform vector $|\mu\rangle$ is always a 0-eigenvector of $L_{G(x)}$ orthogonal to the $|\xi_i\rangle$'s, meaning that the number of components of $G(x)$ is at least $\kappa$. This completes the proof. □

On the other hand, we show a lower bound for the first non-zero eigenphase of $U(x, P)$ when $G(x)$ is connected.

**Lemma 101.** *Let $P_G$ be the st-connectivity span program from Eq. (88). Then $\Delta(U(x,P)) \geq 2\sqrt{\lambda_2(G(x))/d_{\max}(G)}$.*

*Proof.* By [IJ19, Theorem 3.10], for every span program we have $\Delta(U(x,P)) \geq 2\sigma_{\min}(A(x))/\sigma_{\max}(A)$. Plugging in the values that we have derived for the *st*-connectivity span program yields the result. □

Thus, by Lemma 100, in order to determine if $G(x)$ is connected, it is sufficient to detect the presence of *any* 0-phase eigenvector of $U(x,P)$ on $\mathrm{row}(A)$. Let $\{|\psi_i\rangle\}_{i=1}^{n-1}$ be any basis for $\mathrm{row}(A)$, not necessarily orthogonal, and suppose we have access to an operation that generates a normalized version of the state

$$|\psi_{\text{init}}\rangle = \sum_{i=1}^{n-1} |i\rangle|\psi_i\rangle.$$

Such a basis is independent of the input, so we can certainly generate this state with 0 queries. We will later discuss the construction of such basis states for different graphs $G$. Let us, for now, not specify them.

**Algorithm 102.** *Assume there is a known constant $\lambda$ such that if $G(x)$ is connected, then $\lambda_2(G(x)) \geq \lambda$. Let $\{|\psi_i\rangle\}_i$ be some states that span the rowspace of $A$, whose choice determines the cost of the amplitude estimation step.*

1. *Prepare $|\psi_{\text{init}}\rangle = \sum_{i=1}^{n-1} \frac{1}{\sqrt{n-1}}|i\rangle|\psi_i\rangle$.*
2. *Perform the phase estimation of $U(x,P)$ (see Theorem 9) on the second register, to precision $\sqrt{\lambda/d_{\max}(G)}/2$, and accuracy $\epsilon$.*
3. *Use amplitude estimation (see Corollary 12) to determine if the amplitude on $|0\rangle$ in the phase register is 0, in which case, output "connected", or $> 0$, in which case, output "not connected."*

The algorithm works in the following manner. First, suppose that there is a non-trivial $\kappa - 1$-dimensional subspace $\mathcal{B} = \mathrm{span}\{|\phi_j\rangle : j \in [k-1]\}$ spanned by an orthonormal basis of 0-phase eigenvectors of $U(x,P)$ such that $\mathcal{B} \subseteq \mathrm{row}(A)$, and let $\Pi_{\mathcal{B}}$ be the orthonormal projector onto their span. In other words, suppose that $G(x)$ is not connected. By Theorem 9, for each $i$, the phase estimation step will map $|i\rangle(\Pi_{\mathcal{B}}|\psi_i\rangle)$ to $|i\rangle|0\rangle(\Pi_{\mathcal{B}}|\psi_i\rangle)$. Thus, the squared amplitude on $|0\rangle$ in the phase register will be at least:

$$\varepsilon := \frac{\|(I \otimes \Pi_{\mathcal{B}})|\psi_{\text{init}}\rangle\|^2}{\||\psi_{\text{init}}\rangle\|^2} = \frac{1}{\||\psi_{\text{init}}\rangle\|^2} \sum_{i=1}^{n-1} \|\Pi_{\mathcal{B}}|\psi_i\rangle\|^2 > 0. \tag{5.31}$$

On the other hand, suppose $G(x)$ is connected. Then, by Lemmas 100 and 101, all phases will be at least $\Delta(U(x,P)) \geq \sqrt{\lambda_2(G(x))/d_{\max}(G)}/2$, and there is no 0-phase eigenvector in $\text{row}(A)$, hence the phase register will have squared overlap at most $\epsilon$ with $|0\rangle$ if we choose precision $\Delta(U(x,P))/2$ for the phase estimation step.

Setting a phase estimation error of $\epsilon = \varepsilon/2$, we just need to distinguish between a squared amplitude of $\geq \varepsilon$ and a squared amplitude of $\leq \varepsilon/2$ on $|0\rangle$ in the phase register. Using Corollary 12, we can distinguish these two cases in $\frac{1}{\sqrt{\varepsilon}}$ calls to steps 1 and 2. Observe that we do not need the initial state to be an entangled state built from a basis of $\ker(A)^\perp$, but it is in our advantage to choose it this way since we are guaranteed to find $(\kappa - 1)$ 0-phase eigenvectors of $U(x,P)$ in $\ker(A)^\perp$ if $G(x)$ is not connected.

By Theorem 9, Step 2 can be implemented using $\sqrt{\frac{d_{\max}(G)}{\lambda}} \log \frac{1}{\varepsilon}$ calls to $U(x,P)$. Let $\mathsf{U}$ be the cost of implementing, for any $u \in \mathcal{V}$, the map

$$|u,0\rangle \mapsto \sum_{v,\ell:(u,v,\ell)\in\vec{E}(G)} \sqrt{\frac{c(u,v,\ell)}{d_G(u)}}|u,v,\ell\rangle, \tag{5.32}$$

which corresponds to one step of a quantum walk on $G$. Then, by Theorem 13 in [JK17], $U(x,P)$ can be implemented in time $O(\mathsf{U}/\sqrt{\delta(G)})$, where $\delta(G)$ is the spectral gap of a random walk over $G$. We thus get the following:

**Theorem 103.** *Fix $\lambda > 0$. Let $\mathsf{Init}$ denote the cost of generating the initial state $|\psi_{\text{init}}\rangle$, and $\mathsf{U}$ the cost of the quantum walk step in Eq. (5.32). Let $\varepsilon$ be as in Eq. (5.31) for any notion of cost. Then for any family of connected graphs $G$ and $X \subseteq \{0,1\}^{E(G)}$ such that for all $x \in X$, either $\lambda_2(G(x)) \geq \lambda$ or $G(x)$ is not connected, $\text{CONN}_{G,X}$ can be decided by a quantum algorithm with query complexity $\mathcal{O}\left(\frac{1}{\sqrt{\varepsilon}}\left(\sqrt{\frac{d_{\max}(G)}{\lambda}}\log\frac{1}{\varepsilon}\right)\right)$ and cost $\mathcal{O}\left(\frac{1}{\sqrt{\varepsilon}}\left(\mathsf{Init} + \sqrt{\frac{d_{\max}(G)}{\lambda\delta(G)}}\mathsf{U}\log\frac{1}{\varepsilon}\right)\right)$.*

In the next two sections, we will discuss particular implementations of this algorithm and bound their respective time complexities. That is, we will

give two different initial states and study the parameters $\mathsf{Init}$ and $\varepsilon$. If we only care about query complexity, we already have the following.

**Corollary 104.** *Fix any $\lambda > 0$ and $\kappa > 1$. For any family of connected graphs $G$ on $n$ vertices and $X \subseteq \{0,1\}^{E(G)}$ such that for all $x \in X$, either $\lambda_2(G(x)) \geq \lambda$ or $G(x)$ has at least $\kappa$ connected components, the bounded error quantum query complexity of $\mathrm{CONN}_{G,X}$ is $\mathcal{O}\left(\sqrt{\frac{nd_{\max}(G)}{\kappa\lambda}}\right)$.*

*Proof.* First, observe that by [IJ19, Lemma 3.1], $U(x,P)$ can be implemented with 2 queries.

Next, let $\{|\psi_i\rangle\}_{i=1}^{n-1}$ be any orthonormal basis for $\mathrm{row}(A)$. Then in $\mathsf{Init} = 0$ queries, we can generate the state

$$|\psi_{\text{init}}\rangle = \frac{1}{\sqrt{n-1}} \sum_{i=1}^{n-1} |i\rangle|\psi_i\rangle.$$

Then if there are $\kappa - 1$ orthonormal 0-phase vectors of $U(x,P)$ in $\mathrm{row}(A)$, $|\phi_1\rangle, \ldots, |\phi_{\kappa-1}\rangle$, setting $\Pi = \sum_{j=1}^{\kappa-1} |\phi_j\rangle\langle\phi_j|$, we have

$$\varepsilon = \|(I \otimes \Pi)|\psi_{\text{init}}\rangle\|^2 = \frac{1}{n-1} \sum_{j=1}^{\kappa-1} \sum_{i=1}^{n-1} |\langle\phi_j|\psi_i\rangle|^2 = \frac{\kappa-1}{n-1}.$$

Then the result follows. □

### 5.5.1 A construction for any $G$

In this section we will give a general construction of an initial state that, while not having optimal scaling, works for any graph $G$.

Let $|\mu\rangle = \frac{1}{\sqrt{n}} \sum_{u \in V(G)} |u\rangle$, and for $j \in [n-1]$ let $|\hat{j}\rangle = \frac{1}{\sqrt{n}} \sum_{u \in [n]} e^{2\pi i j u/n} |u\rangle$. Then it is easily checked that $|\hat{1}\rangle, \ldots, |\widehat{n-1}\rangle$ form an orthonormal basis of the columnspace of $A$ for a connected graph $G$. Let $|\psi_j\rangle = A^T|\hat{j}\rangle$. Then $\{|\psi_1\rangle, \ldots, |\psi_{n-1}\rangle\}$ is a basis for $\mathrm{row}(A)$, but it is not necessarily orthogonal unless the $|\hat{j}\rangle$'s themselves form an eigenbasis of $L_G$, (as in the case of Cayley graphs, discussed in Section 5.5.2), and in general, the $|\psi_j\rangle$ are not

normalized. We have

$$|\psi_j\rangle = A^T|\hat{j}\rangle = \sum_{(u,v,\ell)\in\vec{E}(G)} \sqrt{c(u,v,\ell)}|u,v,\ell\rangle(\langle u|\hat{j}\rangle - \langle v|\hat{j}\rangle)$$
$$= \sum_{(u,v,\ell)\in\vec{E}(G)} \sqrt{c(u,v,\ell)}\left(\frac{1}{\sqrt{n}}e^{2\pi iju/n} - \frac{1}{\sqrt{n}}e^{2\pi ijv/n}\right)|u,v,\ell\rangle, \quad (5.33)$$

from which we compute

$$\||\psi_j\rangle\|^2 = \left\|A^T|\hat{j}\rangle\right\|^2 = \frac{1}{n}\sum_{(u,v,\ell)\in\vec{E}(G)} c(u,v,\ell)\left|e^{2\pi iju/n} - e^{2\pi ijv/n}\right|^2$$
$$= \frac{1}{n}\sum_{(u,v,\ell)\in\vec{E}(G)} c(u,v,\ell)2\left(1-\cos\frac{2\pi j(v-u)}{n}\right). \quad (5.34)$$

For any graph, we can use as initial state a normalization of $\sum_{j=1}^{n-1}|j\rangle|\psi_j\rangle$. From Eq. (5.34), we have, using Lagrange's identity:

$$\sum_{j=1}^{n-1}\||\psi_j\rangle\|^2 = \frac{2}{n}\sum_{(u,v,\ell)\in\vec{E}(G)} c(u,v,\ell)\sum_{j=1}^{n-1}\left(1-\cos\frac{2\pi j(v-u)}{n}\right)$$
$$= \frac{2}{n}\sum_{(u,v,\ell)\in\vec{E}(G)} c(u,v,\ell)\left(n-1-\left(\frac{1}{2}\frac{\sin\left((n-\frac{1}{2})\frac{2\pi(v-u)}{n}\right)}{\sin\left(\frac{\pi(v-u)}{n}\right)}-1\right)\right)$$
$$= \frac{2}{n}\sum_{(u,v,\ell)\in\vec{E}(G)} c(u,v,\ell)\left(n-\frac{\sin\left(-\frac{\pi(v-u)}{n}\right)}{2\sin\left(\frac{\pi(v-u)}{n}\right)}\right)$$
$$= \frac{2}{n}\left(n+\frac{1}{2}\right)\sum_{u\in V}\sum_{v,\ell:(u,v,\ell)\in\vec{E}(G)} c(u,v,\ell)$$
$$= 2\left(1+\frac{1}{2n}\right)\sum_{u\in V} d_G(u) = 2\left(1+\frac{1}{2n}\right)nd_{\text{avg}}(G), \quad (5.35)$$

where $d_{\text{avg}} = d_{\text{avg}}(G)$ is the average weighted degree in $G$.

Define the initial state as the unit vector:

$$|\psi_{\text{init}}\rangle = \frac{\sum_{j=1}^{n-1}|j\rangle A^T|\hat{j}\rangle}{\sqrt{2(1+\frac{1}{2n})nd_{\text{avg}}}}. \quad (5.36)$$

Then we lower bound $\varepsilon$ in the case where $G(x)$ is not connected as follows:

**Lemma 105.** *Let $|\psi_{\text{init}}\rangle$ be as in Eq. (5.36), and suppose $G(x)$ has $\kappa > 1$ connected components. Let $\Pi$ be the projector onto a $(\kappa-1)$-dimensional subspace of $\text{row}(A)$ that is in the 0-phase space of $U(x,P)$. Then*

$$\varepsilon = \|(I \otimes \Pi)|\psi_{\text{init}}\rangle\|^2 \geq \frac{(\kappa-1)\lambda_2(G)}{(1+\frac{1}{2n})nd_{\text{avg}}}.$$

*Proof.* By Lemma 100, there exist orthonormal vectors $|\phi_1\rangle, \ldots, |\phi_{\kappa-1}\rangle \in \text{row}(A)$ that are fixed by $U(x,P)$. Let $\Pi = \sum_{i=1}^{\kappa-1} |\phi_i\rangle\langle\phi_i|$. We have:

$$\begin{aligned} \|(I \otimes \Pi)|\psi_{\text{init}}\rangle\|^2 &= \frac{\sum_{i=1}^{\kappa-1}\sum_{j=1}^{n-1} |\langle\phi_i|\psi_j\rangle|^2}{2(1+\frac{1}{2n})nd_{\text{avg}}} = \frac{\sum_{i=1}^{\kappa-1}\sum_{j=1}^{n-1} |\langle\phi_i|A^T|\hat{j}\rangle|^2}{2(1+\frac{1}{2n})nd_{\text{avg}}} \\ &= \frac{\sum_{i=1}^{\kappa-1} \|A|\phi_i\rangle\|^2}{2(1+\frac{1}{2n})nd_{\text{avg}}}. \end{aligned} \tag{5.37}$$

Since $|\phi_i\rangle \in \text{row}(A)$ for each $i$, $\|A|\phi_i\rangle\|^2 \geq \sigma_{\min}(A)^2$. Thus

$$\|(I \otimes \Pi)|\psi_{\text{init}}\rangle\|^2 \geq \frac{(\kappa-1)\sigma_{\min}(A)^2}{2(1+\frac{1}{2n})nd_{\text{avg}}} = \frac{(\kappa-1)2\lambda_2(G)}{2(1+\frac{1}{2n})nd_{\text{avg}}}. \tag{5.38}$$

Here we used the fact that $G$ is assumed to be connected, the second smallest eigenvalue of $2L_G$ is the smallest non-zero eigenvalue, so $\lambda_2(2L_G) = \lambda_2(AA^T) = \sigma_{\min}(A)^2$. □

We remark that although this initial state lends itself well to analysis, it might be a particularly bad choice of an initial state, because it ensures lower weight on lower eigenvalue eigenstates of $G$, which may have high overlap with the 0-eigenvalue eigenstates of $G(x)$.

We next describe how we can construct the initial state.

**Lemma 106.** *Let $\mathsf{S}$ be the cost of generating the stationary state of $G$*

$$|\pi\rangle := \sum_{u \in V} \sqrt{\frac{d_G(u)}{d_{\text{avg}}n}}|u\rangle. \tag{5.39}$$

*Let $\mathsf{U}$ be the cost of implementing a step of the quantum walk on $G$ as in Eq. (5.32), and let $\epsilon \in (0,1)$. Then the cost of implementing a map that, with probability at lest $1-\epsilon$, succeeds in mapping $|0\rangle \mapsto |\psi_{\text{init}}\rangle$ is*

$$\mathsf{Init} = \mathcal{O}\left((\mathsf{S} + \mathsf{U} + \log n)\log\frac{1}{\epsilon}\right).$$

*Proof.* We have:

$$\begin{aligned}|\psi_j\rangle &= A^T|\hat{j}\rangle = \sum_{(u,v,\ell)\in\vec{E}(G)} \sqrt{c(u,v,\ell)}\frac{1}{\sqrt{n}}(e^{2\pi iju/n} - e^{2\pi ijv/n})|u,v,\ell\rangle \\ &= \frac{1}{\sqrt{n}}\sum_{u\in V} e^{2\pi iju/n}|u\rangle \sum_{\substack{v,\ell:\\(u,v,\ell)\in\vec{E}(G)}} \sqrt{c(u,v,\ell)}|v,\ell\rangle \\ &\quad -\frac{1}{\sqrt{n}}\sum_{u\in V}|u\rangle \sum_{\substack{v,\ell:\\(u,v,\ell)\in\vec{E}(G)}} e^{2\pi ijv/n}\sqrt{c(u,v,\ell)}|v,\ell\rangle. \end{aligned} \tag{5.40}$$

We first note that we can generate the state $|\pi\rangle$ in cost $\mathsf{S}$ from which we can generate, for any $j \in [n]$,

$$|\pi\rangle \mapsto \sum_{u\in V} e^{2\pi iju/n}\sqrt{\frac{d_G(u)}{nd_{\text{avg}}}}|u\rangle$$

using a generalized $Z_n^j$ gate, which performs the map $|u\rangle \mapsto e^{2\pi iuj/n}|u\rangle$ with complexity $O(\log n)$. From this, with one step of the quantum walk, we can get

$$|\alpha_j\rangle = \frac{1}{\sqrt{nd_{\text{avg}}}}\sum_{u\in V} e^{2\pi iju/n}|u\rangle \left(\sum_{v,\ell:(u,v,\ell)\in\vec{E}(G)} \sqrt{c(u,v,\ell)}|v,\ell\rangle\right) \tag{5.41}$$

in cost $\mathsf{U}$. The total cost of constructing $|\alpha_j\rangle$ is $O(\mathsf{S}+\mathsf{U}+\log n)$.

Similarly, applying first one step of the quantum walk to $|\pi\rangle$ and then a generalized $Z_n^j$ gate on the second register, we get

$$|\beta_j\rangle = \frac{1}{\sqrt{nd_{\text{avg}}}}\sum_{u\in V}|u\rangle \left(\sum_{v,\ell:(u,v,\ell)\in\vec{E}(G)} e^{2\pi ijv/n}\sqrt{c(u,v,\ell)}|v,\ell\rangle\right), \tag{5.42}$$

for total cost $O(\mathsf{U}+\mathsf{S}+\log n)$. One can now see that $|\alpha_j\rangle - |\beta_j\rangle = \frac{1}{\sqrt{d_{\text{avg}}}}|\psi_j\rangle$.

To construct $|\psi_{\text{init}}\rangle$, generate the state:

$$\frac{1}{\sqrt{2(n-1)}}|0\rangle\sum_{j=1}^{n-1}|j\rangle|\alpha_j\rangle - \frac{1}{\sqrt{2(n-1)}}|1\rangle\sum_{j=1}^{n-1}|j\rangle|\beta_j\rangle.$$

This costs $\mathcal{O}(\mathsf{S}+\mathsf{U}+\log n)$. Next, apply a Hadamard gate to the first register to get:

$$\frac{1}{2\sqrt{n-1}}|0\rangle\sum_{j=1}^{n-1}|j\rangle(|\alpha_j\rangle-|\beta_j\rangle)+\frac{1}{2\sqrt{n-1}}|1\rangle\sum_{j=1}^{n-1}|j\rangle(|\alpha_j\rangle+|\beta_j\rangle)$$
$$=\frac{1}{2\sqrt{(n-1)d_{\text{avg}}}}|0\rangle\sum_{j=1}^{n-1}|j\rangle|\psi_j\rangle+\frac{1}{2\sqrt{n-1}}|1\rangle\sum_{j=1}^{n-1}|j\rangle(|\alpha_j\rangle+|\beta_j\rangle). \quad (5.43)$$

By Eq. (5.35), we have $\left\|\sum_{j=1}^{n-1}|j\rangle|\psi_j\rangle\right\| = \sqrt{2(1+1/(2n))nd_{\text{avg}}}$, so the amplitude on the $|0\rangle$ part of the state is at least $\frac{1}{\sqrt{2}}$. Thus, we can measure the first register, and post select on measuring $|0\rangle$ to obtain $|\psi_{\text{init}}\rangle$. With $\log\frac{1}{\epsilon}$ repetitions, we succeed with probability $1-\epsilon$. □

We can now give an upper bound on the complexity of deciding connectivity for any family of parent graphs $G$, in terms of the costs of constructing $|\psi_{\text{init}}\rangle$ and its overlap with the space $\mathcal{B}\subseteq\text{row}(\mathcal{A})$. We first note that it is reasonable to assume that these costs should be low in many natural cases. The cost $\mathsf{U}$ is the cost of implementing a step of a quantum walk on $G$, and note that $G$ is input-independent, so as long as it is sufficiently structured, this shouldn't be a particularly large cost. For example, if for any vertex in $G$, we can efficiently query its degree, and its $i^{\text{th}}$ neighbor for any $i$, then $\mathsf{U}=\mathcal{O}(\log n)$. Note that this is *not* the same as assuming we can efficiently query the $i^{\text{th}}$ neighbor of a vertex in $G(x)$, which is not an operation that we can easily implement in the edge-query input model. Similarly, we might hope that $\mathsf{S}$ is also $\mathcal{O}(\log n)$ in many cases of interest. Indeed, whenever $G$ is $d$-regular, it is simply the cost of generating the uniform superposition over all vertices.

**Theorem 107.** *Fix any $\lambda>0$ and $\kappa>1$. For any family of connected graphs $G$ on $n$ vertices and $X\subseteq\{0,1\}^{E(G)}$ such that for all $x\in X$, either $\lambda_2(G(x))\geq\lambda$, or $G(x)$ has at least $\kappa$ connected components, $\text{CONN}_{G,X}$ can be solved in bounded error in time*

$$\widetilde{\mathcal{O}}\left(\sqrt{\frac{nd_{\text{avg}}(G)}{\kappa\lambda_2(G)}}\left(\mathsf{S}+\sqrt{\frac{d_{\max}(G)}{\lambda\delta(G)}}\mathsf{U}\right)\right).$$

*Proof.* By Lemma 106, the complexity of generating $|\psi_{\text{init}}\rangle$ is $\mathsf{Init}=\mathcal{O}(\mathsf{S}+\mathsf{U}+\log n)$, and by Lemma 105, the initial state has overlap at least $\varepsilon=\Omega\left(\frac{\kappa\lambda_2(G)}{nd_{\text{avg}}}\right)$

with any unit vector in $\ker A(x) \cap \mathrm{row}(A)$. Plugging these values into the expression in Theorem 103 gives (neglecting polylogarithmic factors)

$$\mathcal{O}\left(\frac{1}{\sqrt{\varepsilon}}\left(\mathsf{Init} + \sqrt{\frac{d_{\max}(G)}{\lambda\delta(G)}}\mathsf{U}\log\frac{1}{\varepsilon}\right)\right) = \widetilde{\mathcal{O}}\left(\sqrt{\frac{nd_{\mathrm{avg}}}{\kappa\lambda_2(G)}}\left(\mathsf{S} + \sqrt{\frac{d_{\max}(G)}{\lambda\delta(G)}}\mathsf{U}\right)\right). \qquad \square$$

### 5.5.2 An algorithm for Cayley graphs

When the parent graph $G$ is a Cayley graph for a finite Abelian group, we can use the extra structure to construct an orthonormal basis of $\mathrm{row}(A)$. Cayley graphs are graphs that encode the algebraic structures of groups. They are named after the British mathematician Arthur Cayley (1821-1895), who pioneered the study of groups as abstract objects satisfying binary relations.

**Definition 108** (Cayley graphs)**.** Let $\Gamma$ be a finite abelian group, and let $S$ be a symmetric subset of $\Gamma$, i.e. $s \in S \Leftrightarrow -s \in S$, not containing the identity element. The Cayley graph $\mathrm{Cay}(\Gamma, S)$ is the graph that has vertex set $V(\mathsf{Cay}(\Gamma, S)) = \Gamma$ and edge set $E(\mathrm{Cay}(\Gamma, S)) = \{\{a, b\} : a - b \in S\}$.

Cayley graphs are of special interest to us because of their extensive relation to expander graphs [Chu97; HLW06] and the fact that it is possible to compute many graph properties using the algebraic properties of the underlying group. In particular, we will be interested in computing the eigenvalues and eigenvectors of the Laplacian.

**Lemma 109** ([Bol13])**.** *Let $\Gamma$ be a finite abelian group $\Gamma = \mathbb{Z}/(m_1) \times \cdots \times \mathbb{Z}/(m_k)$. Let $G = \mathsf{Cay}(\Gamma, S)$ be a Cayley graph, with $n = |\Gamma|$ and $d = |S|$. For every element $g \in \Gamma$, let $\chi_g : \Gamma \to \mathbb{C}$ be the character function defined as $\chi_g(s) = \omega_{m_1}^{g_1 s_1} \ldots \omega_{m_k}^{g_k s_k}$, where $\omega_m = e^{2\pi i/m}$ for every integer $m$. Then, for every $g \in \Gamma$, the vector*

$$|\hat{g}\rangle = \frac{1}{\sqrt{n}} \sum_{h \in \Gamma} \chi_g(h)|h\rangle. \tag{5.44}$$

*is an eigenvector of the Laplacian of $G$ with eigenvalue $\lambda_g = d - \sum_{s \in S} \chi_g(s)$.*

For $g \neq 0$, these are also the left-singular vectors of $A$ because $L_G = AA^T$. Most importantly, the vectors $A^T|\hat{g}\rangle$ for $g \neq 0$ are proportional to the right-

singular vectors of $A$ and form an orthogonal basis of $\mathrm{row}(A)$. We define

$$|\psi_g\rangle = A^T|\hat{g}\rangle = \sum_{u\in\Gamma}\sum_{v:v-u\in S}\sum_{h\in\Gamma}\frac{\chi_g(h)}{\sqrt{n}}|u,v\rangle(\langle u| - \langle v|)|h\rangle$$

$$= \sum_{u\in\Gamma}\sum_{v:v-u\in S}\frac{1}{\sqrt{n}}(\chi_g(u)-\chi_g(v))|u,v\rangle \tag{5.45}$$

$$= \sum_{u\in\Gamma}\frac{\chi_g(u)}{\sqrt{n}}\sum_{s\in S}(1-\chi_g(s))|u,u+s\rangle. \tag{5.46}$$

We have $\||\psi_g\rangle\|^2 = \left\|A^T|\hat{g}\rangle\right\|^2 = 2\lambda_g$, where $\lambda_g$ is the eigenvalue of $L_G$ associated with $|\hat{g}\rangle$. Now we define the initial state as:

$$|\psi_{\text{init}}\rangle = \frac{1}{\sqrt{n-1}}\sum_{g\in\Gamma\setminus\{0\}}\frac{1}{\sqrt{2\lambda_g}}|g\rangle|\psi_g\rangle. \tag{5.47}$$

We first lower bound the overlap of the initial state with the 0-phase space of $U(x,P)$ when $G(x)$ is not connected.

