Author:
Cade, Chris W

Title:
Quantum Algorithms and Complexity in Non-standard Models

Quantum Algorithms and Complexity in Non-standard Models

Chris Cade

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Doctor of Philosophy (PhD) in the Faculty of Engineering, Department of Computer Science, September 2019.

34,389 words.

Abstract

This thesis explores the apparent ability of quantum computers to outperform classical computers in a variety of settings. It approaches this goal from two quite different but complementary directions: at first by studying restricted models of quantum computing, and then by considering enhanced models of both quantum and classical computing. Both approaches aim to provide insight into the nature and power of universal quantum computers, understand what sorts of problems might be good candidate tasks for near-term devices, and narrow down exactly what it is about quantum physics that seemingly gives us access to more computational power.

It begins by introducing and describing *span programs*, a tool from the classical literature that has proved useful for designing quantum algorithms for a variety of problems, often with the added benefit that these algorithms are space efficient. Time-optimal and space efficient quantum algorithms for deciding two graph properties, detecting cycles and testing bipartiteness, are developed using span programs as one of their main algorithmic ingredients.

The quantum computational complexity of estimating a natural mathematical quantity, the Schatten p-norm of a matrix, is studied and found to be closely related to the one clean qubit model – a restricted model of quantum computing inspired by NMR. It is shown that this quantity can be estimated efficiently using the one clean qubit model, and that estimating it is at least as hard as simulating this model.

The effect of post-selection on quantum and classical computation is studied in the query complexity setting, where an unbounded separation is found between the post-selected quantum and classical query complexities of computing a certain (total) Boolean function. This separation arises from a lower bound on the post-selected classical query complexity of computing symmetric Boolean functions, which in turn arises from a characterisation of the query complexity in terms of the non-negative rational degree of Boolean functions.

Acknowledgements

First and foremost I would like to thank my supervisor Ashley Montanaro for his help, support, and patience with me throughout this PhD, and for being a wonderful supply of interesting problems and of clever ideas for solving them. I thank my co-authors, and all the members of the Bristol quantum information group, as well as many members from the Cambridge and UCL groups, for many stimulating discussions and useful advice.

For my time here in Bristol, thanks go to all of my other friends and colleagues. Thanks and love go to my mother Susan and sister Stephanie for their love and support during my PhD and the twenty-five years before that, my grandmother Sylvia for her hospitality when I needed to retreat to somewhere quiet, and of course to Charlotte for her unending support and patience with me over the past 7 years, and for making my time here in Bristol so enjoyable.

Author's Declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: .....CHRISTOPHER CADE..... DATE:....15/01/2020....

Contents

Chapter 1

Introduction

As far as we know, quantum computers encapsulate the true computational power of the world that we live in. Using the laws of quantum physics, they can do things that computers built using the rules of classical physics cannot. In the context of computation, this means they are believed to be able to solve certain computational problems much more efficiently than their classical counterparts – the canonical examples of this being Shor's algorithm for integer factorisation [108], which achieves an exponential speedup over all previously known classical algorithms, and Grover's algorithm for unstructured search [54], which can find a marked item from a set quadratically faster than is possible classically. In some settings, the speedups achieved by quantum algorithms can be rigorously proved, as is the case with unstructured search [19, 54, 93].

Despite tremendous progress in quantum computing over the last few decades, it is not completely understood exactly *why* quantum computers can perform computations faster than classical ones. Indeed, precisely which property of quantum mechanics is responsible for this power is hotly debated, with proposals including 'quantum parallelism' arising from quantum superposition [46], quantum entanglement [68, 119], quantum discord [42], and the recently popular notion of quantum contextuality [70, 60]. Similarly, it is not entirely clear what kinds of problems we can expect to obtain significant quantum speedups for, and there is no reliable, general-purpose method for generating efficient quantum algorithms from existing classical ones, although some methods do exist in certain cases [89, 14, 22]. These two open problems are very likely to be related: on the one hand, knowing which properties of a problem that quantum computers can use and classical ones cannot might tell us which properties of quantum mechanics they are exploiting; and on the other, if

we know the precise ingredients of quantum physics that quantum computers make use of, then we may be able to determine exactly which problems allow them to make best use of those ingredients.

There are many directions from which we can approach these questions. For instance, one may attempt to design algorithms for, and to characterise, limited models of quantum computation. Studying such models can offer some insight into the nature and power of universal quantum computers, and may also guide us in understanding what kinds of problems might be good candidate tasks for near-term quantum computers [58]. Moreover, by finding computational problems that even a quantum algorithm running on a restricted quantum computer can solve faster than any known classical algorithm, we can attempt to narrow down exactly what it is about quantum physics that seemingly gives us access to more computational power. Conversely, we might consider what happens when we *add* power to computational models. For example, we have strong evidence[1] that adding the hypothetical power of 'post-selection' to quantum computers makes them vastly more powerful than their classical counterparts augmented with the same power. Exactly why this is the case, however, is not completely clear, and as with many results in computational complexity, the separation strongly relies on widely believed (but unproved) complexity-theoretic conjectures.

This thesis attempts to address some of these questions by using both approaches. In Chapter 2, we begin by introducing the concept of Span Programs [71], a model of computation that was developed to study classical logspace complexity and has recently found applications in designing efficient quantum algorithms for a number of problems, with the span program for solving the s-t connectivity problem taking a central role [22, 64, 67]. Although some of the proofs in this chapter have not appeared elsewhere, the results described here are not new – rather, this chapter is intended to serve as preliminary material for the chapter that follows, as well as to highlight some of the main results in this area and mention some more recent results that were obtained following the publication of the

[1] This evidence arises from the combination of a few previous results – namely a result of Han et al. [56], which places the complexity class PostBPP into the third level of the polynomial hierarchy; a result of Aaronson [5] which shows that PostBQP = PP; and Toda's theorem [118], which puts the polynomial hierarchy inside PP. All of this can be combined to state that "if PostBQP = PostBPP, then the polynomial hiearchy will collapse to its third level", something that is believed to be very unlikely to be true (for many of the same reasons that it is believed that P $\neq$ NP). Here, PostBPP and PostBQP are the complexity classes associated with post-selected classical and quantum computation, respectively. For definitions of the other classes, see e.g. the complexity zoo [1]

work contained in Chapter 3.

In Chapter 3 we introduce our first new result: a quantum algorithm that can be used to decide two graph theoretic problems – detecting whether an (undirected) n-vertex graph contains a cycle, and deciding whether a graph is bipartite. We consider two standard models for accessing the edges of the graph: the *adjacency matrix model* and the *adjacency array model*. In both models, our algorithms achieve the optimal (quantum) time complexity of $\tilde{O}(n^{3/2})$[2] and $\tilde{O}(n)$, respectively. In contrast, the best a classical algorithm can achieve with either matrix input model is $O(n^2)$. At the core of the algorithms is a reduction to the problem of s-t connectivity, which parallels various other works that design time and space efficient quantum algorithms by utilising reductions to the same problem.

As mentioned, the time complexities of both algorithms are optimal up to polylogarithmic factors, but perhaps more interestingly, the space complexity is also optimal – they use only $O(\log n)$ quantum and classical bits of storage – showing that quantum algorithms for these problems can be both time and space efficient, whereas the analogous classical result is not known (the logarithmic-space classical algorithms for these problems do not have optimal time complexity). These results have practical consequences for quantum computing: since early fault-tolerant quantum computers are unlikely to have a very large number of (logical) qubits, it is important to find ways to design efficient quantum algorithms that minimise space usage. For problems where a dramatic quantum speedup is expected only when the input becomes very large – which is likely to be true of many graph-theoretic problems, in which only relatively modest polynomial speedups are possible – it is crucial that the quantum algorithm's space requirements scale only logarithmically with the size of the input.

In Chapter 4 we push the idea of limited space even further, and consider a restricted model of quantum computation in which we have access to only a single 'clean' qubit, and n very noisy qubits – the so-called *One Clean Qubit model* [76]. Inspired by the proposal of NMR[3] as a viable form of quantum computation, this model is weaker than full quantum computation, but can seemingly solve some problems much more efficiently than a classical computer can [76, 109]. One would hope that, by understanding the power of such interme-

[2]The $\tilde{O}$ notation hides polylogarithmic factors in n.
[3]Nuclear magnetic resonance.

diate classes of computation, we can shed light on the types of problems that are efficiently solvable by a fully universal quantum computer. Moreover, non-universal computers like the one clean qubit model are expected to be easier to implement in practice, and finding useful applications for them is especially important in the early stages of quantum computer development, where large fault-tolerant devices are still some years away. In this chapter we investigate the computational complexity of a natural mathematical problem – estimating Schatten p-norms of matrices – and find that this problem is closely related to the one clean qubit model, whose associated complexity class is known as DQC1. In particular, we show that the problem of estimating Schatten p-norms of log-local Hamiltonians up to a certain additive accuracy is in DQC1, and that the problem of estimating them up to a somewhat higher accuracy is DQC1-hard. By building on a previous result of Janzing and Wocjan [63], we also show that these quantities can be estimated for sparse matrices in BQP[4]. The quantum algorithms developed in this chapter can be used to compute various global properties of the spectra of matrices. An interesting quantity of this kind is the graph energy [81], which is defined to be the Schatten 1-norm of the adjacency matrix of a graph, and has applications in chemistry where it is related to the total electron energy of a class of organic molecules [55, 81].

Currently there is also a large interest in sub-universal quantum computing models, such as DQC1, in the setting of quantum computational supremacy[5]. Indeed, some previous works in this setting have shown that DQC1 is hard to classically simulate up to reasonable error, so long as some widely-believed complexity-theoretic conjectures hold true [90]. Adding to the relatively short list of DQC1-hard problems, as well as showing that an apparently intractable classical problem is contained in DQC1, can help us to further characterise the power of this model and provide more evidence that this class is indeed hard to classically simulate.

We then consider what happens when we add hypothetical powers to computational models. There are several reasons to study such non-conventional models: in the quantum setting, studying the computational power of extended versions of quantum mechanics might give insight into why quantum mechanics is the way it is. For instance, changing the

[4]Bounded-error Quantum Polynomial time.

[5]Quantum computational supremacy refers to the experimental realisation of a (perhaps non-universal) quantum computer that can unambiguously outperform any existing (or soon to exist) classical computer for some task, and is considered to be an important milestone in the field of quantum computation.

amplitude-to-probability measurement rule from $|\alpha|^2$ to $|\alpha|^p$ for some $p \neq 2$, or allowing linear but non-unitary evolution, yields a class of computation equivalent to PostBQP = PP [5]. If we consider such computational classes to be unrepresentative of the world that we find ourselves in, then this might help to explain why the measurement rule is the way it is, or why quantum mechanics is unitary.

Sometimes, modifications to the rules of computation can tease out large differences between computational models that, at first glance, are not obviously very different. For example, Aaronson et al. [8] studied the power of quantum computers, supposing that they were given the ability to perform 'non-collapsing' measurements. They found that such a modification only slightly increased the power of quantum computers, allowing them to solve the unstructured search problem with $O(N^{1/3})$ queries instead of the conventional $\Theta(N^{1/2})$. This is a somewhat surprising result, since many modifications of quantum mechanics lead to drastically more powerful computational models (PostBQP being a prominent example), and one would expect that preventing collapse after measurement would be yet another example of this. Subsequent to this work, Aaronson [6] showed that combining non-collapsing measurements with another 'small' enhancement to quantum computation – that of quantum advice[6] – would allow a quantum computer to solve *any* decision problem in polynomial time (corresponding to the complexity class ALL). This is very surprising, since neither enhancement by itself leads to a large increase in computational power. This goes to show that adding an additional power – in this case, quantum advice – to two models of computation that do not, at first glance, appear all that dissimilar, can yield two very different computational classes and hence reveal an underlying difference that was not immediately visible.

In Chapter 5 we consider adding the hypothetical power of post-selection to both quantum and classical computers, and compare the power of the two models in the query complexity setting. The query complexity of post-selected quantum computation was recently studied by Mahadev and de Wolf [86]: they showed that the post-selected quantum query complexity of a Boolean function is equal, up to a constant factor of 2, to the minimum degree of a rational polynomial that approximates that function. In this chapter, we obtain the analogous classical result: that, up to a constant factor of 2, the post-selected classical

[6]Here, we allow our computers access to an 'advice' string (which could be number of qubits or classical bits, depending on the setting. In the main text, we consider the quantum case.) that can be used to aid the computation, and which is provided by a trusted, computationally unbounded agent, but must be independent of the input, instead depending only on its length.

query complexity of a Boolean function is equal to the minimum degree of a 'non-negative' rational polynomial that approximates that function (that is, a rational polynomial that is only allowed to have positive coefficients). The techniques used to prove the two results are somewhat different, but an interesting observation is that in the quantum case, the coefficients of the rational functions correspond to amplitudes in the quantum query algorithms, and to probabilities in the (randomised) classical query algorithms. Hence, this result suggests that the only difference between classical and quantum computation in the post-selected query model, is in the use of amplitudes vs. probabilities – an intuition that fits nicely with our understanding of quantum mechanics. Using the characterisation of post-selected classical query algorithms as rational polynomials with non-negative coefficients, we are able to prove an unbounded (i.e. $O(1)$ vs. $\Theta(N)$) separation between the post-selected quantum and post-selected classical query complexities of computing a particular total Boolean function (as well as a number of separations from other complexity measures). This is in contrast to the ordinary (i.e. not post-selected) case, in which the largest possible separation between the quantum and classical query complexities of computing a total Boolean function is only polynomial [7]. Thus, adding post-selection appears to amplify the difference between classical and quantum computation.

1.1 Publications

Some of the work in this thesis has been published previously, and is joint work with others.

- Chapter 3 is joint work with Ashley Montanaro and Aleksandrs Belovs, and has been published previously as "Time and space efficient quantum algorithms for detecting cycles and testing bipartiteness", *Quantum Information and Computation* vol. 18 no. 1&2, 2016. (`arXiv:1610.00581`).

- Chapter 4 is joint work with Ashley Montanaro, and has been published previously as "The quantum complexity of computing schatten p-norms" In the proceedings of Theory of Quantum Computation, vol. 111, 2018. (`arXiv:1706.09279`).

- Chapter 5 is available as the preprint "Post-selected classical query complexity", 2018, `arXiv:1804.10010`, and is currently under review for the Proceedings of the Royal Society A.

- We note here that some joint work with Mithuna Yoganathan, available as the preprint "The one clean qubit model without entanglement is classically simulable", 2019, `arXiv:1907.08224`, is somewhat related to the content of Chapter 4 but does not form a part of this thesis.

Chapter 2

Span programs

This chapter is intended to serve as preliminary material for the chapter that follows. Some of the content from this chapter is included in the publication "Time and space efficient quantum algorithms for detecting cycles and testing bipartiteness", *Quantum Information and Computation* vol. 18 no. 1&2, 2016, and an earlier version that is available as the pre-print `arXiv:1610.00581`. This includes most of the proof of the existence of a time efficient implementation of the span-program based algorithm for s-t connectivity. Although the details of these proofs have not appeared elsewhere, the results are not new.

Span programs are a linear algebraic model of computation, introduced by Karchmer and Wigderson in 1993 [71], that have many applications in classical complexity theory, and can be used to evaluate decision problems. Reichardt and Špalek [102] introduced a new complexity measure for span programs, the *witness* size, which Reichardt later showed to have strong connections with quantum query complexity [99, 101]. In particular, he showed that the witness size of a span program and the query complexity of a quantum algorithm evaluating that span program are separated by at most a constant factor, and in fact used the concept of span programs to prove that the general adversary bound gives tight quantum query lower bounds for any decision problem. This suggests that span programs may be useful for designing new quantum algorithms.

Span programs have been used to design quantum query algorithms for formula evaluation [102, 103, 100, 66, 67], the matrix rank problem [21], subgraph detection [128, 23, 26], s-t connectivity [26, 64], strong connectivity [17], and estimating the algebraic connectivity of a graph [64], to name a few. In this chapter we only briefly introduce the model; for a

more comprehensive introduction, see e.g. [26, 71].

One appeal of using span programs to design quantum algorithms is that they usually give rise to *space-efficient* algorithms. Indeed, span programs were originally introduced to study classical log-space complexity. Given the limited size of quantum computers in the near future, the design of space-efficient quantum algorithms is of great importance. Some very recent work of Jeffery [65] formalises this observation, albeit in the form of a lower bound. She shows that the amount of space needed for a quantum algorithm to compute some function f with bounded (two-sided) error is lower-bounded by the logarithm of its approximate span program size (a property of span programs that has been well-studied in the classical literature).

Organisation We will begin in Section 2.1 by introducing span programs, and then proceed to present a span program for the problem of s-t connectivity (Section 2.2), which will be used in a later chapter. Following this, we describe a general method for implementing span programs time efficiently (Section 2.3), due to Belovs and Reichardt [26], and then apply this method to the span program for s-t connectivity (following the approach in [26] very closely).

Included in Section 2.2 are some recent results (obtained by others) that appeared subsequent to the work presented in Chapter 3, and so, whilst interesting, were not used to obtain any of the results presented in this thesis. In fact, some of these newer results were used by DeLorenzo et al. [45] to improve upon the work presented in Chapter 3, and we discuss the connection between their work and ours in Section 3.1.2 of that chapter.

2.1 Formal definition

A span program $\mathcal{P}$ takes as input an n-bit string $x \in \mathcal{D} \subseteq \{0,1\}^n$, and either accepts or rejects it. That is, it implements the (partial) boolean function $f_{\mathcal{P}} : \mathcal{D} \to \{0,1\}$.

Definition 1. *A span program is defined by a tuple $\mathcal{P} = (\mathcal{H}, |\tau\rangle, \{M_{i,b}\}, M_{free})$, where $\mathcal{H}$ is a finite-dimensional Hilbert space, $|\tau\rangle \in \mathcal{H}$ is the* 'target vector'*, $\{M_{i,b}\}$ is a set of sets of vectors for $i \in [n], b \in \{0,1\}$, where each $M_{i,b} \subseteq \mathcal{H}$ is a finite set of vectors which we will collectively call* 'input vectors'*, and $M_{free} \subseteq \mathcal{H}$ is a set of* 'free' *input vectors.*
Given an input x, denote by $M(x) = \bigcup\{M_{i,b} : i \in [n], x_i = b\} \cup M_{free}$. Then the span

program accepts if the target vector $|\tau\rangle$ can be written as a linear combination of the vectors in $M(x)$:

$$f(x) = 1 \iff |\tau\rangle \in \text{span}(M(x)).$$

Informally, the span program consists of sets of vectors that are either *available* or *unavailable*, depending on the input given to the span program. Generally speaking, we associate two sets of vectors to each input bit x_i, so that if $x_i = 1$, then the vectors in $M_{i,1}$ are available and those in $M_{i,0}$ are unavailable, and vice versa. The vectors contained in M_{free} are always available, and any set $M_{i,b}$, M_{free} may be empty. The goal is then to construct the target vector $|\tau\rangle$ using the collection of vectors made available to you with the input. If this is possible, then the span program accepts, else it rejects.

Positive and negative witnesses

Call $M(x)$ the set of available input vectors and let d be the dimension of the Hilbert space $\mathcal{H}$ and m be the total number of input vectors and free input vectors (also referred to as the 'size' of $\mathcal{P}$). Write the set of *all* input vectors and free vectors as $\{|v_j\rangle : j \in [m]\}$. Finally, define $M := \sum_{j=1}^{m} |v_j\rangle \langle j|$, which can be thought of as a matrix that enumerates all input vectors as columns.

Then we can consider two cases:

- **Positive case**: If $\mathcal{P}$ accepts x, then we must be able to write $|\tau\rangle$ as a linear combination of available input vectors:

  $$|\tau\rangle = \sum_{v_j \in M(x)} w_j |v_j\rangle .$$

  Then the coefficients w_j give a *positive witness vector* for x, $|w\rangle = \sum_j w_j |j\rangle$, such that $M|w\rangle = |\tau\rangle$. The size of this witness is taken to be $\| |w\rangle \|^2$.

- **Negative case**: If $\mathcal{P}$ rejects x, then it must not be possible to construct $|\tau\rangle$ using a linear combination of the available input vectors. Therefore, there must be some component of $|\tau\rangle$ that is orthogonal to all available input vectors. That is, there must exist some vector $|w'\rangle$ such that $\langle w'|v_j\rangle = 0$ for all $v_j \in M(x)$, and $\langle w'|\tau\rangle \neq 0$. In order for the witness size to be well defined we require that $\langle w'|\tau\rangle = 1$. We call the

vector $|w'\rangle$ the *negative witness vector* for x. The size of the witness is defined as

$$\| M^\dagger |w'\rangle \|^2 = \| \sum_{j=1}^{m} |j\rangle \langle v_j|w'\rangle \|^2 = \sum_{v_j \notin M(x)} |\langle v_j|w'\rangle|^2$$

This equals the sum of the absolute squares of the inner products of $|w'\rangle$ with all *unavailable* input vectors.

Witness size The witness size of $\mathcal{P}$ on input x, $\text{wsize}(\mathcal{P}, x)$, is defined as the minimum size among all witnesses for x. For domain $\mathcal{D} \subseteq \{0,1\}^n$, let

$$\text{wsize}_b(\mathcal{P}, \mathcal{D}) = \max_{x \in \mathcal{D}: f_{\mathcal{P}}(x)=b} \text{wsize}(\mathcal{P}, x).$$

Then the witness size of $\mathcal{P}$ on domain $\mathcal{D}$ is defined as

$$\text{wsize}(\mathcal{P}, \mathcal{D}) = \sqrt{\text{wsize}_0(\mathcal{P}, \mathcal{D}) \text{wsize}_1(\mathcal{P}, \mathcal{D})}.$$

2.2 A span program for s-t connectivity

We present here a span program for solving the problem of s-t connectivity, due to Belovs and Reichardt [26]. Formally, the problem is defined as follows: given an n-vertex graph $G = (V, E)$, and two vertices $s, t \in V$, is there a path from s to t in G?

Before describing it, it is worth noting that this particular span program has proved to be very useful in recent works. For instance, in Chapter 3 we use this span program as a subroutine to solve a number of graph problems; it has also been applied to generic formula evaluation problems [66, 67], and underlies the learning graph framework of Belovs [22]. But perhaps more important is the fact that it is often possible to analyse not only the query complexity of these s-t connectivity based algorithms, but their time complexities as well. This is because it is possible to tightly characterise the positive and negative witness sizes of the associated span programs.

In [26], Belovs and Reichardt gave a tight characterisation of the positive witness size as the *effective resistance*[1] between vertices s and t in the input graph. However, they only

[1]Intuitively, the effective resistance between two vertices s and t of a graph, $R_{s,t}$, is a measure of how 'easy' it is to get from one vertex to the other, perhaps via random walk. If the two vertices are connected by a single edge of weight w, then $R_{s,t} = 1/w$. If they are not connected in the graph, then $R_{s,t} = \infty$. In

loosely bounded the negative witness size of an input in which s and t are not connected by using the total weight of an s-t cut (i.e. the total minimum weight of edges that need to be removed from the underlying graph in order to separate s and t). Following this, Jeffery and Kimmel [67] showed that if the underlying graph is planar, the negative witness size can be characterised exactly as the effective resistance of a graph related to the planar dual of the parent graph. This allowed for an improved analysis of the s-t connectivity based span program algorithms for formula evaluation, yielding improved algorithms for some classes of input.

Finally, in a subsequent work, Jarret et al. [64] showed that the negative witness size of the s-t connectivity span program is actually exactly characterised by the *effective capacitance* of the input graph. This elegant result brings the problem of characterising the s-t connectivity span program to a satisfying conclusion. At a high level, it shows that the negative witness size is related to a measure of the potential difference that the network could store between the component containing s and the component containing t. The more, shorter paths there are between these components in the underlying graph, the greater the capacitance. Intuitively, then, these results tell us that quantum algorithms can quickly decide s-t connectivity on graphs that are promised to either have small effective resistance (it's easy to check a yes-instance) or small effective capacitance (it's easy to check a no-instance).

Below we describe the most basic version of the span program for s-t connectivity, but do not concern ourselves with its characterisation in terms of effective resistance and effective capacitance (since these results are not used in this thesis).

2.2.1 The span program

Define a span program $\mathcal{P}$ using the vector space $\mathbb{R}^n$, with an orthonormal basis $\{|v\rangle : v \in V\}$ – i.e. a basis vector for each vertex in G. We suppose that the input to the program is a bit string of the form x_{ij} for $i, j \in V$, such that $x_{ij} = 1$ iff there is an edge $(i, j) \in E$. Then the span program is defined as follows:

- **Target Vector:** $|\tau\rangle = |t\rangle - |s\rangle$.

general, many, highly weighted paths from s to t will give a low effective resistance, and few, lowly weighted paths will give a high effective resistance. For a rigorous definition, see e.g. [64].

- **Available Input Vectors:** For each edge $(u, v) \in E$, (i.e. $x_{uv} = 1$), we make available the input vector $|v\rangle - |u\rangle$.

There are no free input vectors. It might be useful to note that in this example, the set of all input vectors is $\{|j\rangle - |i\rangle : i \neq j \in V\}$, with an input vector corresponding to every possible edge that might occur in G, given the vertex set V. Therefore, the total number of input vectors is $m = \binom{n}{2}$.

Now we prove correctness and calculate the witness sizes.

Positive case

Suppose s and t are connected in G. Then there must exist some path of length, say, d between them: $s = u_0, u_1, ..., u_d = t$. Then all of the vectors $|u_1\rangle - |s\rangle, |u_2\rangle - |u_1\rangle, ..., |t\rangle - |u_{d-1}\rangle$ are available. Simply adding all these vectors, each with unit weight, gives $|t\rangle - |s\rangle = |\tau\rangle$. Since the positive witness will consist of d entries of $+1$, the positive witness size is $O(d)$.

Negative case

Suppose that s and t are not connected in G, and instead lie in different connected subcomponents of G. We must show that a negative witness $|w'\rangle$ exists, such that $\langle w'|\tau\rangle = 1$, and $\langle w'|v\rangle = 0$ for all available input vectors v. We can define $|w'\rangle$ by its inner product on all basis vectors: let $\langle w'|u\rangle = 1$ if u is in the same connected subcomponent as t, and $\langle w'|u\rangle = 0$ otherwise. Then we have that $\langle w'|\tau\rangle = \langle w'|(|t\rangle - |s\rangle) = \langle w'|t\rangle - \langle w'|s\rangle = 1 - 0 = 1$. If an input vector of the form $|v\rangle - |u\rangle$ is available, then there is an edge between vertices u and v in G, and therefore both u and v belong to the same connected subcomponent in G. Therefore, $\langle w'|v\rangle = \langle w'|u\rangle$ in all such cases, and $|w'\rangle$ is orthogonal to all available input vectors. Since there are at most $\binom{n}{2} = O(n^2)$ unavailable input vectors, and the inner product between $|w'\rangle$ and any basis vector is either 0 or 1, the negative witness size is $O(n^2)$. The span program $\mathcal{P}$ therefore has witness size $O(n\sqrt{d})$.

Using the more sophisticated results from [64], we could instead write the witness size as $\sqrt{R_{s,t} \cdot C_{s,t}}$, with $R_{s,t}$ and $C_{s,t}$ the effective resistance and effective capacitance, respectively, between s and t.

2.3 Time-efficient implementation of the span program

In this section we describe in some detail how to implement a span program in a manner that is time efficient, where the efficiency depends upon some properties of the span program, as we will describe.

2.3.1 Preliminaries

We will require some facts about the eigenspaces of the product of two reflections. Let A and B be matrices each with n rows and orthonormal columns. Let $\Pi_A = AA^\dagger$ and $\Pi_B = BB^\dagger$ be the projections onto the column spaces of A and B, respectively. Let $R_A = 2\Pi_A - I$ and $R_B = 2\Pi_B - I$ be the reflections about the corresponding subspaces, and let $U = R_B R_A$ be their product.

Lemma 1. (Spectral Lemma [117]). *Under the above assumptions, all the singular values of $A^\dagger B$ are at most 1. Let $\cos\theta_1, ..., \cos\theta_l$ be all the singular values of $A^\dagger B$ lying in the open interval $(0, 1)$, and let $\mathcal{C}(A)$ and $\mathcal{C}(B)$ denote the column spaces of A and B, respectively. Then the following is a complete list of the eigenvalues of U:*

- *The $+1$ eigenspace is $(\mathcal{C}(A) \cap \mathcal{C}(B)) \oplus (\mathcal{C}(A)^\perp \cap \mathcal{C}(B)^\perp)$*
- *The -1 eigenspace is $(\mathcal{C}(A) \cap \mathcal{C}(B)^\perp) \oplus (\mathcal{C}(A)^\perp \cap \mathcal{C}(B))$. Moreover, $\mathcal{C}(A)^\perp \cap \mathcal{C}(B) = B(\ker A^\dagger B)$*
- *On the orthogonal complement of the above subspaces, U has eigenvalues $e^{2i\theta_j}$ and $e^{-2i\theta_j}$ for $j \in [l]$.*

Lemma 2 (Effective Spectral Gap Lemma [80])**.** *Let P_Θ be the orthogonal projection onto the span of all eigenvectors of U with eigenvalues $e^{i\theta}$ such that $|\theta| \leq \Theta$. Then, for any vector $|w\rangle$ in the kernel of Π_A, we have*

$$\| P_\Theta \Pi_B |w\rangle \| \leq \frac{\Theta}{2} \| |w\rangle \|.$$

We will also require the following tools, which have been used many times elsewhere in quantum algorithm design:

Theorem 1 (Phase estimation [73][40]). *Given a unitary U as a black box, there exists a quantum algorithm that, given an eigenvector $|\psi\rangle$ of U with eigenvalue $e^{i\phi}$, outputs a real number w such that $|w - \phi| \leq \delta$ with probability at least $9/10$. The algorithm uses $O(1/\delta)$ controlled applications of U and $\frac{1}{\delta}\text{polylog}(1/\delta)$ other elementary operations.*

Theorem 2 (Reflection using phase estimation [85]). *Let $U \in U(n)$ have a unique eigenvector with eigenvalue 1, and let the smallest non-zero phase of U be $\sigma_{\min}$. Then for any integer k there exists a quantum circuit R that acts on $O(\log_2 n) + ks$ qubits, where $s = \log_2\left(\frac{1}{\sigma_{\min}}\right) + O(1)$, such that:*

- *R uses the controlled-U operator $O(k2^s)$ times and contains $O(ks^2)$ other gates.*
- *If $|\psi\rangle$ is the unique 1-eigenvector of U, then $R|\psi\rangle|0^{ks}\rangle = |\psi\rangle|0^{ks}\rangle$.*
- *If $|\phi\rangle$ lies in the subspace orthogonal to $|\psi\rangle$, then $\|(R+I)|\phi\rangle|0^{ks}\rangle\| = O(1/2^k)$.*

The latter point of Theorem 2 tells us that the circuit R implements a reflection about the eigenvalue-1 eigenspace of U up to some precision 2^{-k}, determined by the value of k. In particular, if U has a constant spectral gap, then $s = O(1)$ and the number of calls to the controlled-U operator is $O(k)$, and depends only on our desired precision for the circuit.

2.3.2 Implementing span programs

In this section we outline a general method, due to Belovs and Reichardt [26], for implementing span programs in a time-efficient manner, and apply it to the span program for evaluating s-t connectivity. Before we do so, however, it will be useful to describe a quantum query algorithm for evaluating span programs and, in doing so, note an interesting connection between span programs and quantum query complexity:

Theorem 3 (Reichardt [99]). *For any (partial) boolean function $f : \mathcal{D} \to \{0,1\}$, there is a (bounded-error) quantum algorithm that requires $O(wsize(\mathcal{P}, \mathcal{D}))$ queries to a quantum oracle for the bits of x, where $\mathcal{P}$ is any span program for which $f_{\mathcal{P}}$ agrees with f on the domain $\mathcal{D}$.*

Proof: *(From [26], included for completeness)*

The quantum algorithm works in the space $\mathbb{R}^{m+1}$ with orthonormal basis elements $\{|j\rangle\}_{j=0}^{m}$. Let $I(x) = \{j : v_j \in M(x)\}$ be the indices corresponding to the available input vectors, and let W_0, W_1 and $W = \sqrt{W_0 W_1}$ be, respectively, the negative witness size, positive witness size, and witness size of $\mathcal{P}$.

We perform phase estimation on the operator $U = (2\Lambda - I)(2\Pi_x - I)$, the product of two reflections about the images of the projection operators Λ and Π_x, which are defined as follows: Let $\Lambda : \mathbb{R}^{m+1} \to \mathbb{R}^{m+1}$ be the orthogonal projection onto the kernel of $\tilde{M}$, where:

$$\tilde{M} = \frac{1}{\alpha} |\tau\rangle \langle 0| + M = \frac{1}{\alpha} |\tau\rangle \langle 0| + \sum_{j=1}^{m} |v_j\rangle \langle j| \text{ for some } \alpha \in \mathbb{R} \text{ yet to be defined,}$$

$$\text{and let } \Pi_x : \mathbb{R}^{m+1} \to \mathbb{R}^{m+1} \text{ be defined by } \Pi_x = |0\rangle \langle 0| + \sum_{j \in I(x)} |j\rangle \langle j|$$

The algorithm accepts x if and only if, on the input of $|0\rangle$, phase estimation on U, with precision Θ, outputs a phase of zero (corresponding to an eigenvalue of 1). We need to find values of α and Θ for which this procedure works.

First we consider the positive case. Take an optimal positive witness $|w\rangle = \sum_j w_j |j\rangle$, and use it to construct an eigenvalue 1 eigenvector of U: let $|u\rangle = \alpha |0\rangle - |w\rangle = \alpha |0\rangle - \sum_j w_j |j\rangle$. Since $|u\rangle$ consists only of $|0\rangle$ and the positive witness vector, we have $\Pi_x |u\rangle = |u\rangle$ and $\tilde{M} |u\rangle = |\tau\rangle - |\tau\rangle = 0$. So $|u\rangle$ is in the kernel of $\tilde{M}$, which implies that $\Lambda |u\rangle = |u\rangle$. Thus, $|u\rangle$ is an eigenvalue 1 eigenvector of U, and phase estimation on $|u\rangle$ will output a phase of zero with certainty. Therefore, the probability of phase estimation on $|0\rangle$ outputting a phase of zero depends on the overlap of $|0\rangle$ with $|u\rangle$, and is at least:

$$\frac{|\langle 0|u\rangle|^2}{\| |u\rangle \|^2} = \frac{\alpha^2}{\alpha^2 + \sum w_j^2} \geq \frac{1}{1 + W_1/\alpha^2}. \tag{2.1}$$

Therefore, if we choose $\alpha = C\sqrt{W_1}$, for some constant C, we can increase the value of C to make this probability arbitrarily close to 1. In particular, we can ensure that $CW > 1$

Now consider the negative case. Let $|w'\rangle$ be an optimal negative witness, and define $|v\rangle = \alpha \tilde{M}^\dagger |w'\rangle$. Since $|w'\rangle$ is a negative witness, we have $\langle w'|\tau\rangle = 1$ and so $|v\rangle = |0\rangle +$

$\alpha M^\dagger |w'\rangle$. So $\Pi_x |v\rangle = |0\rangle$ and

$$\| |v\rangle \|^2 \leq 1 + \alpha^2 W_0 = 1 + C^2 W_1 W_0 = 1 + C^2 W^2 \leq 2C^2 W^2 \tag{2.2}$$

since the negative witness size W_0 is defined by $W_0 = \|M^\dagger |w'\rangle\|^2$, and the final inequality follows from the restriction that $CW > 1$.

Let Θ be the precision of the phase estimation algorithm, and let P_Θ be the projection operators onto the space of eigenvectors of U of phase less than Θ. So the probability that phase estimation outputs a phase of zero on the input $|0\rangle$ is $\|P_\Theta |0\rangle\|^2 = \|P_\Theta \Pi_x |v\rangle\|^2$.

Using the Effective Spectral Gap Lemma (Lemma 2), we have that $\|P_\Theta \Pi_x |v\rangle\| \leq \frac{\Theta}{2} \| |v\rangle \|$, as long as $\Lambda |v\rangle = 0$. This last condition is easy to verify, since $|v\rangle$ lies in the image of $\tilde{M}^\dagger$. So, we have

$$\|P_\Theta |0\rangle\| = \|P_\Theta \Pi_x |v\rangle\| \leq \frac{\Theta}{2} \| |v\rangle \| \leq \Theta CW \tag{2.3}$$

and we can choose the precision to be $\Theta = 1/C'W$, for some constant C'. Therefore, we can choose a large value of C' such that the probability $\|P_\Theta |0\rangle\|^2$ is arbitrarily small.