**Lemma 110.** *Suppose $G(x)$ has at least $\kappa > 1$ components. Let $|\psi_{\text{init}}\rangle$ be as in Eq. (5.47), and let $\mathcal{B} = \mathrm{span}\{|\phi_i\rangle : i \in [\kappa-1]\}$ be spanned by the orthonormal 0-phase vectors of $U(x,P)$ in $\mathrm{row}(A)$. Let $\Pi = \sum_{i=1}^{\kappa-1}|\phi_i\rangle\langle\phi_i|$. Then*

$$\|(I\otimes\Pi)|\psi_{\text{init}}\rangle\|^2 \geq \frac{\kappa-1}{n-1}.$$

*Proof.* Let $|\overline{\psi}_g\rangle = |\psi_g\rangle/\sqrt{2\lambda_g}$. Then $\{|\overline{\psi}_g\rangle\}_{g\neq 0}$ is an orthonormal basis for $\mathrm{row}(A)$. We have

$$\|(I\otimes\Pi)|\psi_{\text{init}}\rangle\|^2 = \frac{1}{n-1}\sum_{i=1}^{\kappa-1}\sum_{g\in\Gamma\setminus\{0\}}|\langle\phi_i|\overline{\psi}_g\rangle|^2 = \frac{1}{n-1}\sum_{i=1}^{\kappa-1}1 = \frac{\kappa-1}{n-1}. \tag{5.48}$$

□

Next, we give an upper bound on the time complexity of constructing the initial state. We will use the following fact:

**Claim 4** (See, for example, [Bol13])**.** Let $G$ be any connected graph, with non-zero eigenvalues $\lambda_2,\ldots,\lambda_n$. Then

$$R_{\text{avg}}(G) = \frac{1}{n-1}\sum_{i=2}^{n}\frac{1}{\lambda_i}.$$

**Lemma 111.** *Let* $\mathsf{U}$ *be the cost of implementing a step of the quantum walk on* $\mathsf{Cay}(\Gamma, S)$*. Let* $\Lambda$ *be the cost of implementing, for* $g \in \Gamma$*,* $|g\rangle|0\rangle \mapsto |g\rangle|\lambda_g\rangle$*. Then the cost of generating the state* $|\psi_{\text{init}}\rangle$ *from Eq. (5.47) with success probability* $1-\epsilon$ *is*

$$\mathcal{O}\left(\left(\sqrt{R_{\text{avg}}(G)d}\,(\mathsf{U}+\log n)+\Lambda\sqrt{\frac{d}{\lambda_2(G)}}\right)\log\frac{1}{\epsilon}\right).$$

*Proof.* The proof is similar to that of Lemma 106. Using a Fourier transform over $\Gamma$, we can generate the state

$$|g\rangle \mapsto \frac{1}{\sqrt{n}}\sum_{u\in\Gamma}\chi_g(u)|u\rangle \tag{5.49}$$

for any $g \in \Gamma$ in time $\mathcal{O}(\log n)$. We can then generate $\sum_{s\in S}\frac{1}{\sqrt{d}}|s\rangle$ in time $\mathsf{U}$ because $\{|s\rangle : s \in S\}$ are the neighbors of the identity of $\Gamma$ by definition of $\mathsf{Cay}(\Gamma, S)$. Now, perform the addition over $\Gamma$ map

$$\frac{1}{\sqrt{n}}\sum_{u\in\Gamma}\chi_g(u)|u\rangle\sum_{s\in S}\frac{1}{\sqrt{d}}|s\rangle \mapsto \frac{1}{\sqrt{dn}}\sum_{u\in\Gamma}\chi_g(u)|u\rangle\sum_{s\in S}|u+s\rangle =: |\alpha_g\rangle \tag{5.50}$$

for a total complexity of $\mathcal{O}(\log n + \mathsf{U})$ to generate $|\alpha_g\rangle$.

Alternatively, we can use the generalized $Z_\Gamma^g$ gate, which maps $|s\rangle$ to $\chi_g(s)|s\rangle$ in time $\mathcal{O}(\log n)$ to get

$$\begin{aligned}\frac{1}{\sqrt{nd}}\sum_{u\in\Gamma}\chi_g(u)|u\rangle\sum_{s\in S}|s\rangle &\mapsto \frac{1}{\sqrt{nd}}\sum_{u\in\Gamma}\chi_g(u)|u\rangle\sum_{s\in S}\chi_g(s)|s\rangle \\ &\mapsto \frac{1}{\sqrt{nd}}\sum_{u\in\Gamma}\chi_g(u)|u\rangle\sum_{s\in S}\chi_g(s)|u+s\rangle =: |\beta_g\rangle\end{aligned} \tag{5.51}$$

for a total complexity of $\mathcal{O}(\log n + \mathsf{U})$ to generate $|\beta_g\rangle$. Observe that $|\alpha_g\rangle - |\beta_g\rangle = \frac{1}{\sqrt{d}}|\psi_g\rangle$.

Now to construct $|\psi_{\text{init}}\rangle$, we first construct $\sum_{g\in\Gamma\setminus\{0\}}\frac{1}{\sqrt{\lambda_g}}|g\rangle$ as follows. We first generate:

$$\sum_{g\in\Gamma\setminus\{0\}}\frac{1}{\sqrt{n-1}}|g\rangle|\lambda_g\rangle, \tag{5.52}$$

in cost $\Lambda$. Next, we map this to:

$$\sum_{g\in\Gamma\setminus\{0\}} \frac{1}{\sqrt{n-1}}|g\rangle|\lambda_g\rangle \left(\sqrt{\frac{\lambda_2(G)}{\lambda_g}}|0\rangle + \sqrt{1-\frac{\lambda_2(G)}{\lambda_g}}|1\rangle\right). \tag{5.53}$$

We can perform this map because, for all $g \in \Gamma \setminus \{0\}$, $\lambda_2(G) \leq \lambda_g$. We uncompute $|\lambda_g\rangle$, and then do amplitude amplification on $|0\rangle$ in the last register to get the desired state. The squared amplitude on $|0\rangle$ is:

$$\left\| \sum_{g\in\Gamma\setminus\{0\}} \frac{1}{\sqrt{n-1}} \sqrt{\frac{\lambda_2(G)}{\lambda_g}}|g\rangle \right\|^2 = \frac{\lambda_2(G)}{n-1} \sum_{g\in\Gamma\setminus\{0\}} \frac{1}{\lambda_g} = \lambda_2(G) R_{\text{avg}}(G). \tag{5.54}$$

So we can generate the normalized state

$$\frac{1}{\sqrt{(n-1)R_{\text{avg}}(G)}} \sum_{g\in\Gamma\setminus\{0\}} \frac{1}{\sqrt{\lambda_g}}|g\rangle \tag{5.55}$$

with constant success probability in time complexity $O(\Lambda/\sqrt{\lambda_2(G)R_{\text{avg}}(G)})$. Thus far, we have shown how to map $|g\rangle$ to $|\alpha_g\rangle$ and $|\beta_g\rangle$, and how to generate a superposition over $|g\rangle$ with the right weights. Now we combine the two to obtain:

$$\frac{1}{\sqrt{2(n-1)R_{\text{avg}}}} \left( |0\rangle \sum_{g\in\Gamma\setminus\{0\}} \frac{1}{\sqrt{\lambda_g}}|g\rangle|\alpha_g\rangle - |1\rangle \sum_{g\in\Gamma\setminus\{0\}} \frac{1}{\sqrt{\lambda_g}}|g\rangle|\beta_g\rangle \right), \tag{5.56}$$

and we follow up by applying a Hadamard gate to the first qubit to get

$$\begin{aligned} &\frac{1}{2\sqrt{(n-1)R_{\text{avg}}}} \left( |0\rangle \sum_{g\in\Gamma\setminus\{0\}} \frac{1}{\sqrt{\lambda_g}}|g\rangle(|\alpha_g\rangle - |\beta_g\rangle) \right. \\ &\qquad \left. + |1\rangle \sum_{g\in\Gamma\setminus\{0\}} \frac{1}{\sqrt{\lambda_g}}|g\rangle(|\alpha_g\rangle + |\beta_g\rangle) \right) \\ &= \frac{1}{2\sqrt{(n-1)R_{\text{avg}}d}} \left( |0\rangle \sum_{g\in\Gamma\setminus\{0\}} \frac{1}{\sqrt{\lambda_g}}|g\rangle|\psi_g\rangle \right. \\ &\qquad \left. + |1\rangle \sum_{g\in\Gamma\setminus\{0\}} \frac{1}{\sqrt{\lambda_g}}|g\rangle(|\alpha_g\rangle + |\beta_g\rangle) \right). \end{aligned} \tag{5.57}$$

The total cost to make one copy of this state with constant success probability is $\mathcal{O}(\mathsf{U}+\log n+\mathsf{\Lambda}/\sqrt{\lambda_2(G)R_{\text{avg}}})$. We can then get $|\psi_{\text{init}}\rangle$ by doing amplitude amplification on the $|0\rangle$ part of this state. The amplitude on the $|0\rangle$ part of the state is given by

$$\left\| \frac{1}{2\sqrt{(n-1)R_{\text{avg}}d}} \sum_{g\in\Gamma\setminus\{0\}} \frac{1}{\sqrt{\lambda_g}}|g\rangle|\psi_g\rangle \right\| = \frac{1}{2\sqrt{R_{\text{avg}}d}},$$

so using $\mathcal{O}\left(\sqrt{R_{\text{avg}}(G)d}\right)$ rounds of amplitude amplification is sufficient to generate $|\psi_{\text{init}}\rangle$ with constant probability, for a total cost of (neglecting constants):

$$\begin{aligned} &\sqrt{R_{\text{avg}}(G)d}\left(\mathsf{U}+\log n+\frac{\mathsf{\Lambda}}{\sqrt{\lambda_2(G)R_{\text{avg}}(G)}}\right) \\ &= \sqrt{R_{\text{avg}}(G)d}\,(\mathsf{U}+\log n)+\mathsf{\Lambda}\sqrt{\frac{d}{\lambda_2(G)}}. \end{aligned} \tag{5.58}$$

We can amplify this to success probability $1-\epsilon$ at the cost of a $\log(1/\epsilon)$ multiplicative factor. □

As a quick corollary to these lemmas we have the following:

**Theorem 112.** *Fix any $\lambda > 0$ and $\kappa > 1$. For any family of connected graphs $G$ on $n$ vertices such that each $G$ is a degree-$d$ Cayley graph over an Abelian group, and $X \subseteq \{0,1\}^{E(G)}$ such that for all $x \in X$, either $\lambda_2(G(x)) \geq \lambda$ or $G(x)$ has at least $\kappa$ connected components, $\text{CONN}_{G,X}$ can be solved in bounded error at cost*

$$\widetilde{\mathcal{O}}\left(\sqrt{\frac{nd}{\kappa\lambda_2(G)}}\left(\sqrt{\frac{d}{\lambda}}\mathsf{U}+\mathsf{\Lambda}\right)\right).$$

*Proof.* Combining Lemma 111 and Lemma 110 with Theorem 103, we get complexity:

$$\widetilde{\mathcal{O}}\left(\sqrt{\frac{n}{\kappa}}\left(\sqrt{dR_{\text{avg}}(G)}\mathsf{U}+\sqrt{\frac{d}{\lambda_2(G)}}\mathsf{\Lambda}+\sqrt{\frac{d}{\lambda\delta(G)}}\mathsf{U}\right)\right). \tag{5.59}$$

Since $G$ is $d$-regular, we have $\delta(G) = \lambda_2(G)/d$. By Claim 4, we can see that $R_{\text{avg}}(G) \leq \frac{1}{\lambda_2(G)}$. The claim follows. □

We now look at specific examples where it is particularly efficient to compute $\lambda_g$, as well as prepare a step of the walk $\sum_{s\in S}|s\rangle$. We first consider the complete graph on $n$ vertices, in which $\Gamma = \mathbb{Z}_n$, and $S = \Gamma \setminus \{0\}$.

**Corollary 113.** *Fix any $\lambda > 0$, and integer $\kappa > 1$ and let $G$ be the complete graph. Let $X \subseteq E(G)$ be such that for all $x \in X$, either $\lambda_2(G(x)) \geq \lambda$, or $G(x)$ has at least $\kappa$ components. Then $\mathrm{CONN}_{G,X}$ can be solved in bounded error in time*

$$\widetilde{\mathcal{O}}\left(\frac{n}{\sqrt{\kappa\lambda}}\right).$$

*Proof.* It is easily verified that for $G$ a complete graph, $L_G = (n-1)I-(J-I)$, where $J$ is the all-ones matrix, so the eigenvalues consist of a single 0, and $n$ with multiplicity $n-1$ — that is, all non-zero eigenvalues are $n$. Thus, the mapping $|g\rangle \mapsto |g\rangle|\lambda_g\rangle = |g\rangle|n\rangle$ can be implemented trivially in $\mathcal{O}(\log n)$ complexity.

Next, we can generate the state $\sum_{s\in S}\frac{1}{\sqrt{d}}|s\rangle = \sum_{s=1}^{n-1}\frac{1}{\sqrt{n-1}}|s\rangle$ in complexity $\mathsf{S} = \mathcal{O}(\log n)$. Then the result follows from Theorem 112. □

Next, we consider the Boolean hypercube, in which $\Gamma = \mathbb{Z}_2^d$, so $n = 2^d$, and $S = \{e_i\}_{i=1}^d$, where $e_i$ is 0 everywhere except the $i^{\text{th}}$ entry, which is 1.

**Corollary 114.** *Fix any $\lambda > 0$, and integer $\kappa > 1$ and let $G$ be the Boolean hypercube on $n = 2^d$ vertices. Let $X \subseteq E(G)$ be such that for all $x \in X$, either $\lambda_2(G(x)) \geq \lambda$, or $G(x)$ has at least $\kappa$ components. Then $\mathrm{CONN}_{G,X}$ can be solved in bounded error in time*

$$\widetilde{O}\left(\sqrt{\frac{n}{\kappa\lambda}}\right).$$

*Proof.* The eigenvalues of the Boolean hypercube are well known to be $\lambda_g = 2|g|$, for $g \in \mathbb{Z}_2^d$, where $|g|$ denotes the Hamming weight of $g$. Thus, the map $|g\rangle \mapsto |g\rangle|\lambda_g\rangle$ can be implemented in cost $\mathcal{O}(\log n)$. Finally, the state $\sum_{s\in S}|s\rangle = \sum_{i=1}^d|e_i\rangle$ can be generated in time $\mathcal{O}(\log n)$. Then by Theorem 112, the time complexity is (neglecting polylog factors):

$$\sqrt{\frac{nd}{\kappa\lambda}} = \widetilde{\mathcal{O}}\left(\sqrt{\frac{n}{\kappa\lambda}}\right). \tag{5.60}$$

□

### 5.5.3 Estimating the connectivity when $G = K_n$

For the remainder of this section, let $G$ be the complete graph on $n$ vertices, $K_n$. In that case, we give an algorithm for estimating $\lambda_2(G(x))$when $G(x)$ is connected, in addition to the one that decides if $G(x)$ is connected. The idea is the following. Let us assume that we know $G(x)$ is connected, and let $\Delta(U(x,P))$ denote the smallest eigenphase of $U(x,P)$, which we want to estimate. First, we relate $\Delta(U(x,P))$ to $\lambda_2(G(x))$ when $G = K_n$ in Lemma 115. The algorithm that we outline now will estimate $\Delta(U(x,P))$.

Assume, for simplicity, that we run the phase estimation algorithm to precision $\theta$ and no error on a state $|\varphi\rangle$. If the amplitude on $|0\rangle$ in the phase register is non-zero, that means that $|\varphi\rangle$ has support on eigenvectors with eigenphase $|\theta_j| \leq \theta$, which can only happen if such eigenvectors exist. We conclude that $\Delta(U(x,P)) \leq \theta$.

If, on the other hand, the amplitude on $|0\rangle$ in the phase register is zero, that means simply that $|\varphi\rangle$ has no overlap with eigenvectors of $U(x,P)$ with phases smaller than $\theta$. What it does not mean is that all eigenphases are above $\theta$. For that, we would need to show that $|\varphi\rangle$ has some overlap with the $\Delta(U(x,P))$-phase eigenvectors of $U(x,P)$. This is exactly what we do in Lemma 5, where we show that there is a vector in row$(A)$ that is in the $\pm\Delta(U(x,P))$-phase eigenspace of $U(x,P)$. Therefore, we can choose as initial state of the phase estimation procedure the state

$$|\psi_{\text{init}}\rangle = \frac{1}{\sqrt{n-1}} \sum_{i=1}^{n-1} |i\rangle|\psi_i\rangle$$

where $\{|\psi_i\rangle\}$ are an orthonormal basis of row$(A)$. With such state, we can determine an interval for $\Delta(U(x,P))$ by performing consecutive runs of phase estimation with different precision. Naturally, the actual algorithm will require amplitude amplification after every phase estimation run to really distinguish zero from non-zero amplitudes, and on top of that, we will have to deal with the error of phase estimation. We begin by formally proving the connection between $\Delta(U(x,P))$ and $\lambda_2(G(x))$ and then formally present the algorithm for estimating $\Delta(U(x,P))$ and analyse its correctness and complexity.

#### Connection between $\lambda_2(G(x))$ and $\Delta(U(x,P))$

We use Jordan's Lemma, which we restate for convenience.

**Lemma.** *Let $A, B$ be two subspaces of $\mathcal{H}$ and let $U = (2\Pi_A - I)(2\Pi_B - I)$ be a unitary with discriminant $D = \Pi_A \Pi_B$. Let $D = \sum_{j=1}^{d} \cos\varphi_j |\theta_j\rangle\langle\psi_j|$ be a singular value decomposition with $\varphi_j \in [0, \pi/2]$ for all $j$. Then the vectors $|\theta_j\rangle - e^{\pm i\varphi_j}|\psi_j\rangle$ are eigenvectors of $U$ with eigenvalue $e^{\pm i2\varphi_j}$ respectively. Furthermore, the $(+1)$-eigenspace of $U$ is $(A \cap B) \oplus (A^\perp \cap B^\perp)$ and the $(-1)$-eigenspace of $U$ is $(A \cap B^\perp) \oplus (A^\perp \cap B)$.*

We derive several consequences of this lemma, and specialize them to our particular setting.

**Lemma 115.** *Let $G$ be a complete graph on $n$ vertices, and $x \in \{0,1\}^N$ define a subgraph $G(x)$. Then $\lambda_2(G(x)) = n \sin^2(\Delta(U(x,P))/2)$.*

*Proof.* It is well known that for connected graphs the uniform vector $|\mu\rangle = \frac{1}{\sqrt{n}} \sum_{u\in[n]} |u\rangle$ is the only 0-eigenvector of $L_G$. Moreover, for the complete graph we have

$$L_G = (n-1)I - (J - I) = nI - J = nI - n|\mu\rangle\langle\mu| = n\sum_{i=1}^{n-1} |b_i\rangle\langle b_i|,$$

where $\{|b_i\rangle\}_{i=1}^{n-1}$ is any orthonormal basis for $\mathrm{span}\{|\mu\rangle\}^\perp$, which is $\mathrm{col}(A)$. Since $2L_G = AA^T$, the eigenvectors of $L_G$ are left-singular vectors for $A$ and the singular values of $A$ are $\sqrt{2n}$. This implies that the following is a singular value decomposition of $A$:

$$A = \sum_{i=1}^{n-1} \sqrt{2n} |b_i\rangle\langle\psi_i|,$$

for some orthonormal basis for $\mathrm{row}(A)$, $\{|\psi_i\rangle\}_{i=1}^{n-1}$ of right-singular vectors of $A$. Thus

$$A^+ = \sum_{i=1}^{n-1} \frac{1}{\sqrt{2n}} |\psi_i\rangle\langle b_i|.$$

Next, note that since $L_G|\mu\rangle = 0$ for any $G$, and $2L_{G(x)} = A(x)A(x)^T$, we have $A(x)^T|\mu\rangle = 0$, so the columnspace of $A(x)$ is in $\mathrm{span}\{|\mu\rangle\}^\perp$, and in particular, if $G(x)$ is connected, it is exactly $\mathrm{span}\{|\mu\rangle\}^\perp$. The basis $\{|b_i\rangle\}_{i=1}^{n-1}$ can be chosen to be any basis of $|\mu\rangle^\perp$, so let's choose it to be the left singular basis of $A(x)$. That is, there exist $|\phi_i\rangle$ and $\sigma_i$ such that the following is a singular value decomposition of $A(x)$.

$$A(x) = \sum_{i=1}^{n-1} \sigma_i |b_i\rangle\langle\phi_i|$$

Consider the discriminant of $U(x,P)$, $D = \Pi_{\ker(A)}\Pi_{\mathcal{H}(x)}$. Since $A(x) = A\Pi_{\mathcal{H}(x)}$, and $A^+A = Pi_{\ker(A)}$ we have

$$D = A^+A(x) = \sum_{i=1}^{n-1} \frac{\sigma_i}{\sqrt{2n}} |\psi_i\rangle\langle\phi_i|.$$

Since $2L_{G(x)} = A(x)A(x)^T$, the $\sigma_i$ are just the square roots of twice the nonzero eigenvalues $\lambda_2, \ldots, \lambda_n$ of $L_{G(x)}$, so the singular values of $D$ are

$$\left\{ \sqrt{\frac{2\lambda_2}{2n}}, \ldots, \sqrt{\frac{2\lambda_n}{2n}} \right\}.$$

We conclude that $\sigma_{\min}(D) = \sqrt{\frac{\lambda_2(G(x))}{n}}$, which, combined with Corollary 2 gives $\lambda_2(G(x)) = n\sin^2(\Delta(U(x,P))/2)$. □

Another consequence of Lemma 1, proven in Lemma 3, is that there is a vector in row$(A)$ in the span of the $\pm\Delta(U(x,P))$-phase eigenvectors of $U(x,P)$.

**Claim 5.** Let $U = (2\Pi_{\ker(A)} - I)(2\Pi_{\mathcal{H}(x)} - I)$, and let $\cos\theta_j$ be the singular values of $D = \Pi_{\ker(A)}\Pi_{\mathcal{H}(x)}$. Then for every $j \in \text{rank}(D)$ there exists a a vector $|u_j\rangle$ in row$(A)$ such that $|u_j\rangle \in E_{|2\theta_j|}$. In particular, if $|\Delta_\pm\rangle$ are $\pm\Delta(U(x,P))$-phase eigenvectors of $U(x,P)$, then there exists a vector $|u\rangle$ in row$(A)$ such that $|u\rangle \in E_{|\Delta|} = \text{span}\{|\Delta_+\rangle, |\Delta_-\rangle\}$.

### Algorithm for estimating $\Delta(U(x,P))$

We will actually estimate the value $\tau = \Delta(U(x,P))/\pi$, getting an estimate in $[0,1]$, which we will then transform into an estimate of $\lambda_2(G(x))$. At every iteration, $c$ will denote a lower bound for $\tau$ and $C$ will denote the current upper bound. At the beginning of the algorithm we have $c = 0$, $C = 1$, and every iteration will result in updating either $C$ or $c$ in such a manner that the new interval for $\tau$ is reduce by a fraction of $2/3$. Let $\{|\psi_i\rangle\}_{i=1}^{n-1}$ be a basis of row$(A)$, and let $\varepsilon$ be the precision with which we want to estimate $\lambda_2(G(x))$. The algorithm is described as follows.

**Algorithm 116.** *To begin, let $c = 0$ and $C = 1$.*

1. *Set $\varphi = \frac{C-c}{3}$, $\epsilon = \frac{1}{\sqrt{2n}}$, $\delta = c + \varphi$.*
2. *For $j = 1, \ldots, 4\log(n/\varepsilon)$:*
   (a) *Prepare $|\psi_{init}\rangle \sum_{i=1}^{n-1} \frac{1}{\sqrt{n-1}} |i\rangle|\psi_i\rangle|0\rangle_C|0\rangle_P$.*
   (b) *Perform the gapped phase estimation algorithm $GPE(\varphi, \epsilon, \delta)$ of Theorem 10 applying $U(x, P)$ on the second register.*
   (c) *Use amplitude estimation (see Theorem 11) to distinguish between the case when the amplitude on $|0\rangle_C$ is $\geq \frac{1}{\sqrt{n}}$, in which case output "$a_j = 0$", and the case where the ampltiude is $\leq \frac{1}{\sqrt{2n}}$, in which case, output "$a_j = 1$".*
3. *Compute the majority $\tilde{a} = Maj(a_1, \ldots, a_{4\log(n/\varepsilon)})$. If the result is 0, set $C = \delta + \varphi$. If the result is 1, set $c = \delta$. If $C - c \leq 2\varepsilon c$, then output $n \sin^2\left(\frac{\pi(C+c)}{4}\right)$. Otherwise, return to Step 1.*

**Analysis of the algorithm** We say an iteration of the algorithm *succeeds* if $\tilde{a} = Maj\left(a_1, \ldots, a_{4\log(n/\varepsilon)}\right)$ correctly indicates whether the amplitude on $|0\rangle_C$ is $\geq \frac{1}{\sqrt{n}}$ or $\leq \frac{1}{\sqrt{2n}}$. This happens with probability $\Omega(1-(\varepsilon/n)^4)$. Since we will shortly see that the algorithm runs for at most $\widetilde{\mathrm{O}}\left(\frac{n}{\varepsilon\sqrt{\lambda_2(G(x))}}\right) \leq \widetilde{\mathrm{O}}\left(\frac{n^2}{\varepsilon}\right)$ steps, the probability that every iteration succeeds is at least

$$\left(1 - (\varepsilon/n)^4\right)^{(n/\varepsilon)^2} = 1 - \mathcal{O}(\varepsilon/n)^4(n/\varepsilon)^2 = 1 - \mathcal{O}(\varepsilon/n)^2. \tag{5.61}$$

It is therefore reasonable to assume that every iteration succeeds, since this happens with high probability. We first prove that if every iteration succeeds, throughout the algorithm we have $\tau = \Delta(U(x, P))/\pi \in [c, C]$.

**Lemma 117.** *Let $\tau = \Delta(U(x, P))/\pi$. For any $\varphi$ and $\delta$, if $\tau \geq \delta + \varphi$, applying $GPE(\varphi, \epsilon, \delta)$ to $|\psi_{\text{init}}\rangle$ results in a state with amplitude at most $\frac{1}{\sqrt{2n}}$ on $|0\rangle_C$ in register $C$; and if $\tau \leq \delta$, this results in a state with amplitude at least $\frac{1}{\sqrt{n}}$ on $|0\rangle_C$ in register $C$. Thus, if every iteration succeeds, at every iteration, we have $\tau \in [c, C]$.*

*Proof.* First, suppose $\tau \geq \delta + \varphi$. By Lemma 100, when $G(x)$ is connected there is no vector in $\mathrm{row}(A)$ in the 1-eigenspace of $U(x,P)$, so the 1-eigenspace of $U(x,P)$ is contained in $\ker(A) \cap H(x) \subseteq \ker A$. Thus each of the $|u_j\rangle$ from Claim 5 is in the span of $e^{i\pi\theta}$-eigenvectors of $U(x,P)$ with $|\theta| \geq \delta + \varphi$. Thus, applying $GPE(\varphi, \epsilon, \delta)$ will map each $|u_j\rangle_R|0\rangle_C|0\rangle_P$ to a state $\beta_0|0\rangle_C|\gamma_0\rangle_{PR} + \beta_1|1\rangle_C|\gamma_1\rangle_{PR}$ such that $|\beta_0| \leq \epsilon$. Then, by linearity, the total amplitude on $|0\rangle_C$ in register $C$ will be at most $\epsilon = \frac{1}{\sqrt{2n}}$.

On the other hand, suppose $\tau \leq \delta$. By Claim 5, there exists a vector $|u_1\rangle \in \mathrm{row}(A)$ such that $|u_1\rangle$ is in the span of the $e^{\pm i\pi\tau}$-eigenvectors of $U(x,P)$. Applying $GPE(\varphi, \epsilon, \delta)$ will map $|u_1\rangle_R|0\rangle_C|0\rangle_P$ to a state $\beta_0|0\rangle_C|\gamma_0\rangle_{PR} + \beta_1|1\rangle_C|\gamma_1\rangle_{PR}$ such that $|\beta_1| \leq \epsilon$, so $|\beta_0| \geq \sqrt{1-\epsilon^2}$. Let $|u_2\rangle, \ldots, |u_{n-1}\rangle$ be any orthonormal set such that $|u_1\rangle, \ldots, |u_{n-1}\rangle$ is an orthonormal basis for $\mathrm{row}(A)$. Then there exists some (unknown) orthonormal set $\{|\tilde{j}\rangle\}_{j=1}^{n-1}$ such that

$$|\psi_{\text{init}}\rangle = \frac{1}{\sqrt{n-1}} \sum_{j=1}^{n-1} |\tilde{j}\rangle |u_j\rangle |0\rangle_C |0\rangle_P. \tag{5.62}$$

So after applying $GPE(\varphi, \epsilon, \delta)$ to $|\psi_{\text{init}}\rangle$, the amplitude on $|0\rangle_C$ in register $C$ will be at least $\sqrt{\frac{1-\epsilon^2}{n-1}} \geq \frac{1}{\sqrt{n}}$. This proves the first part of the statement.

By Corollary 12, we can distinguish the case when the amplitude on $|0\rangle_C$ is at least $\frac{1}{\sqrt{n}}$ or at most $\frac{1}{\sqrt{2n}}$ with bounded error using $\mathcal{O}\left(\frac{\sqrt{p_0}}{p_0 - p_1}\right) = \mathcal{O}(\sqrt{n})$ calls to $GPE(\varphi, \frac{1}{\sqrt{2n}}, \delta)$ where $p_0 := \frac{1}{n}$ and $p_1 := \frac{1}{2n}$. Thus, by repeating the procedure $4\log(n/\varepsilon)$ times and taking the majority, we succeed at every iteration with high probability.

We now prove by induction that we always have $\tau \in [c, C]$, as long as every iteration succeeds. At the beginning of the first iteration, we have $[c, C] = [0, 1]$. Since $\Delta(U(x,P)) \in [0, \pi]$, $\tau \in [0, 1]$. Next, suppose in some arbitrary iteration, we have $\tau \in [c, C]$. If $\tau \geq \delta + \varphi$, then there will be amplitude at most $\frac{1}{\sqrt{2n}}$ on $|0\rangle_C$, and assuming the iteration succeeds, we will have $\tilde{a} = 1$. In that case, we will set $c = \delta \leq \delta + \varphi \leq \tau$, so we will still have $\tau \in [c, C]$. If $\tau \leq \delta$, then there will be amplitude at least $\frac{1}{\sqrt{n}}$ on $|0\rangle_C$, and assuming the iteration succeeds, we will have $\tilde{a} = 0$. In that case, we will set $C = \delta + \varphi \geq \delta \geq \tau$, so we will still have $\tau \in [c, C]$.

We finally consider what happens if $\delta \leq \tau \leq \delta + \varphi$. In that case, there is no guarantee on the output of amplitude estimation; it can either output 0 or 1. However, we can still use the result to update our bounds for $\tau$. If we get $\tilde{a} = 1$, and set $c = \delta$, we have $\delta \leq \tau$, so $\tau \in [c, C]$. If we get $\tilde{a} = 0$, and

set $C = \delta + \varphi$, we have $\delta + \varphi \geq \tau$, so $\tau \in [c, C]$. □

Next, we analyse the running time of Algorithm 116.

**Theorem 118.** *With probability* $1-\mathcal{O}((\varepsilon/n)^2)$*, Algorithm 116 will terminate after time* $\widetilde{\mathcal{O}}\left(\frac{n}{\varepsilon\sqrt{\lambda_2(G(x))}}\right)$.

*Proof.* With probability $1-\mathcal{O}((\varepsilon/n)^2)$, each of the first $(n/\varepsilon)^2 \geq \widetilde{\mathcal{O}}\left(\frac{n}{\varepsilon\sqrt{\lambda_2(G(x))}}\right)$ iterations of the algorithm will succeed, so we assume this to be the case. We first bound the number of (successful) iterations before the algorithm terminates. Fig. 5.4 shows the interval $[c, C]$, which represents the algorithm's


Figure 5.4: The interval $[c, C]$

current state of knowledge of where $\tau = \Delta(U(x, P))/\pi$ lies. The values $\delta$ and $\delta + \varphi$ are at 1/3 and 2/3 of the interval, respectively. it is not difficult to convince ourselves that at every iteration, the interval $[c, C]$ will be 2/3 the size it had in the previous iteration. Hence, since the interval initially has length 1, after $k$ iterations, we will have an interval of size $\left(\frac{2}{3}\right)^k$. The execution terminates when the interval becomes sufficiently small. Specifically, let $T$ be the smallest integer such that $T \geq \frac{\log \frac{2}{\tau\varepsilon}}{\log \frac{3}{2}}$, and let $[c, C]$ be the interval after $T$ steps, so $C - c = (2/3)^T \leq \tau\varepsilon/2$. Suppose $(2/3)^T = C - c \geq 2\varepsilon c$, so $\tau \geq 4c$. This implies that $C \geq 4c$, so $C/c \geq 4$. We will argue that this is a contradiction.

First, suppose $c = 0$. That means that

$$\tau \leq C = (2/3)^T \leq \tau\varepsilon/2 \leq \tau/2,$$

which is a contradiction, since $\tau > 0$. Thus, we must have $c > 0$. Consider the first setting of $c$ and $C$ such that $c \neq 0$. Since the previous value of $c$ was 0, we set the new value as $c = \delta = 0 + \varphi = (C - 0)/3 = C/3$, so the ratio $C/c$ satisfies $C/c = 3$. This ratio can only decrease, because subsequent steps either decrease $C$, or increase $c$. Thus, after $T$ steps, $C/c \leq 3$. Thus, $C - c \geq 2\varepsilon c$ leads to a contradiction, so we can conclude that after $T$ steps, $C - c \leq 2\varepsilon c$, so the algorithm terminates in at most $T$ steps.

We can now analyse the total running time by adding up the cost of all iterations. Step 1 of the algorithm is defining the variables $\varphi = \frac{C-c}{4}$, $\delta = c+\varphi$ and $\epsilon = \frac{1}{\sqrt{2n}}$, which will contribute negligibly to the complexity.

Step 2(a) begins by constructing the initial state

$$|\psi_{\text{init}}\rangle = \sum_{i=1}^{n-1} \frac{1}{\sqrt{n-1}} |i\rangle|\psi_i\rangle|0\rangle_P|0\rangle_C,$$

where $\{|\psi_i\rangle\}_{i=1}^{n-1}$ is a basis of $\text{row}(A)$. Because $G = K_n$ can be seen as a particularly simple kind of Cayley graph with group $\Gamma = \mathbb{Z}/n\mathbb{Z}$ and $S = \Gamma \setminus \{0\}$, we can use the construction of Section 5.5.2 to generate $|\psi_{\text{init}}\rangle$. In fact, combining the remarks in the proof of Corollary 113 with Lemma 111 it follows that this state can be constructed in time $\mathcal{O}(\log n \log \frac{n}{\varepsilon})$ with success probability $1 - (\varepsilon/n)^4$.

Step 2(b) consists of applying the unitary procedure $GPE(\varphi, \epsilon, \delta)$ described in Theorem 10 on the last three registers with $\varphi$, $\epsilon = \frac{1}{\sqrt{2n}}$, $\delta$ defined in Step 1. By Theorem 10, this makes $\mathcal{O}(\varphi^{-1} \log \epsilon^{-1}) = \mathcal{O}(\varphi^{-1} \log n)$ calls to $U(x, P)$.

Step 2(c) then uses amplitude estimation, repeating Steps 2(a) and 2(b) $\mathcal{O}(\sqrt{n})$ times, by Corollary 12. Let $\varphi^{(i)}$ denote the value of $\varphi$ at the $i^{\text{th}}$ iteration of the algorithm. Neglecting $\text{polylog}(n/\varepsilon)$ factors, the running time of the $i^{\text{th}}$ iteration is

$$Q_i := \frac{\sqrt{n}}{\varphi^{(i)}}. \tag{5.63}$$

During the $i^{\text{th}}$ iteration, we begin with $C - c = (2/3)^{i-1}$, and so $\varphi^{(i)} = \frac{1}{3}(2/3)^{i-1}$. Thus, we can compute the total complexity of the algorithm as (neglecting polylogarithmic factors):

$$\begin{aligned}\sum_{i=1}^{T} Q_i =& \sqrt{n} \sum_{i=1}^{T} 3(3/2)^{i-1} = 3\sqrt{n} \frac{\left(\frac{3}{2}\right)^T - 1}{3/2 - 1} = \widetilde{\mathcal{O}}\left(\sqrt{n}(3/2)^{\frac{\log(2/(\tau\varepsilon))}{\log(3/2)}}\right) \\ =& \widetilde{\mathcal{O}}\left(\frac{\sqrt{n}}{\tau\varepsilon}\right).\end{aligned}$$

By Lemma 115, we have $\lambda_2(G(x))/n = \sin^2(\Delta(U(x,P))/2) \leq \Delta(U(x,P))^2/4$, so Filling in $\tau = \Delta(U(x,P))/\pi \geq \sqrt{\lambda_2(G(x))/(2n)}$, we get a total query complexity of $\widetilde{\mathcal{O}}\left(\frac{n}{\varepsilon\sqrt{\lambda_2(G(x))}}\right)$. □

Finally, we prove that the algorithm outputs an estimate that is within $\frac{\pi^2 3}{4}\varepsilon$ multiplicative error of $\lambda_2(G(x))$.

**Theorem 119** (Correctness)**.** *With probability at least* $1-\mathcal{O}(\varepsilon/n)^2$*, Algorithm 116 outputs an estimate* $\tilde{\lambda}$ *such that* $\left|\lambda_2(G(x))-\tilde{\lambda}\right| \leq \frac{\pi^2 3}{4}\varepsilon\lambda_2(G(x))$.

*Proof.* We will assume that all iterations succeed, which happens with probability at least $\Omega(1-(\varepsilon/n)^2)$. Then the algorithm outputs $\tilde{\lambda} := n\sin^2\left(\frac{\pi(C+c)}{4}\right)$ for some $c$ and $C$ such that $c \leq \tau \leq C$, and $C-c \leq 2\varepsilon c \leq 2\varepsilon\tau$. Using $\tau = \Delta(U(x,P))/\pi$ and $\lambda_2(G(x)) = n\sin^2\left(\Delta(U(x,P))/2\right)$, we have:

$$\left|\lambda_2(G(x))-\tilde{\lambda}\right| = \left|n\sin^2\left(\pi\tau/2\right) - n\sin^2\left(\pi(C+c)/4\right)\right|. \tag{5.64}$$

From $c \leq \tau \leq C$ and $C-c \leq 2\varepsilon\tau$, we have

$$\left|\frac{\pi(C+c)}{4} - \frac{\pi\tau}{2}\right| \leq \frac{\pi\varepsilon\tau}{2}. \tag{5.65}$$

Let $\delta = \pi(C+c)/4 - \pi\tau/2$, so $\pi(C+c)/4 = \pi\tau/2+\delta$. Then we have:

$$\begin{aligned}
\left|\sin^2(\pi\tau/2) - \sin^2(\pi\tau/2+\delta)\right| &= \left|\tfrac{1-\cos(\pi\tau)}{2} - \tfrac{1-\cos(\pi\tau+2\delta)}{2}\right| \\
&= \tfrac{1}{2}\left|\cos(\pi\tau+2\delta)-\cos(\pi\tau)\right| = \left|\sin(\pi\tau+\delta)\sin(-\delta)\right| \\
&\leq \left|\delta(\pi\tau+\delta)\right| \;\leq\; \pi^2\tau^2\tfrac{\varepsilon}{2}\left(1+\tfrac{\varepsilon}{2}\right) \;\leq\; \tfrac{3\varepsilon}{4}\pi^2\tau^2
\end{aligned} \tag{5.66}$$

where we used $|\delta| \leq \pi\varepsilon\tau/2$. Then, plugging this into Eq. (5.64), we have:

$$\begin{aligned}
\left|\lambda_2(G(x))-\tilde{\lambda}\right| &\leq \frac{3\varepsilon}{4}n\pi^2\tau^2 \\
&= \frac{3\varepsilon}{4}n\pi^2\frac{\Delta(U(x,P))^2}{\pi^2} \;\leq\; \frac{3\varepsilon}{4}n\pi^2\sin^2\left(\frac{\Delta(U(x,P))}{2}\right) \\
&= \pi^2\frac{3\varepsilon}{4}\lambda_2(G(x)),
\end{aligned} \tag{5.67}$$

using the fact that $\frac{x^2}{\pi^2} \leq \sin^2(x/2)$ when $x \in [-\pi,\pi]$. □

We have proven the following theorem.

**Theorem 120.** *Let* $G$ *be the complete graph on* $n$ *vertices. There exists a quantum algorithm that, on input* $x$*, with probability at least 2/3, outputs an estimate* $\tilde{\lambda}$ *such that* $\left|\tilde{\lambda}-\lambda_2(G(x))\right| \leq \varepsilon\lambda_2(G(x))$*, where* $\lambda_2(G(x))$ *is the algebraic connectivity of* $G(x)$*, in time* $\widetilde{\mathcal{O}}\left(\frac{1}{\varepsilon}\frac{n}{\sqrt{\lambda_2(G(x))}}\right)$.

## 5.6 Graph connectivity without graph surgery

In this chapter we have given two algorithms for graph connectivity based on the *st*-connectivity span program. The first one, outlined in Section 5.4, directly transforms graph connectivity into an instance of *st*-connectivity on a bigger graph $\mathcal{G}$. The advantage in this case is that the algorithm for *st*-connectivity is at the ready and all we need to do is bound the witness sizes, which we can do accurately given the symmetry of the graph $\mathcal{G}$.

This approach has worst-case optimal query complexity, but has the problem that the time complexity scales as $\delta(\mathcal{G})^{-1/2}$, where $\delta(\mathcal{G})$ is the spectral gap of $\mathcal{G}$. It inverse is the relaxation time of the graph $\mathcal{G}$, which is within a logarithmic factor of the mixing time. Unfortunately, because $\mathcal{G}$ is a long chain of $\sim n^2$ copies of $G$, this graph mixes rather slowly. It would be perhaps possible to modify $\mathcal{G}$, adding a few parallel edges orthogonal to $\mathcal{H}(x)$ that bypass the chain and make it efficiently walk-able, but that analysis remains highly non-trivial.

Our second approach circumvents this difficulty by working directly on $G$, but at the cost of having to construct a new algorithm that is not worst-case optimal in query complexity. In the end, both the time and query complexities of this algorithm are non-comparable to those of the first one.

In this section, we will give a span program for graph connectivity that uses the span program algorithms in Section 3.4 and [JK17] off the shelf, while operating directly on $G$. We will compare all three algorithms in more detail in Section 5.8. We organize the section as follows. First, we will define a span program for graph connectivity in Def. 121. That is followed by a study on its positive and negative witnesses in Lemmas 122, and 123. Then we will show how we can eliminate the dependence on the choice of a particular vertex that acts as a seed for our span program. This leads us to Theorem 124, which states the query complexity of our algorithm. We finish the section with an analysis of the time complexity of our algorithm, resulting in Theorem 126.

**Definition 121.** Let $G$ be a graph on $n$ vertices, $\mathcal{H}, \mathcal{V}, A, \mathcal{H}(x)$ be the spaces and map of the standard *st*-connectivity span program in Definition 88. Take a vertex $s \in V(G)$ and define the state

$$|\tau_s\rangle = \sum_{t \in V(G), t \neq s} \frac{|s\rangle - |t\rangle}{n-1}. \tag{5.68}$$

We define the span program for graph connectivity with source $s$ as $P_s = (\mathcal{H}, \mathcal{V}, A, |\tau_s\rangle)$.

Then, we claim that this span program exactly decides $\text{CONN}_{G,X}$. We prove this by finding explicit positive witnesses for every connected subgraph $G(x)$, and negative witnesses for every disconnected subgraph $G(x)$.

**Lemma 122.** *Let $P_s$ be the span program in Definition 121, and let $G(x)$ be a connected subgraph of $G$. Then*

$$w_+(x, P_s) \leq \frac{1}{2} \sum_{t:t\neq s} \frac{R_{s,t}(G(x))}{n-1}.$$

*Proof.* For all $t \in V(G)$ such that $t \neq s$ let $P_{st} = (\mathcal{H}, \mathcal{V}, A, |s\rangle - |t\rangle)$ be the $st$-connectivity span program, and let $|w_t\rangle$ be the optimal positive witness for $x$ in $P_{st}$. In particular, that means that $A|w_t\rangle = |s\rangle - |t\rangle$.

By Lemma 89, we know that

$$\||w_t\rangle\|^2 = \frac{1}{2} R_{st}(G(x)),$$

Observe that we can combine all the different vectors $|w_t\rangle$ to create the witness for $|\tau\rangle$

$$|w_x\rangle = \sum_{t:t\neq s} \frac{|w_t\rangle}{n-1}.$$

Clearly, $A|w_x\rangle = |\tau\rangle$. Let us now bound its squared norm.

$$\||w_x\rangle\|^2 = \frac{1}{(n-1)^2} \left( \sum_{t:t\neq s} \||w_t\rangle\|^2 + \sum_{t,t'\neq s} 2|\langle w_t|w_{t'}\rangle| \right) \leq \frac{\left(\sum_{t:t\neq s} \||w_t\rangle\|\right)^2}{(n-1)^2} \tag{5.69}$$

$$= \left( \sum_{t:t\neq s} \frac{\sqrt{R_{st}(G(x))}}{\sqrt{2}(n-1)} \right)^2 \leq \sum_{t:t\neq s} \frac{R_{st}(G(x))}{2(n-1)}, \tag{5.70}$$

where we have used Cauchy-Schwarz in the first inequality and Jensen's inequality in the second one, given that the quadratic function is convex. □

In the electrical network analogy, this approach to graph connectivity would correspond to a network in which $s$ is a unit source of flow and each one of the $(n-1)$ other vertices is a sink draining $\frac{1}{n-1}$ units of flow. Alternatively, it is a network in which the net flow must be a combination of $st$-flows, each with intensity $1/(n-1)$. The graph is connected iff such a flow exists.

This combination of small-intensity flows could potentially have a rather small resistance due to the fact that the resistance scales quadratically with the amount of flow. For instance, if $G(x)$ were a star graph with $s$ at its center, then all flows would be orthogonal and the bound in Eq. (5.70) could be improved to

$$\||w_x\rangle\|^2 = \sum_{t:t\neq s} \frac{R_{st}(G(x))}{2(n-1)^2},$$

instead of $\sum_{t:t\neq s} \frac{R_{st}(G(x))}{2(n-1)}$.

Unfortunately, the different $st$-flows will generally have some overlap, meaning that $\||w_x\rangle\|^2$ will be somewhere between the average over $t \neq s$ of $R_{st}(G(x))$, and $1/(n-1)$ times that.

Yet, the bound is necessarily not tight because optimal flows going to two different vertices must differ somewhere. This means that the use of Cauchy-Schwarz is always not tight. In addition, Jensen's inequality is only tight when there is only one term in the sum.

Another reason why the bound might not be tight is that we do no not claim the witness we give is the optimal one, although we suspect it.

Nonetheless, in the worst-case scenario of a chain graph of length $n$ with $s$ at one end, the optimal witness is $\sim n/3$, while $\sum_{t:t\neq s} \frac{R_{st}(G(x))}{2(n-1)} \sim n/2$, meaning that the bound in Eq. (5.70) is not too far from being tight in the worst case.

Now, we shall provide negative witnesses when $G(x)$ is not connected.

**Lemma 123.** *Let $P_s$ be the span program in Definition 121, and let $G(x)$ be a disconnected subgraph of $G$, then*

$$w_-(x, P_s) \leq \frac{n^2}{(n-n_s)^2} C_s,$$

*where $n_s$ is the size of the component $H_s \subset V(G)$ containing $s$ and $C_s$ is the size of the cut $H_s$, $H_s^c$ in $G$.*

*Proof.* We simply need to find a witness of disconnectedness. Let $H_s \subset V(G)$ be the component of $G(x)$ that contains $s$, and $|H_s| = n_s$. Some of the

intuition we built in the characterization of the *st*-connectivity span program still applies. Specifically, any negative witness needs to be constant on the components of $G(x)$.

Consider the following state:

$$\langle w| = \sum_{t \in H_s} \langle t|.$$

Then, $\langle w|\tau\rangle = 1 - \frac{n_s - 1}{n-1} = \frac{n - n_s}{n-1}$. Since $\langle w|$ is constant over all the connected component and zero elsewhere we have $\langle w|A\Pi_{\mathcal{H}(x)} = 0$. Therefore, the vector $\langle \omega_x| = \frac{\langle w|}{\langle w|\tau\rangle}$ is exactly a negative witness for $x$ in $P_s$.

We readily see that

$$\|\langle w|A\|^2 = \sum_{\substack{(u,v,\ell) \in \vec{E}(G) \\ u \in H_s, v \in H_s^c}} 2c(u,v,\ell) =: 2C_s \tag{5.71}$$

There are a few ways of understanding this magnitude $C_s$. We can see this as the weighted size of the cut in $G$ that separates $H_s$ from its complement $H_s^c$. Since $H_s$ is a component in $G(x)$, this cut is over edges in $G \setminus G(x)$. In [Chu97], the set of edges between a subset $S \in V(G)$ and its complement $S^c$ is called its *edge boundary* and is denoted as $\partial(S)$ and so $C_s = |\partial(H_s)|$, where the size is taken with respect to the weight function $c(e)$. Another way of understanding $C_s$ is as the (weighted) out-degree of $H_s$. Putting all together, we have that the negative witness size is bounded as

$$w_-(x, P_s) \leq \|\langle \omega_x|A\|^2 = \left(\frac{n-1}{n-n_s}\right)^2 2C_s. \tag{5.72}$$

When all the weights are 1, this takes the form $w_-(x, P_s) \leq \frac{(n-1)^2}{n-n_s} 2d_{\max}$. □

Observe that the upper bound on $w_+(x, P_s)$ can be made independent of our choice of $s$ at little cost. That is because $\mathbb{E}_s\left(\sum_{t:t\neq s} \frac{R_{st}(G(x))}{n-1}\right) = R_{\text{avg}}(G(x))$, so by Markov's inequality we have that sampling $s$ uniformly at random from the vertex set we have

$$\mathbb{P}\left(w_+(x, P_s) \geq 10R_{\text{avg}}(G(x))\right) \leq \frac{\mathbb{E}(w_+(x, P_s))}{10R_{\text{avg}}(G(x))} \leq \frac{1}{10}.$$

Hence, we can proceed forward with the span program algorithm as if $w_+(x, P_s) \leq R_{\text{avg}}(G(x))$, which is true with probability at least 9/10. Now,

the span program algorithm in Theorem 45, even with the right bounds for $W_+$ and $W_-$, decides the span program with bounded error, say 9/10. We conclude that choosing a vertex $s$ at random and applying the algorithm from Theorem 45 decides $\text{CONN}_{G,X}$ with total probability of error $\leq 1-(9/10)^2 \leq 1/5$. In other words, we have proven the following theorem.

**Theorem 124.** *For any family of graphs $G$ and $X \subseteq \{0,1\}^{E(G)}$ there exists a procedure that decides $\text{CONN}_{G,X}$ with bounded error and with query complexity*

$$\mathcal{O}\left(\sqrt{RC\frac{n^2}{(n-n_{\max})^2}}\right),$$

*where $R$ is a known upper bound on $R_{\text{avg}}(G(x))$ for all connected $G(x)$, and for all disconnected $G(x)$, $C \geq C_s$ is an upper bound for the largest out-degree of any component of $G(x)$ and $n_{\max}$ upper bounds the size of the largest component of $G(x)$.*

Let us end this section by talking about the time complexity of deciding $P_s$. By Lemmas 54, 55, and 56, we know that the time complexity of the span program unitary depends on the complexity of reflecting around $\ker A$ and generating the initial state $|w_0^{(s)}\rangle := A^+|\tau_s\rangle$.