The phase estimation algorithm on U with precision Θ requires $O(1/\Theta)$ queries to U, and each query to U requires only one query to the oracle for x. Therefore the algorithm evaluates the span program on an input x with $O(1/\Theta) = O(W)$ queries to x . □

The above algorithm defines a unitary operator $U = (2\Lambda - I)(2\Pi_x - I)$, which is the product of two reflections - the first, $R_\Pi := (2\Pi_x - I)$, is an input dependent reflection, and the second, $R_\Lambda := (2\Lambda - I)$, is an input independent reflection. Since the algorithm requires repeated applications of this operator, it may only be implemented time-efficiently if we can implement both R_Π and R_Λ time-efficiently. R_Π is generally quite straightforward to implement - since all it requires is some efficient map from the bits of the input to the corresponding input vectors. However, the implementation of R_Λ is more subtle, and will be the main focus of the rest of this section.

Implementing R_Π and R_Λ

We begin by describing a general approach for implementing the two reflections, which is due to Belovs and Reichardt [26].

We consider the $d \times (m+1)$ matrix $\tilde{M} = \frac{1}{\alpha} |\tau\rangle \langle 0| + \sum_{j=1}^m |v_j\rangle \langle j|$ as the biadjacency matrix for a bipartite graph on $d + m + 1$ vertices, and run a Szegedy-type quantum

walk [117] on it. The structure of this graph is as follows: we have two disjoint sets of vertices – one consisting of a vertex for every basis vector in our vector space, and one consisting of a vertex for the target vector plus every input vector in the span program. An edge exists between two vertices when a basis vector makes up some non-zero component of one or more of the input/target vectors.

To perform a quantum walk, we must factor $\tilde{M}$ into two sets of unit vectors: vectors $|a_i\rangle \in \mathbb{R}^m$ for each row $i \in [d]$ and vectors $|b_j\rangle \in \mathbb{R}^d$ for each column $j \in [m]$, so that $\langle i|b_j\rangle \langle a_i|j\rangle = M'_{ij}$, where M' differs from $\tilde{M}$ only by a rescaling of its rows, since rescaling the rows of a matrix does not affect its nullspace.

Given such a factorisation, let $A = \sum_{i=1}^{d} (|i\rangle \otimes |a_i\rangle) \langle i|$ and $B = \sum_{j=1}^{m} (|b_j\rangle \otimes |j\rangle) \langle j|$, so that $A^\dagger B = M'$. Let R_A and R_B be the reflections about the column spaces of A and B, respectively.

Embed $\mathcal{H}$ into $\tilde{\mathcal{H}} = \mathbb{R}^d \otimes \mathbb{R}^m$ using the isometry B. Then R_Λ can be implemented on $B(\mathcal{H})$ as the reflection about the -1 eigenspace of $R_B R_A$. By Lemma 1, this eigenspace equals $(\mathcal{C}(A) \cap \mathcal{C}(B)^\perp) \oplus (\mathcal{C}(A)^\perp \cap \mathcal{C}(B))$, which is equal to $B(\ker A^\dagger B) = B(\ker M)$ plus a part that is orthogonal to $\mathcal{C}(B)$ and is therefore irrelevant. The reflection about the -1 eigenspace of $R_B R_A$ can then be implemented using phase estimation, which will give us a reflection about the kernel of M in the larger space $\tilde{\mathcal{H}}$, whose basis is given by $|i\rangle \otimes |j\rangle$ for $i \in [d], j \in [m]$. R_Π may be implemented by reflections controlled by j – i.e. given a state in $\tilde{\mathcal{H}}$, multiply the phase by -1 if $|v_j\rangle$ is an unavailable input vector.

Intuitively, A and B are matrices that give us the *local spaces* for each vertex $|a_i\rangle$ and $|b_j\rangle$, respectively. By local space, we are referring to the neighbours of a vertex in the graph described by M. Therefore, the reflection about the column spaces of A and B are equivalent to reflections about the local spaces $|a_i\rangle$ and $|b_j\rangle$, controlled by columns i and j, respectively.

The efficiency of the algorithm thus depends on two factors:

1. The implementation costs of R_A and R_B. Since these reflections decompose into local reflections, they can be easier to implement than R_Λ.

2. The spectral gap around the -1 eigenvalue of $R_B R_A$, on which the efficiency of the phase estimation sub-routine will depend. By Lemma 1, this gap is determined by the gap of $A^\dagger B = M'$ around singular value 0.

Since, for any given span program, M' describes a bipartite graph, the reflections R_A and R_B can usually be implemented efficiently. To calculate the properties of the spectral

gap around singular value 0 of M', we may calculate the spectral gap around the eigenvalue 0 of $\Delta := M'M'^{\dagger}$ (since the singular values of M' are the square roots of the eigenvalues of Δ).

We use phase estimation to perform the reflection about the -1 eigenspace of $R_B R_A$. If the smallest non-zero singular value of M' is $\sigma_{\min}$, then by Theorem 2 this will require $O((1/\sigma_{\min})\log(1/\delta))$ controlled applications of R_A and R_B, plus $O(\log(1/\delta)\text{polylog}(1/\sigma_{\min}))$ other elementary operations, where δ is the precision of the circuit. Thus, if R_A takes time T_A and R_B takes time T_B, the entire process will require time $\tilde{O}(\frac{T_A+T_B}{\sigma_{\min}})$ for constant δ. If M' has a constant spectral gap (i.e. $\sigma_{\min} = \Omega(1)$), the time required to implement the span program depends only on the complexity of the reflections R_A and R_B.

2.3.3 Implementation of the s-t connectivity span program

In this section we will apply the general approach described above to the span program for s-t connectivity as described in Section 2.2. Recall that we are given as input a graph $G = (V, E)$, and the indices of two vertices s and t from V, and we want to know whether or not there is a path from s to t in G.

Theorem 4 (Combination of Theorems 3 and 9 from [26])**.** *Consider the s-t connectivity problem on a graph G given by its adjacency matrix. Assume there is a promise that if s and t are connected by a path, then they are connected by a path of length at most d. Then there exists a bounded-error quantum algorithm that determines whether s and t are connected in $\tilde{O}(n\sqrt{d})$ time and uses $O(\log n)$ bits and qubits of storage, and which fails with probability at most $1/10$.*

Proof. In order to make the implementation straightforward, we modify the span program from Section 2.2 slightly. We assume that s and t are not directly connected by an edge – a fact that can be checked in $O(n)$ time beforehand, if necessary. Then, alongside the normal scaled-down target vector $|\tilde{\tau}\rangle = \frac{1}{\alpha}(|t\rangle - |s\rangle)$, we introduce a 'never-available' input vector $|\tilde{\sigma}\rangle = \sqrt{1 - 1/\alpha^2}(|t\rangle - |s\rangle)$. We may assume that $\alpha = C_1\sqrt{W_1} \geq 1$. To introduce a never-available input vector, we introduce a dummy input bit that is always set to zero, and associate the input vector with it. It is easy to verify that this modification changes neither the behaviour of the span program nor its witness size.

Define the set of input vectors (not including the one corresponding to an edge between s

and t)

$$M_{in} := \{|v_{xy}\rangle := |y\rangle - |x\rangle : x < y \in V\} \setminus \{|v_{st}\rangle\},$$

and let $\{|xy\rangle : |v_{xy}\rangle \in M_{in}\} \cup \{|st\rangle, |\overline{st}\rangle\}$ be an orthonormal basis for the set of indices of vectors in M_{in}, with the extra vectors $|st\rangle$ and $|\overline{st}\rangle$ indexing the scaled target vector $|\tilde{\tau}\rangle$ and the 'never-available' input vector $|\tilde{\sigma}\rangle$, respectively. Let

$$\tilde{M} = \frac{1}{\alpha} |\tau\rangle \langle st| + \sqrt{1 - 1/\alpha^2} |\sigma\rangle \langle \overline{st}| + \sum_{|v_{xy}\rangle \in M_{in}} |v_{xy}\rangle \langle xy| .$$

Now we define some vectors $|a_x\rangle$ for each $x \in V$:

- For $x \notin \{s,t\}$, $|a_x\rangle = \frac{1}{\sqrt{n-1}} \sum_{y \in V \setminus \{x\}} |xy\rangle$
- For $x \in \{s,t\}$, $|a_x\rangle = \frac{1}{\alpha\sqrt{n-1}} |st\rangle + \sqrt{\frac{1-1/\alpha^2}{n-1}} |\overline{st}\rangle + \frac{1}{\sqrt{n-1}} \sum_{y \in V \setminus \{s,t\}} |xy\rangle$

and some vectors $|b_{ij}\rangle$ for each input vector $|v_{ij}\rangle \in M_{in}$:

- $|b_{ij}\rangle = \frac{1}{\sqrt{2}}(|j\rangle - |i\rangle)$

Then these $|a_x\rangle$ and $|b_{ij}\rangle$ give a factorisation of the matrix $\tilde{M}$, up to a rescaling of the rows. That is, for $x \notin \{s,t\}$,

$$\begin{aligned} \langle a_x|ij\rangle \langle x|b_{ij}\rangle &= \frac{1}{\sqrt{2}\sqrt{n-1}} \left(\sum_{y \in V \setminus \{x\}} \langle xy|ij\rangle \right) \langle x| (|j\rangle - |i\rangle) \\ &= \frac{1}{\sqrt{2(n-1)}} \langle x|v_{ij}\rangle \end{aligned}$$

The cases $x \in \{s,t\}$ can be verified separately, and give the desired final result, implying that $M' = \frac{1}{\sqrt{2(n-1)}} M$.

Now we may define $A = \sum_{u \in V} (|u\rangle \otimes |a_u\rangle) \langle u|$ and $B = \sum_{u,v \in V} (|b_{uv}\rangle \otimes |uv\rangle) \langle uv|$, and proceed as in Section 2.3.2 – i.e. we can now implement the reflection R_Λ by using phase estimation to reflect about the -1 eigenspace of $R_B R_A$, where R_B and R_A are the reflections about $\mathcal{C}(B)$ and $\mathcal{C}(A)$, respectively. This reflection is independent of the input – that is, we reflect about the null-space of the biadjacency matrix given by the basis vectors and the set of (all possible) input vectors, which is formally described above. Our approach is then to alternate reflections about this space and the space of all *available* input vectors,

with the latter being achieved by the reflection operator R_Π. We can implement R_Π by querying the input graph, and multiplying by a phase of -1 if the edge corresponding to a given input vector is not present.

Spectral gap of M'

Since we use phase estimation to implement the reflection about the -1 eigenspace of $R_B R_A$, the efficiency of the algorithm will depend upon the spectral gap around the -1 eigenvalue of $R_B R_A$. By Lemma 1, this gap is determined by the spectral gap around singular value 0 of $A^\dagger B = M'$. The non-zero singular values of M' are the square roots of the non-zero eigenvalues of $\Delta := M'M'^\dagger$. Recall that $m = |M_{\text{in}}|$ gives the total number of input vectors in the span program, and that $M' = \frac{1}{\sqrt{2(n-1)}} M = \frac{1}{\sqrt{2(n-1)}} \sum_{i \in V} \sum_{j \in [m]} \langle i|v_j\rangle \, |i\rangle\langle j|$, where each $|v_j\rangle$ corresponds to an 'ordinary' input vector of the form $|y\rangle - |x\rangle$ for some $x \neq y \in V$, or to one of the special input vectors $|\tilde{\tau}\rangle = \frac{1}{\alpha}(|t\rangle - |s\rangle)$ or $|\tilde{\sigma}\rangle = \sqrt{1 - \frac{1}{\alpha^2}}(|t\rangle - |s\rangle)$. We have that

$$M'M'^\dagger = \sum_{i,i' \in V} \left(\frac{1}{2(n-1)} \sum_{j \in [m]} \langle i|v_j\rangle \, \langle v_j|i'\rangle \right) |i\rangle\langle i'| \,,$$

and therefore we can compute Δ by inspecting the individual values of $M' M'^\dagger_{ii'}$ for different cases of i and i':

- If $i \notin \{s,t\}$, and/or $i' \notin \{s,t\}$, then we consider two cases:
  1. $i = i'$: In this case, the term inside the brackets contributes a value of $\frac{n-1}{2(n-1)} = \frac{1}{2}$ to $\Delta_{ii'}$, since each vertex i has degree $(n-1)$. Alternatively, we note that each basis vector has $(n-1)$ input vectors with which it has inner product 1, and thus $\sum_j |\langle i|v_j\rangle|^2 = (n-1)$.
  2. $i \neq i'$: In this case, there is exactly one edge between vertices i and i', which contributes a term of $-\frac{1}{2(n-1)}$ to $\Delta_{ii'}$.
- If $i, i' \in \{s,t\}$, the situation is slightly different, but the result is the same. Again, we will deal with two cases:
  1. $i = i'$: In this case, vertex i has n neighbours. $(n-2)$ of them correspond to ordinary edges, whilst the remaining two correspond to the scaled target vector $|\tilde{\tau}\rangle$, and the never-available input vector $|\tilde{\sigma}\rangle$. Then $|\langle i|\tilde{\tau}\rangle|^2 = 1/\alpha^2$ and

$|\langle i|\tilde{\sigma}\rangle|^2 = (1 - 1/\alpha^2)$. So the term inside the brackets contributes a value of $(1 + (n-2) + 1/\alpha^2 - 1/\alpha^2)/2(n-1) = 1/2$ to Δ_{ii}.

2. $i \neq i'$: In this case, there are two edges between vertices i and i' (since one must be s, and the other t). The first edge, $|\tilde{\tau}\rangle$, contributes a value of $-1/\alpha^2$, and the other, $|\tilde{\sigma}\rangle$, contributes a value of $(1/\alpha^2 - 1)$. Together, they contribute a term of $-\frac{1}{2(n-1)}$ to $\Delta_{ii'}$.

The result is an $n \times n$ square matrix, whose diagonal elements are $1/2$, and off-diagonal elements are $-1/2(n-1)$. By taking out a factor of $1/2(n-1)$, we obtain the Laplacian matrix for the complete graph on n vertices, which has the form $nI_n - J_n$, where I_n is the $n \times n$ identity matrix and J_n the $n \times n$ all-ones matrix. This Laplacian has a single eigenvalue of 0, and $(n-1)$ eigenvalues of n. Therefore, the eigenvalues of Δ are 0 (multiplicity 1) and $n/2(n-1)$ (multiplicity $(n-1)$), and thus the non-zero singular values of M' are all at least $1/\sqrt{2}$, giving us a constant spectral gap as desired.

Implementing R_A and R_B

Now it remains to show that we can implement the reflection operators R_A and R_B efficiently. Since these reflections decompose into local reflections, it suffices to describe operators that implement local reflections about each $|a_u\rangle$ and $|b_{uv}\rangle$.

Recall that the algorithm works in the Hilbert space spanned by the vectors $|i\rangle \otimes |j\rangle$, where i varies over the vertices in the graph, and j over the input vectors and the target vector. We will describe the implementation of R_A first. For all $u \notin \{s, t\}$, $|u\rangle \otimes |a_u\rangle$ is the uniform superposition of the states $\{|u\rangle \otimes |uv\rangle : v \in V \setminus \{u\}\}$, and so the reflection is a Grover diffusion operator.

For $u = s$, the transformation is slightly more complex. Let F be the Fourier transform on the space spanned by $\{|s\rangle \otimes |sv\rangle : v \in V \setminus \{s\}\}$ that maps $|s\rangle \otimes |st\rangle$ to the uniform superposition; let K be a unitary on the space spanned by $\{|s\rangle \otimes |st\rangle, |s\rangle \otimes |\overline{st}\rangle\}$ that maps $|s\rangle \otimes |st\rangle$ to $\frac{1}{\alpha}(|s\rangle \otimes |st\rangle) + \sqrt{1 - \frac{1}{\alpha^2}}(|s\rangle \otimes |\overline{st}\rangle)$; and let L be a unitary that multiplies the phase of all states except $|s\rangle \otimes |st\rangle$ by -1. Then the local reflection can be implemented by $FKLK^{-1}F^{-1}$. Intuitively, this is still similar to the Grover diffusion operator: the unitary K acts to spread out the amplitude on the st edge between the target vector $|\tilde{\tau}\rangle$ and the input vector $|\tilde{\sigma}\rangle$, and when combined with F it creates the desired superposition over edges adjacent to s. Finally, the unitary L performs the reflection, analogously to a standard diffusion operator. A very similar operation works for $u = t$.

The implementation of R_B is relatively straightforward. We apply the negated swap to all pairs $(|u\rangle \otimes |uv\rangle, |v\rangle \otimes |uv\rangle)$, which maps a pair $(|x\rangle, |y\rangle) \mapsto (-|y\rangle, -|x\rangle)$, for some arbitrary states $|x\rangle, |y\rangle$. This can be achieved in logarithmic time.

To implement R_Π, the algorithm checks, for each state $|i\rangle \otimes |j\rangle$, whether input vector j is available by querying the input oracle for the presence of edge j. If it is available, it does nothing, otherwise it negates the phase of the state.

The states $|i\rangle \otimes |j\rangle$ can be stored using a logarithmic number of qubits. In particular, for a graph with n vertices, we require $O(\log n)$ qubits. □

There may be cases where we do not know an upper bound on the length of the path between s and t ahead of time. In such situations, we may want to 'guess' an upper bound on the length of the path. Provided that our guess is only wrong by at most a constant factor, the s-t connectivity algorithm of Theorem 5 will still fail with probability at most 1/10, which follows directly from the statement of Theorem 5. In the case that our guess is smaller than the actual length of the path (by more than a constant factor), then the algorithm may fail with a high probability.

Chapter 3

Time and space efficient quantum algorithms for detecting cycles and testing bipartiteness

The work in this chapter is joint work with Ashley Montanaro and Aleksandrs Belovs, and has been published previously as "Time and space efficient quantum algorithms for detecting cycles and testing bipartiteness", *Quantum Information and Computation* vol. 18 no. 1&2, 2016, and an earlier version is available as the pre-print `arXiv:1610.00581`. The content of this chapter is mostly identical to the published content, although we mention here some recent results (by other authors) that were obtained following publication of our work.

3.1 Introduction

Graph-theoretic problems are an important class of problems for which quantum algorithms appear to achieve a significant speedup over their classical counterparts. Examples include deciding whether there is a path between two vertices, or whether a graph is planar. The latter is exemplary of a subclass of graph problems that involve deciding whether or not a graph has a given property, which (besides planarity) includes properties such as containing a triangle, being bipartite, or being a forest. Many such properties are known to have efficient quantum algorithms [47, 38, 15].