Since the span program $P_s$ uses the same $\mathcal{H}$, $\mathcal{V}$ and $A$ as the $st$-connectivity span program, the time complexity of $2\Pi_{\ker A} - I$ has already been shown to be $\mathcal{O}(\mathsf{U}/\delta(G))$ in Ref. [JK17], where $\mathsf{U}$ is the cost of the quantum walk operator and $\delta(G)$ is the spectral gap of the symmetric normalized Laplacian of $G$.

The time complexity of generating the minimal witness for $P_s$ is not much different from that of generating the minimal witness for the $st$-connectivity span program.

**Lemma 125.** *Let $\mathcal{H}$, $\mathcal{V}$, $A$ be as defined in Definition 88, $|\tau_s\rangle$ be the state in Eq. (5.68), and $P_s = (\mathcal{H}, \mathcal{V}, A, |\tau_s\rangle)$. Then, with constant probability the state $\frac{|w_0^{(s)}\rangle}{\||w_0^{(s)}\rangle\|}$ can be constructed using polylog$(|V|, \varepsilon)$ queries to the reflection $2\Pi_{\ker A} - I$.*

*Proof.* First, we add the edges of a star graph $\{|s,t,\emptyset\rangle\}_{t:t\neq s}$ to $\mathcal{H}_{\text{false}}$ with weight $c(s,t,\emptyset) = 1/\alpha^2$, and extend the action of $A$ on those edges to act as

it normally would. Call $\widetilde{A}$ this map acting on $E(G) \cup \{|s,t,\emptyset\rangle\}_{t:t\neq s}$. Adding edges to a graph cannot decrease the spectral gap of its symmetric normalized Laplacian, so the cost of reflecting around $\ker(\widetilde{A})$ is not bigger than the cost of reflecting around $\ker(A)$. Observe that adding edges to $G$ and running the *st*-connectivity span program algorithm with this extended graph would affect the negative complexity, but this is not what we are doing here. We are adding the edges *only* for the subroutine that generates $|w_0^{(s)}\rangle$.

Define the unnormalized state $|\hat{0}\rangle$ as

$$|\hat{0}\rangle = \sum_{t:t\neq s} \frac{|s,t,\emptyset\rangle}{n-1}. \tag{5.73}$$

Then, $\Pi_{(\ker\widetilde{A})^\perp}|\hat{0}\rangle = \widetilde{A}^+\widetilde{A}|\hat{0}\rangle = \widetilde{A}^+\frac{1}{\alpha}|\tau\rangle = \frac{1}{\alpha}|w_0'\rangle$, where the vector $|w_0'\rangle = \widetilde{A}^+|\tau\rangle$ is the shortest vector such that $\widetilde{A}|w_0'\rangle = |\tau\rangle$. Hence, $|\hat{0}\rangle$ has the following decomposition

$$|\hat{0}\rangle = \frac{1}{\alpha}|w_0'\rangle + |w_\perp\rangle$$

where $|w_0'\rangle \in (\ker\widetilde{A})^\perp$ and $|w_\perp\rangle \in \ker\widetilde{A}$. If follows that $|w_0'\rangle$ is in the $+1$ eigenspace of $Ref(\ker\widetilde{A})$ and $|w_\perp\rangle$ is in the $-1$ eigenspace of $Ref(\ker\widetilde{A})$.

Running standard phase estimation on $Ref(\ker\widetilde{A})$ with initial state $\frac{|\hat{0}\rangle}{\||\hat{0}\rangle\|}$ yields the state

$$\frac{\sqrt{n-1}}{\alpha}|w_0'\rangle|0\rangle + |w_\perp\rangle|1\rangle$$

with only one query to $Ref(\mathrm{Ker}\widetilde{A})$. We need to understand a bit more about $|w_0'\rangle$.

Remember that $\widetilde{A}|w_0'\rangle = |\tau\rangle$, and that $\mathrm{span}\,|\hat{0}\rangle$ are the only vector in the span of the new edges that map to $\mathrm{span}\{|\tau\rangle\}$. Thus, $|w_0'\rangle$ has to be a combination of the minimal witness for $A$, $|w_0^{(s)}\rangle$ and $\alpha|\hat{0}\rangle$, because it is the only witness.

Altogether we have

$$|w_0'\rangle = \gamma|w_0^{(s)}\rangle + (1-\gamma)\alpha|\hat{0}\rangle, \tag{5.74}$$

where $\gamma$ is such that $\||w_0'\rangle\|$ is minimal. It is a simple exercise to see that $\gamma = \frac{\alpha^2}{(n-1)\left\||w_0^{(s)}\rangle\right\|^2+\alpha^2}$.

Observe that running phase estimation with constant precision on $2(n-1)|\hat{0}\rangle\langle\hat{0}| - I$ maps $|w_0'\rangle$ to the state $\gamma|w_0\rangle|0\rangle + (1-\gamma)\alpha|\hat{0}\rangle|1\rangle$, and the choice for $\alpha^2$ that maximizes $\gamma$ is

$$\alpha^2 = (n-1)\left\||w_0^{(s)}\rangle\right\|^2.$$

Combining one call to phase estimation on the unitary $Ref(\ker\widetilde{A})$ with one call to phase estimation on $2(n-1)|\hat{0}\rangle\langle\hat{0}| - I$ we can generate the state

$$\frac{\sqrt{n-1}}{\alpha}\left(\gamma\||w_0^{(s)}\rangle\|\frac{|w_0^{(s)}\rangle}{\||w_0^{(s)}\rangle\|}|0\rangle + (1-\gamma)\alpha|\hat{0}\rangle|1\rangle\right)|0\rangle + |w_\perp\rangle|0\rangle|1\rangle \tag{5.75}$$

$$= \frac{\alpha\sqrt{n-1}\||w_0^{(s)}\rangle\|}{(n-1)\||w_0^{(s)}\rangle\|^2 + \alpha^2}\frac{|w_0^{(s)}\rangle}{\||w_0^{(s)}\rangle\|}|0\rangle|0\rangle + \tag{5.76}$$

$$\left(1 - \frac{\alpha^2}{(n-1)\||w_0^{(s)}\rangle\|^2 + \alpha^2}\right)\frac{|\hat{0}\rangle}{\||\hat{0}\rangle\|}|1\rangle|0\rangle + |w_\perp\rangle|0\rangle|1\rangle \tag{5.77}$$

At this stage all we need to do is measure the two extra qubit registers. If the result is $|0\rangle|0\rangle$ then we are left with $\frac{|w_0^{(s)}\rangle}{\||w_0^{(s)}\rangle\|}$ in the first register. The probability of this event is

$$p_{\text{succ}} = \left(\frac{\alpha\sqrt{n-1}\||w_0^{(s)}\rangle\|}{(n-1)\||w_0^{(s)}\rangle\|^2 + \alpha^2}\right)^2.$$

If we choose an $\alpha$ such that $\alpha^2 \ll (n-1)\left\||w_0^{(s)}\rangle\right\|^2$, then the probability of success goes as $\frac{\alpha^2}{(n-1)\left\||w_0^{(s)}\rangle\right\|^2} \ll 1$, whereas $\alpha^2 \gg (n-1)\left\||w_0^{(s)}\rangle\right\|^2$ leads to a success probability $\frac{(n-1)\left\||w_0^{(s)}\rangle\right\|^2}{\alpha^2} \ll 1$.

As we said, the optimal choice of $\alpha$ is $\alpha^2 = (n-1)\left\||w_0^{(s)}\rangle\right\|^2$, for which the success probability is $1/4$. It is easy to see that any choice of $\alpha$ that is $\Theta\left(\sqrt{n-1}\left\||w_0^{(s)}\rangle\right\|\right)$ will lead to a success probability of $\mathcal{O}(1)$.

If the value $\||w_0^{(s)}\rangle\|$ is not known, the iterative process in which we vary $\alpha$ of Lemma 47 results in an estimation of $\alpha^2 = (n-1)\left\||w_0^{(s)}\rangle\right\|^2$ within

multiplicative error $1/2$ in $\mathcal{O}(\log(n-1)\left\||w_0^{(s)}\rangle\right\|^2)$ steps. Once we have this estimate, we can generate the state $\frac{|w_0^{(s)}\rangle}{\||w_0^{(s)}\rangle\|}$ with constant probability using $\mathcal{O}(1)$ queries to $2\Pi_{\ker A} - I$ and $2\frac{|\hat{0}\rangle\langle\hat{0}|}{\left\||\hat{0}\rangle\right\|^2} - I$. □

Putting together all the results and insight in this section, together with Theorem 45 and Theorem 53 we have proven the following:

**Theorem 126.** *For any family of graphs $G$ and $X \subseteq \{0,1\}^{E(G)}$, let $\mathsf{U}$ be the cost of the quantum walk operator defined in Equation (5.32), and $\delta(G)$ be the spectral gap of the symmetric normalized Laplacian of $G$. Then there exists a procedure that decides $\text{CONN}_{G,X}$ with bounded error and cost (neglecting logarithmic factors)*

$$\mathcal{O}\left(\sqrt{\frac{RC\frac{n^2}{(n-n_{\max})^2}}{\delta(G)}}\mathsf{U}\right).$$

*where $R$ is a known upper bound on $R_{\text{avg}}(G(x))$ for all connected $G(x)$, and for all disconnected $G(x)$, $C \geq C_s$ is an upper bound for the largest out-degree of any component of $G(x)$ and $n_{\max}$ upper bounds the size of the largest component of $G(x)$.*

## 5.7 Witness generation for *st*-connectivity

In Theorem 48, we gave an algorithm that generated an $\varepsilon$-approximation of the positive witness for a general span program using $\mathcal{O}\left(\frac{1}{\varepsilon}\sqrt{w_+(x)\tilde{w}_-(x)}\right)$ queries to the span program unitary. In this section we will use that algorithm to generate an optimal positive witness for the *st*-connectivity span program over a particular graph. We will then measure that state in the computational basis of $\mathcal{H}$, obtaining an edge, which will indicate an *st*-path in $G$. We want to stress that this is simply a proof of concept aimed at proving the utility of the witness generation algorithm. The witness generation algorithm does *not* produce an *st*-path in general, but rather a combination of paths in superposition, or one edge in an *st*-path after measuring. How to generate *st*-paths with small space complexity remains an interesting open problem, which we hope we can tackle using our witness generation algorithms.

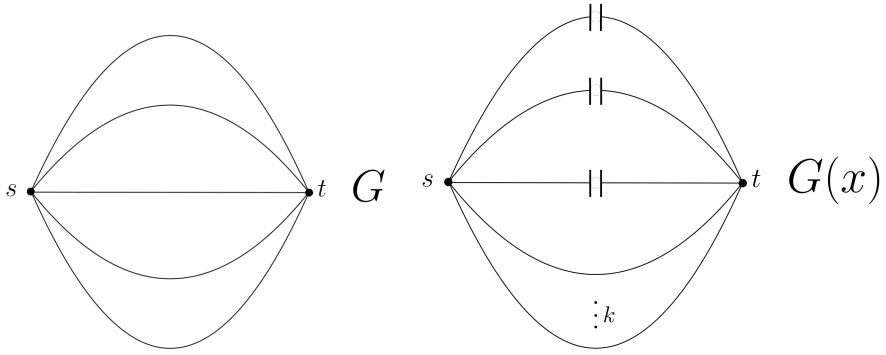



Figure 5.5: Representation of the graph $G$ and a subgraph $G(x)$ where all but $k$ paths are missing exactly one edge.

Let $G$ be the graph consisting of $m$ edge-disjoint paths of length $d$ connecting two points $s$ and $t$. For simplicity imagine that all weights are equal to one, see Figure 5.5. Then consider the problem of $st$-connectivity on $G$.

This problem is equivalent to solving the function $\mathrm{OR}_m$ composed with $\mathrm{AND}_d$ because any one of the $m$ paths makes $s$ and $t$ connected and every edge in a given path must be present for it to be a valid $st$-path in $G(x)$. The randomized query complexity of these problems is $\Omega(md)$. See, for example, [ABK16, Theorem 5] or [JK09, Theorem 4]. Quantumly, the only algorithm that we know of that can generate $st$-paths is the algorithm in [DHH+06] that generates spanning trees in time $\mathcal{O}(n^{3/2})$, where $n = md$ in this case. Since this lower bound applies to all graphs, not just the one we use here, we do not claim that our algorithm outperforms theirs in terms of query complexity. However, the algorithm by Dürr *et al.* will necessarily store a spanning tree of the graph, which in their approach requires $\mathcal{O}(md)$ space (the number of edges in this particular graph). Since our algorithm simply makes calls to the $st$-connectivity span program unitary, we achieve our goal of finding an $st$-path with only $\mathcal{O}(\log n) = \mathcal{O}(\log md)$ space.

Let's return to our problem. In Figure 5.5 we can see that any graph $G(x)$ such that $s$ and $t$ are connected is composed of $k$ disjoint $st$-paths of length $d$, for some $k > 1$, and a bunch of loose ends. By Lemma 89, the optimal positive witness corresponds to an $st$-flow on $G(x)$ which is nothing but a superposition of $st$-paths. It is easy to see that since all $st$-paths are disjoint and of equal length, the optimal flow will split equally among them,

each carrying $1/k$ flow. Moreover, there will be no flow through any of the edges that are *not* part of a valid $st$-path.

Let $|p_1\rangle, \ldots, |p_k\rangle$ be these paths, where $|p_i\rangle = |s, u_{i,2}\rangle + \cdots + |u_{i,d-1}, t\rangle$. Then what we just said is that $|w\rangle = \frac{1}{k}\sum_{i=1}^{k}|p_i\rangle$ is the optimal positive witness for $G(x)$. By Theorem 48, we can generate a state $|\tilde{w}\rangle$ $\varepsilon$-close to $|w\rangle / \||w\rangle\|$ using $\mathcal{O}\left(\frac{1}{\varepsilon}\sqrt{w_+(x)\widetilde{w}_-(x)}\right)$ queries to a unitary $U_x$ which by Theorem 53 has query complexity exactly 1. Since $|w\rangle$ is orthogonal to all edges that are not part of any $st$-path, measuring $|\tilde{w}\rangle$ in the computational basis of $\mathcal{H}$ will produce an edge $|u, v\rangle$ which will *not* be an element of an $st$-path with probability less than $\varepsilon$. In other words, generating $|\tilde{w}\rangle$ and measuring it will produce an edge in one of the $k$ $st$-paths of $G(x)$ with probability bigger than $1 - \varepsilon$. If that is the case, the result of the measurement itself is enough for us to *find* an $st$-path, since each edge belongs to only one path.

The positive witness size is $w_+(x) = \frac{d}{k}$ by direct calculation. The min. error negative witness size $\widetilde{w}_-(x)$ will be a bit harder to compute. Just to make our life a little bit harder, let us assume that each broken path is missing exactly one edge.

Observe that the graph $G$ is a planar graph. This means that we can use the *dual graph* construction in [JK17] to produce a graph $G'$ constructed in the following manner. Fix a planar embedding of $G$ and for each internal face add a vertex of $G'$. Then add an edge from the vertex $s$ to infinity and an edge from $t$ to infinity. This will split the exterior face into two faces call one such face $s'$ and the other $t'$. Then, for each edge in $G$ that is adjacent to two faces, add an edge in $G'$ that connects their respective vertices. For our particular $G$, the graph $G'$ can be seen in Figure 5.6. This graph encodes $\mathrm{AND}_k \circ \mathrm{OR}_m$. Observe that every edge of $G'$ cuts exactly one edge in $G$. A subgraph $G'(x)$ of $G$ will include an edge $|u', v'\rangle$ if and only if the edge in $G$ that it cuts is *not* an edge in $G(x)$. The beauty of this correspondence is that negative witnesses in $G(x)$ correspond exactly to positive witnesses in $G'(x)$. The same thing is true for non-exact witnesses. It is rather easy to convince ourselves that the min. error negative witness in $G(x)$ will be the min. error positive witness in $G'(x)$, which is surprisingly easy to compute.

Indeed, consider the graph $G'(x)$. Any positive witness in $G'(x)$, exact or not, is an $s't'$-flow, but since this graph is a chain, the flow through every vertex is always 1. Therefore, finding the min. error flow through the chain amounts to finding the min. error flow through each link of the chain. Each complete path in $G(x)$ will be $d$ parallel edges between the same two vertices

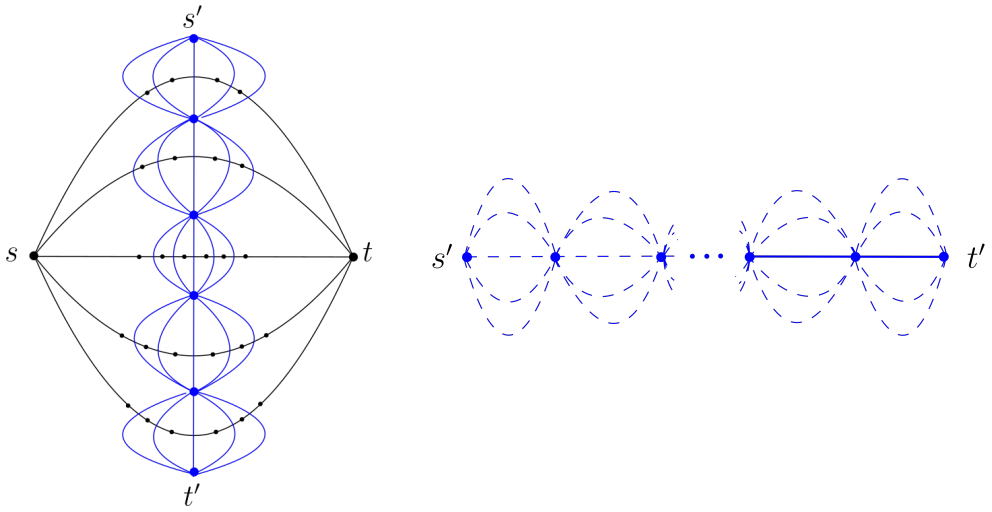



Figure 5.6: Left: The dual graph $G'$, represented in blue, has a vertex for every interior face of $G$ and an edge between two faces if the faces share an edge in $G$. Right: The dual graph $G'(x)$. Dashed edges represent edges in $G' \setminus G'(x)$, while the solid edges are edges in $G'(x)$, which correspond to the missing edges of $G(x)$.

of $G'$, and since every edge in the path is in $G(x)$, the dual edges in $G'$ are all in $G' \setminus G'(x)$. We conclude that the flow through this link of the chain all contributes towards the negative error of the flow. In these sections, the flow that minimizes the error is the one that is split along the $d$ parallel edges, which contribute a $1/d$ term to the min. error (i.e. the energy of the flow through the edges in $G' \setminus G(x)$), and to the overall energy of the flow. Since there are $k$ such links, the error of the flow through these links is $k/d$.

For the sections that correspond to paths in $G(x)$ that are broken, there are $d$ parallel edges through which a unit flow must go, but one of them is an edge in $G'(x)$, while all the others are edges in $G' \setminus G(x)$. It follows that the flow that minimizes the error is the one that goes all through the solid edge. Therefore, the min. error flow, in going through this sections does not contribute to the error. You can tell that we are on the right track because the error of this flow is $k/d = 1/w_+(x)$, exactly as Theorem 42 would predict. Overall, given that there are $k$ sections in $G'(x)$ with no solid edges and $m-k$ sections with exactly one solid edge, we conclude that the min. error positive

witness size in $G'(x)$ has norm

$$\widetilde{w}_-(x) = \frac{k}{d} + m - k. \tag{5.78}$$

Together with our computation of the positive witness size we have the following.

**Theorem 127.** *Let $G$ be the graph consisting of $m$ parallel paths of length $d$ between two vertices $s$ and $t$. Let $G(x)$ be a connected subgraph of $G$ with $k$ st-paths. Then, there exists a quantum algorithm that, with bounded error, finds an st-path on $G(x)$ with query complexity*

$$\mathcal{O}\left(\sqrt{w_+(x)\widetilde{w}_-(x)}\right) = \mathcal{O}\left(\sqrt{\frac{md}{k}}\right). \tag{5.79}$$

Our algorithm is a quadratic speed-up compared to the randomized query complexity lower bound, and much faster than the general purpose $\mathcal{O}\left(n^{3/2}\right) = \mathcal{O}\left((md)^{3/2}\right)$ complexity in [DHH+06], although it is possible that their algorithm, adapted to this specific problem, would achieve the same complexity as ours. Nonetheless, we expect the advantage of our algorithm with respect to theirs in terms of space complexity to remain.

### Path-finding in more general graphs

As we have already stated, the witness generation algorithm applied to the *st*-connectivity span program does not produce a path. Instead, it approximately produces a superposition of edges belonging to *st*-paths, with edges that sit in shorter paths having a bigger amplitude than edges on longer paths.

Measuring this state in the computational basis produces a single edge, call it $|u, v\rangle$, which will be in an *st*-path with high probability — A constant error $\varepsilon$ in Theorem 48 suffices because we can always run $su$-$\text{CONN}_{G,X}$ and $vt$-$\text{CONN}_{G,X}$ a few times to confirm it.

In the example above, that edge is part of a segment in $G$ having $s$ and $t$ as its endpoints. The measurement result $|u, v\rangle$ indicates not only that $|u, v\rangle$ belongs to an *st*-path in $G(x)$, but that *every* edge in the segment belongs to an *st*-path in $G(x)$. In this particular case, it just so happens that the segment is itself an *st*-path. Needless to say, this need not be the case in

more general graphs, but the logic of identifying the whole segment to which $|u, v\rangle$ belongs as part of an $st$-path still remains.

Now that we have identified one segment $(u_*, \ldots, u, v, \ldots, v_*)$ belonging to an $st$-path in $G(x)$, we could, for example, contract it. This would shorten every $st$-paths that go through that segment, thus increasing the amplitude through their edges while decreasing the amplitude through the $st$-paths disjoint from the contracted segment. We can see that this is true by drawing from the electrical analogy. Contracting a segment reduces the resistance of any path containing said segment, and so more flow would follow those paths, to the detriment of the paths disjoint to the segment. Moreover, since the total resistance would decrease, finding the optimal positive witness on the contracted graph would now be cheaper.

Another thing that we could do is remove the segment $(u_*, \ldots, u, v, \ldots, v_*)$ and break the problem into two, one of finding an $su_*$-path, and one of finding a $v_*t$-path. This would again de-emphasize $st$-paths that do not contain the segment $(u_*, \ldots, u, v, \ldots, v_*)$.

Regardless of which way we choose to proceed, iterating on the process would eventually lead to an $st$-path with high probability, if only because a graph can only be contracted or cut so many times. We hope that we can use these ideas to construct a path-finding algorithm that uses an amount of space of the order of the length of the path, rather that the size of the whole graph. How exactly remains, you guessed it, an open question.

## 5.8 Discussion and open problems

In this chapter we have provided a tight characterization of the negative witness of the $st$-connectivity span program. Since $st$-connectivity is fairly ubiquitous, it seems that our approach may, in turn, help analyse future applications of $st$-connectivity. Additionally we provide two algorithms for deciding the problem of graph connectivity. The first one, discussed in Section 5.4 has a query complexity of

$$Q_1 = \mathcal{O}\left(\sqrt{\frac{Rn^2}{\kappa}}\right),$$

where $R$ is an upper bound on $R_{\text{avg}}(G(x))$ for all connected subgraphs $G(x)$. Given the complete characterization of the $st$-connectivity span program, we

think that the query complexity of this algorithm is actually $\Theta(n\sqrt{R/\kappa})$. In the worst case, $\kappa = 2$ and $R < n - 1$ so the algorithm has complexity $\Theta(n^{3/2})$, which is known to be optimal. However, we want to remark that this is a promise problem, and given a strong promise one could go well below the $n^{3/2}$ complexity of the worst case. For instance, it is known that random graphs with $n$ vertices and edge probability $p$, usually denoted as $G(n,p)$, will be almost surely connected if $p \geq c \ln n/n$ for any $c > 1$ and almost surely disconnected if $p = \Omega(1/n)$. In that last case the number of components will be $\mathcal{O}(n/\ln n)$ components. Under this promise, we would then have $R \leq \text{diam}(G(x)) = \mathcal{O}(\log n/\log\log n)$ and $\kappa \geq \mathcal{O}(n/\ln n)$ for a total complexity of $\widetilde{\mathcal{O}}(\sqrt{n})$.

On the other hand, the spectral algorithm from Section 5.5 has a query complexity of

$$Q_2 = \mathcal{O}\left(\sqrt{\frac{nd_{max}(G)}{\kappa\lambda}}\right),$$

where $\lambda$ is a lower bound for the algebraic connectivity of all connected instances. Unlike the other algorithm, where the complexity depended uniquely on the number of vertices and our promises on positive and negative witnesses, the query complexity of this algorithm "also" depends on the parent graph through the $d_{max}(G)$ factor. In the unweighted worst case, $\lambda_2(G(x)) \geq 2/n^2$ and $d_{\max} = n-1$, which gives a sub-optimal $\mathcal{O}(n^2)$ algorithm. However, for some classes of inputs, this algorithm may perform better than our first algorithm. Indeed, let us consider the quotient

$$\frac{Q_1}{Q_2} = \sqrt{\frac{nR\lambda}{d_{max}(G)}}.$$

Using the fact that $R_{\text{avg}}(G(x)) \leq 1/\lambda_2(G(x))$ and $d_{max} \leq n-1$ we have that as long as $d_{max}/n \sim 1$ then the first algorithm will outperform the second. If $d_{max} \ll n$ it is possible that $Q_2 < Q_1$ as long as $R$ is not much smaller than $1/\lambda$. We could imagine a scenario where the parent graph $G$ had low degree but still have high algebraic connectivity for its connected subgraphs. For example, it is known that expander graphs have those properties. Naively it seems like our two algorithms are incomparable even though they are based on similar unitaries. It would be worthwhile to understand whether the two approaches are fundamentally different.

We remark that our spectral connectivity algorithms apply for any choice of edge weights on $G$: $d_{\text{avg}}(G)$ and $d_{\max}(G)$ should be interpreted as the

average and maximum *weighted* degrees in $G$, and $\lambda_2(G)$ and $\lambda_2(G(x))$ the second-smallest eigenvalue of the *weighted* Laplacian of $G$ and $G(x)$ respectively. The choice of weights may also impact the costs $\mathsf{U}$ and $\mathsf{S}$. Therefore, another open question is to determine how to set the weights of edges in our graphs since these weights can have a significant effect on query and time complexity.