Most of these algorithms have space requirements that grow linearly with the input

size – i.e. require $\Omega(n)$ bits and/or qubits. Two exceptions are the work of Belovs and Reichardt [26], who improve on the connectivity algorithm of [47] by providing a time-efficient ($\tilde{O}(n^{3/2})$) span-program-based algorithm for s-t connectivity that requires only logarithmic space, as well as the query-efficient ($O(n^{3/2})$) and space-efficient ($O(\log n)$) (but not time efficient) algorithm of Āriņš [17] for deciding bipartiteness. Prior to this work, little was known about the quantum space requirements for graph problems. Subsequent to the publication of the work in this chapter, a few more recent results have appeared that either present space-efficient algorithms for graph problems [67, 64, 45], or study the relationship between span programs and space complexity directly [65].

Since early quantum computers will be limited in size, it is important to design space efficient (i.e. $O(\log n)$ space) as well as time-efficient algorithms. This is even more important when one considers the fact that the graphs that will be good candidates for quantum algorithms are likely to be extremely large, and accessed implicitly (i.e. never entirely stored in memory). Hence, reducing the amount of space (both classical and quantum bits) required to process them is important; moreover, it is interesting to study how the space requirements of quantum algorithms relate to those of their classical counterparts.

In this chapter, we study space-efficient algorithms for deciding two graph properties: the property of being a forest – that is, deciding whether the input graph contains a cycle; and the property of bipartiteness, which is characterised by containing no odd length cycle as a subgraph. We note here that the property of being a forest is *minor-closed* with a single forbidden minor (a triangle), whereas the property of bipartiteness is only *subgraph-closed*. Equivalently, both properties can be characterised by an infinite number of *forbidden subgraphs* (all cycles and all odd-length cycles, respectively). We explain these terms in more detail in Section 3.1.1.

In this work we consider two models for the input of an undirected graph G with vertex set V and edge set E:

- **The adjacency matrix model** – The input is given as the adjacency matrix $A \in \{0,1\}^{n\times n}$, where A_{ij} is 1 if and only if $(i,j) \in E$.
- **The adjacency array model** – We are given the degrees of the vertices $d_1, d_2, ..., d_n$ and for every vertex i an array with its neighbours $f_i : [d_i] \to [n]$, so that $f_i(j)$ returns the j^{th} neighbour of vertex i, according to some arbitrary but fixed numbering of the outgoing edges of i. Following Dürr et al. [47], we assume that the degrees are

provided for free as a part of the input, and we account only for queries to the arrays f_i. Moreover, we assume that the graph is not a multigraph (that is, each f_i is injective).

Classically, in the adjacency matrix model, each of the problems requires $\Theta(n^2)$ queries to the input adjacency matrix, since both the randomised and deterministic query complexities of any (non-trivial) subgraph-closed graph property are $\Theta(n^2)$, which can be shown via a reduction from the unstructured search problem [38]. Known classical algorithms that achieve this bound are based on breadth first search, and hence require more than logarithmic space; however, by allowing more time the space requirement can be reduced: by using random walks, Aleliunas et al. [13] provide $O(n^3)$ time and $O(\log n)$ space algorithms for deciding bipartiteness and s-t connectivity. By taking into account a reduction given in this work, this implies a similar algorithm for detecting arbitrary cycles.

We describe a bounded-error quantum algorithm in the adjacency matrix model, which, given as input a graph G with vertex set V and edge set E, returns some vertex $v \in V$ that is a part of a cycle in G if such a cycle exists, and otherwise returns false. The algorithm runs in time $\tilde{O}(n^{3/2})$ and requires $O(\log n)$ bits and qubits of storage. Our algorithms are based on quantum walks and can hence be seen as quantum analogues of the approach of Aleliunas et al. [13]. By a simple modification of the original algorithm, we obtain an algorithm that can decide whether or not a graph is bipartite (i.e. contains no odd-length cycles), and has the same time and space requirements. For both problems our algorithms are optimal up to poly-logarithmic factors, almost matching the $\Omega(n^{3/2})$ quantum query lower bounds for bipartiteness [127] and cycle detection [38].

The main new technical ingredient of the algorithms is a reduction from the problem of cycle detection to the problem of s-t connectivity in some ancillary graph. We then apply the span-program-based s-t connectivity algorithm of Belovs and Reichardt [26] to this ancillary graph without explicitly constructing it. Following this, we make use of a variant of Grover search to look for a vertex that makes up a part of a cycle in the input graph.

We then turn to the adjacency array model. Dürr et al. prove a tight $\Omega(n)$ quantum query lower bound for the s-t connectivity problem in the adjacency array model [47], which we extend to give a $\Omega(n)$ bound on the problems of deciding bipartiteness and cycle detection in the array model. By combining our reduction to s-t connectivity with the s-t connectivity algorithm in [47], it is possible to construct algorithms that decide

bipartiteness and detect cycles in time $\tilde{O}(n)$. These algorithms are therefore optimal up to poly-logarithmic factors, but require $O(n \log n)$ space [47]. In order to preserve space efficiency, we construct a quantum walk-based algorithm to decide s-t connectivity, but at the expense of time efficiency. Our quantum walk-based algorithm takes time $\tilde{O}(n\sqrt{d_m})$, where d_m is the maximum degree of any vertex in the graph.

3.1.1 Previous work

Dürr et al. [47] previously gave quantum query lower bounds for some graph problems in both the adjacency matrix model and the adjacency array model, and in particular showed that the quantum query complexity of st-connectivity is $\Theta(n^{3/2})$ in the adjacency matrix model. In the following, unless explicitly stated, we will assume that any bounds given are applicable to the adjacency matrix model. Ambainis et al. [15] showed that deciding planarity also has quantum query complexity $\Theta(n^{3/2})$, and Zhang [127] gave a lower bound of $\Omega(n^{3/2})$ for the problems of bipartiteness and perfect matching. More generally, Sun et al. [116] showed that all non-trivial graph properties have quantum query complexity $\Omega(\sqrt{n})$, and gave a non-monotone property for which this lower bound is tight (up to polylogarithmic factors).

An interesting graph property for which a tight lower bound has not been found is the H-subgraph containment problem. In the most general form of this problem, we are asked to determine whether or not the input graph contains the fixed graph H as a subgraph. The best known lower bound for this property is only $\Omega(n)$. A special case is the property of containing a triangle, and the best known lower bound for this problem is also $\Omega(n)$. Le Gall [79] has described a quantum query algorithm that detects triangles using $O(n^{5/4})$ queries. Under the promise that the input graph either contains a triangle as a subgraph, or does not contain it as a minor[1] (the so-called *subgraph/not-a-minor problem*), Belovs and Reichardt [26] provide an $O(n)$ quantum query algorithm based on span-programs, which can also be implemented time efficiently. Also under the promise of the subgraph/not-a-minor problem, Wang [120] gives a span-program-based algorithm capable of detecting a given tree as a subgraph in $\tilde{O}(n)$ time.

Monotone graph properties are those that are subgraph-closed – that is, every subgraph of a graph with the property also has that property. Likewise, a graph property

[1] A graph H is said to be a minor of a graph G if H can be formed by deleting vertices from G and contracting edges.

is minor-closed if every graph minor of a graph with the property also has that property. Over a series of papers, Robertson and Seymour [104] showed that all minor-closed graph properties can be described by a finite set of forbidden minors – graphs that do not appear as a minor of any graph possessing the property. Some minor-closed properties can also be characterised by a finite set of forbidden *subgraphs*.

The widely believed Aanderaa-Karp-Rosenberg conjecture [105] states that the deterministic and randomised query complexities of all monotone graph properties are $\Theta(n^2)$. Childs and Kothari [38] show that all minor-closed properties that cannot be characterised by a finite set of forbidden *subgraphs* have quantum query complexity $\Theta(n^{3/2})$. On the other hand, they show that all minor-closed properties and sparse graph properties that can be characterised by finitely many forbidden subgraphs can be determined in $o(n^{3/2})$ queries. Belovs and Reichardt [26] extended this result to show that any minor-closed property that can be characterised by exactly one forbidden subgraph, which must necessarily be a path or a (subdivided) claw, has query complexity $O(n)$.

Some of these previous results can be applied to finding cycles and detecting bipartiteness. To decide whether or not a graph is bipartite, Ārinš [17] designed a span program that gives rise to an (optimal) $O(n^{3/2})$ quantum query algorithm. Our algorithm is inspired by this span program, and makes use of the s-t connectivity span program of Belovs and Reichardt as a sub-routine in a similar manner. We also provide a time (and space) efficient implementation.

Piddock [97] described a span-program-based quantum query algorithm for detecting cycles of constant fixed length, subject to the subgraph/not-a-minor promise, which requires $O(n^{3/2})$ queries to the input for odd length cycles, and $O(n)$ queries for even length cycles. This is in contrast to the present work, which detects cycles of arbitrary length (with no promise on the input). Within the adjacency array model, Dürr et al. [47] suggest a quantum query algorithm for deciding bipartiteness in $O(n)$ queries, and using $O(n \log n)$ space. An example of a reduction to s-t connectivity given in terms of span programs is the work of Jeffery and Kimmel [66], in which the problem of evaluating NAND-trees is reduced to the problem of s-t connectivity on certain graphs.

The algorithms presented in this chapter achieve optimal time complexity up to polylogarithmic factors, but require only $O(\log n)$ classical and quantum bits of storage. Thus, we emphasise that our algorithms are also space efficient, with respect to the number of classical and quantum bits of storage that they require. We assume that we have

access to quantum RAM[2] (QRAM [51]), which we use for storage, and that we are given access to an oracle that lets us evaluate edges of G, and which isn't counted towards the space bound. Therefore our measure of space efficiency differs somewhat to other, alternative definitions. For example, in investigating the computational power of space-bounded quantum Turing machines, Watrous [121, 122] measured the space requirements of the quantum (and classical) Turing machines in terms of the number of bits required to encode configurations of these machines (i.e. the position of the work tape head, and the work tape itself). Instead, we work in the (more standard) circuit model of computation, and consider the size of the QRAM required for our algorithms to run. Our algorithms could be simulated in the space-bounded quantum Turing machine model, but this might require a longer run-time due to the use of input, work, and output tapes in place of a QRAM and input oracle.

3.1.2 Subsequent related work

Following the completion and publication of the work in this chapter, DeLorenzo, Kimmel, and Witter [45] described span-program-based quantum algorithms for detecting cycles and testing bipartiteness, based on similar reductions to s-t connectivity. They made use of the improved characterisations of span program witness sizes from Jarret et al [64], which enabled them to construct simpler reductions from cycle detection to s-t connectivity than it was possible for us to use in this chapter. This characterisation also allows them to obtain improved performance when the input graph is promised to have either large circuit rank, or a small number of edges. They also describe an algorithm for estimating the circuit rank of a graph, all whilst retaining the logarithmic space complexity that arises from span-program-based algorithms.

These results demonstrate the effectiveness of the improved characterisations of span program witness sizes for designing algorithms using span programs for s-t connectivity. This leaves open the question of whether an improved characterisation of span program witness size for more general span programs (i.e. not just span programs for s-t connectivity) might yield faster quantum algorithms for a wider variety of problems.

[2]For our purposes, we assume that we have access to some memory device (the QRAM), which stores classical information (i.e. the information about the edges of the input graph), but can be accessed in quantum superposition.

3.1.3 Organisation of chapter

We begin by introducing some useful results and preliminary material in Section 3.2. In Section 3.3, we present a reduction of the problem of cycle detection in a graph G to the problem of s-t connectivity in some ancillary graph that is constructed from G, which is the main ingredient for the algorithms that follow. Section 3.4 presents a randomised algorithm for deciding whether a given vertex in a graph is a part of a cycle, and discusses the probability with which this algorithm fails. Section 3.5 describes a more general algorithm that allows the detection of arbitrary cycles, and then Section 3.6 explains how to modify this algorithm to decide whether or not a graph is bipartite. Section 3.7 discusses how to obtain an efficient algorithm in the adjacency array model, by using a quantum walk in place of the span-program-based s-t connectivity algorithm used in the previous sections, and ends with some lower bounds on the two problems in this model. Finally, section 3.8 provides a proof of correctness of the reduction given in Section 3.3.

3.2 Preliminaries

We will make use of the following result of Belovs and Reichardt [26] from Chapter 2, restated here for convenience.

Theorem 5. *[Combination of Theorems 3 and 9 from [26]] Consider the s-t connectivity problem on a graph G given by its adjacency matrix. Assume there is a promise that if s and t are connected by a path, then they are connected by a path of length at most d. Then there exists a bounded-error quantum algorithm that determines whether s and t are connected in $\tilde{O}(n\sqrt{d})$ time and uses $O(\log n)$ bits and qubits of storage, and which fails with probability at most $1/10$.*

The proof of this theorem can be found in Section 2.3.3.

We will also require some facts about *k-wise independent hash functions*:

Definition 2 ([88])**.** *Let U be a collection of items with $|U| \geq n$ and let $V = \{0, 1, ..., n-1\}$. A family of hash functions $\mathcal{H}$ from U to V is said to be* strongly k-universal *if, for any elements $x_1, x_2, ..., x_k \in U$, any values $y_1, y_2, ..., y_k \in V$, and a hash function h chosen*

uniformly at random from $\mathcal{H}$, we have

$$\Pr[(h(x_1) = y_1) \cap (h(x_2) = y_2) \cap \cdots \cap (h(x_k) = y_k)] = \frac{1}{n^k}.$$

We will be interested in the case where $k = n = 2$. In this case, the values $h(x_1), h(x_2)$ are pairwise independent, since the probability that they take on any pair of values is $\frac{1}{n^2} = \frac{1}{4}$. The simplest construction, which suffices for our purposes, is to use functions $h : \{0,1\}^m \to \{0,1\}$ of the form $h(x) = \langle a, x \rangle + b \mod 2$, where $\langle a, x \rangle = \sum_{i=1}^{m} a_i x_i \mod 2$ [88]. Each function is parameterised by two values $a \in \{0,1\}^m, b \in \{0,1\}$. To achieve pairwise independence of the values $h(x_1), h(x_2)$ for $x_1, x_2 \in \{0,1\}^m$, we therefore require $m + 1$ truly random bits to specify a and b. Doing so gives us $N = 2^m$ pairwise independent 'random' bits.

To use a hash function to assign a value in $\{0,1\}$ to each of N elements, we require $O(\log N)$ bits to specify the hash function h, from which $h(x)$, $x \in [N]$ can be calculated in $O(\log^2 N)$ time [88].

3.3 Reduction of Cycle Detection to s-t Connectivity

Let $G = (V, E)$ be a connected, undirected graph on n vertices. Fix some arbitrary orientation of the edges $(u, v) \in E$ by directing edges from $u \to v$ if $v > u$, $v \to u$ otherwise. G is now a directed graph.

Now consider an ancillary graph $H = (V', E')$, where $V' = \{s, t\} \cup \{v_b : v \in V, b \in \{0,1,2\}\}$ and $E' = \{(u_b, v_{b+1 \mod 3}) : (u, v) \in E, b \in \{0,1,2\}\} \cup \{(s, k_0), (t, k_1)\}$ for some $k \in V$. Intuitively, we split each vertex $v \in V$ into three vertices v_0, v_1, and v_2. Then, for each (directed) edge $(u, v) \in E$, we create three edges $(u_0, v_1), (u_1, v_2)$, and (u_2, v_0) in H. Finally, we add an edge between s and k_0 and between t and k_1, for some arbitrarily chosen vertex k.

To analyse the reduction to s-t connectivity, we introduce the notion of 'clockwise' and 'anticlockwise' edges. Given an undirected cycle, fix an arbitrary vertex v in the cycle that has at least 1 outgoing edge that makes up a part of the cycle (it is easy to verify that such a vertex must exist). Starting with one of the outgoing edges, we traverse the cycle from v back to v. Any edge that is oriented in the direction of traversal is defined as clockwise, and any edge oriented against the direction of traversal is defined as anticlockwise. Figure 3.1 provides an example to illustrate the notion of clockwise and anticlockwise edges, and gives

two examples of the form of the graph H constructed from a cycle on 4 vertices, showing how the reduction fails when the number of clockwise and anticlockwise edges is equal modulo 3 (more on this later).

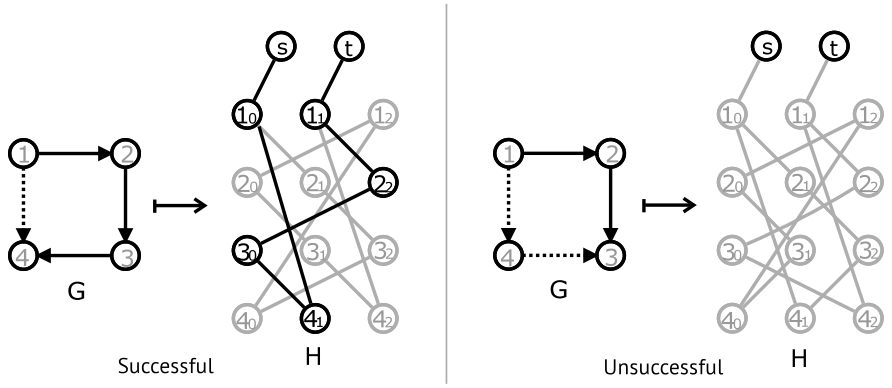

Figure 3.1: A successful (left) and unsuccessful (right) reduction to s-t connectivity for $G = C_4$, a cycle on 4 vertices, with clockwise and anticlockwise edges represented by solid and dashed arrows, respectively

The following lemma shows how the problem of detecting cycles can be reduced to the problem of s-t connectivity.

Lemma 3. *Let $G = (V, E)$ be a connected undirected graph, and let $H = (V', E')$ be defined as above. Then there is a path from s to t in H if and only if there is a cycle present in G, such that the difference $D := p - q$ between the number of clockwise edges p and anticlockwise edges q satisfies $D \not\equiv 0 \mod 3$. Furthermore, if k is chosen to be a vertex on the cycle, then the length of the path between s and t is at most $2c + 2$, where c is the length of the cycle.*

We defer to Section 3.8 for a full technical proof of this Lemma, and instead opt here to outline the intuition behind the reduction. To detect cycles, we wish to know whether it is possible to depart from some vertex on a walk around the graph, and arrive back at that vertex without having to retrace our steps. One way to tell if this is possible is to (arbitrarily) orient the edges of the graph, and keep track of how many times we traverse

an edge oriented *against* the direction of travel ('anti-clockwise' edges), and how many times we traverse an edge oriented *with* the direction of travel ('clockwise' edges), on a walk around the graph. If we arrive back at the starting vertex with an unequal number of each, then we must have, at some point in the walk, traversed a cycle. On the other hand, if we returned simply by retracing our steps, then the number of each type of edge encountered would be equal. In some cases, we may have traversed a cycle, but the number of clockwise and anticlockwise edges encountered could still be equal; however, randomly flipping the orientation of the edges in the graph and repeating is enough to reveal such cases with high probability.