The third connectivity algorithm shares features, for good or ill, with both of the approaches discussed above. Like the first one, its query complexity is described by electrical quantities and the number of vertices as

$$Q_3 = \mathcal{O}\left(\sqrt{RC\frac{n^2}{(n-n_{\max})^2}}\right),$$

where $R$ is a known upper bound on $R_{\text{avg}}(G(x))$ for all connected $G(x)$, and for all disconnected $G(x)$, $C$ is an upper bound for the largest out-degree of any component of $G(x)$ and $n_{\max}$ upper bounds the size of the largest component of $G(x)$.

When the weights are uniform and we have no multi-edges, this complexity reduces to $\mathcal{O}\left(\sqrt{\frac{n^3 d_{\max}}{n-n_{\max}}}\right) = \mathcal{O}(n^2)$, worse that the worst case $\mathcal{O}(n^{3/2})$ of the algorithm in Theorem 98, and on-par with the spectral algorithm of Section 5.5.

When the number of components is $\kappa = \mathcal{O}(1)$ and no component is much bigger than the others for all disconnected graphs, then the complexity reduces to $\mathcal{O}(\sqrt{RC})$, or $\mathcal{O}\left(\sqrt{Rnd_{\max}}\right)$. Since $d_{\max} \leq n$ and $R \leq 1/\lambda_2(G(x))$, this algorithm can be better than either of the algorithms in sections 5.4 and 5.5 when these inequalities are not tight.

Discussions about time complexity, sometimes referred to as *cost*, are necessarily more tentative as it has not been the focus of this chapter. First of all, because these algorithms have different starting assumptions. The first one only works for unweighted graphs, and works best for *simple* graphs (without multi-edges), while we have given two versions of the spectral algorithm, one that works for any graph and can be rather inefficient, and a more efficient one for Cayley graphs.

Moreover, the time complexity depends on quantities that can vary wildly depending on the structure of the parent graph, some of which we have not fully characterized. For example, the time complexity of the first algorithm

depends on $\delta(\mathcal{G})$, the spectral gap of the symmetric Laplacian of $\mathcal{G}$, which is yet to be characterized. Intuitively one would expect this to be significantly bigger than $\delta(G)$ since the spectral gap of the symmetric normalized Laplacian is equal to the spectral gap of the random walk over $\mathcal{G}$. This spectral gap, in turn, is proportional to the mixing time of $\mathcal{G}$, which is a a chain of $\sim n^2$ copies of $G$. The mixing time of a chain scales quadratically with the length of the chain, and since each node of the chain is an entire copy of $G$ that has to be traversed, one would expect the mixing time to be $\sim n^4 CT(G)$, where $CT(G)$ is the average commute time in $G$, which is known to be related to the average resistance [CRR+96].

The time complexity of the second algorithm is not much easier to characterize in the general case, although it does benefit from only ever using the parent graph $G$. This allows its time complexity to depend on $\delta(G)$, the spectral gap of $L_G^{\mathrm{sym}}$, rather than $\delta(\mathcal{G})$. In contrast, the time complexity of the third algorithm is neatly characterized as the query complexity times $\mathsf{U}/\delta(G)$ for every parent graph (ignoring logarithmic factors), weighted or unweighted. If such graph is sufficiently structured (e.g. expander graphs, or the complete graph), these two terms can be as small as $\mathcal{O}(\log|V|)$, which would make this third algorithm time efficient.

In Section 5.7 we applied the witness generation algorithm in Theorem 48 to the *st*-connectivity span program to find *st*-paths in a particular graph. Taking inspiration from this toy problem, we sketched a couple of ideas that we believe could lead to algorithms for path-finding with small space complexity in other graphs. For the sake of brevity we will not repeat them here, but we do want to mention the three main avenues of future research related to this problem. The first thing to do is formalize such an algorithm and prove its correctness and complexity. It might be that a rigorous proof of correctness and account of complexity is just not possible. If that is so, it can still be the case that the algorithm works on a heuristic level, with numerical simulations providing the necessary evidence. Once an algorithm is formalized, it would be wise to restrict ourselves, as a first step, to particular families of graphs, such as *planar* graphs or *series-parallel* graphs. Last, it remains an open question whether there exists applications of the witness generation algorithm for *st*-connectivity other than path-finding.

Finally, it would be interesting to see whether one can extend our algorithm for estimating algebraic connectivity to accept more general parent graphs than the complete graph. Looking closely at our algorithm, it is clear

that the algorithm works because the Laplacian of the complete graph is degenerate (all non-zero eigenvalues are equal). Absent that, it is unclear how one could determine the spectrum of $D = A^+A(x) = \Pi_{\ker(A)^\perp}\Pi_{\mathcal{H}(x)}$. A last open question is whether or not it would be possible to adapt this algorithm to graphs with *almost degenerate* Laplacians. For example, it is well known that the spectrum of the completely bipartite graph $K_{n,n}$ is $n$ with multiplicity $2(n-1)$ and $2n$ with multiplicity 1. It would be worthwhile to see if an argument can be made that can give us useful bounds on the spectrum of $D$ when $G = K_{n,n}$.

# Chapter 6

# Span programs for boundary problems

## 6.1 Overview

In the last chapter of this dissertation we will dust off our old topology books and show that the problem of *st*-connectivity is a particular instance of a much larger class of problems. That of simplicial homology.

We assume no previous knowledge of the topic, and so we will define all the concepts we need and explain them as they appear. The contents of this chapter are based on a paper in preparation and is joint work with Maris Ozols and Stacey Jeffery.

**Boundary problems in graphs** In Section 6.2, we take a broader look at the *st*-connectivity span program and frame the problem it solves as a boundary problem. This allows us to imagine different boundaries encoding other graph problems. We show how one can modify the *st*-connectivity span program to decide if there exist flows with multiple sources and sinks, and obtain algorithms for the OR and the AND of *st*-connectivity problems.

**Simplicial complices** Further generalizing *st*-connectivity to higher dimensions requires us to first generalize graphs to higher dimensions. We do just that in Section 6.3.1, where we define the concepts of *simplices*, $n$-dimensional counterparts to triangles, *simplicial $k$-complices*, collection of $k$ dimensional simplices and all the smaller-dimensional simplices they contain

within their boundaries, and *simplicial space*, which is the inner product space generated by a simplicial complex. The importance of simplicial complices does not come just from the fact that they generalize graphs. It is a classic theorem in topology [DM68; Moi13] that any topological manifold of dimension $\leq 3$ accepts a *triangulation*, that is, a simplicial complex that is topologically equivalent (i.e. there exists a continuous bijection with continuous inverse) to the manifold. Many smooth manifolds of higher dimension accept a triangulation too.

**Simplicial homology** We continue in Section 6.3.2 with the definitions of *simplex orientation*, *boundary maps* $\partial_k$ and $k$*-cycles* as elements of $\ker(\partial_k)$ for all $k \in \mathbb{N}$.

For example, the 2-simplex (i.e. triangle) $|(v_1, v_2, v_3)\rangle$ is mapped to the clockwise oriented cycle $|(v_1, v_2)\rangle + |(v_2, v_3)\rangle - |(v_1, v_3)\rangle$ by the map $\partial_2$. The map $\partial_1$ maps a 1-simplex $|(u, v)\rangle$ to $|(u)\rangle - |(v)\rangle$, and so $\partial_1 \circ \partial_2 |(v_1, v_2, v_3)\rangle = 0$. In fact, the property $\partial_k \circ \partial_{k+1} = 0$ is true for any $k \in \mathbb{N}$. A simplicial space, together with its boundary maps (or any collection of maps such that $\partial_k \circ \partial_{k+1} = 0$ for all $k$), forms what is known as a *chain complex*.

Equipped with these definitions, we then introduce the central concept of this chapter, namely, that of *simplicial homology*. Fix a simplicial complex $\mathcal{K}$, and let $|\sigma\rangle \in \mathbb{C}^{\mathcal{K}}$ correspond to a combination of $(k+1)$-simplices for some $k$. From the equality $\partial_k \circ \partial_{k+1} = 0$, it follows that $\partial_{k+1}|\sigma\rangle \in \ker(\partial_k)$, which is precisely the definition of a $k$-cycle. The converse is not true. There are $k$-cycles that are not in the image of $\partial_{k+1}$. For any $k$, the *homology group* $H_k$ is a way to classify the cycles in a complex in equivalence classes according to the rule $|\tau\rangle \sim |\eta\rangle$ iff $|\tau\rangle - |\eta\rangle \in \mathrm{Im}(\partial_{k+1})$ for any $|\tau\rangle, |\eta\rangle \in \ker(\partial_k)$. Two cycles in the same equivalence class are called *homologous*.

Homology is studied because it is a *topological invariant*. If a manifold accepts a triangulation, then the manifold's and triangulation's homology groups will be the same. If two simplicial complices $\mathcal{K}$ and $\mathcal{K}'$ are isomorphic, so will be their homology groups, and if a simplicial complex $\mathcal{K}'$ is a subdivision of another complex $\mathcal{K}$, then their homology groups will be the same.

We conclude Section 6.3 with a discussion of non-simplicial complices, commonly known as *cellular complices*, and define sub-complices of a complex specified by a variable $x \in \{0, 1\}^N$ for $N \in \mathbb{N}$ in Section 6.3.4.

**A span program for simplicial homology** Having defined simplicial complices and sub-complices, boundary operators and $k$-cycles, and simplicial homology, we finally present the span program for simplicial homology in Section 6.4. Fix a simplicial $k$-complex $\mathcal{K}$ and a cycle $|\tau\rangle \in \ker(\partial_{k-1})$, we define $\mathcal{H}$ as the space generated by the $k$-simplices in $\mathcal{K}$, $\mathcal{H}(x)$ as the space generated by the $k$-simplices in $\mathcal{K}(x)$, $\mathcal{V}$ as the space generated by the $(k-1)$-simplices in $\mathcal{K}$ and $A = \partial_k$. Then, we prove that the span program $P = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$ decides whether $|\tau\rangle$ is homologous to the null-vector in $H_k(\mathcal{K}(x))$. This implies that there exists a quantum algorithm that decides this function. Unfortunately, the only way we know of estimating its complexity is to do it case by case. In Section 6.4.1, we show that the span program for $st$-connectivity from Definition 88 is a particular case of the general span program for simplicial homology that computes whether two vertices are equal in the zero-th homology group of a graph.

**A span program for homology of surfaces** In Section 6.5 we define orientable surfaces and show that cycles on such a surface are null in the first homology group of the surface iff they are *separating*, i.e. they separate the surface into two sub-surfaces, *and* are the sole boundary of at least one of them. We give the example of the torus, where the only non-separating cycles are the ones that go around the handle, called *meridians*, and the ones that go around the major circumference of the torus.

The homology group of a surface contains a lot of information about its topology and has been studied in the classical literature extensively and used to construct algorithms, e.g. to find minimum cuts in a surface-oriented graph [CEN09]. As we mentioned before, all surfaces admit a triangulation, i.e. a simplicial 2-complex $\mathcal{K} = (F, E, V)$ composed of faces, edges, and vertices, with a continuous, invertible map from the surface to the complex that makes them topologically equivalent. In particular, homology in the surface is captured by homology in the simplicial complex.

With that in mind, we consider the span program for simplicial homology as it applies to triangulations of surfaces. In Lemma 135 we characterize the positive witness size of a null-homologous 1-cycle $|\tau\rangle$ in a subcomplex $\mathcal{K}(x) \subseteq \mathcal{K}$ as the size of the subset of $F(x)$ that covers the subsurface with boundary $|\tau\rangle$. The negative witness size of a non-null cycle in $\mathcal{K}(x)$ is characterized in Lemma 136 as being at most 2. Together, these lemmas prove Theorem 137, which states that the quantum query complexity of deciding whether a cycle

$|\tau\rangle$ on a sub-complex $\mathcal{K}(x)$ of a surface triangulation $\mathcal{K}$ is $\mathcal{O}(\sqrt{n+g})$, where $n$ is the number of vertices in $\mathcal{K}$ and $g$ is the genus of the surface.

## 6.2 Boundary problems in graphs

An old professor of the author used to say that in math there are only three tricks —multiply by one, add zero, and make two different things the same— and a handful of problems. There are computation problems, there are identification/classification problems, there are boundary problems, and there is probably something else somewhere. We will be interested in the third kind. Boundary problems are, at their core, variations of the same question. Is there an object with some given properties that has this particular boundary? Derivatives and integrals or differential equations are examples of boundary problems.

The *st*-connectivity problem is a particular kind of boundary problem. Indeed, asking if two vertices $s$ and $t$ of a graph $G$ are connected is equivalent to asking if there exists a path, i.e. a subgraph, with $s$ and $t$ as boundaries (naturally, the boundary of a line, curve, or path is just two points). In the *st*-connectivity span program, positive witnesses are merely combinations of such paths. Moreover, this problem maps naturally to the one of optimal flow in an electrical network.

By changing the target, we can ask about different kinds of boundaries. For example, fix two sets of vertices $\{|s_i\rangle : i \in [k]\}$, and $\{|t_i\rangle : i \in [k]\}$ for some $k$. If we let the target be

$$|s_1\rangle + \cdots + |s_k\rangle - |t_1\rangle - \cdots - |t_k\rangle,$$

then the span program tells you if there is a set of $k$ paths, each one connecting a unique $s_i$ to a unique $t_j$ (note, this does not solve perfect matching as there are cases where such paths exist but there is no perfect matching). Again, this maps perfectly to the electrical network with $k$ sources and $k$ sinks. Note that the number of sources and sinks need not be the same as long as the total in-flow equals the total out-flow. An extreme case would be to take as target the state

$$|\tau\rangle = k|s\rangle - |t_1\rangle - \cdots - |t_k\rangle.$$

A span program with this target would decide if $s$ is connected to *all* $t_i$. In other words, the elements of the *st*-connectivity span program with this

target would make a span program for

$$\bigwedge_{i=1}^{k} (st_i\text{-CONN}_{G,X}) \,. \tag{6.1}$$

If the set $\{t_i : i \in [k]\}$ is $V(G) \setminus \{s\}$, then we would have that this span program computes graph connectivity. Indeed, we have already encountered and analysed this span program in Section 5.6.

If we set the target to be $|s\rangle$, and we make vertices $|t_1\rangle, \ldots, |t_k\rangle$ into "singularities" by making them the images of free vectors in $\mathcal{H}$, then the span program decides if there's a path from $|s\rangle$ to any of the $|t_i\rangle$. In other words, we would obtain a span program for

$$\bigvee_{i=1}^{k} (st_i\text{-CONN}_{G,X}) \,. \tag{6.2}$$

Imagine that we are promised that for any subgraph of $G$, $s$ is connected to one endpoint $t_i$ out of $k$ different possibilities. One way to find which one is the right endpoint would be to run a Grover search over the possible targets, using the $st$-connectivity span program algorithms to mark the right vertex. Since this algorithm would have different complexity for each pair $st_i$, we would have to use the variable time search from [Amb10] or the algorithm we construct in Section 4.6. In any case, the complexity of that algorithm would scale with $k$ as $\sqrt{k}$.

Using this span program we can test many endpoints at a time, and so we could find the right one through binary elimination with $\mathcal{O}(\log k)$ evaluations of the span program algorithm for $\bigvee_{i=1}^{k} (st_i\text{-CONN}_{G,X})$. However, the positive witness for that span program would be the same as that of every $st_i$-connectivity span program, while the negative witness would be the sum of all $k$ negative witness for the $st_i$-connectivity span programs, for a total complexity that scales as $\widetilde{\mathcal{O}}(\sqrt{k})$. Rather similar but with fewer moving parts. Together with the witness generation algorithm for $st$-connectivity, slightly modified for the occasion, we could generate the optimal witness between $s$ and the right $t_i$ without needing to know which one that is.

We will not develop any of these span programs any further, rather we mention them to show how seeing the $st$-connectivity span program as a boundary problem can help us generalize it.

## 6.3 Simplicial complices and homology

A different way of generalizing the *st*-connectivity span program is going to higher dimensions. A graph is a collection of points and segments, which can be seen as the simplest 0-dimensional and 1-dimensional objects around. Continuing this reasoning, the generalization of a segment is a polygonal area, the simplest example being a triangle. After that comes a volume, with tetrahedrons and so on. And just like the boundary of a segment is a combination of vertices, the boundary of a polygon is a combination of segments.

We can imagine lifting the ideas of the *st*-connectivity span program one dimension up, and define a span program that maps polygons to lines, which is nothing but taking a graph and mapping faces to edges. If that is what we are going to do, we should also think about how the targets should look like, and what kind of functions we can decide. Answering these questions will require us to delve a little bit into topology theory. For the battery of definitions coming we apologize.

### 6.3.1 Simplicial complices

As we said at the beginning, boundary problems abound in mathematics. They are the bread and butter of differential equations, complex analysis and topology. Thus, characterizing the *st*-connectivity problem as a boundary problem only registers as a mild surprise. In fact, graphs are the simplest instance of a category of topological objects known as *simplicial complices*. Let us break down this name, starting from the top.

A *simplex* is a generalization of the concept of a triangle to arbitrary dimensions. A 0-simplex is a point, a 1-simplex is a line segment, a 2-simplex is a triangle, a 3-simplex is a tetrahedron, and so on.

In geometry, a $k$-simplex is determined by its $k+1$ vertices as follows.

**Definition 128** (Geometric simplex)**.** Let $v_0, \ldots, v_k \in \mathbb{R}^k$ such that $\{v_i - v_0\}_{i=1}^k$ are linearly independent. Then the simplex determined by these points is the set

$$\sigma = \left\{ \alpha_0 v_0 + \cdots + \alpha_k v_k : \sum_{i=0}^{k} \alpha_i = 1, \alpha_i \geq 0, \forall i \right\}. \tag{6.3}$$

Now, we will not be concerned from now on about coordinates and linear constraints, but this definition is useful because it illustrates a few things. First, that a $k$-simplex is determined by $k+1$ vertices $\{v_0, \ldots, v_k\}$. Second, that a simplex is the *convex hull* of the vertex set (all the points between the vertices). This is a complicated way to remind the reader that, for instance, when we say that a 2-simplex is a triangle, we do not mean just three points joined by three lines, the simplex is the surface inside the triangle. Third, that the set of points obtained by fixing one of the $\alpha_i = 0$, called the *faces* of a $k$-simplex, are $(k-1)$-simplices themselves.

With a slight abuse of notation, we can forget about coordinates and just say that a $k$-simplex is a formal tuple $\sigma = (v_0, \ldots, v_k)$ determined by $k+1$ vertices, with the convention that two tuples determine the same simplex if they differ by an even permutation of their elements. We use tuples instead of sets because we will later use the order of the elements to define the *orientation* of the simplex.

The *faces* of a simplex are the tuples formed by subsets of $\{v_0, \ldots, v_k\}$. Thus, the tuple $(v_0, v_1, v_2)$ denotes a triangle with edges $(v_0, v_1)$, $(v_1, v_2)$ and $(v_2, v_0)$ and vertices $(v_0)$, $(v_1)$, and $(v_2)$. This allows us to think of undirected graphs as collections of 0-simplices, the vertices, and 1-simplices, the edges. This is an instance of a *simplicial complex*.

**Definition 129.** A *simplicial complex* $\mathcal{K}$ is a set of simplices such that:

1. every face of a simplex in $\mathcal{K}$ is also in $\mathcal{K}$,
2. if two simplices $\sigma_1$ and $\sigma_2$ share vertices $v_1, \ldots, v_\ell$, then the simplex $(v_1, \ldots, v_\ell)$ is a face of both $\sigma_1$ and $\sigma_2$.

These requirements are necessary because when we define simplices as formal tuples of vertices, we lose the property that the faces of the simplex are contained in the simplex, since tuples don't formally contain other tuples. Let's give a couple of examples.

**Example 6.1.** Assume we have a 2-simplex $\sigma = (v_0, v_1, v_2)$ defined only as a tuple of points. The smallest simplicial complex that contains $\sigma$ is

$$\mathcal{K} = \{(v_0), (v_1), (v_2), (v_0, v_1), (v_0, v_2), (v_1, v_2), \sigma\}.$$

**Example 6.2.** Consider the following 2-complex, $\mathcal{K}$ shown in Figure 6.1.

$$\mathcal{K} = \mathcal{K}_0 \cup \mathcal{K}_1 \cup \mathcal{K}_2,$$

where

$$\begin{aligned}\mathcal{K}_0 &= \{(v_0), (v_1), (v_2), (v_3), (v_4)\},\\ \mathcal{K}_1 &= \{(v_0, v_1), (v_0, v_2), (v_1, v_2), (v_2, v_3), (v_2, v_4), (v_3, v_4)\},\\ \mathcal{K}_2 &= \{(v_0, v_1, v_2)\}.\end{aligned}$$

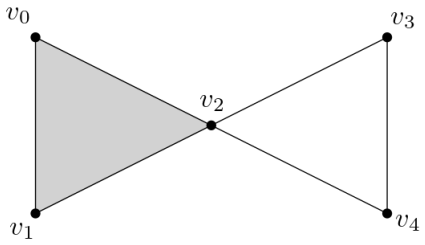



Figure 6.1: A 2-complex with one triangle, 6 edges and 5 vertices.

In this definition, simplicial complices are just finite sets of tuples grouped by length. This is great for storage, but in order to do calculations, we would like to define operations. For that reason we are going to embed the complex into the inner product vector space that it generates. See linear algebra notation in Section 2.1. Enter the *simplicial space*.

**Definition 130** (Simplicial space)**.** Let $\mathcal{K} = \mathcal{K}_0 \cup \cdots \cup \mathcal{K}_\ell$ be a simplicial complex, and $\mathcal{K}_j$ be the set of $j$-simplices in $\mathcal{K}$. For all $j \in [\ell]$ we formally define the $j$-simplicial space of $\mathcal{K}$ as

$$C_j := \mathbb{C}^{\mathcal{K}_j}. \tag{6.4}$$

We call $C = (C_\ell, \ldots, C_0)$ the *simplicial space* generated by $\mathcal{K}$.

Now, each $C_j$ is a complex inner product space, which also means that it is an Abelian group with respect to addition.

From now on we will not make much of a distinction between a simplicial complex $\mathcal{K}$ — which is a combinatorial object — and its simplicial space $C$. We will also make liberal use of the bra-ket notation later on, especially when we discuss span programs and quantum algorithms for simplicial spaces. In a slight abuse of notation, we will often say that a vector $|\tau\rangle$ is in a complex $\mathcal{K}$ even if $|\tau\rangle$ is not an element of the standard basis of any $C_j$ but a linear combination of those.

### 6.3.2 Simplicial homology

Having generalized graphs to simplicial complices, we now turn our attention to generalizing the very problem of *st*-connectivity to a problem relating higher dimensional simplicial complices with complices one dimension smaller. This will take us to the concept of homology. Before that, we need a few definitions.

**Orientations** A really important notion will be that of orientation of a simplex. An orientation is given by an ordering of the vertices of a simplex with an equivalence relation that two orderings are the same if they differ by an even permutation. For example, the ordered tuples $(v_0, v_1, v_2)$ and $(v_2, v_0, v_1)$ define the same oriented simplex, and the tuple $(v_1, v_0, v_2)$ has opposite orientation. We will always use the convention that if two oriented simplices $\sigma, \widetilde{\sigma} \in \mathcal{K}$ are the same simplex with opposite orientation, then their representatives in $\mathbb{C}^{\mathcal{K}}$ are opposite to each other. That is, $|\sigma\rangle = -|\widetilde{\sigma}\rangle$.

**Boundary maps** We have already said that a simplex is naturally bounded by a set of smaller degree simplices. We can exactly define a map, called *boundary operator* that maps a simplex to its boundary.

**Definition 131** (Boundary maps)**.** Let $\mathcal{K}$ be an oriented simplicial complex, $C$ be its simplicial space, and let $\sigma = (v_0, \ldots, v_k) \in \mathcal{K}_k \subset \mathcal{K}$ be an oriented $k$-simplex. We define the $k$-boundary operator as the map

$$\begin{aligned} \partial_k : C_k &\to C_{k-1} \\ |(v_0, \ldots, v_k)\rangle &\mapsto \sum_{i=0}^{k} (-1)^i |(v_0, \ldots, v_{i-1}, v_{i+1}, \ldots, v_k)\rangle. \end{aligned} \tag{6.5}$$

Given that there are no $(-1)$-simplices, the boundary map of the vertex set $C_0$ is defined as $\partial_0 : C_0 \to \{0\}$, $\partial_0(v) = 0$. A simple calculation gives us that this family of maps has the crucial property that $\partial_{k-1} \circ \partial_k = 0$. Another way of expressing this is to say that $\mathrm{Im}(\partial_k) \subset \mathrm{Ker}(\partial_{k-1})$.

**Chain complices** Together with the boundary maps, a simplicial complex forms an object known as a *chain complex*.

A chain complex is an algebraic structure formed by a sequence of Abelian groups $(A_k, \ldots, A_1, A_0)$ and a sequence of boundary maps $(\partial_k, \ldots, \partial_0)$ taking

one group to the next, and such that $\partial_i \circ \partial_{i+1} = 0$. Without going into too much detail, chain complices are very rich objects that contain a lot of information about the groups and how they all sit inside each other. They are studied in many contexts, from Galois theory, to abstract algebra, geometry, or topology. It is in this last context, topology, where we will use them.

It is a classic theorem [DM68; Moi13] that any topological manifold of dimension $\leq 3$ accepts a triangulation. That is, a simplicial complex that is topologically equivalent (i.e. there exists a continuous bijection with continuous inverse) to the manifold.

The opposite is also true. Any simplicial 3-complex can be embedded into a sufficiently complicated, smooth manifold (although it need not be a complete triangulation if the complex is not boundary-less). In low dimensions, we will think of points sitting on manifolds and forming simplicial complices on top of them. Sometimes we will not distinguish the manifold from its triangulation, for example, when we talk about tori.[1]

Triangulations and chain complices are used in topology because there are certain topological features, namely the homology group (hurrah, more definitions!), that are shared between a manifold and its triangulation. However, it is not necessary to embed a complex in a manifold to define the homology group. Before we define the concept of homology, lets talk about cycles.

**$k$-Cycles** Intuitively, a cycle is a curve that comes back to the initial point. Another way of saying that is that a cycle is a curve without boundaries. This is the property that we use to generalize cycles to arbitrary dimensions.