We encode the direction of the edges of G into an ancillary graph H, which formalises the above intuition. That is, each of the three vertices in H associated to a vertex in G represents a walker being at that vertex with a 'parity' that is either 0,1 or 2, where we use parity to keep track of how many clockwise and anticlockwise edges we have seen. Every time a clockwise edge is traversed, we add one to the parity (modulo 3), and every time an anticlockwise edge is traversed, we subtract one from the parity. Then a path from, say, k_0 to k_1 or k_2 in H represents a path in G in which it is possible to perform a walk from vertex k back to vertex k, whilst encountering a differing number of clockwise and anticlockwise edges, and hence, a cycle. It can be shown that if there is a path from k_0 to k_2 in H, then there is also a path from k_2 to k_1, which shows that there is a path from s to t in this case. In the case that there is a cycle, but the orientations of the edges of the graph are such that the number of clockwise and anticlockwise edges that must be traversed on the path from k, through the cycle, and then back to k are equal modulo 3, then there will be no such path from k_0 to k_1 or k_2. Instead, there will only be a path from k_0 back to k_0 (and from k_1 to k_1, etc.). In this case, we say that the reduction fails.

Finally, if there is no cycle in G, then there will be no path from k back to k that traverses an unequal number of clockwise and anticlockwise edges, and in this case there will never be a path from s to t. Therefore, we see that there is a path from s to t in H if there is a cycle in G (except where the reduction fails), and no such path if there is no cycle in G.

3.4 Algorithm for Cycle Detection

In this section, we describe an algorithm that makes use of both the reduction of cycle detection to s-t connectivity, and the s-t connectivity algorithm of Belovs and Reichardt.

In particular, we prove:

Theorem 6. *There exists a quantum algorithm which, given as input a graph $G = (V, E)$, a vertex $k \in V$, and an integer d, outputs true with probability $\geq 9/20$ if G contains a cycle of length $l \leq d$ that includes k, and returns false with probability $\geq 9/10$ if it does not contain any cycle. The algorithm takes $\tilde{O}(n\sqrt{d})$ time and requires $O(\log n)$ space.*

Proof: By Lemma 3, the problem of detecting if a vertex k is included in a cycle reduces to the problem of s-t connectivity on an ancillary graph H, which is constructed from G. It is worth noting that Lemma 3 actually gives a stronger result – that there is a path from s to t in H when there exists some *path* from k to a cycle in G, provided that the cycle satisfies some constraints on the orientations of its edges. However, as we will see, for some inputs, the algorithm could fail to detect such cases with certainty. Thus, we restrict ourselves to the worst case – that in which the algorithm can only detect the presence of a cycle that includes the vertex k.

If there is some (efficient) map from the edges of H to the edges of G, then the s-t connectivity algorithm can be run on the graph H whilst only querying the input oracle for G. Additionally, we must show that the algorithm fails only with some constant probability, even when given a 'bad' input (one with an equal number of clockwise and anticlockwise edges (modulo 3)).

We begin by describing a randomised approach which causes the reduction to s-t connectivity to fail with only constant probability when given a bad input. Suppose we were to run the s-t connectivity algorithm on the graph H associated with an input graph G, which contains a cycle. If the adjacency matrix of the graph were such that the number of clockwise edges and the number of anticlockwise edges on the cycle were equal modulo 3, then the algorithm, as it stands, would fail to detect the cycle with certainty, as a result of Lemma 3.

We can prevent the algorithm from failing by flipping the direction of a single edge on the cycle. Recall that we are given some vertex $k \in V$, and add edges (s, k_0) and (t, k_1) in H before testing for a path between s and t. Our solution is to flip the direction of some random subset of the edges adjacent to vertex k, and show that this flips exactly one edge on the cycle with high probability ($\geq 1/2$). To choose a random subset of edges to flip, we colour every vertex in G with a colour chosen from $\{0, 1\}$. Then, for every edge adjacent to

k in G, if the vertex at the end of the edge is coloured 1, we flip the direction of the edge, and otherwise do nothing. The colouring is achieved using a family of pairwise independent hash functions $\mathcal{H}$ from $[n]$ to $\{0,1\}$. The pairwise independence gives the constraint that, for $x, y \in [n]$ and $a, b \in \{0,1\}$, and a hash function h chosen uniformly at random from $\mathcal{H}$,

$$\Pr[h(x) = a \cap h(y) = b] = \frac{1}{4}.$$

Let the two vertices adjacent to k in the cycle be a and b, and choose a pairwise independent hash function h uniformly at random from $\mathcal{H}$. Then, by pairwise independence, we have

$$\Pr[h(a) = h(b) = 0] = \Pr[h(a) = h(b) = 1] = \frac{1}{4} \qquad \text{and} \qquad \Pr[h(a) \neq h(b)] = \frac{1}{2}.$$

So the above method will fail to flip either of the edges $(a,k), (b,k)$ with probability $\frac{1}{4}$. Otherwise, with probability $\frac{1}{2}$ exactly one of the two edges will be flipped, and with probability $\frac{1}{4}$ both edges will be flipped. Thus, with probability at least $\frac{1}{2}$, the number of clockwise edges will no longer equal the number of anticlockwise edges modulo 3. By colouring the vertices of the graph using a pairwise independent hash function, the algorithm will fail with probability at most $\frac{1}{2}$ when given a 'bad' input.

On the other hand, if the algorithm is given some 'good' input, then this process may cause the algorithm to fail; however, this will happen with probability at most $1/2$. To see this, we consider what happens when one or both of the edges adjacent to k are flipped. Let p' and q' be the number of clockwise and anticlockwise edges present after the flipping process has taken place. Suppose that one of the edges (a,k) and (b,k) is clockwise, and the other anticlockwise. Then flipping both of the edges or flipping neither results in $p' = p$ and $q' = q$, in which case $p' - q' = p - q$. If the clockwise edge is flipped, then $p' - q' = p - q + 1$, and if the anticlockwise edge is flipped, then $p' - q' = p - q - 1$. In the former case, $p' - q' \equiv 0 \mod 3$ only if $p - q \equiv 2 \mod 3$, and in the latter case $p' - q' \equiv 0 \mod 3$ only if $p - q \equiv 1 \mod 3$. Hence, for such 'good' inputs, the algorithm will fail with probability at most $1/4$.

Suppose that both (a,k) and (b,k) are clockwise edges. Then flipping exactly one of them (which happens with probability $1/2$) results in $p' - q' = p - q + 1$. If $p - q \equiv 2 \mod 3$, then $p' - q' \equiv 0 \mod 3$, causing the algorithm to fail. If both edges are flipped (which happens with probability $1/4$), then $p' - q' = p - q - 1$ and so $p' - q' \equiv 0 \mod 3$ only when $p - q \equiv 1 \mod 3$. Thus, in this case, the algorithm transforms a good input to

a bad input with probability at most 1/2. A similar argument holds when both (a, k) and (b, k) are anticlockwise edges, hence showing that, given a 'good' input, the algorithm will fail with probability at most 1/2.

Now we consider the map from the edges of H to the edges of G. We can query the adjacency matrix of H implicitly by querying the entries of G's adjacency matrix. Given two vertices u_b and $v_{b'}$ in H, we test for the presence of the edge $(u_b, v_{b'})$ as follows. First we determine the direction of the edge (u, v) in G, if it were to exist. If $v > u$, then the edge is directed from $u \to v$, otherwise it is directed from $v \to u$. If $u = k$, then we look up the colour of vertex v as determined by our hash function, and vice versa if $v = k$. If the colour is 0, we do nothing; if it is 1, we flip the edge. Next we test whether the edge is allowed to exist. If the edge is directed from $u \to v$, then it is allowed only if $b' \equiv b + 1 \mod 3$. Similarly, if the edge is directed from $v \to u$, then it is allowed only if $b' \equiv b - 1 \mod 3$. Finally, if the edge is allowed to exist, then we test for its presence in G by querying the uv entry of G's adjacency matrix. If the result of the query is 1, then the edge exists and we return 1. In all other cases (the query returns 0, or the edge is not allowed), we return 0. We can use this map to run the s-t connectivity algorithm on H without explicitly constructing it. That is, rather than allowing the algorithm to query the input oracle for G, we allow it to query the circuit that implements the process described above, which will query the input oracle for G as appropriate.

We have shown that if the input graph G contains a cycle that includes vertex k, then there will be a path from s to t in H with probability at least 1/2. Now we consider the probability of detecting this path using the s-t connectivity algorithm of Belovs and Reichardt. Recall that their algorithm takes as input an upper bound d on the length of the path between s and t. By Theorem 5, if $2l + 2 \leq d$, then the s-t connectivity algorithm detects the presence of a path of length $2l + 2$ with probability at least 9/10 if one exists, and otherwise says that no path exists with probability at least 9/10. Therefore, if G contains a cycle of length $l \leq \frac{d-2}{2}$ that includes k, then with probability $p \geq 1/2$ there will be a path from s to t in H. By running the s-t connectivity algorithm on H, we will detect this path with probability 9/20. Conversely, if G does not contain a cycle, then there will be no path from s to t in H, and the s-t connectivity algorithm will return false with probability $\geq 9/10$.

Now we analyse the time (and space) complexity of this algorithm. The map from H to G runs in time polylog(n) and uses $O(\log n)$ space. The colouring step requires the use of a pairwise-independent hash function, which requires $O(\log n)$ space and $O(\log^2 n)$ time. By Theorem 5, we can test for s-t connectivity in an n vertex graph in $\tilde{O}(n\sqrt{d})$ time and $O(\log n)$ space, where d is an upper bound on the length of the path connecting s and t. Lemma 3 states that, if vertex k is contained within a cycle of length l, then there is a path from s to t in H of length at most $2l+2$. Taking $d \geq 2l+2$, we can therefore detect the presence of a cycle in G in time $\tilde{O}(n\sqrt{d})$. The s-t connectivity algorithm uses $O(\log n)$ space, and so the algorithm of Theorem 6 requires $\tilde{O}(n\sqrt{d})$ time and $O(\log n)$ space in total . □

3.5 Detecting Arbitrary Cycles

In the previous sections we described an algorithm that, given a graph $G = (V, E)$, a vertex $k \in V$, and an upper bound d on the length of a cycle in G, outputs 1 with probability $\geq 9/20$ if there is a cycle of length $l \leq d$ in G that contains k, and outputs 0 with probability $\geq 9/10$ if G does not contain a cycle. By repeating the algorithm $O(\log 1/\epsilon)$ times and using majority voting, we obtain an algorithm $\mathcal{A}$ that fails (i.e. returns false positives or false negatives) with probability at most ϵ. In particular, we could reduce the probability of failure of $\mathcal{A}$ to $\frac{1}{\text{poly}(n)}$ with only an $O(\log n)$ overhead. As the overall algorithm calls $\mathcal{A}$ poly(n) times, we can reduce the probability of failure of the overall algorithm to an arbitrary constant. This holds even for quantum algorithms calling $\mathcal{A}$ in superposition [28, 61].

We can use this algorithm as a sub-routine for a more general algorithm that is capable of detecting the presence of arbitrary cycles in G. In order to do this we need to overcome two (related) obstacles. The first is that, ahead of time, we do not know an upper bound on the length of a cycle in G. The second is that we do not know which vertices in the graph G might be a part of a cycle. Both of these problems can be overcome by repeatedly guessing at the length of the cycle in G, and then using a variant of Grover search to look for a vertex that is a part of a cycle. In particular, we make use of the following result from [31].

Theorem 7. *Given oracle access to some Boolean function $f : [N] \to \{0, 1\}$, such that the set of 'solutions' $M = \{x \in [N] : f(x) = 1\}$ has unknown size $t = |M|$, and a lower*

bound $a \leq t$ on the number of solutions. Then there exists a quantum algorithm $\mathbf{Grover}_a$ *that returns a solution if there is one with probability $\geq 2/3$, or returns 'no solution' with probability $\geq 2/3$ otherwise, using $O(\sqrt{N/a})$ queries to the oracle, and $O(\log N)$ space.*

Note that this theorem is not explicitly stated in [31], but follows easily from their results.

We use this algorithm repeatedly, with exponentially increasing values of $a = d/2$, to search over the set of vertices in the input graph, using the algorithm $\mathcal{A}$ as an oracle. That is, for each vertex $v \in V$, we will call $\mathcal{A}$ with our guess d at the length of the cycle, and with the vertex k set to v. Then, starting at $d = 4$, we double the size of d at each iteration until either we find a cycle, or $d \geq N$. In particular, we perform the following:

1. For $i = 2$ to $\lceil \log_2 n \rceil$:
   (a) Run $\mathbf{Grover}_a$ on the vertices of the graph with $d = 2^i, a = 2^{i-1}$.
   (b) If the algorithm returns a solution, then output that solution and exit, otherwise continue.
2. Output 'no cycle exists'.

This detects the presence of a cycle with high probability if one exists, since, as soon as i becomes large enough that $d = 2^i \geq l$, the cycle detection algorithm detects the presence of cycles with (arbitrarily) high probability, as a consequence of Theorem 6. At this point, $\mathbf{Grover}_a$ finds a good solution (i.e. a vertex that causes the cycle detection algorithm to accept) with high probability, since the number of marked elements is equal to l, and satisfies $d/2 + 1 \leq l \leq d$, and therefore $a < l$. If G does not contain a cycle, then $\mathcal{A}$ will return false with high probability on all vertices, and therefore $\mathbf{Grover}_a$ will return 'no solution' with high probability every time it is run, and the above algorithm will output 'no cycle exists' with high probability.

To analyse the time complexity of the algorithm, we will consider what happens when there is no cycle present. If a cycle is present, then it will be found with high probability in one of the iterations and the algorithm will exit early, requiring less time. In the i^{th} round of the algorithm, we run $\mathbf{Grover}_a$ with $a = 2^{i-1}$, which requires $O(\sqrt{\frac{n}{2^{i-1}}})$ applications of $\mathcal{A}$ and $\mathcal{A}^{-1}$, each of which take time $\tilde{O}(n\sqrt{d}) = \tilde{O}(n\sqrt{2^i})$. Therefore, the time taken to run the i^{th} round is $O(\sqrt{\frac{n}{2^{i-1}}}) \cdot \tilde{O}(n\sqrt{2^i}) = \tilde{O}(n^{3/2})$. We run at most $\lceil \log_2 n \rceil$ rounds, requiring

$\tilde{O}(n^{3/2})$ time in total.

In summary, by making use of Grover search and repeatedly guessing at increasing cycle lengths, we are able to find a vertex $k \in V$ that is part of a cycle in G with probability $\geq 2/3$ if such a vertex exists, and return false with probability $\geq 2/3$ if G contains no cycle. This requires $\tilde{O}(n^{3/2})$ time and $O(\log n)$ bits and qubits of storage.

3.6 Deciding Bipartiteness

We can view the algorithm for cycle detection as a special case of a more general algorithm. Currently, we arbitrarily orient the edges of an initially undirected graph, and accept if this forms a cycle in which the number of clockwise and anticlockwise edges are unequal modulo 3. We might ask what happens when we look for cycles with an unequal number of clockwise and anticlockwise edges modulo some other constant s. This would change the reduction to s-t connectivity by modifying the structure of the graph H that is constructed from G. In particular, each vertex in G would be split into s sub-vertices in H, and for each edge (u, v) in G we would have s corresponding edges $(u_0, v_1), (u_1, v_2), ..., (u_{s-2}, v_{s-1}), (u_{s-1}, v_0)$.

The cases $s > 3$ behave similarly to the case $s = 3$, and are uninteresting. However, the case $s = 2$ proves useful. In this case, the original algorithm (i.e. without random colouring) fails when the number of clockwise edges and the number of anticlockwise edges in every cycle C differs by some multiple of 2. This will be the case if C is of even length, and is independent of the orientation of the individual edges. Conversely, no odd-length cycle will cause the algorithm to 'fail'. This means that, given some graph $G = (V, E)$ and a vertex $k \in V$, the algorithm will accept (with certainty) if k is a part of an odd-length cycle, and reject otherwise. A graph is bipartite if and only if it contains no odd-length cycles. Thus, setting $s = 2$ (and omitting the colouring step) allows us to decide whether or not a graph is bipartite. Since we have only removed a step of the algorithm, it still runs in time $\tilde{O}(n^{3/2})$ and requires $O(\log n)$ space.

3.7 Cycle Detection in the Adjacency Array Model

Rather than an adjacency matrix, we may be provided with an adjacency array description of an undirected graph as an input. In this model, we are given a list of neighbours for each vertex of the graph. Following [47], we assume that we are given the following information:

- The degrees of the vertices $d_1, d_2, ..., d_n$ and for every vertex u an array with its neighbours $f_i : [d_i] \to [n]$. So $f_i(j)$ returns the j^{th} neighbour of vertex i, according to some arbitrary but fixed numbering of the outgoing edges of i.

Note that if we are not given the degrees ahead of time, it is possible to determine the degree of each vertex to within a factor of two of its true value in $O(\log d_m)$ time and $O(\log n)$ space by using a form of binary search, where d_m is the maximum degree of a vertex in the graph. This is sufficient to be able to count the number of edges (up to a factor of two), and to implement the quantum walk operators described below. Assuming that we know the degrees of the vertices, we can calculate the number of edges m as $m = \frac{1}{2}\sum_{u \in V} d_u$. In this way, we can discard any n-vertex input graph with $m \geq n$, since such a graph must necessarily contain a cycle. This means that we only need to consider graphs with $m < n$.

If we can map from the edges of H to the edges of G in the adjacency array model, then we can run a quantum walk on H, starting from s and with t as the single marked vertex, which, by a result of Belovs [24, 25], can be used to decide s-t connectivity. By making use of the reduction of Lemma 3, and the version of Grover search outlined in Section 3.5, we can use a quantum walk in place of the span-program-based s-t connectivity algorithm to detect cycles in the adjacency array model in time $\tilde{O}(n\sqrt{d_m})$. For this approach to work, we need to be able to map from the edges of H to the edges of G in a similar manner to before, and we need to be able to implement the quantum walk operators efficiently.

3.7.1 Map from the edges of H to the edges of G

Given a vertex u_b in H, we want to be able to produce an array of its neighbours: i.e. we need a function $g_{u_b} : [d'_{u_b}] \to [3n]$, where d'_{u_b} is the degree of vertex u_b in H, so that $g_{u_b}(j)$ returns the j^{th} neighbour of vertex u_b in H. First, note that the degree of the vertex u_b in H is the same as the degree of the vertex u in G – that is, $d'_{u_b} = d_u$, unless u_b is connected to s or t. Also, recall that the neighbours of vertex u_b in H are all of the form $v_{b'}$, where v is a neighbour of u in G, and b' depends on the orientation of the edge (u, v).

In general, suppose that we want to compute $g_{u_b}(j)$, the j^{th} neighbour of vertex u_b. We know that it will be $v_{b'}$, for $v = f_u(j)$ (i.e. the j^{th} neighbour of vertex u in G) and some $b' \in \{0, 1, 2\}$. To calculate b', we use the same process described in Section 3.4, which begins by determining the direction of the edge (u, v) as follows: if $v > u$, then the edge is directed $u \to v$, otherwise it is directed from $v \to u$. If $u = k$, then we look up the colour of vertex v as determined by our hash function, and vice versa if $v = k$. If the colour is

0, we do nothing; if it is 1, we flip the edge. Finally, if the edge is directed $u \to v$, then $b' = b + 1 \mod 3$, otherwise $b' = b - 1 \mod 3$. We then return the answer: $g_{u_b}(j) = v_{b'}$.

The functions g_{u_b} for every vertex u_b in H can be computed using the function f_u given by the adjacency array for vertex u, as well as some other operations that require $O(\text{polylog}(n))$ time and $O(\log n)$ space. Therefore, we can implement a quantum walk on the graph H by implicitly querying the adjacency arrays for the vertices in G. The following section describes such a quantum walk.

3.7.2 Quantum walk for s-t connectivity

We use the quantum walk algorithm presented by Belovs in [24] and [25] for detecting a marked vertex, by setting the starting vertex to s and setting t to be the only marked vertex in the graph. Then the presence of a path from s to t can be detected in $\tilde{O}(\sqrt{ln})$ steps of the quantum walk, where l is an upper bound on the length of the path from s to t. Thus, given a vertex k in the graph and some upper bound d on the length of a cycle, we can detect the presence of a cycle that includes k in $\tilde{O}(\sqrt{dn})$ steps of the quantum walk. We can then use the same variant of Grover search as in Section 3.5 to detect the presence of an arbitrary cycle in $\tilde{O}(n)$ steps. Later in this section, we show that a single step of the appropriate quantum walk can be implemented in a time that depends on the degrees of the vertices in the input matrix.

We begin by considering the graph $H = (V', E')$ corresponding to the graph $G = (V, E)$. In order to apply the quantum walk, we must first make H bipartite (which, in general, it will not be to begin with). To do this, we transform $H = (V', E')$ into $H' = (V'', E'')$ with vertex set $V'' = V' \times \{0, 1\}$ and edge set $E'' = \{((u, b), (v, b \oplus 1)) : (u, v) \in E', b \in \{0, 1\}\}$. The graph is now bipartite, and we can still efficiently compute the neighbours of each vertex. Let A be the set of vertices $\{(u, 0) : u \in V'\}$ and B the set of vertices $\{(u, 1) : u \in V'\}$, and let d_u denote the degree of vertex u.

For the algorithm to work, we first set $s = (k_0, 0)$ and $t = (k_1, 0)$. This choice of s and t is sufficient to enable us to detect cycles that create even length paths from k_0 to k_1 in H. We can then set $s = (k_0, 0)$ and $t = (k_1, 1)$ to detect those cycles that create odd length paths in H. Then by running the algorithm twice, and accepting if at least one of these runs accept, we can detect arbitrary cycles containing vertex k.

The vectors $\{|s\rangle \otimes |e_s\rangle\} \cup \{|u\rangle \otimes |v\rangle : (u, v) \in E''\}$ give the basis for the vector space of the quantum walk, which starts in the state $|s\rangle |e_s\rangle$. Here, we view e_s as a 'dangling'

edge incident to vertex s, which is defined to be the $(d_s+1)^{\text{th}}$ neighbour of s, where d_s is the degree of vertex s. That is, we add an entry to the adjacency array for s so that $f_s(d_s+1) = e_s$.

Let $\mathcal{H}_u = \text{span}(\{|u\rangle\,|v\rangle : (u,v) \in E''\})$ denote the local space of vertex $u \neq s$, and $\mathcal{H}_s = \text{span}(\{|s\rangle\,|v\rangle : (s,v) \in E''\} \cup \{|s\rangle\,|e_s\rangle\})$. A step of the quantum walk is defined as $R_A R_B$ where $R_A = \bigoplus_{u\in A} D_u$ and $R_B = \bigoplus_{u\in B} D_u$. Each diffusion operator D_u acts only on $\mathcal{H}_u$, and is defined as follows:

- D_t is the identity.
- If $u \notin \{s,t\}$, then $D_u = I - 2\,|\zeta_u\rangle\,\langle\zeta_u|$, where

$$|\zeta_u\rangle = \frac{1}{\sqrt{d_u}} \sum_{(u,v)\in E} |u\rangle\,|v\rangle\,.$$

- $D_s = I - 2\,|\zeta_s\rangle\,\langle\zeta_s|$, where

$$|\zeta_s\rangle = \frac{1}{\sqrt{1+d_s C d}} \left(|s\rangle\,|e_s\rangle + \sqrt{Cd} \sum_{(s,v)\in E} |s\rangle\,|v\rangle \right), \tag{3.1}$$

  for some constant C.

Then we have the following result, which follows directly from Theorem 4 of [24]:

Theorem 8. *Given a graph $G = (V,E)$ such that $|E| \leq n$, two vertices s and t in G, and an integer d, then by applying R_A and R_B $O(\sqrt{dn})$ times we can detect a path from s to t with probability $\geq 2/3$ if a path of length $l \leq d$ exists, or otherwise say that no path exists with probability $\geq 2/3$.*

To see how this follows from the results of [24], we recall that the quantum algorithm described in [24] is able to detect one or more marked vertices in a graph, starting from some subset of the vertices of that graph. By setting the starting vertex to s, and choosing t to be the only marked vertex, the algorithm will accept with probability $\geq 2/3$ if there is a path from s to t in the graph, and reject with probability $\geq 2/3$ otherwise.

By using the same arguments given in Section 3.5, we can make use of the algorithm of Theorem 8 to detect arbitrary cycles by applying the operators R_A and R_B $\tilde{O}(n)$ times. Furthermore, we may use a special case of this algorithm to decide bipartiteness, also

requiring $\tilde{O}(n)$ applications of R_A and R_B. The efficiency of the algorithm then depends on the efficiency with which we can implement the reflections R_A and R_B.

3.7.3 Implementing R_A and R_B

We will restrict our attention to R_A; R_B is implemented similarly (and is actually easier, since the vertex s, which requires a more complex diffusion operator, is in A). We need to implement

$$\begin{aligned} R_A = \bigoplus_{v\in A} D_v &= \bigoplus_{v\in A} (I - 2\left|\zeta_u\right\rangle\left\langle\zeta_u\right|) \\ &= \bigoplus_{v\in A} (I - 2\left|v\right\rangle\left\langle v\right| \otimes \left|\phi_v\right\rangle\left\langle\phi_v\right|) \\ &= I - 2\sum_{v\in A} (\left|v\right\rangle\left\langle v\right| \otimes \left|\phi_v\right\rangle\left\langle\phi_v\right|) \end{aligned}$$

where we define $\left|\phi_v\right\rangle := \frac{1}{\sqrt{d_v}} \sum_{i\in[d_v]} \left|f_v(i)\right\rangle$ for $v \neq s$ (recall that we are given the degrees $d_1, d_2, ..., d_n$ of each vertex, and for each vertex v a function $f_v : [d_v] \to [n]$, so that $f_v(j)$ returns the j^{th} neighbour of vertex v.). $\left|\phi_s\right\rangle$ is defined slightly differently: there is an extra $\left|s\right\rangle$ term in Eq. (3.1) originating from the dangling edge e_s incident to s. Then we may define

$$\left|\phi_s\right\rangle := \frac{1}{\sqrt{1 + d_s Cd}} \left(\left|f_s(d_s + 1)\right\rangle + \sqrt{Cd} \sum_{i\in[d_s]} \left|f_s(i)\right\rangle \right).$$

Intuitively, the $\left|\phi_v\right\rangle$ states represent the neighbours of the vertex v, in correspondence with the states $\left|\zeta_v\right\rangle$ given above.

If we can implement a map $\left|v\right\rangle\left|0\right\rangle \mapsto \left|v\right\rangle\left|\phi_v\right\rangle$ for each vertex $v \in A$, then we may implement R_A by performing the local reflections $I - 2\left|v\right\rangle\left\langle v\right| \otimes \left|\phi_v\right\rangle\left\langle\phi_v\right|$ in parallel for each $v \in A$. It is possible to implement the map efficiently for every vertex in superposition – in particular, we have the following result:

Lemma 4. *R_A and R_B can be implemented using $O(\sqrt{d_m})$ queries to the adjacency array, and* polylog(n) *additional operations per query, where d_m is the maximum degree of any vertex in the graph.*

Proof: We can query the adjacency array of each vertex in superposition. In particular,

let

$$|\psi_v\rangle = \frac{1}{\sqrt{d_v}} \sum_{i\in[d_v]} |i\rangle\,|f_v(i)\rangle$$

be the state that results from querying the adjacency array of vertex v in superposition. Recall that we define $|\phi_v\rangle := \frac{1}{\sqrt{d_v}} \sum_{i\in[d_v]} |f_v(i)\rangle$. We want to produce $|\phi_v\rangle$ from the state $|\psi_v\rangle$.

Let $|+_v\rangle := \frac{1}{\sqrt{d_v}} \sum_{i\in[d_v]} |i\rangle$. If we perform the $\{|+_v\rangle\langle+_v|, I-|+_v\rangle\langle+_v|\}$ measurement on the first register, then the first outcome is obtained with probability $1/d_v$, and in this case the second register collapses to $|\phi_v\rangle$. We want to maximise the probability of measuring $|+_v\rangle$ in the first register, in order to produce the desired state in the second register.

In fact, we can increase the probability of measuring $|+_v\rangle$ to certainty using exact amplitude amplification, which also gives us the desired state in the second register. Define

$$|\Phi_v\rangle := |+_v\rangle\,|\phi_v\rangle\,,$$

and two projectors

$$\begin{aligned} P_+ &:= |+_v\rangle\langle+_v| \otimes I \\ P_\psi &:= |\psi_v\rangle\langle\psi_v|\,. \end{aligned}$$

We see that

$$\begin{aligned} P_+\,|\psi_v\rangle &= \frac{1}{\sqrt{d_v}} \sum_{i\in[d_v]} \left(|+_v\rangle\langle+_v|i\rangle \otimes |f_v(i)\rangle\right) \\ &= \frac{1}{\sqrt{d_v}} \sum_{i\in[d_v]} \left(\frac{1}{\sqrt{d_v}}\,|+_v\rangle \otimes |f_v(i)\rangle\right) \\ &= \frac{1}{\sqrt{d_v}}\,|+_v\rangle \otimes \frac{1}{\sqrt{d_v}} \sum_{i\in[d_v]} |f_v(i)\rangle \\ &= \frac{1}{\sqrt{d_v}}\,|\Phi_v\rangle \\ &= |\Phi_v\rangle\langle\Phi_v|\psi_v\rangle \end{aligned}$$

and

$$\begin{aligned} P_+ |\Phi_v\rangle &= |+_v\rangle \langle +_v|+_v\rangle \otimes \frac{1}{\sqrt{d_v}} \sum_{i\in[d_v]} |f_v(i)\rangle \\ &= |+_v\rangle \otimes \frac{1}{\sqrt{d_v}} \sum_{i\in[d_v]} |f_v(i)\rangle \\ &= |+_v\rangle \otimes |\phi_v\rangle = |\Phi_v\rangle \\ &= |\Phi_v\rangle \langle \Phi_v|\Phi_v\rangle \end{aligned}$$

That is, in the subspace spanned by $\{|\psi_v\rangle, |\Phi_v\rangle\}$, the projector P_+ acts as the projector $|\Phi_v\rangle \langle \Phi_v|$. We can define two operators $R_+ = I - 2P_+$ and $R_\psi = I - 2P_\psi$, which, taking into account the observation noted above, are inversions about the spaces spanned by $|\Phi_v\rangle$ and $|\psi_v\rangle$, respectively (within the subspace spanned by $\{|\psi_v\rangle, |\Phi_v\rangle\}$). Thus, by alternating the two reflections, we can use the exact variant of amplitude amplification in the standard way to produce the state $|\Phi_v\rangle$ from the state $|\psi_v\rangle$. In particular, we apply $Q^m := (R_\psi R_+)^m$ to the initial state $|\psi_v\rangle$ for some integer m, followed by one application of a modified version of Q which performs smaller rotations in order to make the algorithm exact. Since Q preserves the subspace spanned by $\{|\psi_v\rangle, |\Phi_v\rangle\}$, then (by the arguments above) the algorithm will produce the desired state $|\Phi_v\rangle$. We have that $|\langle \psi_v|\Phi_v\rangle|^2 = \frac{1}{d_v}$, and so we can choose an $m = \Theta(\sqrt{d_v})$ to obtain the state $|\Phi_v\rangle$ in $\Theta(\sqrt{d_v})$ time [34].

In other words, we can produce a state $|v\rangle |+_v\rangle |\phi_v\rangle$ from an initial state $|v\rangle |0\rangle |0\rangle$ in $\Theta(\sqrt{d_v})$ time. We can then uncompute the value in the second register, giving us $|v\rangle |\phi_v\rangle |0\rangle$ (where we have swapped the final two registers for clarity). Let U be the operator that maps the state $|v\rangle |0\rangle |0\rangle$ to $|v\rangle |\phi_v\rangle |0\rangle$. Then the diffusion operator D_v may be implemented by US_0U^{-1}, where S_0 changes the sign of the amplitude if and only if the final two registers are in the all zero state. That is, it performs the map $S_0 : |v\rangle |0\rangle |0\rangle \mapsto -|v\rangle |0\rangle |0\rangle$ for all $v \in A$ and implements the reflection

$$S_0 = I - 2 \sum_{v\in A} |v\rangle \langle v| \otimes |0\rangle \langle 0| \otimes |0\rangle \langle 0| .$$

Therefore,

$$\begin{aligned} US_0U^{-1} &= U(I - 2\sum_{v\in A} |v\rangle\langle v| \otimes |0\rangle\langle 0| \otimes |0\rangle\langle 0|)U^{-1} \\ &= I - 2\sum_{v\in A} |v\rangle\langle v| \otimes |\phi_v\rangle\langle\phi_v| \otimes |0\rangle\langle 0| \\ &= R_A \end{aligned}$$

by the definition of R_A.

Since we are applying U to all vertices in superposition, it will be necessary to clarify how the amplitude amplification part of U can be applied to a superposition over vertices. In order to produce the desired state for some vertex v, amplitude amplification needs to be performed for a number of iterations that depends upon the degree of that vertex. Since different vertices will have different degrees, the number of iterations required will vary between vertices. To address this, we can define an algorithm $AA(v, d_v, d_m)$, where d_m is the maximum degree of any vertex. The algorithm will perform amplitude amplification for the correct number of iterations for vertex v, and then will do nothing (i.e. apply the identity operator) for the remaining iterations. In this way, the amplitude amplification routines stop and wait for the vertex with the largest degree, and thus the amplitude amplification step can be applied to all vertices in superposition, requiring time $O(\sqrt{d_m})$.

Since the diffusion operators required for the vertices s and t are different, it is worth discussing their implementations separately. The diffusion operator for vertex t is easy to implement, since it is the identity. Vertex s has a more complicated operator; however, all we need to change is the operation that maps the state $|s\rangle|0\rangle|0\rangle$ to the state $|s\rangle|\psi_s\rangle$. For all $v \notin \{s, t\}$, we simply produce a uniform superposition over the neighbours of v. For s, as discussed above, we have an additional term corresponding to an additional edge incident to vertex s, and therefore we need to be able to produce the state

$$|+_s\rangle = \frac{1}{\sqrt{1 + d_sCd}} |d_s\rangle + \sqrt{\frac{Cd}{1 + d_sCd}} \sum_{i=0}^{d_s-1} |i\rangle$$

in the second register, which will be used to produce $|\phi_s\rangle$. In order to do this, let K be a unitary operator on the space spanned by $\{|0\rangle, |d_s\rangle\}$ that maps $|0\rangle$ to $\frac{1}{\sqrt{1+d_sCd}} |d_s\rangle + \sqrt{\frac{d_sCd}{1+d_sCd}} |0\rangle$. Then let F be the Fourier transform on the space spanned by $\{|i\rangle : i \in$

$\{0, \ldots, d_s - 1\}\}$ that maps $|0\rangle$ to the uniform superposition. Then the required state can be produced by applying FK to the state $|0\rangle$. We can then proceed as in the more general case to implement the local reflection.

In general, the time taken to implement the operators R_A and R_B will depend on the degrees of the vertices in the graph. In particular, if the maximum degree of any one vertex is d_m, then we will have to use at most $O(\sqrt{d_m})$ iterations of amplitude amplification in order to implement the local reflections in parallel. Therefore $O(\sqrt{d_m})$ queries are required to implement $R_A R_B$, and the time complexity is the same up to polylog factors in n . □

Here we have assumed that we only have access to an oracle O that acts on basis states as

$$O\,|v\rangle\,|j\rangle\,|0\rangle = |v\rangle\,|j\rangle\,|f_v(j)\rangle\,.$$

If instead we assume that we have access to a (more powerful) oracle O' that acts on basis states as

$$O'\,|v\rangle\,|j\rangle = |v\rangle\,|f_v(j)\rangle\,,$$

then we can implement R_A and R_B using a only a constant number of queries, by simply querying O' in superposition. This would remove the $O(\sqrt{d_m})$ overhead from the running time of the algorithm from Section 3.7.