**Definition 132** ($k$-cycles and boundaries)**.** Let $C = (C_\ell, \ldots, C_0)$ be a chain complex. We say $|\sigma\rangle \in C_k$ is a *$k$-cycle* if $\partial_k |\sigma\rangle = 0$. We say $|\sigma\rangle \in C_k$ is a *$k$-boundary* if $|\sigma\rangle \in \mathrm{Im}(\partial_{k+1})$. We denote the space of all $k$-cycles and $k$-boundaries as $\mathcal{Z}_k := \ker(\partial_k)$ and $\mathcal{B}_k := \mathrm{Im}(\partial_{k+1})$ respectively.

For example, any vertex is a 0-cycle, any loop in a graph is a 1-cycle, any closed surface is a 2-cycle (e.g. a sphere, a torus, a Klein bottle), and so on. Now, because $\partial_k \partial_{k+1} = 0$, any $k$-cycle has no boundary, and any element in $\mathcal{B}_k$ is a $k$-cycle. We call this kind of cycles *separating*, because they are the boundary of a sub-complex, and therefore, they separate the complex into

[1] Pedant for toruses. Not to be confused with torii, ceremonial gates in Japanese temples.

two. The converse, however, is not always true, as there can exist $k$-cycles that are nobody's boundary.

For example, in a torus we have two different kinds of non-separable $k$-cycles, the ones that go around the handle, and the ones that go around the circumference of the torus, see Figure 6.2. We would like to have a tool that differentiates these two different kinds of cycles while classifying all meridians together, and all longitudes together. That tool is the homology group of the torus.

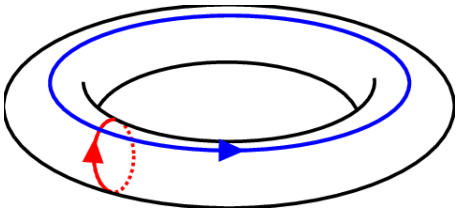


Figure 6.2: Two different kinds of cycles. The red ones are called *meridians*, while the blue ones are *longitudes*.

### Homology

We said at the beginning that the number of tricks available to a mathematician is rather limited. There is adding zero, multiplying by 1, and taking two different things and calling them the same. Homology is the third kind of trick applied to cycles.

Let $\mathcal{K}$ be a simplicial complex and let $\mathcal{C}$ be its chain complex. From the basic properties of boundary maps we have that $\mathrm{Im}(\partial_{k+1}) \subset \mathrm{Ker}(\partial_k)$. Now, since the elements of a chain complex are Abelian groups, this is an inclusion of Abelian subgroups. That means that one can take the quotient of $\ker(\partial_k)$ modulo $\mathrm{Im}(\partial_{k+1})$. That is what we call the *$k$-Homology group* of $\mathcal{C}$ (or $\mathcal{K}$),

$$H_k(\mathcal{K}) := H_k(\mathcal{C}) = \frac{\ker(\partial_k)}{\mathrm{Im}(\partial_{k+1})}, \tag{6.6}$$

This group divides $\ker(\partial_k)$ into different equivalence classes. On the one hand, we have the $k$-cycles in $\mathrm{Im}(\partial_{k+1})$, which we named *separating cycles* or boundaries. These cycles are equivalent to the null cycle in $H_k$. *Non-separating* cycles are sent to non-null elements. We say that two non-

separating $k$-cycles $\gamma$ and $\eta$ are *homologous* if $\gamma - \eta \in \mathrm{Im}(\partial_{k+1})$. This is the same as saying that together they are the boundary of a sub-manifold.

Going back to our example torus, we have that our two non-separating 1-cycles would be non-null in $H_1$. Moreover, any two meridians with opposite orientation would be the boundary of a cylinder, which means they belong in the same equivalence class in $H_1$. Similarly, any two longitudes define a ring, which makes them equivalent in $H_1$. And finally, we have that the red and blue cycle do not define the boundary of a sub-manifold, meaning that they are in different equivalence classes in $H_1$.

**Example 6.3** (Homology of a graph)**.** We have said this before and we repeat it now, graphs are equivalent to simplicial 1-complices, where the vertices are 0-simplices, and the edges are 1-simplices. It is therefore possible to define the plethora of concepts we have recently introduced, in particular, the simplicial space associated to a graph, the boundary operators $\partial_i$ from definition 131 and the homology groups $H_k(G)$.

For a graph $G$ with simplicial space $C(G) = (C_0, C_1)$ the only non-trivial boundary operator is $\partial_1$, and so the only non-zero homology groups are $H_0(C(G))$ and $H_1(C(G))$. From Eq. (6.5), this map acts on edges as $\partial_1|(u,v)\rangle = |(u)\rangle - |(v)\rangle$, which is (up to weights) the map $A$ in the *st*-connectivity span program of Definition 88.

The 0-th homology group is defined as $H_0(G) := \frac{\ker(\partial_0)}{\mathrm{Im}(\partial_1)}$. The numerator of this expression is $\ker(\partial_0) = C_0$, because $\partial_0$ is trivial. This means that all vertices are 0-cycles and $H_0(G)$ is in fact a splitting of the vertex set into equivalence classes. The denominator of the expression, which determines the equivalence classes in which $V(G)$ is split, is

$$\mathrm{Im}(\partial_1) = \mathrm{span}\{|u\rangle - |v\rangle : |u,v\rangle \in C_1\}.$$

Two vertices $u, v$ are equivalent if $|u\rangle - |v\rangle \in \mathrm{Im}(\partial_1)$. At first sight this means that a vertex $u$ is equivalent to all its neighbors, but by the transitivity of the equivalence relation, we conclude that, in fact, two vertices $u, v$ are equivalent if and only if they are connected.

Therefore, $H_0(G)$ has an equivalence class for every connected component of $G$. Remember that $C_0$ is not the set of vertices but the linear space *generated* by the vertices, $\mathbb{C}^{V(G)}$. If $u$ and $v$ are connected, not only $|u\rangle \sim |v\rangle$, but for any $z \in \mathbb{C}$, $z|u\rangle \sim z|v\rangle$ too. Therefore, $H_0(G)$ has exactly one

equivalence class for every component and every $z \in \mathbb{C}$. Denoting the number of connected components by $\kappa$, we arrive at

$$H_0(G) \cong \mathbb{C}^\kappa.$$

The other non-trivial homology group is $H_1(G)$. Recall that $\partial_2$ is trivial because $C_2 = \mathbb{C}^\emptyset = \{0\}$. It follows that $\mathrm{Im}(\partial_2) = \{0\}$ and so

$$H_1(G) := \frac{\ker(\partial_1)}{\mathrm{Im}(\partial_2)} \cong \ker \partial_1.$$

This is precisely the space generated by the cycles of the graph, which has already appeared in Chapter 5 and is integral to the *st*-connectivity span program. Its dimension is the cyclotomic number, also known as the cycle rank.

### 6.3.3 Cellular complices

We have focused on simplices so far because they are the simplest objects with which we can construct complices. It is possible, however, to envision a set similar to a simplicial complex, i.e. fulfilling the conditions of Definition 129, but using polytopes instead of simplices. After all, a polytope is characterized by being delimited by other polytopes of smaller dimension, and the intersection of polytopes is a polytope. Such a non-simplicial complex is called a *cellular complex*.

Cellular complices are interesting in their own right, and many times they are the more natural choice for a problem. For example, every finite graph (i.e. 1-complex) can be embedded on a surface in such a way that no two edges cross. The complement of the graph in the surface is a series of polygonal regions delimited by edges which we call *faces* of the graph. A graph $G = (E, V)$ together with the set of faces $F$ it forms with respect to such an embedding would be an example of a cellular complex.

Since any polytope can be subdivided into a collection of simplices — for example, through a process called barycentric subdivision, see [GT01] — we can think of cellular complices as insufficiently subdivided simplicial complices. For example, it is always possible to add edges to a planar graph so that every face is a triangle.

It is possible to directly define the spaces generated by cellular complices and boundary operators between them, thus obtaining the concepts necessary

for defining homology. However, the process can get messy, and is ultimately unnecessary. Let us explain why.

Let $\mathcal{K}$ be a cellular $k$-complex. As we just said, we can always break any polytope down and express it as the union of simplices. Let $\mathcal{K}'$ be a simplicial $k$-complex defined by a given subdivision of the polytopes of $\mathcal{K}$. For any dimension $\ell$, pick any polytope $\pi \in \mathcal{K}_\ell$, and let $\{\sigma_i : i \in [d]\} \subset \mathcal{K}'$ for some $d$ be the set of oriented simplices that add up to $\pi$, that is, their orientation is consistent with that of $\pi$. Then we can identify the vector generated by $\pi$ within $\mathbb{C}^{\mathcal{K}_\ell}$ with $\sum_{i=1}^{d} |\sigma_i\rangle \in \mathbb{C}^{\mathcal{K}'}$, and in this way see all of $\mathbb{C}^{\mathcal{K}_\ell}$ as a subspace of $\mathbb{C}^{\mathcal{K}'_\ell}$, see Example 6.4. The boundary operator acts on $\mathbb{C}^{\mathcal{K}_\ell}$ as

$$\partial_\ell : \mathbb{C}^{\mathcal{K}_\ell} \subseteq \mathbb{C}^{\mathcal{K}'_\ell} \to \mathbb{C}^{\mathcal{K}_{\ell-1}} \subseteq \mathbb{C}^{\mathcal{K}'_{\ell-1}}$$

$$\partial_\ell |\pi\rangle := \sum_{i}^{d} \partial_\ell |\sigma_i\rangle.$$

In this way, we can see a cellular complex as a simplicial complex in which we have "glued" together some adjacent simplices. For the remainder of the chapter, we will forget about cellular complices and stick to simplicial complices.

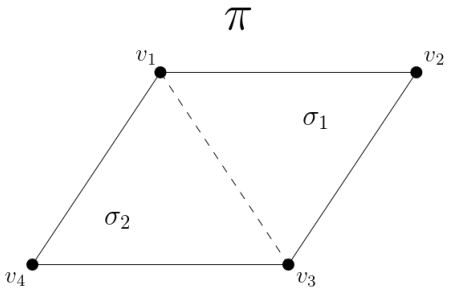



Figure 6.3: A very simple non-simplex.

**Example 6.4.** In Figure 6.3 we have a square $\pi = (v_1, v_2, v_3, v_4)$ oriented clockwise which has been subdivided into two triangles $\sigma_1 = (v_1, v_2, v_3)$ and $\sigma_2 = (v_1, v_3, v_4)$. Observe that these triangles preserve the clockwise orientation of the original square, and that a new 1-simplex has been added, the

edge $(v_1, v_3)$. Then, the boundary operator acts on $|\pi\rangle$ as

$$\begin{aligned}\partial_2|\pi\rangle =&\partial_2|\sigma_1\rangle + \partial_2|\sigma_2\rangle \\ =&|(v_1,v_2)\rangle + |(v_2,v_3)\rangle + |(v_3,v_1)\rangle + |(v_1,v_3)\rangle + |(v_3,v_4)\rangle + |(v_4,v_1)\rangle \\ =&|(v_1,v_2)\rangle + |(v_2,v_3)\rangle + |(v_3,v_4)\rangle + |(v_4,v_1)\rangle.\end{aligned}$$

In the last equality, we have used that $|(v_i, v_j)\rangle = -|(v_j, v_i)\rangle$.

### 6.3.4 Sub-complices of a complex

Span programs decide substructure problems by nature. They decide Boolean functions $f : X \subseteq [q]^n \to \{0,1\}$ in which an input $x \in X$ determines a subspace $\mathcal{H}(x) = \bigoplus_i \mathcal{H}_{i,x_i} \subseteq \mathcal{H}$ of a larger space $\mathcal{H}$. Thus, if we want to mix span programs and simplicial complices, we better be able to define substructures instantiated by a variable $x \in [q]^n$, for some $q,\ n \in \mathbb{N}$. In the case of the *st*-connectivity span program, we assumed that we had a parent graph, $G$, and that every string $x \in \{0,1\}^N$ determined a subgraph $G(x) = (E(G(x)), V(G))$, where $E(G(x)) \subseteq E(G)$. We construct sub-complices the same way we constructed subgraphs in Section 2.3.1.

Let $\mathcal{K} = \{\mathcal{K}_k, \ldots, \mathcal{K}_0\}$ be a simplicial $k$-complex. Let $N \in \mathbb{N}$, and consider the set of variables and their negations $\{x_1, \ldots, x_N, \bar{x}_1, \ldots, \bar{x}_N\}$, which we call *literals*. We assume there exists a map $L : \mathcal{K}_k \to \{x_1, \ldots, x_N, \bar{x}_1, \ldots, \bar{x}_N\}$. Then, for every $x \in \{0,1\}^N$, we define the sub-complices $\mathcal{K}_k(x)$ and $\mathcal{K}(x)$ as

$$\mathcal{K}_k(x) := \bigcup_{i\in[N]:x_i=1} \{\sigma \in \mathcal{K}_k : L(\sigma) = x_i\} \cup \bigcup_{i\in[N]:x_i=0} \{\sigma \in \mathcal{K}_k : L(\sigma) = \bar{x}_i\},$$

$$\mathcal{K}(x) := \{\mathcal{K}_k(x), \mathcal{K}_{k-1}, \ldots, \mathcal{K}_0\}$$

In particular, each $k$-simplex in $\mathcal{K}_k$ is associated with a literal $x_i$ or its negation $\bar{x}_i$, and is included in $\mathcal{K}_k(x)$ if and only if its associated literal evaluates to 1. In the simplest case, $N = |\mathcal{K}_k|$, but this need not be the case. This way of defining sub-complices of a complex is also suitable for non-simplicial complices.

## 6.4 A span program for simplicial homology

The setup of chain complices, cycles, boundaries, and homological equivalence translates well to the language of span programs.

Let $\mathcal{K}$ be an oriented simplicial $k$-complex, and $\mathcal{C}$ be its chain complex. Because span programs are substructure problems specified by some input, let $x \in \{0,1\}^N$ specify a sub-complex $\mathcal{K}(x) \subseteq \mathcal{K}$ in the manner of Section 6.3.4. Then, for every $(k-1)$-cycle $|\tau\rangle$, we can define the span program $P = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$, where

$$\mathcal{H}_{i,1} = \text{span}\{|\sigma\rangle : \sigma \in \mathcal{K}_k, L(\sigma) = x_i\}, \quad \mathcal{H}_{i,0} = \text{span}\{|\sigma\rangle : \sigma \in \mathcal{K}_k, L(\sigma) = \overline{x}_i\},$$
$$\mathcal{H} = \bigoplus_{i\in[N], b\in\{0,1\}} \mathcal{H}_{i,b} = C_k, \qquad \mathcal{H}(x) = \bigoplus_{i\in[N]} \mathcal{H}_{i,x_i} =: C_k(x),$$
$$\mathcal{V} = C_{k-1} = \mathbb{C}^{\mathcal{K}_{k-1}}, \qquad A = \partial_k, \qquad |\tau\rangle \in \mathcal{Z}_{k-1}. \tag{6.7}$$

Remember that a span program decides whether $|\tau\rangle \in A\Pi_{\mathcal{H}(x)} = \text{Im}(\partial_k(x))$, where $\partial_k(x)$ denotes the restriction $\partial_k|_{C_k(x)}$. In other words, this span program decides whether the $(k-1)$-cycle $|\tau\rangle$ is null-homologous in $C(x)$, or, for the humans among us, whether $|\tau\rangle$ is a boundary in $\mathcal{K}(x)$. Alternatively, choosing as a target the difference $|\tau\rangle = |\gamma\rangle - |\eta\rangle$, $|\gamma\rangle, |\eta\rangle \in \mathcal{Z}_{k-1}$, our span program decides whether $\gamma$ and $\eta$ are homologous to each other in $\mathcal{K}(x)$. This is exactly what the *st*-connectivity span program decides.

Any sub-complex (for reference, think sub-manifold, sub-surface, subgraph, etc.) of $C_k(x)$ that has $|\tau\rangle$ as its boundary would be a positive witness.

Similarly, if $|\tau\rangle \notin \text{Im}(\partial_k(x))$ then there must exist negative witnesses. That is, there must exist elements $|\omega\rangle \in \mathcal{V}$ such that $\langle\omega|\tau\rangle \neq 0$ and $\langle\omega|\partial_k(x) = 0$. And they do! Take, for example, $|\omega\rangle = \Pi_{\text{Im}(\delta_k(x))^\perp}|\tau\rangle$. This vector must be non-zero since $|\tau\rangle \notin \text{Im}(\partial_k(x))$ and $\text{Im}(\partial_k(x))$ is a subspace of $\mathcal{V}$.

By framing these simplicial homology problems in terms of span programs we add new concepts to our vocabulary in the form of witnesses. It is not just that a cycle is or is not in a given subspace, in the span program there exist vectors in $\mathcal{H}(x)$ or $\mathcal{V}$ to witness that. Moreover, we can make a quantum algorithm that decides it without explicitly finding the witnesses.

Let us put what we just said in formal statements.

**Claim 6.** Let $\mathcal{K}$ be a $k$-complex and $x \in \{0,1\}^N$ specify a sub-complex $\mathcal{K}(x) \subseteq \mathcal{K}$. Let $|\tau\rangle \in \mathcal{Z}_{k-1}$. Then the span program $P = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$

defined in Eq. (6.7) exactly decides the function

$$f_{\mathcal{K},X,\tau} : X \subseteq \{0,1\}^N \to \{0,1\},$$
$$f_{\mathcal{K},X,\tau}(x) = \begin{cases} 1 & \text{if } |\tau\rangle \text{ null-homologous in } C(x) \\ 0 & \text{else.} \end{cases} \tag{6.8}$$

By Theorem 45, any span program deciding a function $f$ can be compiled into a quantum algorithm that decides the same function with bounded error. Thus, we obtain:

**Theorem 133.** *Let $f_{\mathcal{K},X,\tau}$ be the function defined in Eq. (6.8). Then, there exists a quantum algorithm that computes $f$ with bounded error.*

Unfortunately, this problem is so general that we cannot make any statements for the complexity of this algorithm. Every dimension requires its own analysis, as we will shorty illustrate for dimension 1 (graphs) and 2 (surfaces).

### 6.4.1 *st*-connectivity revisited

And so we return to the origin, *st*-connectivity. In *st*-connectivity we have a graph $G = (E, V)$, two connected vertices $s$ and $t$, and we would like to know if $s$ and $t$ are still connected in a subgraph $G(x) = (E(x), V)$ for every $x \in X$.That is to say that $s - t$ is the boundary of a subgraph of $G(x)$. We can picture $G$ as a simplicial 1-complex, where the edges are 1-dimensional simplices (because they are tuples of two), and the vertices are the 0-dimensional simplices. The *st*-connectivity span program maps the space $\mathcal{H} = \text{span}\{|e\rangle : e \in E\}$ to $\mathcal{V} = \text{span}\{|v\rangle : v \in V\}$, which are nothing but the simplicial spaces of $G$. The map taking $\mathcal{H}$ to $\mathcal{V}$ is $A = \sum_{(u,v)\in E(G)} (|u\rangle - |v\rangle) \langle u, v| = \partial_1$ and the target is the difference between two vectors in $\ker(\partial_0) = \mathcal{V}$.

Following the reasoning of the previous section, this span program tests if the 0-cycles $s$ and $t$ are homologically equivalent. As we have seen in Example 6.3, the equivalence classes of $H_0(G(x))$ are the components of $G(x)$, so the span program tests whether $s$ and $t$ are in the same component; that is, whether $s$ and $t$ are connected.

The complexity of the span program algorithm deciding *st*-connectivity is exactly characterized in Corollary 91 by the product of an effective resistance and an effective capacitance.

## 6.5 A span program for homology of surfaces

A surface is a compact, connected subset of $\mathbb{R}^3$ that is locally 2-dimensional. That is, for every point $z$ in the surface, there exists a neighbourhood $\mathcal{B}$ of $z$ and a continuous, invertible map $\varphi$ from $\mathcal{B}$ to either the disc $\{(x,y) \in \mathbb{R}^2 : x^2+y^2 \leq 1\}$ or the half-disc $\{(x,y) \in \mathbb{R}^2 : x^2+y^2 \leq 1, x \geq 0\}$. A continuous, invertible map between two topological spaces (read, surfaces, curves, points) with a continuous invertible inverse is called a *homeomorphism*. A surface is boundary-less if it has no points with a half-disc neighbourhood. Otherwise the set of such points forms the boundary of the surface. Any connected component of the boundary is homeomorphic to a $S_1$, i.e. a circle.

Surfaces can be orientable or non-orientable. A non-orientable surface is any surface that contains a sub-surface homeomorphic to the Möbius strip. The projective plane, the Klein bottle and the Möbius strip itself are all non-orientable surfaces. An orientable surface is a surface that does not contain a Möbius strip. In the remainder of this section we will deal with orientable surfaces.

All oriented surfaces without boundary are characterized up to homeomorphism by their *genus*, which is the maximum number of disjoint simple cycles on the surface whose complement is connected. For visualization, these are tori with multiple handles. The number of handles determining the genus of the surface. Surfaces with boundaries are characterized by their genus and the number of connected components of the boundary.

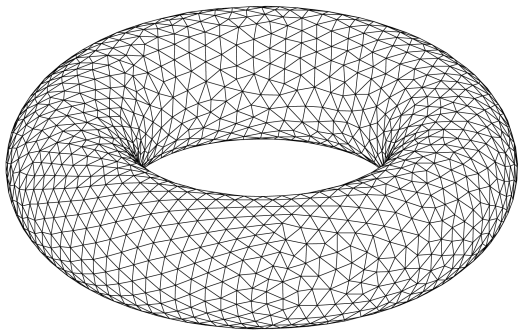


Figure 6.4: A triangulation of a torus.

Let $\Sigma$ be a surface. A triangulation of $\Sigma$ is a simplicial 2-complex $\mathcal{K} = (F, E, V)$ consisting of triangles, also known as faces $f \in F$, edges $e \in E$

and vertices $v \in V$, together with a map that takes vertices of $\mathcal{K}$ to points in $\Sigma$, edges to simple, disjoint paths, and faces to the disjoint subsets of $\Sigma$ homeomorphic to discs that are delimited by the edges of the face (or rather, their paths in $\Sigma$). In a slight abuse of language, we will often use the term triangulation to refer to the simplicial complex therein.

Observe that the subcomplex $G = (E, V) \subset \mathcal{K}$ with this same embedding forms what is known as a *graph embedding*. There is an extensive literature on surface embedded graphs [Cab10; Eri11; CVL11], in which graphs drawn on a surface are used to study its topology.

We want to stress that a triangulation and a graph embedding are not the same thing. In a graph embedding, the faces are merely the interior of "minimal" cycles on the surface, whereas in a triangulation, the faces are part of the complex, and it matters whether a triangle is present or not.

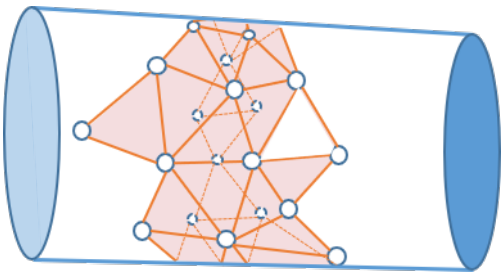


Figure 6.5: A 2-complex embedded on a cylinder. Notice that it is missing a face.

When reduced to triangulations, the span program for simplicial homology described in Eq. (6.7) is the following. Let $\mathcal{K} = (F, E, V)$ be a triangulation of a surface $\Sigma$, and let $x \in \{0,1\}^{|F|}$ define a sub-complex $\mathcal{K}(x) = (F(x), E, V)$. Then the span program that decides if the target cycle is null-homologous in the first homology group $H_1(\mathcal{K})$ is defined as $P = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$, where

$$\begin{aligned} \mathcal{H} &= \mathbb{C}^F, \qquad & \mathcal{H}(x) &= \mathbb{C}^{F(x)}, \\ \mathcal{V} &= \mathbb{C}^E, \qquad & A &= \partial_2, \qquad & |\tau\rangle &\in \mathcal{Z}_1. \end{aligned} \tag{6.9}$$

Remember that the first homology group classifies 1-cycles in the complex. In other words, it classifies the cycles in the edgeset of the triangulation, and since the complex and the surface are homeomorphic to each other, it too classifies the cycles of the surface.

**Definition 134** (Contractible and separating cycles)**.** A cycle $\sigma \in \Sigma$ is *contractible* if it is *homotopous* to a disc, that is, it can be continuously deformed to a single point. These are sometimes referred to as *trivial* or *null-homotopic* cycles. A cycle $\sigma \in \Sigma$ is *separating* if it is the boundary of a sub-surface of $\Sigma$. If $\Sigma$ is boundary-less, then any separating cycle is actually the boundary of two sub-surfaces.

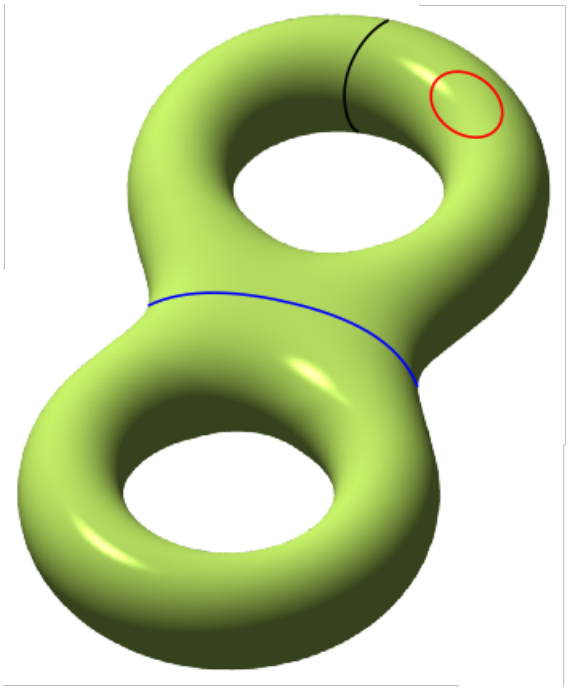


Figure 6.6: Different kinds of cycles in a genus-2 surface.

From these definitions we see that any contractible cycle is also separating since it is the boundary of its interior disc. In Figure 6.6 we have a genus-2 surface with three cycles depicted. The red cycle is contractible (and separating), while the blue cycle is separating but not contractible. Indeed, if we cut the double torus along the blue cycle we are left with two surfaces not homeomorphic to a disc. Thus, these two cycles belong to the same homology class but cannot be continuously deformed to each other. They are not homotopous. The black cycle is non-separating. Other non-separating cycles would be the ones that go around the holes, the ones that go around

the contour of the double torus, the ones that go around the other handle or the ones that go around both holes, perpendicular to the blue cycle.

All cycles within these families are clearly homologous, any pair defines the boundary of a subsurface, and these families are pairwise non-homologous (and only pairwise!).