In summary, since s-t connectivity can be decided using $\tilde{O}(\sqrt{dn})$ steps of the quantum walk (Theorem 8), and each step takes $\tilde{O}(\sqrt{d_m})$ time (Lemma 4), the algorithm for deciding s-t connectivity in the adjacency array model takes $\tilde{O}(\sqrt{d_m dn})$ time. Combining this with the Grover search method outlined in Section 3.5, this means that we can detect cycles of arbitrary length in $\tilde{O}(\sqrt{d_m}n)$ time. Moreover, the quantum walk can be implemented with only a logarithmic number of qubits, and therefore the algorithm requires only $O(\log n)$ space, as before. Note that if we are given no promise on the maximum degree of the graph, then the algorithm will take $\tilde{O}(n^{3/2})$ time, matching the time complexity of the algorithm in the adjacency matrix model. It is also worth pointing out that we can reduce the time complexity to $\tilde{O}(n)$, by using a different algorithm for deciding s-t connectivity (e.g. [47]). However, in this case, we lose space efficiency, and require $O(n \log n)$ space.

3.7.4 Lower bounds

In this section we obtain $\Omega(n)$ quantum query lower bounds for both problems in the adjacency array model, which follow from almost the same reduction used by Dürr et al. in [47] to prove a lower bound on s-t connectivity – namely a reduction from the PARITY problem. The PARITY problem is defined as follows: given a bit-string $x \in \{0,1\}^n$ of length n, is there an even or an odd number of bits set to 1? It is well known that the quantum (and classical) query complexity of solving the PARITY problem is $\Theta(n)$. We reproduce the reduction here, and show how it leads to lower bounds for both cycle detection and bipartiteness.

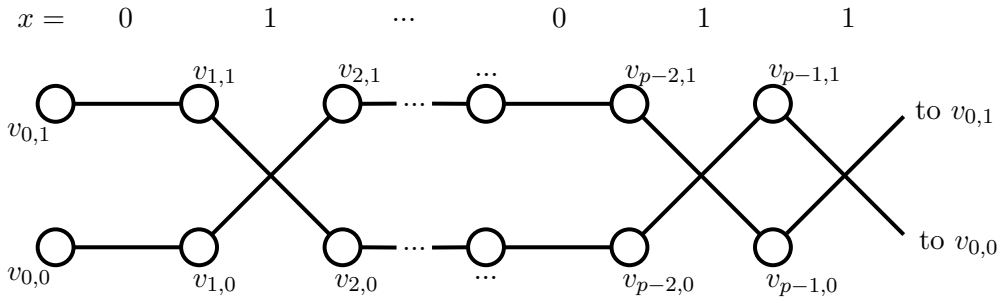

Figure 3.2: Reduction from PARITY (similar to [47])

Lemma 5. *Bipartiteness testing and cycle detection both require $\Omega(n)$ queries in the adjacency array model.*

Proof. Let $x \in \{0,1\}^p$ be an instance of the parity problem. We construct a function f on $\{v_{i,b} : i \in \{0,\cdots,p\}, b \in \{0,1\}\}$ which has exactly 1 or 2 cycles, depending on the parity of x. We define $f_{v_{i,b}}(0) = v_{i+1,b\oplus x_i}$ and $f_{v_{i+1,b\oplus x_i}}(1) = v_{i,b}$ (so that f is symmetric), where the addition is modulo $2p$ (and $\oplus$ denotes addition modulo 2). The (undirected) graph defined by f has two levels and p columns, each corresponding to a bit of x (see Figure 3.2). A walk starting at vertex $v_{0,0}$ and using each edge at most once will go from left to right, changing level whenever the corresponding bit in x is 1. So when x is even, the walk returns to $v_{0,0}$ while having only explored half of the graph, otherwise it moves to $v_{0,1}$, and then connects from there to $v_{0,0}$ by p more steps.

If we were to arbitrarily remove a single edge from the graph defined by f, we would either have no cycle present (if x is odd), or exactly one cycle present (when x is even). Therefore, if we can detect cycles in this modified graph, then we can decide the parity of x. This gives a $\Omega(n)$ quantum query lower bound for cycle detection in the adjacency array model.

In the case of bipartiteness, we ensure that x has an odd number of bits by adding a 'dummy' bit $x_p = 0$ if p is even. We fix this dummy bit to zero, so that it doesn't affect the parity of x. This has the effect of adding two additional vertices $v_{p,0}$ and $v_{p,1}$ to the graph such that $f(v_{p,0}) = v_{0,0}$ and $f(v_{p,1}) = v_{0,1}$ (and vice versa). After this modification, we have a single cycle of length $2(p+1)$ in the graph if x is odd, or two disjoint cycles of length $p+1$ if x is even. Since $p+1$ is an odd integer, there is an odd cycle in the graph if and only if x is odd. In the case where p is not even, we do not add the dummy bit, and so we also have an odd cycle in the graph if and only if x is odd. In both cases, the graph is bipartite if and only if the parity of x is odd. This gives the required bound. □

These lower bounds are actually tight, since the quantum query complexity of s-t connectivity is $\Theta(n)$ in the adjacency array model [47], implying the existence of $O(n)$ quantum query algorithms for both problems, which can be obtained by applying an $O(n)$ query algorithm for s-t connectivity to the ancillary graph used in the reduction.

3.8 Proof of Lemma 3 (reduction to s-t connectivity)

Here we provide the full technical proof of Lemma 3, which is restated below.

Lemma 3. *Let $G = (V, E)$ be a connected undirected graph, and let $H = (V', E')$ be defined as above. Then there is a path from s to t in H if and only if there is a cycle present in G, such that the difference $D := p - q$ between the number of clockwise edges p and anticlockwise edges q satisfies $D \not\equiv 0 \mod 3$. Furthermore, if k is chosen to be a vertex on the cycle, then the length of the path between s and t is at most $2c + 2$, where c is the length of the cycle.*

Proof. First we show that if there is a cycle in G such that $D \not\equiv 0 \mod 3$, then there is also a path from s to t in H. We will assume that the vertices of G are labelled arbitrarily

by the integers $1, \ldots, n$, and (without loss of generality) that $k = 1$. To find a path from s to t, it suffices to find a path from 1_0 to 1_1. We assume that the edges of the cycle are oriented in such a way that $D \not\equiv 0 \mod 3$. It is useful to recall that all edges in H are of the form $(u_b, v_{b+1 \mod 3})$, for $b \in \{0, 1, 2\}$, and such an edge only exists if the edge (u, v) is present in G. Suppose that the cycle is of length c, and is composed of the vertices $2, 3, 4, \ldots, c, 2$, where each vertex label is arbitrary. First we show that this implies that there is a path from 2_0 to 2_1 in H. In fact, we prove something stronger: that there must exist a cycle of length $3c$ in H that contains all vertices i_b for $i \in \{2, \ldots, c\}, b \in \{0, 1, 2\}$.

To see this, suppose that, starting at the vertex 2_0 in H, we follow the edges of H that correspond to the edges of the cycle in G. Depending on the orientation of the first edge, we first move to either vertex 3_1 (if the edge is directed $2 \to 3$) or 3_2 (if the edge is directed $3 \to 2$). In general, at each step, we move from a vertex u_d to a vertex $v_{d \pm 1 \mod 3}$, where the clockwise edges add 1 to the value of d, and the anticlockwise edges subtract 1. We will refer to the value of d as the 'parity' of the vertex. After taking c steps, we will have traversed p clockwise edges and q anticlockwise edges, and so we will arrive at vertex $2_{p-q \mod 3}$. If $p - q = D \not\equiv 0 \mod 3$, then we must be at either 2_1 or 2_2, depending on the value of D. By traversing the cycle again, we arrive at vertex $2_{2D \mod 3}$. Traversing the cycle one final time, we arrive at $2_{3D \mod 3} = 2_0$. Since $2D \not\equiv D \mod 3$, unless $D \equiv 0 \mod 3$, vertices $2_{D \mod 3}$ and $2_{2D \mod 3}$ are distinct. Therefore, we have a path from 2_0 to 2_b for some $b \in \{1, 2\}$, from 2_b to $2_{b'}$ for $b' \in \{1, 2\} \setminus \{b\}$, and from $2_{b'}$ to 2_0. Each path must necessarily be disjoint, since each traversal around the edges of the cycle in G adds the same sequence of $+1$s and -1s to the parity, and therefore starting with a different initial parity ensures a unique path through the vertices of H. Since we have 3 disjoint paths of length c, combining them gives us a cycle of length $3c$ that includes all vertices in H corresponding to vertices in G that make up the cycle.

A straightforward consequence of this is that there exists a path from 2_b to $2_{b'}$ in H if there exists a cycle in G containing the vertex 2, for $b \neq b' \in \{0, 1, 2\}$. Since G is connected, there must be a path from 1 to 2 in G. By following the edges of this path, we can find a corresponding path in H from 1_0 to 2_b and from 1_1 to $2_{b+1 \mod 3}$, for some $b \in \{0, 1, 2\}$. Since there exists a path from 2_b to $2_{b+1 \mod 3}$ in H, there must also exist a path from 1_0 to 1_1.

The length of this path will depend upon two things: the length of the shortest path from 1 to 2 in G, and the length of the cycle in G. The former is determined by the length l of the path from 1 to 2. In particular, following the edges of H that correspond to this path

in G will lead to paths of length l from 1_0 to 2_b, and from 1_1 to $2_{b'}$, for $b \neq b' \in \{0,1,2\}$. The length of the path from 2_b to $2_{b'}$ in H is then at most $2c$. To see this, consider traversing the edges of H corresponding to the cycle in G, starting at 2_b. As argued above, after c steps we will arrive at a vertex $2_{\tilde{b}}$, for $\tilde{b} \neq b \in \{0,1,2\}$. If $\tilde{b} = b'$, then the length of the path is c. On the other hand, if $\tilde{b} \neq b'$ then we can take c more steps, at which point we will arrive at $2_{b'}$ after a total of $2c$ steps. In order to prove the final part of the lemma, we note that when the vertex k (which we have assumed without loss of generality is the vertex 1 in this case) is contained in the cycle, then the path from s to t will be determined only by the length of the path from 2_0 to 2_1. By the arguments given here, this is at most of length $2c$. Adding in the two edges incident to vertices s and t, we obtain the upper bound of $2c+2$.

We will now show that if G does not contain a cycle, or if it contains a cycle such that $D = p - q \equiv 0 \mod 3$, then s and t are not connected in H.

Assume that G does not contain a cycle at all. Suppose that there is a path P from v_0 to v_1 in H, of the form $v_0, v'_b, v''_{b'}, ..., v_1$. Since, for every edge $(u_b, v_{b'}) \in E'$ there must be a corresponding edge $(u, v) \in E$, we can construct a path $Q = v, v', v'', ..., v$ in G from P. However, this gives a cycle in G, and hence, a contradiction.

Suppose instead that there is a cycle of length c in G such that $D \equiv 0 \mod 3$. We have shown that this implies the existence of a path of length c from v_b back to v_b for any vertex v in the cycle and for all $b \in \{0,1,2\}$. Since each such cycle must be disjoint (by the same argument as before), there cannot be a path from any v_b to $v_{b'}$ for $b \neq b'$, else the cycles would necessarily share vertices. □

Chapter 4

The quantum complexity of computing Schatten p-norms

The work in this chapter is joint work with Ashley Montanaro, and has been published previously as "The quantum complexity of computing Schatten p-norms" in the *Proceedings of Theory of Quantum Computation*, vol. 111, 2018, and an earlier version is available as the pre-print `arXiv:1706.09279`. The majority of the content of this chapter is identical to the published content, however in this chapter we prove a (much) stronger result than was shown in the published version – namely that estimating Schatten p-norms up to reasonable additive error is DQC1-complete. In contrast, the previously obtained result showed that estimating Schatten p-norms up to reasonable additive error is in DQC1, and that estimating them up to higher accuracy is DQC1-hard. Here, we reconcile this difference, and obtain a single completeness result. We also mention here some recent results (by other authors) that were obtained following publication of our work.

4.1 Introduction

It is widely believed that quantum computers will be capable of solving certain computational problems more efficiently than any classical computer. However, the exact characterisation of the class of problems that allow for a quantum speedup is the subject of ongoing research. In complexity theory, this class is known as BQP [123] – the set of languages efficiently decidable by a uniform family of polynomial-size quantum circuits with bounded error. A useful way to understand and identify the types of problems that are

efficiently solvable by a quantum computer, but unlikely to be efficiently solvable by a classical computer, is to find problems that are complete[1] for BQP; that is, problems that can be solved by a polynomial-time quantum computer, and that any other problem in BQP can be reduced to. Intuitively, these are the very hardest problems in BQP.

Several BQP-complete problems are known, including approximating the Jones polynomial [9], estimating quadratically signed weight enumerators (QSWEs)[77], and estimating diagonal entries of powers of sparse matrices [63]. The latter problem is particularly interesting, since it is a relatively natural problem that is not obviously 'quantum' in nature.

Knill and Laflamme [77] showed that a more constrained version of the QSWE problem is efficiently solvable in the one clean qubit model of computation – an apparently non-universal model of quantum computation that is weaker than full quantum computation, but that can seemingly solve some problems more efficiently than a classical computer [106]. Understanding the power of such intermediate classes of computation could shed light on the types of problems that are efficiently solvable by a fully universal quantum computer, and help us to understand which properties of quantum computers allow them to achieve speedups for certain computational problems.

Here we consider the computational complexity of estimating Schatten p-norms of matrices. We find that for certain values of p and certain families of matrices, this problem is closely related to the one clean qubit model of computation. We also consider similar quantities related to the spectra of matrices, such as the so-called "energy" of graphs [81, 55], and provide quantum algorithms for estimating them that are more efficient than any known classical algorithms.

4.1.1 The One Clean Qubit Model of Computation

The one clean qubit model of quantum computation initially arose as an idealised model for computation on very noisy initial states, such as those that appear in NMR implementations [76]. In this model, we are given a quantum state consisting of a single 'clean' qubit in the pure state $|0\rangle$, and n qubits in the maximally mixed state. This can be represented by the density matrix

$$\rho = |0\rangle\langle 0| \otimes \frac{I_{2^n}}{2^n}.$$

[1] We note that what we are really referring to here are PromiseBQP-complete problems, since there are in fact no known BQP-complete problems. For a detailed discussion on this point see [63, 52].

We then apply an arbitrary polynomial-sized quantum circuit to ρ, and measure the first qubit in the computational basis. Following [76], we will refer to the class of problems that can be solved in polynomial time using this model of computation as DQC1 – deterministic quantum computation with a single clean qubit.

The canonical problem that can be solved in this model is that of estimating the normalised trace of a $2^n \times 2^n$ unitary matrix U corresponding to a polynomial-sized quantum circuit. This is achieved by applying a controlled version of U to ρ, where the clean qubit is used as the control qubit and is put into the state $(|0\rangle + |1\rangle)/\sqrt{2}$ using a Hadamard gate. More precisely, we apply the controlled-U operator to the state

$$\rho' = \frac{1}{2}(|0\rangle + |1\rangle)(\langle 0| + \langle 1|) \otimes \frac{I_{2^n}}{2^n}$$

and then apply a Hadamard gate to the first qubit, before measuring it. The probability of measuring zero is $\frac{1}{2} + \frac{1}{2}\frac{\mathrm{Re}(\mathrm{Tr}(U))}{2^n}$, which can be estimated up to accuracy ϵ by repeating the procedure $O(1/\epsilon^2)$ times. The imaginary part of the trace of U can be estimated similarly by starting with the first qubit in the state $\frac{1}{\sqrt{2}}(|0\rangle - i\,|1\rangle)$. This problem has been shown to be complete for the class DQC1 [109].

One might wonder how things change if we allow more clean qubits. Indeed, we might consider the class DQCk: deterministic quantum computation with k pure qubits. If $k = O(\log(n))$, then DQCk = DQC1 [109]. This result is important for us since the quantum circuit that we apply may require a number of ancilla qubits initialised to $|0\rangle$ in order to correctly perform its computation. For example, if the quantum circuit implementing the unitary U performs the phase estimation routine, then it will usually require an additional $O(\log n)$ clean qubits. In the context of estimating the trace of a unitary matrix, this result tells us that it is possible in DQC1 to compute the trace of a sub-matrix whose size is an inverse-polynomially large fraction of the size of the input matrix.

DQC1-complete Problems

The list of DQC1-complete problems is relatively short, and many of the problems are of the same 'flavour'. Shortly after formally introducing the model, Knill and Laflamme [76] showed that the problem of estimating a coefficient in the Pauli decomposition of a quantum circuit, up to polynomial accuracy, is complete for the class DQC1. In fact, the aforementioned problem of estimating the normalised trace of a quantum circuit is a spe-

cial case of this problem [109]. More recently, Shor and Jordan [109] added to the set of DQC1-complete problems by showing that the problem of estimating the 'trace closure' of Jones polynomials is also complete for the class DQC1. Finally, Brandão [33] showed that two problems related to Hamiltonians were DQC1-complete: computing the partition function of a class of (quantum) Hamiltonians, and computing the sum of all eigenvalues of a Hamiltonian that fall between two given energy levels.

These quantities appear to be hard to compute classically, and therefore the one clean qubit model of computation seems to be more powerful than classical computation. However, it is unlikely that DQC1 contains all of BQP [106], and thus this model of computation appears to have a computational power that is somewhere in between BPP and BQP. Some evidence in this direction was recently provided by Morimae [91], who built on earlier work ([90]) to show that the output distribution of the one clean qubit model is difficult to sample from classically up to constant total variation distance error, provided that some complexity theoretic conjectures hold.

Here we show that the problem of computing Schatten p-norms of matrices is also DQC1-complete.

4.1.2 Schatten p-norms and Graph Energy

Schatten p-norms are ubiquitous in Quantum Information theory (see for example [96, 27, 59]). This family of matrix norms includes the three most commonly used norms in quantum information theory: the Schatten 1-norm is more commonly called the trace norm, the Schatten 2-norm is also known as the Frobenius norm, and the Schatten ∞-norm is called the operator norm or spectral norm. Here we consider a normalised version of the Schatten p-norm, defined to be

$$\|A\|_p := \left(\frac{\sum_j |\lambda_j|^p}{2^n}\right)^{1/p}$$

for a $2^n \times 2^n$ Hermitian matrix A, where the sum ranges over the eigenvalues of A.

For instance, the Schatten 1-norm is the average of the absolute values of the eigenvalues of A,

$$\|A\|_1 = \frac{\mathrm{Tr}(|A|)}{2^n} = \frac{\sum_j |\lambda_j|}{2^n}.$$

If we consider the matrix A to be the adjacency matrix of a graph, this quantity is known

as the 'Graph Energy', and has applications in chemistry, where it is related to the total electron energy of a class of organic molecules [81, 55]. It is quite typical for quantities relating to the spectra of adjacency matrices to be used throughout Graph Theory to reveal information about the graphs that they represent. In the present work, we consider some 'global' properties of the spectra of matrices and graphs – i.e. those of the form $\mathrm{Tr}(f(A))/2^n$, for some suitably chosen function f.