In short, what we have seen is that for boundary-less surfaces, null cycles in $H_1(\mathcal{K})$ are separating, and non-null cycles in $H_1(\mathcal{K})$ are non-separating cycles. For surfaces with boundaries the situation is a little more complicated. Null cycles are still the boundary of a subsurface, so they are still separating. Non-separating cycles are still non-null. The difference now is that there can be non-null, separating cycles. Indeed, the sub-surfaces that result from cutting along a separating cycle have it as part of their boundary, but it need not be the only element in the boundary. Hence, we see that the null cycles are actually separating cycles that are the sole boundary of at least one of the sub-surfaces they generate.

Regardless of a surface boundary, the triangulation is homeomorphic to the surface, and so all these notions hold for the complex. Sub-surfaces are equivalent to sub-complices formed by contiguous triangles and their edges and vertices. Cycles on the surface are equivalent to cycles on the edge-set of the sub-complex, and null-cycles, in particular, are rather simple. They are simply the boundary of a subsurface. For a triangulated surface, let $|\Sigma|$ be the number of triangles in its corresponding complex.

**Lemma 135.** *Let $\mathcal{K} = (F, E, V)$ be a triangulation of a surface and $P = (\mathcal{H}, \mathcal{V}, A, |\tau\rangle)$ be the span program defined in Eq. (6.9). Let $x$ define a sub-triangulation $\mathcal{K}(x)$ such that $|\tau\rangle$ is null-homologous in $\mathcal{K}(x)$, then $w_+(x, P) \leq \|\partial_2^{-1}|\tau\rangle\| \leq |F(x)|$.*

*Proof.* Let $\Sigma(x)$ be the subsurface of $\Sigma$ that is covered by $F(x)$ by the homeomorphism of the triangulation. Since $|\tau\rangle$ is null-homologous, it is separating, so there exists a subsurface of $\Sigma(x)$ that has (a cycle that maps to) $|\tau\rangle$ as its boundary, call it $\Sigma_\tau$. The image of that surface through the homeomorphism must be a sum of contiguous, non-overlapping triangles in $F(x)$. In other words, there exists $|w\rangle = \sum_{i=1}^{d} |f_i\rangle$ for some $f_i \in F(x)$ and some $d$ such that $A|w\rangle = |\tau\rangle$. Since this is contained in $F(x)$ and has no repeated triangles (because it is a proper sub-surface) the number of terms is $d = |\Sigma_\tau| \leq |F(x)|$, and so $w_+(x, P) = d \leq |F(x)|$. □

The reader would be excused to think that this proof sounded like a

Monthy Python sketch. The negative witnesses are a little bit more nuanced.

**Lemma 136.** *Let $\mathcal{K} = (F, E, V)$ be a triangulation of an orientable surface without boundaries. and $P = (\mathcal{H}, \mathcal{V}, A, \tau)$ be the span program defined in Eq. (6.9). Let $x$ define a sub-triangulation $\mathcal{K}(x)$ such that $|\tau\rangle$ is a self-avoiding non-null cycle in $\mathcal{K}(x)$. If $|\tau\rangle$ is also non-null in $\mathcal{K}$, there exists a non-zero negative witness $|\omega\rangle$ with negative witness size $\|\langle\omega|A\|^2 = 0$, and if $|\tau\rangle$ is null-homologous in $\mathcal{K}$, then $w_-(x, P) \leq 2$.*

*Proof.* As we have already discussed, homology in surfaces with boundaries is a bit more complicated than just separability. While it is true that null-homologous cycles are separating, it is possible to have separating cycles that are also not null-homologous. That means we have to split the proof in three different cases.

**Case 1** We begin by proving the case where $|\tau\rangle$ is non-separating in $\mathcal{K}(x)$. Since $\mathcal{K}(x)$ defines a sub-surface of the surface $\Sigma$ triangulated by $\mathcal{K}$, $|\tau\rangle$ is also non-separating in $\mathcal{K}$ (hence, non-null). Consider the surface $\widetilde{\Sigma}$ obtained by cutting $\Sigma$ along $\tau$ (see Figure 6.7). By assumption, this surface is connected and has (at least) two boundaries corresponding to two copies of $|\tau\rangle$. Pick a vertex $|v\rangle$ in $|\tau\rangle$. After the cut, $|v\rangle$ has two copies in $\widetilde{\Sigma}$, call them $|v\rangle$ and $|v'\rangle$. Since the surface is connected, there exists a shortest path $p$ going from $|v\rangle$ to $|v'\rangle$. This path is simple and so it turns into a simple directed cycle $|\gamma\rangle$ in $\mathcal{K}$ that crosses $|\tau\rangle$ exactly once.

By assumption the surface is orientable, so all cycles are what is known as two-sided. That is, given a direction for a cycle, the notions of left and right neighbours of the vertices in the cycle are well defined. Let $|u\rangle$ be a vertex in a directed cycle $|\gamma\rangle$, the left neighbourhood of $|u\rangle$ is the set

$$L_\gamma(u) = \{v \in V : |(u, v)\rangle \in E, v \text{ is immediately to the left of } u\}. \tag{6.10}$$

Let us assume that $|\gamma\rangle$ crosses $|\tau\rangle$ from right to left through the vertex $|u^*\rangle$. The negative witness is the state

$$|\omega\rangle = \sum_{\substack{u\in\gamma \\ v\in L_\gamma(u)}} |(u, v)\rangle. \tag{6.11}$$

Clearly, $|\tau\rangle$ has one edge, and only one, in $L_\gamma(u^*)$, so $\langle\omega|\tau\rangle = 1$. We claim that this state is orthogonal to $\mathrm{Im}(A)$. Indeed, remember that every non-null

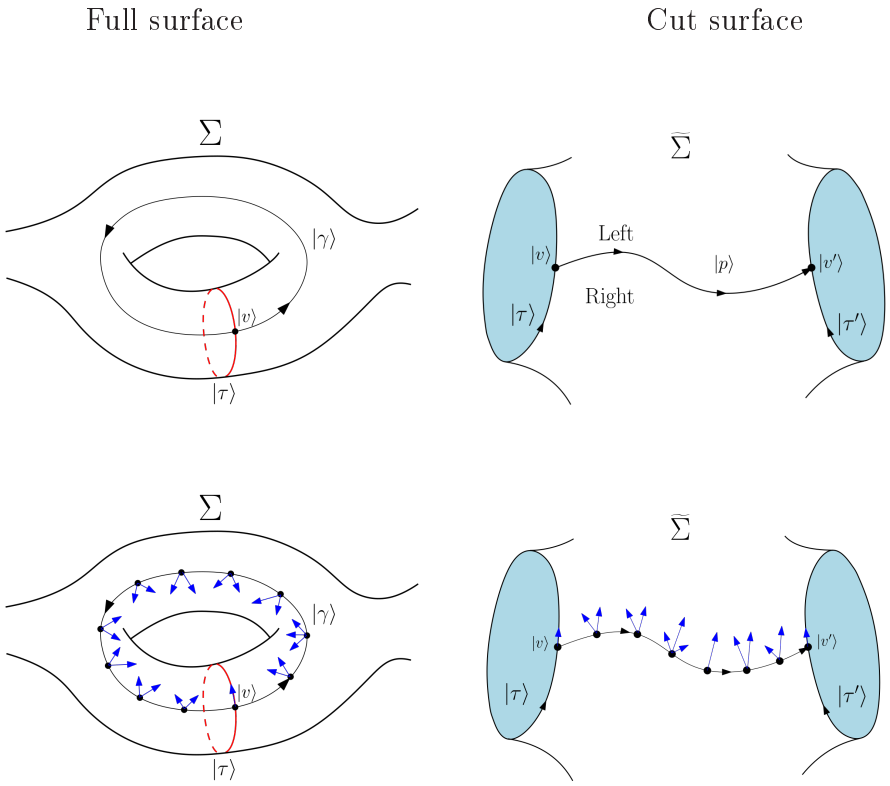



Figure 6.7: Example of a surface with a non-separating cycle $|\tau\rangle$, before and after cutting along $|\tau\rangle$. The negative witness is the set of directed edges to the left of the cycle (blue).

cycle $\sigma$ is separating and splits $\Sigma$ into two sub-surfaces. Let us consider a walker that walks around the cycle $\gamma$ and starts its walk in one of the two sub-surfaces. Since the walker eventually goes back to the start, the number of times $\gamma$ crosses into the second sub-surface equals the number of times it crosses back into the first one. The inner product $\langle\omega|\sigma\rangle$ is then zero because for every time $\sigma$ crosses $\gamma$ from left to right there is one crossing from right to left. We conclude that $\langle\omega|A = 0$.

In the other two cases, the cycle $|\tau\rangle$ is separating but not null-homologous in $\mathcal{K}(x)$. The difference will be the behaviour of $|\tau\rangle$ in $\mathcal{K}$.

**Case 2** First we deal with the case that $|\tau\rangle$ is separating in $\mathcal{K}$ but not null-homologous. In this situation we have that cutting $\mathcal{K}$ along $|\tau\rangle$ splits the surface $\Sigma$ in two sub-surfaces, $\Sigma_1$ and $\Sigma_2$. However, since $|\tau\rangle$ is non-null in $\mathcal{K}$, $|\tau\rangle$ cannot be the only element of the boundary of either sub-surface, see Fig. 6.8 for an example. Let $|\sigma_1\rangle$ be a cycle in the boundary of $\Sigma_1$ and $|\sigma_2\rangle$ be a cycle in the boundary of $\Sigma_2$. Observe that $|\sigma_1\rangle$ and $|\sigma_2\rangle$ can overlap with $|\tau\rangle$, but there must exist at least one edge in $|\sigma_1\rangle$ that is not in $|\tau\rangle$, because $|\tau\rangle$ is self-avoiding. In fact, we can assume that there exists a vertex $|u\rangle$ in $\sigma_1$ that is not adjacent to $|\tau\rangle$ because we can always subdivide $\mathcal{K}$ using barycentric subdivision, which would split the edge that is not in $|\tau\rangle$ into two edges with a vertex in between, all not in $|\tau\rangle$. Since this edge sits on $\sigma_1$ but not in $|\tau\rangle$, it must be in $\Sigma_1$ but not in $\Sigma_2$ The same is true for $|\sigma_2\rangle$. Let $u, v$ be vertices in $|\sigma_1\rangle$ and $|\sigma_2\rangle$ not adjacent to $|\tau\rangle$. Then the shortest path $|p\rangle$ from $u$ to $v$ must be a simple path that crosses $|\tau\rangle$ an odd number of times.

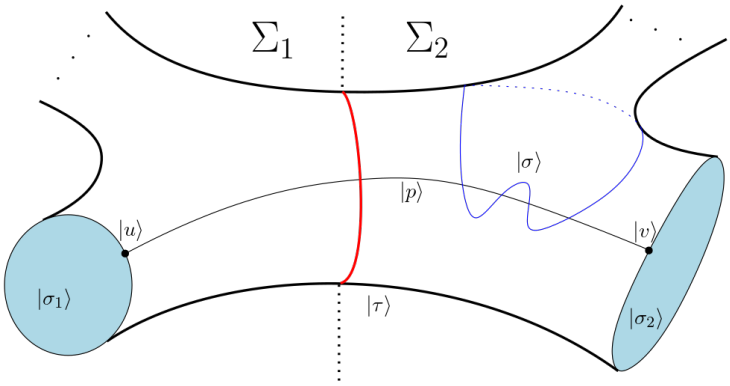



Figure 6.8: $|\sigma_1\rangle$ and $|\sigma_2\rangle$ are both part of the boundary of $\Sigma$. In this case $|\tau\rangle$ is separating but not null-homologous in $\mathcal{K}$.

Just like we did for the cycle $|\gamma\rangle$ in the previous case, there exist left and right neighbourhoods of $|p\rangle$ so we define the negative witness as

$$|\omega\rangle = \sum_{\substack{u \in p \\ v \in L_p(u)}} |(u, v)\rangle. \tag{6.12}$$

Now, $\langle\omega|\tau\rangle$ is the number of times $|\tau\rangle$ crosses $|p\rangle$ from right to left minus the times it crosses from left to right, and since the number of crossings is odd, we conclude that $\langle\omega|\tau\rangle = \pm 1$. We can assume that it is actually 1,

otherwise we pick $-|\omega\rangle$ as negative witness and proceed on to computing $\langle\omega|A$. Let $|\sigma\rangle$ be any null-homologous cycle in $\mathcal{K}$, that is $|\sigma\rangle \in \mathrm{Im}(A)$. This cycle necessarily cuts $\Sigma$ into two sub-surfaces, and, $|\sigma\rangle$ is the sole boundary of one of such sub-surfaces. That means that both $|\sigma_1\rangle$ and $|\sigma_2\rangle$, which are elements in the boundary of $\Sigma$, are in the same sub-surface. Therefore, if the path $p$ crosses the cycle $|\sigma\rangle$ it must do so an even number of times, for both ends are on the same side of $|\sigma\rangle$. This means that $\langle\omega|\sigma\rangle = 0$. We conclude that $\langle\omega|A = 0$.

**Case 3** Last but not least, we tackle the case where $|\tau\rangle$ is separating in $\mathcal{K}(x)$ and null-homologous in $\mathcal{K}$. Since $|\tau\rangle$ is null-homologous in $\mathcal{K}$, it separates $\Sigma$ into two sub-surfaces $\Sigma_1$ and $\Sigma_2$ and it is the sole boundary of at least one of them, say, $\Sigma_1$. But $|\tau\rangle$ is also separating and non-null in $\mathcal{K}(x)$, so cutting $\mathcal{K}(x)$ along $|\tau\rangle$ results in two disconnected sub-surfaces, $\Sigma_1(x) \subseteq \Sigma_1$ and $\Sigma_2(x) \subseteq \Sigma_2$, that have $|\tau\rangle$ as part of their boundary. Observe that $|\tau\rangle$ cannot be all of their boundary, otherwise $|\tau\rangle$ would be null-homologous in $\mathcal{K}(x)$. Hence, there must exist at least two cycles $|\sigma_1\rangle$ and $|\sigma_2\rangle$ in the boundaries of $\Sigma_1(x)$ and $\Sigma_2(x)$, and, at least, $|\sigma_1\rangle$ is not in the boundary of $\Sigma_1$. In other words, $\Sigma_1(x)$ has a new hole, see Figure 6.9 for an example.

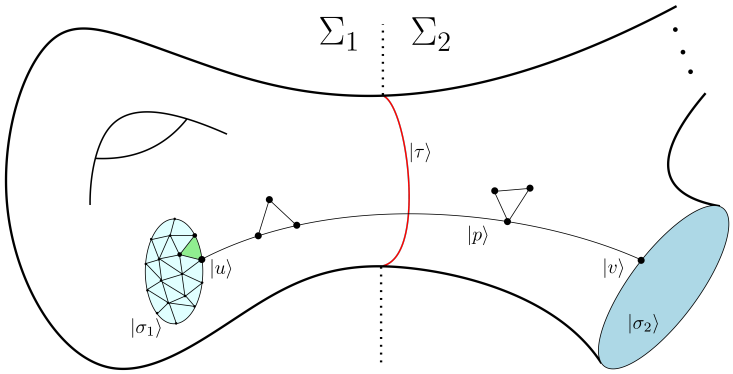



Figure 6.9: The target $|\tau\rangle$ divides $\Sigma$ into two subsurfaces and is actually the boundary of one, $\Sigma_1$. The cycle $|\sigma_1\rangle$ is a boundary in $\mathcal{K}(x)$ but not in $\mathcal{K}$, and the green triangle is the only face in $\mathcal{K}$ that is not orthogonal to $\langle\omega|A$.

Following exactly the same construction as before, we take a path $p$ from $|\sigma_1\rangle$ to $|\sigma_2\rangle$ that crosses $|\tau\rangle$ an odd number of times. The state $|\omega\rangle$ from Eq. (6.12) is still a witness since any null-homologous cycle in $\mathcal{K}(x)$ crosses

the path an even number of times. Hence $\langle\omega|A\Pi_{\mathcal{H}(x)} = 0$. Let us compute the witness size.

$$\|\langle\omega|A\|^2 = \sum_{f\in F} \|\langle\omega|A|f\rangle\|^2. \tag{6.13}$$

For a face $|f\rangle = |(u,v,w)\rangle$, let $|c(f)\rangle := A|f\rangle = |(u,v)\rangle + |(v,w)\rangle + |(w,u)\rangle$ be its boundary. For every $|f\rangle \in \mathcal{K}(x)$, $\langle\omega|c(f)\rangle = 0$ since $|c(f)\rangle$ is contractible, and thus, null-homologous. Furthermore, if $|f\rangle$ has two vertices in $p$ and the third one is to the left of $p$, then necessarily $\langle\omega|c(f)\rangle = 0$, and if $|f\rangle = |(u,v,w)\rangle$ has one vertex $u$ in $p$ which is neither the first or last, then $\langle\omega|c(f)\rangle = \langle\omega|u,v\rangle - \langle\omega|w,u\rangle = 0$. It follows that any face not orthogonal to $\langle\omega|A$ must have one vertex in $p$ that is either the first or the last one and only one edge incident on $|\omega\rangle$. Finally, we recall that by assumption, $|\sigma_1\rangle$ is a null cycle in $\Sigma_1$, so the interior of $|\sigma_1\rangle$ must be covered by a subsurface of $\mathcal{K}$. Then it must be the case that there exists a face in $\mathcal{K}$ adjacent to $|\sigma_1\rangle$ that has overlap 1 with $\langle\omega|A$. If, in addition, $|\sigma_2\rangle$ is also not in the boundary of $\Sigma_2$, then there can be another triangle $|f\rangle$ adjacent to $p$ such that $\langle\omega|A|f\rangle = 1$. We conclude that

$$\|\langle\omega|A\|^2 = \sum_{f\in F} \|\langle\omega|A|c(f)\rangle\|^2 \leq 2. \tag{6.14}$$

□

The fact that in this last lemma we have $w_-(x) = 0$ for two of the three cases should immediately raise an eyebrow. It would seem that the formalism in Chapter 3 is not equipped to deal with instances that have non-zero witnesses with zero witness size. Indeed, if that were the case, Ref. [ACK20] implies that the query complexity of deciding those instances would be 0, which seems rather odd. This seemingly impossible result is explained by the fact that in those two cases, the function $f_{\mathcal{K},X,\tau}$ decided by the surface homology span program is *constant*. More importantly, we can check that it is so without making any queries to $x$. That is because in both cases $|\tau\rangle$ is non-null in $\mathcal{K}(x)$ but also in $\mathcal{K}$, but that implies that $|\tau\rangle$ must be non-null in $\mathcal{K}(x) \subset \mathcal{K}$ for all $x$. We discuss why have bothered to find negative witnesses to a constant function in Section 6.6.

Together, Lemma 135 and Lemma 136 prove the following theorem.

**Theorem 137.** *Let $\mathcal{K} = (F, E, V)$ be a triangulation of an orientable surface $\Sigma$, let $|\tau\rangle$ be a self-avoiding 1-cycle in $\mathcal{K}$, let $X \subseteq \{0,1\}^N$, and let $f_{\mathcal{K},X,\tau}$ be defined as in Eq. (6.8). Then, there exists a quantum algorithm that decides $f_{\mathcal{K},X,\tau}$ with bounded error making $\mathcal{O}(\sqrt{n+g})$ queries, where $n = |V|$ and $g$ is the genus of $\Sigma$.*

*Proof.* By Lemma 135, for every $x \in f^{-1}_{\mathcal{K},X,\tau}(1)$, we have that $w_+(x) \leq |F(x)|$. Hence, we bound the positive complexity as $W_+ \leq |F|$. By Euler's formula $|V| - |E| + |F| + 2g = 2$ (see, e.g. [MT01]), so we have that $|F| = \mathcal{O}(|V| + g)$ and $|E| = \mathcal{O}(|V| + g)$. We conclude that $W_+ = \mathcal{O}(|V| + g)$.

By Lemma 136, the negative witness size is $w_-(x) \leq 2$ for all $x \in f^{-1}_{\mathcal{K},X,\tau}(0)$. We conclude that $W_- = \mathcal{O}(1)$. By Theorem 45 the result follows. □

## 6.6 Discussion

In this chapter we have seen the *st*-connectivity problem as a particular instance of a boundary problem. We have then proposed modifications of the *st*-connectivity span program for other boundaries, arriving at span programs for the AND and OR of $s_i t_j$-CONN$_{G,X}$, for some given pairs of vertices $s_i, t_j$.

One such example was already discussed in Section 5.6, where we used the span program for $\bigwedge_i st_i$-CONN$_{G,X}$ to solve the problem of graph connectivity. The same span program can be repurposed to simply compute the AND of $k$ bits (or the OR). Indeed, consider the star graph $G$ with vertex set $V(G) = \{s, v_1, \ldots, v_k\}$ and edges $E(G) = \{(s, v_i)\}_{i=1}^k$. Let $x \in \{0,1\}^N$ determine a subgraph $G(x)$ in the usual way, i.e. each edge is included if its corresponding literal $x_i = 1$. Clearly this graph is connected if and only if $\bigwedge_{i=1}^k x_i = 1$. Following the analysis in Section 5.6, one arrives at $W_- = \mathcal{O}(k^2)$, and $W_+ = (1/k)$, and so, this span program computes AND$_k$ with query complexity $\mathcal{O}\left(\sqrt{k}\right)$. Since the graph is so simple, the time complexity is the same up to logarithmic factors.

In [JK17], the authors give a way to encode AND/OR formulas in terms of *st*-connectivity problems on graphs. The graph that encodes AND$_k$ in their formulation is a single path of length $k$ with the end-points identified as $s$ and $t$. Naturally, that graph also decides AND$_k$ with query complexity $\mathcal{O}\left(\sqrt{k}\right)$. The difference, is that our graph for computing AND$_k$ has diameter 1 instead of $k$. It remains an open question whether these new span

programs for AND and OR of $s_i t_j$-$\text{CONN}_{G,X}$ can be used to map formula evaluation problems to boundary problems on graphs with small diameter, and most importantly, whether that would be a useful thing to do.

We have also discussed extensively simplicial homology and argued that span programs are a natural way of encoding questions about simplicial homology. A prime example of a span program for simplicial homology is the $st$-connectivity span program. In Section 6.5 we give a second example, a span program that decides whether a cycle on a surface is homologically null on an input-dependent subsurface. The role of cycle homology on surfaces is often a subject of study in the classical literature on surface-embedded graphs [Eri11; Cab10; CVL11; CEN09]. However, the focus in those papers has been to *find* short non-trivial cycles on a surface, rather than deciding whether a cycle is null-homologous on a sub-surface.

In some ways, the function we have constructed is a very artificial one. A big open question, then, is to find a utility for the span program algorithm implied by Theorem 137. Even if there is none, the algorithm serves us to prove a point, i.e. that span programs can be used to construct quantum algorithms for topology problems. In this particular case, we have a strong suspicion that the quantum algorithm from Theorem 137 offers a quadratic speed-up over any classical algorithm deciding the same function. That is because for any separating cycle $|\tau\rangle$ on a triangulation $\mathcal{K}$, a single missing triangle on the surface of $\mathcal{K}(x)$ is enough to make $|\tau\rangle$ non-null in $\mathcal{K}(x)$. Therefore, any classical algorithm must at least check that all the triangles interior to $|\tau\rangle$ in $\mathcal{K}$ are still present in $\mathcal{K}(x)$. In the worst case, the sub-surface of $\mathcal{K}$ delimited by $|\tau\rangle$ has size $\mathcal{O}(n+g)$, which is quadratically bigger than $\mathcal{O}\left(\sqrt{n+g}\right)$, our span program algorithm's query complexity.

As a last remark, we want to comment on why we provided negative witnesses for the cases 1 and 2 in Lemma 136, where the function $f_{\mathcal{K},X,\tau}$ is constant.

There are two reasons for this. First, in computational geometry one is not typically given a triangulation and an oracle to a sub-triangulation. Rather, the initial promise is simply that one is given a triangulation of a surface with $f$ faces, $m$ edges and $n$ vertices. In the query oracle model, that would correspond to having $\mathcal{K}$ be the complete simplicial 2-complex on $n$ vertices (which is not homeomorphic to any surface) with the promise that for every $x$, $\mathcal{K}(x)$ is a triangulation of a surface. With this promise on $\mathcal{K}$ and $x$, and the additional promise that $|\tau\rangle$ is a cycle on $\mathcal{K}(x)$, the witnesses we

have constructed in Lemma 135 and Lemma 136, would still be witnesses, which is why we computed them in the first place, but the negative witness sizes would be non-zero. Analysing the complexity of deciding $f_{\mathcal{K},X,\tau}$ with these promises remains an open problem.

Second, we believe that understanding what constitutes a negative witness for a target $|\tau\rangle$ composed of a single cycle will help us find witnesses for targets of the form $|\tau\rangle = |\gamma\rangle - |\eta\rangle$, where $|\gamma\rangle$, $|\eta\rangle$ are non-separating cycles on $\mathcal{K}(x)$. In some sense this is the more natural generalization of $st$-connectivity, where every vertex by itself is non-null, but a combination $|s\rangle - |t\rangle$ can be null in $H_0$, the zero-th homology group of the graph. For example, Lemma 136 tells us that in a simple torus, any meridian defines a witness for any longitude. In higher genus, we can argue that if $|\gamma\rangle$ and $|\eta\rangle$ are cycles around different "handles", then there must exist witnesses for one of the cycles that are orthogonal to the other cycle. Formalizing this argument is also an open problem.

# Bibliography


[ABK16] Scott Aaronson, Shalev Ben-David, and Robin Kothari. *Separations in query complexity using cheat sheets.* Proceedings of the forty-eighth annual ACM symposium on Theory of Computing (2016). DOI: 10.1145/2897518.2897644.

[Abr07] Nair Abreu. *Old and new results on algebraic connectivity.* Linear Algebra Appl. **423** (2007), pp. 23–53.

[ACK20] Noel T. Anderson, Jay-U Chung, and Shelby Kimmel. *Leveraging Unknown Structure in Quantum Query Algorithms.* 2020. arXiv: 2012.01276 [quant-ph].

[Amb02] Andris Ambainis. *Quantum Lower Bounds by Quantum Arguments.* Journal of Computer and System Sciences **64**, 4 (2002), pp. 750–767. DOI: https://doi.org/10.1006/jcss.2002.1826.

[Amb07] Andris Ambainis. *Quantum walk algorithm for element distinctness.* SIAM Journal on Computing **37**, 1 (2007), pp. 210–239. DOI: 10.1137/S0097539705447311.

[Amb10] Andris Ambainis. *Quantum search with variable times.* Theory of Computing Systems **47**, 3 (2010). DOI: 10.1007/s00224-009-9219-1.

[Āri16] Agnis Āriņš. *Span-program-based quantum algorithms for graph bipartiteness and connectivity.* Mathematical and Engineering Methods in Computer Science (MEMICS 2015). Ed. by Jan Kofroň and Tomáš Vojnar. Vol. 9548. Lecture Notes in Computer Science. Springer International Publishing, 2016, pp. 35–41. ISBN: 978-3-319-29817-7. DOI: 10.1007/978-3-319-29817-7_4.

[Bel12a] Aleksandrs Belovs. *Learning-graph-based quantum algorithm for k-distinctness.* Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2012). 2012, pp. 207–216. DOI: 10.1109/FOCS.2012.18.

[Bel12b] Aleksandrs Belovs. *Span programs for functions with constant-sized 1-certificates.* Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing. STOC '12. New York, New York, USA: Association for Computing Machinery, 2012, pp. 77–84. ISBN: 9781450312455. DOI: 10.1145/2213977.2213985.

[BHM+02] Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. *Quantum amplitude amplification and estimation*. Contemporary Mathematics **305** (2002), pp. 53–74.

[Bol13] Béla Bollobás. *Modern graph theory*. Vol. 184. Springer Science & Business Media, 2013.

[BR12] Aleksandrs Belovs and Ben W. Reichardt. *Span programs and quantum algorithms for st-connectivity and claw detection*. Algorithms – ESA 2012. Ed. by Leah Epstein and Paolo Ferragina. Vol. 7501. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 193–204. ISBN: 978-3-642-33090-2. DOI: 10.1007/978-3-642-33090-2_18.

[BT20] Salman Beigi and Leila Taghavi. *Quantum Speedup Based on Classical Decision Trees*. Quantum **4**, 241 (2020). DOI: 10.22331/q-2020-03-02-241.

[Cab10] Sergio Cabello. *Finding shortest contractible and shortest separating cycles in embedded graphs*. ACM Transactions on Algorithms (TALG) **6**, 2 (2010), pp. 1–18.

[CEM+98] Richard Cleve, Arthur Ekert, Chiara Macchiavello, and Michele Mosca. *Quantum algorithms revisited*. Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences **454**, 1969 (1998), pp. 339–354.

[CEN09] Erin W. Chambers, Jeff Erickson, and Amir Nayyeri. *Minimum Cuts and Shortest Homologous Cycles*. Proceedings of the Twenty-Fifth Annual Symposium on Computational Geometry. SCG '09. New York, NY, USA: Association for Computing Machinery, 2009, pp. 377–385. ISBN: 9781605585017. URL: https://doi.org/10.1145/1542362.1542426.

[Chi21] Andrew M. Childs. *Lecture notes on quantum algorithms*. University of Maryland, 2021. URL: http://www.cs.umd.edu/~amchilds/qa/qa.pdf.

[Chu97] Fan RK Chung. *Spectral graph theory*. 92. American Mathematical Soc., 1997.

[CJO+20] Arjan Cornelissen, Stacey Jeffery, Maris Ozols, and Alvaro Piedrafita. *Span Programs and Quantum Time Complexity*. 45th International Symposium on Mathematical Foundations of Computer Science (MFCS 2020). Ed. by Javier Esparza and Daniel Kráľ. Vol. 170. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, 26:1–26:14. ISBN: 978-3-95977-159-7. DOI: 10.4230/LIPIcs.MFCS.2020.26.

[CKS17] Andrew M Childs, Robin Kothari, and Rolando D Somma. *Quantum algorithm for systems of linear equations with exponentially improved dependence on precision*. SIAM Journal on Computing **46**, 6 (2017), pp. 1920–1950.

[CMB18] Chris Cade, Ashley Montanaro, and Aleksandrs Belovs. *Time and space efficient quantum algorithms for detecting cycles and testing bipartiteness*. Quantum Information and Computation **18** (2018), pp. 0018–0050. DOI: 10.26421/QIC18.1-2.

[CRR+96] Ashok K. Chandra, Prabhakar Raghavan, Walter L. Ruzzo, Roman Smolensky, and Prasoon Tiwari. *The Electrical Resistance of a Graph Captures its Commute and Cover Times*. Computational Complexity **6**, 4 (1996), pp. 312–340.

[CVL11] Sergio Cabello, Eric Colin de Verdiere, and Francis Lazarus. *Finding cycles with topological properties in embedded graphs*. SIAM Journal on Discrete Mathematics **25**, 4 (2011), pp. 1600–1614.

[deW19] Ronald de Wolf. *Quantum computing: Lecture notes* (2019). ArXiv: 1907.09415 (quant-ph).

[DHH+06] Christoph Dürr, Mark Heiligman, Peter Høyer, and Mehdi Mhalla. *Quantum query complexity of some graph problems*. SIAM Journal on Computing **35**, 6 (2006), pp. 1310–1328.

[DM68] Patrick H. Doyle and Daniel A. Moran. *A short proof that compact 2-manifolds can be triangulated*. Inventiones mathematicae **5**, 2 (1968), pp. 160–162.

[Eri11] Jeff Erickson. *Shortest non-trivial cycles in directed surface graphs*. Proceedings of the twenty-seventh annual symposium on Computational geometry. 2011, pp. 236–243.

[Fie89] Miroslav Fiedler. *Laplacian of graphs and algebraic connectivity*. Banach Center Publications **25**, 1 (1989), pp. 57–70.

[Gro96] Lov K Grover. *A fast quantum mechanical algorithm for database search*. Proceedings of the twenty-eighth annual ACM symposium on Theory of computing. 1996, pp. 212–219.

[GT01] Jonathan L Gross and Thomas W Tucker. *Topological graph theory*. Courier Corporation, 2001.

[HLS07] Peter Hoyer, Troy Lee, and Robert Spalek. *Negative weights make adversaries stronger*. Proceedings of the thirty-ninth annual ACM symposium on Theory of computing - STOC '07 (2007). DOI: 10.1145/1250790.1250867.

[HLW06] Shlomo Hoory, Nathan Linial, and Avi Wigderson. *Expander graphs and their applications*. Bulletin of the American Mathematical Society **43**, 4 (2006), pp. 439–561.

[IJ19] Tsuyoshi Ito and Stacey Jeffery. *Approximate span programs*. Algorithmica **81**, 6 (2019), pp. 2158–2195. DOI: 10.1007/s00453-018-0527-1.

[Jef14] Stacey Jeffery. *Frameworks for quantum algorithms*. PhD thesis. University of Waterloo, 2014. URL: https://uwspace.uwaterloo.ca/handle/10012/8710.

[Jef20] Stacey Jeffery. *Span programs and quantum space complexity*. 11th Innovations in Theoretical Computer Science Conference (ITCS 2020). Ed. by Thomas Vidick. Vol. 151. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2020, 4:1–4:37. ISBN: 978-3-95977-134-4. DOI: 10.4230/LIPIcs.ITCS.2020.4.

[JJK+18] Michael Jarret, Stacey Jeffery, Shelby Kimmel, and Alvaro Piedrafita. *Quantum algorithms for connectivity and related problems*. 26th Annual European Symposium on Algorithms (ESA 2018). Ed. by Yossi Azar, Hannah Bast, and Grzegorz Herman. Vol. 112. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 49:1–49:13. ISBN: 978-3-95977-081-1. DOI: 10.4230/LIPIcs.ESA.2018.49.

[JK09] Rahul Jain and Hartmut Klauck. *The Partition Bound for Classical Communication Complexity and Query Complexity*. 2009. arXiv: 0910.4266 [cs.CC].

[JK17] Stacey Jeffery and Shelby Kimmel. *Quantum algorithms for graph connectivity and formula evaluation*. Quantum **1** (2017), p. 26. DOI: 10.22331/q-2017-08-17-26.

[Kit95] A Yu Kitaev. *Quantum measurements and the Abelian stabilizer problem*. arXiv preprint quant-ph/9511026 (1995).

[KW93] Mauricio Karchmer and Avi Wigderson. *On span programs*. Proceedings of the 8th Annual IEEE Conference on Structure in Complexity Theory. 1993, pp. 102–111. DOI: 10.1109/SCT.1993.336536.

[LMR+11] Troy Lee, Rajat Mittal, Ben W Reichardt, Robert Špalek, and Mario Szegedy. *Quantum query complexity of state conversion*. 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science. IEEE. 2011, pp. 344–353.

[Moi13] Edwin E Moise. *Geometric topology in dimensions 2 and 3*. Vol. 47. Springer Science & Business Media, 2013.

[MT01] Bojan Mohar and Carsten Thomassen. *Graphs on surfaces*. Vol. 16. Johns Hopkins University Press Baltimore, 2001.

[NC02] Michael A Nielsen and Isaac Chuang. *Quantum computation and quantum information*. 2002.

[NT95] Noam Nisan and Amnon Ta-Shma. *Symmetric Logspace is Closed Under Complement*. Proceedings of the Twenty-seventh Annual ACM Symposium on Theory of Computing (STOC 1995). ACM, 1995, pp. 140–146. ISBN: 0-89791-718-9. DOI: 10.1145/225058.225101.

[PP20] Subhasree Patro and Alvaro Piedrafita. *An Overview of Quantum Algorithms: From Quantum Supremacy to Shor Factorization*. 2020 IEEE International Symposium on Circuits and Systems (ISCAS). 2020, pp. 1–5. DOI: 10.1109/ISCAS45731.2020.9180793.

[PR17] Álvaro Piedrafita and Joseph M. Renes. *Reliable Channel-Adapted Error Correction: Bacon-Shor Code Recovery from Amplitude Damping*. Phys. Rev. Lett. **119** (25 2017), p. 250501. DOI: 10.1103/PhysRevLett.119.250501.

[Rei09] Ben W. Reichardt. *Span programs and quantum query complexity: The general adversary bound is nearly tight for every Boolean function*. Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2009). 2009, pp. 544–551. DOI: 10.1109/FOCS.2009.55.

[Rei10] Ben W. Reichardt. *Span programs and quantum query algorithms*. Electronic Colloquium on Computational Complexity (ECCC) **17** (2010), p. 110.

[RŠ12] Ben W. Reichardt and Robert Špalek. *Span-program-based quantum algorithm for evaluating formulas*. Theory of Computing **8**, 13 (2012), pp. 291–319. DOI: 10.4086/toc.2012.v008a013.

[San08] Miklos Santha. *Quantum walk based search algorithms*. International Conference on Theory and Applications of Models of Computation. Springer. 2008, pp. 31–46.

[Sze04] Mario Szegedy. *Quantum speed-up of Markov chain based algorithms*. 45th Annual IEEE Symposium on Foundations of Computer Science. 2004, pp. 32–41.

[WY20] Qisheng Wang and Mingsheng Ying. *Quantum random access stored-program machines*. 2020. eprint: 2003.03514.

# Abstract

This dissertation is an in-depth discussion of the theory and applications of span programs. These objects, first introduced in the context of classical counting branching programs by Karchmer and Wigderson, are a model of computation (i.e. they encode 2-output functions) and can be compiled into quantum query algorithms. Moreover, it was known before our work that the resulting span-program-based algorithms can be query and space optimal. The thesis is divided in three parts. **Part I** contains the introduction and mathematical preliminaries.

In **Part II**, we discuss the theory of span programs, meaning that we study the features and structure that are common to all — or at least large families — of span programs. Chapter 3 begins with a review of two existing formulations of span programs before presenting a new formulation that contains and generalizes the previous two. The second part of the chapter presents a plethora of different quantum algorithms that can be compiled out of a span program. Special emphasis is put on the flexibility of span programs for the purposes of algorithm design, a feature that has often been underappreciated. In Chapter 4 we study a correspondence between quantum query algorithms and span programs. We conclude that for a broad category of functions, an optimal time, query and space span-program-based algorithm always exists, and perhaps more importantly, that all these flavours of optimality are simultaneously achievable.

In **Part III** we use span programs to design quantum algorithms. In Chapter 5, we use the *st*-connectivity span program to design quantum algorithms for graph connectivity and other related graph problems, some of which are not decision problems. We end the dissertation in Chapter 6 with a discussion of the *st*-connectivity problem as a boundary problem and a particular instance of the simplicial homology problem and give span programs (and thus, quantum algorithms) for a few instances of this problem.

# Nederlandse samenvatting

Dit proefschrift biedt een diepgaande blik op de theorie en toepassingen van omhulselprogramma's. Deze objecten, die voor het eerst zijn geïntroduceerd in de context van klassieke forktellingprogramma's door Karchmer en Wigderson, vormen een rekenmodel (d.w.z. ze coderen 2-outputfuncties) en kunnen worden gecompileerd naar kwantumquery-algoritmen. Bovendien was het al voor ons werk bekend dat de resulterende op omhulselprogramma's gebaseerde algoritmen zowel query- als ruimteoptimaal kunnen zijn. Het proefschrift is verdeeld in drie delen. **Deel I** bevat de inleiding en wiskundige voorbereidingen.

In **Deel II** bespreken we de theorie van omhulselprogramma's, wat betekent dat we de kenmerken en de structuur bestuderen die alle omhulselprogramma's — of tenminste grote families daarvan — gemeen hebben. Het hoofdstuk begint met een bespreking van twee bestaande formuleringen van omhulselprogramma's alvorens een nieuwe formulering voor te stellen die de vorige twee bevat en veralgemeniseert. Het tweede deel van het hoofdstuk presenteert een overvloed aan verschillende kwantumalgoritmen die uit een omhulselprogramma's kunnen worden gecompileerd. Speciale nadruk wordt gelegd op de flexibiliteit van omhulselprogramma's voor het ontwerpen van algoritmen, een eigenschap die vaak ondergewaardeerd wordt. In Hoofdstuk 4 bestuderen we een overeenkomst tussen kwantumqueryalgoritmen en omhulselprogramma's. We concluderen dat er voor een breed assortiment van functies altijd een tijd-, query- en ruimteoptimaal algoritme bestaat dat is ontstaan uit een omhulselprogramma's, en misschien nog wel belangrijker, dat al deze soorten optimaliteit gelijktijdig haalbaar zijn.

In **Deel III** gebruiken we omhulselprogramma's om kwantumalgoritmen te ontwerpen. In Hoofdstuk 5 gebruiken we het bereikbaarheidsprobleemomhulselprogramma om kwantumalgoritmen te ontwerpen voor graafsamenhangendheid en andere verwante graafproblemen, waarvan sommige geen beslissingsproblemen zijn. We eindigen het proefschrift in Hoofdstuk 6 met een bespreking van het bereikbaarheidsprobleem als een grensprobleem en een bijzonder voorbeeld van het simpliciale homologieprobleem en we geven omhulselprogramma's (en dus kwantumalgoritmen) voor enkele voorbeelden van dit probleem.

# Acknowledgements

First, I want to thank my advisor, Stacey Jeffery, and my promotor, Harry Buhrman, for giving me the chance of doing a PhD at QuSoft. Fact is, when I started, I was but a naïve physicist with a background in quantum information. I knew virtually nothing of theoretical computer science, complexity theory, or classical algorithms, and yet I was certain I wanted to research quantum algorithms for my PhD.

You believed in me and patiently introduced me to the world of complexity theory, CS, algorithms, and most importantly, span programs. Special thanks go to Stacey. I was your fist student, which is fine because you were my first supervisor, and I know you put a lot of hard work into mentoring me. You took me under your wing and then encouraged me to go out and find my own way. This thesis is the result of the trust you put in me.

I also want to thank my co-authors, Maris, Michael, Shelby, Arjan, and again, Stacey. All my best ideas came in one way or another out of a conversation with one of you. Working with you made the work fun and kept me motivated to pull through the inevitable ugly parts of writing a thesis.

I want to thank my thesis committee members for agreeing to be part of the committee and read this 250-page hunk of chunk. In particular Ronald, whose door was always open for questions. CWI is full to the brim with learned men and women. Ronald, however, is a rara avis, a renaissance man. If there is any consistency in hyphenation throughout this document it is due entirely to him. All remaining inconsistencies, of course, are my fault.

As high praise and gratitude goes to Stacey as it must go to my officemate, co-author and friend, Arjan Cornelissen. I could not have started my PhD without her, and I would not have finished without you. I cherish our collaboration above all the things I did at QuSoft. Getting you interested in span programs is, without a doubt, my biggest contribution to their study.

Now, I have a reputation to maintain, so before we get sentimental, I want to shift gears and express my gratitude for my colleagues at CWI. In no special order I want to thank András, Yfke, Koen, Tom, Freek, Joris, Jana, Srini, Mathys, Ralph, Jeroen, the Chris's, the Sebastians, Ido, Simon, Alex, Michael, Yinan, Yan-Lin and Sander. I thoroughly enjoyed taking your ELO. Big thank you to the support staff, Susanne, Doutzen and Victor. Your tireless work makes a big difference. I also want to thank all the new people

I barely got to meet due to the circumstances of the pandemic. You know who you are. Particularly Maarten. I have a lot of faith in you, son.

Special mention to Jan, Jonas (another renaissance man), Esteban, Wessel, Willem Jan, Brinn, Joran and Isabella. Playing with you has been a fixed point of joy and merriment, especially during the pandemic, and I want you to know that if (more like, when) I killed or maimed your characters, I did it out of love.

To my office mates in the peanut-butter room, Farrokh, Harold, Arjan and Subha, a toast! We had a lot of fun, perhaps too much. I apologize for all the pranking and time I made you waste with my shenanigans. Subha, our after-lunch discussions were always a delight. To Farrokh, Harold and Arjan, I bequeath unto you the stewardship of the old ways of foosball. I taught you almost everything I know the way I myself was taught. Through yelling, sarcasm and repeated humiliating defeat. I know you will make me proud.

Last but not least, I want to thank my friends and family. ¡Luis and Aitor, hasta la victoria siempre! Guillem and Joana, per casi quatre anys heu estat els meus millors amics. Gràcies per adopter-me i formar part de la sit-com que es la meva vida.

Gracias a Isa por escucharme y apoyarme. Bedankt, Yente, voor de dans. Thanks Leen, Dilan, Fanny and Sarai. Tack, Clara, vi ses igen. Nuala, here's looking at you, kiddo.

Gracias a mis abuelos, tíos y primos. A Nico por no darme la razón cuando no la tengo y a Gabi por no dársela a él. Gracias a mis padres, por toda una vida detrás de mí. Esta tesis es la culminación de tres décadas de esfuerzo y cariño.

And to you, whoever you may be, thank you for reading these lines. I hope you enjoyed this thesis, I promise I won't do it again.

*Titles in the ILLC Dissertation Series:*

ILLC DS-2016-01: **Ivano A. Ciardelli**
*Questions in Logic*

ILLC DS-2016-02: **Zoé Christoff**
*Dynamic Logics of Networks: Information Flow and the Spread of Opinion*

ILLC DS-2016-03: **Fleur Leonie Bouwer**
*What do we need to hear a beat? The influence of attention, musical abilities, and accents on the perception of metrical rhythm*

ILLC DS-2016-04: **Johannes Marti**
*Interpreting Linguistic Behavior with Possible World Models*

ILLC DS-2016-05: **Phong Lê**
*Learning Vector Representations for Sentences - The Recursive Deep Learning Approach*

ILLC DS-2016-06: **Gideon Maillette de Buy Wenniger**
*Aligning the Foundations of Hierarchical Statistical Machine Translation*

ILLC DS-2016-07: **Andreas van Cranenburgh**
*Rich Statistical Parsing and Literary Language*

ILLC DS-2016-08: **Florian Speelman**
*Position-based Quantum Cryptography and Catalytic Computation*

ILLC DS-2016-09: **Teresa Piovesan**
*Quantum entanglement: insights via graph parameters and conic optimization*

ILLC DS-2016-10: **Paula Henk**
*Nonstandard Provability for Peano Arithmetic. A Modal Perspective*

ILLC DS-2017-01: **Paolo Galeazzi**
*Play Without Regret*

ILLC DS-2017-02: **Riccardo Pinosio**
*The Logic of Kant's Temporal Continuum*

ILLC DS-2017-03: **Matthijs Westera**
*Exhaustivity and intonation: a unified theory*

ILLC DS-2017-04: **Giovanni Cinà**
*Categories for the working modal logician*

ILLC DS-2017-05: **Shane Noah Steinert-Threlkeld**
*Communication and Computation: New Questions About Compositionality*

ILLC DS-2017-06: **Peter Hawke**
*The Problem of Epistemic Relevance*

ILLC DS-2017-07: **Aybüke Özgün**
*Evidence in Epistemic Logic: A Topological Perspective*

ILLC DS-2017-08: **Raquel Garrido Alhama**
*Computational Modelling of Artificial Language Learning: Retention, Recognition & Recurrence*

ILLC DS-2017-09: **Miloš Stanojević**
*Permutation Forests for Modeling Word Order in Machine Translation*

ILLC DS-2018-01: **Berit Janssen**
*Retained or Lost in Transmission? Analyzing and Predicting Stability in Dutch Folk Songs*

ILLC DS-2018-02: **Hugo Huurdeman**
*Supporting the Complex Dynamics of the Information Seeking Process*

ILLC DS-2018-03: **Corina Koolen**
*Reading beyond the female: The relationship between perception of author gender and literary quality*

ILLC DS-2018-04: **Jelle Bruineberg**
*Anticipating Affordances: Intentionality in self-organizing brain-body-environment systems*

ILLC DS-2018-05: **Joachim Daiber**
*Typologically Robust Statistical Machine Translation: Understanding and Exploiting Differences and Similarities Between Languages in Machine Translation*

ILLC DS-2018-06: **Thomas Brochhagen**
*Signaling under Uncertainty*

ILLC DS-2018-07: **Julian Schlöder**
*Assertion and Rejection*

ILLC DS-2018-08: **Srinivasan Arunachalam**
*Quantum Algorithms and Learning Theory*

ILLC DS-2018-09: **Hugo de Holanda Cunha Nobrega**
*Games for functions: Baire classes, Weihrauch degrees, transfinite computations, and ranks*

ILLC DS-2018-10: **Chenwei Shi**
*Reason to Believe*

ILLC DS-2018-11: **Malvin Gattinger**
*New Directions in Model Checking Dynamic Epistemic Logic*

ILLC DS-2018-12: **Julia Ilin**
*Filtration Revisited: Lattices of Stable Non-Classical Logics*

ILLC DS-2018-13: **Jeroen Zuiddam**
*Algebraic complexity, asymptotic spectra and entanglement polytopes*

ILLC DS-2019-01: **Carlos Vaquero**
*What Makes A Performer Unique? Idiosyncrasies and commonalities in expressive music performance*

ILLC DS-2019-02: **Jort Bergfeld**
*Quantum logics for expressing and proving the correctness of quantum programs*

ILLC DS-2019-03: **András Gilyén**
*Quantum Singular Value Transformation & Its Algorithmic Applications*

ILLC DS-2019-04: **Lorenzo Galeotti**
*The theory of the generalised real numbers and other topics in logic*

ILLC DS-2019-05: **Nadine Theiler**
*Taking a unified perspective: Resolutions and highlighting in the semantics of attitudes and particles*

ILLC DS-2019-06: **Peter T.S. van der Gulik**
*Considerations in Evolutionary Biochemistry*

ILLC DS-2019-07: **Frederik Möllerström Lauridsen**
*Cuts and Completions: Algebraic aspects of structural proof theory*

ILLC DS-2020-01: **Mostafa Dehghani**
*Learning with Imperfect Supervision for Language Understanding*

ILLC DS-2020-02: **Koen Groenland**
*Quantum protocols for few-qubit devices*

ILLC DS-2020-03: **Jouke Witteveen**
*Parameterized Analysis of Complexity*

ILLC DS-2020-04: **Joran van Apeldoorn**
*A Quantum View on Convex Optimization*

ILLC DS-2020-05: **Tom Bannink**
*Quantum and stochastic processes*

ILLC DS-2020-06: **Dieuwke Hupkes**
*Hierarchy and interpretability in neural models of language processing*

ILLC DS-2020-07: **Ana Lucia Vargas Sandoval**
*On the Path to the Truth: Logical & Computational Aspects of Learning*

ILLC DS-2020-08: **Philip Schulz**
*Latent Variable Models for Machine Translation and How to Learn Them*

ILLC DS-2020-09: **Jasmijn Bastings**
*A Tale of Two Sequences: Interpretable and Linguistically-Informed Deep Learning for Natural Language Processing*

ILLC DS-2020-10: **Arnold Kochari**
*Perceiving and communicating magnitudes: Behavioral and electrophysiological studies*

ILLC DS-2020-11: **Marco Del Tredici**
*Linguistic Variation in Online Communities: A Computational Perspective*

ILLC DS-2020-12: **Bastiaan van der Weij**
*Experienced listeners: Modeling the influence of long-term musical exposure on rhythm perception*

ILLC DS-2020-13: **Thom van Gessel**
*Questions in Context*

ILLC DS-2020-14: **Gianluca Grilletti**
*Questions & Quantification: A study of first order inquisitive logic*

ILLC DS-2020-15: **Tom Schoonen**
*Tales of Similarity and Imagination. A modest epistemology of possibility*

ILLC DS-2020-16: **Ilaria Canavotto**
*Where Responsibility Takes You: Logics of Agency, Counterfactuals and Norms*

ILLC DS-2020-17: **Francesca Zaffora Blando**
*Patterns and Probabilities: A Study in Algorithmic Randomness and Computable Learning*

ILLC DS-2021-01: **Yfke Dulek**
*Delegated and Distributed Quantum Computation*

ILLC DS-2021-02: **Elbert J. Booij**
*The Things Before Us: On What it Is to Be an Object*

ILLC DS-2021-03: **Seyyed Hadi Hashemi**
*Modeling Users Interacting with Smart Devices*

ILLC DS-2021-04: **Sophie Arnoult**
*Adjunction in Hierarchical Phrase-Based Translation*

ILLC DS-2021-05: **Cian Guilfoyle Chartier**
*A Pragmatic Defense of Logical Pluralism*

ILLC DS-2021-06: **Zoi Terzopoulou**
*Collective Decisions with Incomplete Individual Opinions*

ILLC DS-2021-07: **Anthia Solaki**
*Logical Models for Bounded Reasoners*

ILLC DS-2021-08: **Michael Sejr Schlichtkrull**
*Incorporating Structure into Neural Models for Language Processing*

ILLC DS-2021-09: **Taichi Uemura**
*Abstract and Concrete Type Theories*

ILLC DS-2021-10: **Levin Hornischer**
*Dynamical Systems via Domains: Toward a Unified Foundation of Symbolic and Non-symbolic Computation*