4.1.3 Main results

We study the complexity of approximately computing the Schatten p-norms of sparse matrices and relate this to quantum computation. For clarity, we focus on estimating the quantity $\|A\|_p^p$, and discuss later on how this can provide a good estimate for the quantity $\|A\|_p$. We consider Hermitian matrices of size $2^n \times 2^n$, where at most $d = \mathrm{poly}(n)$ entries in each row are non-zero, and call such matrices d-sparse. One fairly natural class of sparse matrices that can be expressed concretely is the class of 'log-local' Hamiltonians. That is, k-local n-qubit Hamiltonians, with $k = O(\log n)$ - i.e. Hermitian matrices that can be written as a sum $A = \sum_{j=1}^m A_j$, for some m, where each A_j is a Hermitian matrix that acts non-trivially on at most $k = O(\log n)$ qubits. We assume that we are given the individual matrices A_j directly, that $\|A_j\| = O(\mathrm{poly}(n))$ for all j, and that $m = \mathrm{poly}(n)$. Throughout this chapter, we will use $\|A\|$ to denote the operator norm of A. Our main two results are:

Theorem 9. *Let A be a sparse Hermitian matrix on n qubits, and let $p \in \mathbb{R} = O(\mathrm{poly}(n))$, $1/\epsilon = O(\mathrm{poly}(n))$. Then the problem of estimating $\frac{\mathrm{Tr}(|A|^p)}{2^n}$ up to additive accuracy $\epsilon\|A\|^p$ is contained in* BQP*. If the matrix A is* log*-local, then this problem is also contained in* DQC1*.*

Theorem 10. *Let A be a* log*-local Hermitian matrix on n qubits. Then the problem of estimating $\frac{\mathrm{Tr}(|A|^p)}{2^n}$ up to additive accuracy $\frac{1}{2^p}\epsilon\|A\|^p$ for arbitrary $p = O(\mathrm{poly}(n)), 1/\epsilon = O(\mathrm{poly}(n))$ is hard for the class* DQC1*.*

Note that the accuracy required in Theorem 10 is exponentially smaller than that in Theorem 9. Combining these two theorems, we obtain the following completeness result:

Corollary 1. *Let A be a sparse Hermitian matrix on n qubits, and let $p, 1/\epsilon = O(\mathrm{poly}(n))$. Then the problem of estimating $\frac{\mathrm{Tr}(|A|^p)}{2^n}$ up to additive accuracy $\epsilon\|A\|^p$ is* DQC1*-complete.*

The BQP case of Theorem 9 follows from a result of Janzing and Wocjan [62], who gave a BQP algorithm for estimating diagonal entries of $f(A)$, for a sparse matrix A and an appropriate function f which for our purposes can be taken to be $f(x) = |x|^p$.

We therefore see that the problem of computing Schatten p-norms for $p = O(\text{poly}(n))$ is closely related to the one clean qubit model of computation. For different values of p the problem is related to other classes of computation. For instance, $\|A\|_\infty$ is the operator norm of A, and the problem of computing it approximately is QMA-complete[2], even for 2-local Hamiltonians. To see this, suppose we have some upper bound $\Delta = O(\text{poly}(n))$ on the largest eigenvalue of a 2-local n-qubit Hamiltonian A. Define the matrix $B := \Delta I_{2^n} - A$. Then the largest eigenvalue of B (in absolute value) corresponds to the smallest eigenvalue of A. Hence, if we can compute the smallest eigenvalue of A, then we can compute $\|B\|$, and vice versa. Since the problem of estimating the smallest eigenvalue of a k-local Hamiltonian is QMA-complete for $k \geq 2$ [72], this implies QMA-completeness of the problem of estimating the operator norm of a 2-local Hamiltonian.

Theorem 9 gives us the following corollary, by setting $p = 1$:

Corollary 2. *Let A be a* log*-local matrix corresponding to the adjacency matrix of a 2^n-vertex graph G, and let $1/\epsilon = O(\text{poly}(n))$. The normalised Graph Energy of G, $\text{Tr}(|A|)/2^n$, can be estimated up to additive accuracy $\epsilon\|A\|$ in* DQC1*.*

In proving Theorem 9, we also show that there exists a polynomial-time quantum algorithm (in DQC1) for estimating $\text{Tr}(A^p)/2^n$ up to error $\epsilon\|A\|^p$ for $1/\epsilon, p \in O(\text{poly}(n))$. This is useful in the context of graph theory because it allows for an estimation of the expected number of closed walks that start from each vertex in a 2^n-vertex graph. To obtain these algorithms, we prove a more general result:

Lemma 6. *For a log-local Hamiltonian A, and any* log*-space polynomial-time computable function $f : I \to [-1, 1]$ (where I contains the spectrum of A) that is Lipschitz continuous with constant K (i.e. $|f(x) - f(y)| \leq K|x - y|$ for all $x, y \in I$), there exists a* DQC1 *algorithm to estimate $\text{Tr}(f(A))/2^n = \sum_j f(\lambda_j)/2^n$ up to additive accuracy $\epsilon(K + 1)$, where λ_j denote the eigenvalues of A, and $\epsilon = \Omega(1/\text{poly}(n))$.*

Often, one is interested in calculating the properties of general sparse matrices. We note that it is easy to give a quantum algorithm for estimating the above quantities for sparse

[2]For a definition of the class QMA, see for example [123].

matrices by making use of a result of Janzing and Wocjan [63, 62], who give a BQP algorithm for estimating the diagonal entries of $f(A)$, for some function f that satisfies certain continuity constraints, but this comes at the expense of moving to the class BQP.

It is interesting to note that the results of Brandão [33] also make use of log-local Hamiltonians. In both these and our results, it is not clear how to drop the restriction of log-locality without losing the fact that the various problems are contained in DQC1.

Estimating $\|A\|_p$

Given a log-local n-qubit Hamiltonian A, the algorithm of section 4.3 outputs

$$\mathrm{Tr}(|A|^p)/2^n \pm \epsilon \|A\|^p.$$

By taking the pth root, we obtain an estimate of $\|A\|_p$ of the form

$$\left(\frac{\mathrm{Tr}(|A|^p)}{2^n} \pm \epsilon\|A\|^p\right)^{1/p} = \|A\|_p \left(1 + \frac{2^n \epsilon \|A\|^p}{\mathrm{Tr}(|A|^p)}\right)^{1/p}.$$

The error will be small when $\mathrm{Tr}(|A|^p)$ takes a value close to its maximum of $2^n\|A\|^p$. In the best case, the relative error is close to $(1+\epsilon)^{1/p}$. This suggests that in these 'good' cases, our algorithm can estimate $\|A\|_p$ up to a reasonable additive error in polynomial time. On the other hand, we can always bound

$$\frac{2^n\|A\|^p}{\mathrm{Tr}(|A|^p)} \leq \frac{2^n\|A\|^p}{2^n|\lambda_{\min}|^p} = \kappa(A)^p,$$

where $\lambda_{\min}$ is the minimal eigenvalue of A in absolute value, and $\kappa(A) = \|A\|\|A^{-1}\|$ is the condition number of A. In this case the relative error is at most $(1+\epsilon\kappa(A)^p)^{1/p}$.

Since we consider $p = \mathrm{poly}(n)$, the algorithm allows us to achieve relative error close to $\kappa(A)$ by taking $\epsilon = 1 - 1/\kappa(A)^p \approx 1$. Alternatively, we could achieve relative error $(1+\delta)$ for some $\delta = O(1/\mathrm{poly}(n))$ by setting $\epsilon = ((1+\delta)^p - 1)/\kappa(A)^p$. In this case, we sacrifice the run-time of the algorithm in order to improve the accuracy.

We cannot reasonably expect to achieve a better accuracy than this – if we could, then we could also obtain a good estimate of the operator norm $\|A\|$, which as discussed is

QMA-hard to compute. To see this, we first note that the following inequality holds:

$$\frac{1}{2^{n/p}}\|A\| \leq \|A\|_p \leq \|A\|.$$

The upper bound on $\|A\|_p$ is clear by definition. For the lower bound, we observe that

$$\|A\|_p^p = \frac{1}{2^n}\sum_j |\lambda_j|^p \geq \frac{1}{2^n}\|A\|^p,$$

and so

$$\frac{1}{2^n}\|A\|^p \leq \|A\|_p^p \leq \|A\|^p,$$

from which the inequality stated above follows.

Write $d = 2^n$, and let $p = \text{poly}(n) = \text{poly}(\log d) \approx (\log d)^k$ for some constant k. Then

$$1/2^{n/p} = \frac{1}{d^{1/p}} = \frac{1}{2^{\log d/p}} = \frac{1}{2^{\log d/(\log d)^k}} = \frac{1}{2^{1/n^{k-1}}}$$

and so

$$\frac{1}{2^{1/n^{k-1}}}\|A\| \leq \|A\|_p \leq \|A\|.$$

$\frac{1}{2^{1/n^{k-1}}}$ approaches 1 exponentially quickly (in n) for any constant $k \geq 1$, implying that choosing $p = \text{poly}(n)$ (as we did above) will mean that $\|A\|_p$ rapidly approaches $\|A\|$.

Suppose that we obtained an estimate of $\|A\|_p$ up to additive error ϵ. In the worst case (i.e. when the lower bound becomes an equality), we have that $\|A\|_p = \frac{1}{2^{1/n^{k-1}}}\|A\|$. To recover an estimate of $\|A\|$, we must multiply our original estimate of $\|A\|_p$ by $2^{1/n^{k-1}}$. This will increase the additive error, such that our estimate of $\|A\|$ becomes

$$\|A\| \pm \epsilon \cdot 2^{1/n^{k-1}}$$

which rapidly (i.e. exponentially quickly) approaches $\|A\| \pm \epsilon$.

This implies that, given the ability to efficiently estimate the quantity $\|A\|_p \pm \epsilon$ for $p = O(\text{poly}(n))$, we can obtain a good approximation of $\|A\|$ efficiently. This essentially rules out the possibility of obtaining a good estimate of $\|A\|_p$ in polynomial in DQC1 (or indeed in BQP), unless DQC1 = QMA.

4.1.4 Relation to Previous Work

Our techniques are similar to those used in [63] and [57]. In particular, we use the same combination of Hamiltonian simulation and phase estimation for estimating and manipulating the eigenvalues of a Hermitian matrix. To show DQC1-hardness, we use techniques from the Hamiltonian complexity literature, and in particular ideas due to Kitaev et al. [74, 72].

By using a previous result of Janzing and Wocjan [63], we can obtain a BQP algorithm for estimating $\mathrm{Tr}(A^p)/2^n$ for general sparse matrices; however, it is not clear how to implement this algorithm in DQC1, since it uses $O(n)$ ancilla qubits for the Hamiltonian simulation step. In [63], the authors describe a polynomial-time quantum algorithm for estimating the diagonal entries of the matrix A^p up to error $\epsilon\|A\|^p$, for $\epsilon = O(1/\operatorname{poly}(n))$, and show that this problem is in fact BQP-complete for sparse symmetric matrices. The problem remains BQP-complete even for matrices with only $0, \pm 1$ entries.

4.1.5 Comparison with Classical Algorithms

We were not able to find any previous results in the literature regarding the complexity of estimating the above quantities for sparse matrices. One would expect that the quantity $\mathrm{Tr}(|A|^p)/2^n$ for odd p is difficult to compute classically, since it would seemingly require the algorithm to first diagonalise the matrix A to obtain its eigenvalues. In contrast, we know that quantum phase estimation can efficiently obtain eigenvalues without explicitly diagonalising the matrix A. To look for efficient classical algorithms, it makes sense to turn instead to the quantity $\mathrm{Tr}(A^p)/2^n$, and in Section 4.4, we give a classical algorithm for estimating this quantity for sparse matrices, and prove some bounds on the accuracy that this algorithm can achieve.

We find that for some types of matrix, the value $\mathrm{Tr}(A^p)/2^n$ can be estimated efficiently classically, and for others, a quantum algorithm appears to have an advantage. In Section 4.4 we prove the following:

Theorem 11. *Given a $2^n \times 2^n$, d-sparse matrix A, there exists a classical algorithm to estimate $\mathrm{Tr}(A^p)/2^n$ up to accuracy $\epsilon d^p \|A\|_{\max}^p$ in time that is polynomial in n, p and $1/\epsilon$, where $\epsilon = O(1/\operatorname{poly}(n))$ and $\|A\|_{\max}$ is used to denote the maximum absolute size of an entry in A.*

Therefore in the cases where $\|A\| \ll d\|A\|_{\max}$, we can get an advantage by making use of the algorithm of Theorem 9. We find that for certain classes of random graph (namely

power-law graphs), the BQP algorithm for computing $\mathrm{Tr}(A^p)/2^n$ obtains a quadratic improvement in accuracy over the corresponding classical algorithm.

For log-local Hamiltonians and *constant* p, there exists an efficient *exact* classical algorithm for computing $\mathrm{Tr}(A^p)$. By using conventional matrix multiplication, it is possible to calculate the value of A^p by multiplying the individual matrices A_j. This can be seen from the expression for $\mathrm{Tr}(A^p)$:

$$\mathrm{Tr}(A^p) = \sum_{j_1, j_2, \ldots, j_p} \mathrm{Tr}(A_{j_1} A_{j_2} \cdots A_{j_p}),$$

where each index j_i ranges from 1 to m. Every A_{j_i} is k-local, and the complexity of multiplying a k-local matrix by an l-local matrix is $O(2^{3(k+l)})$ (using a naive algorithm), and results in a $(k+l)$-local matrix. If we perform the matrix multiplications from left to right, then, for each term in the sum, the first multiplication will take time $O(2^{3(2k)})$, the second $O(2^{3(3k)})$, and so on, until the final multiplication takes time $O(2^{3(pk)})$. There will be $p-1$ of these multiplications performed in total, with each taking at most $O(2^{3 \cdot pk})$ time, and hence the trace of $A_{j_1} A_{j_2} \cdots A_{j_p}$ can be calculated in $O(2^{3 \cdot pk})$ steps. There are m^p terms in the sum, and therefore the complexity of the entire computation is $O(m^p 2^{3 \cdot pk})$.

If we take $k = O(\log n)$ (i.e. take A to be a log-local Hamiltonian), the time complexity is $m^p n^{O(p)}$. For $p = O(1)$, this time complexity is polynomial and the output of this algorithm is better than the corresponding quantum algorithm, as it computes the desired value exactly.

Finally, we recall from the discussion above that the problem of computing $\mathrm{Tr}(|A|^p)$ appears to be substantially harder classically for odd p, since it cannot be found by simply computing powers of a matrix, and instead requires more knowledge about the eigenvalues of A.

4.1.6 Subsequent related work

Following the completion and publication of the work in this chapter, there have been some recent results that are somewhat related. Subramanian and Hsieh [115] describe a method for estimating Renyi entropies of unknown quantum states by combining the recent technique of quantum singular value transformations with the method of estimating normalised traces in the one clean qubit model. The α-Renyi entropy of a positive semidefinite

operator ρ is defined as

$$S_\alpha(\rho) := \frac{1}{1-\alpha} \log \left[\mathrm{Tr}(\rho^\alpha) \right].$$

The term inside the logarithm is reminiscent of the quantities that we have considered in this chapter, and these results show that we can replace the matrix A in our algorithms with the density matrix of some unknown quantum state by using recent techniques of quantum singular value transformations and block encodings [37, 50].

A more loosely related result is that of Yoganathan and the current author [126]. This result demonstrates that when there is no entanglement present in a one clean qubit computation of a certain kind, then the computation can be classically simulated, in the sense that the output can be sampled from efficiently by a classical algorithm. The role of entanglement in mixed state quantum computation has been debated for some time, and this result shows that, at least for the one clean qubit model, it is indeed necessary for quantum speedups to be possible.

The work in this chapter provides an example of a natural problem that appears to be hard for classical computers to solve, but can be solved efficiently using a quantum computer with a single clean qubit, suggesting that this model of quantum computation is indeed more powerful than classical computation. In a complementary way, the classical simulability result suggests that entanglement is at least one of the resources required for a mixed state quantum computer to have more computational power than its classical counterpart. Hence, we have evidence that even restricted mixed-state quantum computers are more powerful than classical computers, and that entanglement must play a key role in this power.

4.1.7 Organisation

We begin by providing a proof of Theorem 10 in Section 4.2. Section 4.3 describes a proof of Theorem 9, via an algorithm in the one clean qubit model that can estimate $\mathrm{Tr}(f(A))/2^n$ for a $2^n \times 2^n$ log-local matrix A and an appropriately continuous function f. Following this, in Section 4.4 we describe some classical algorithms for estimating the quantity $\mathrm{Tr}(A^p)/2^n$, and compare their performance to the corresponding quantum algorithms in Section 4.5.

4.2 Estimating $\mathrm{Tr}(|A|^p)/2^n$ is DQC1-hard

In this section, we show that the problem of estimating $\mathrm{Tr}(|A|^p)/2^n$ for a $2^n \times 2^n$ log-local Hamiltonian A up to a given accuracy is hard for the class DQC1. More precisely, we assume that we have access to an algorithm that can estimate $\mathrm{Tr}(|A|^p)/2^n$ up to accuracy $\epsilon\|A\|^p$, for $\epsilon = O(1/\operatorname{poly}(n))$ and $p = \operatorname{poly}(n)$, and show that this allows us to solve any problem contained in DQC1.

To do this we show that, given as input a real unitary U (implemented by some polynomial-sized quantum circuit acting on n qubits), it is possible to construct a log-local Hamiltonian A such that $\mathrm{Tr}(|A|^p)/2^n = \mathrm{Tr}(U)/2^n$, for some $p = \operatorname{poly}(n)$. We show that an estimation accuracy of $\epsilon\|A\|^p$ is sufficient to provide an estimate of $\mathrm{Tr}(U)/2^n$ up to accuracy $1/\operatorname{poly}(n)$. This problem is complete for the class DQC1 [109], which implies that the problem of estimating $\mathrm{Tr}(|A|^p)/2^n$ up to the stated accuracy is DQC1-hard.

The construction is based on ideas from the Hamiltonian complexity literature, and in particular Kitaev's clock construction for the local Hamiltonian problem [10]. We assume that we have a decomposition $U = U_{M-1}...U_1U_0$ of the circuit into M elementary gates. Since U is described by a polynomial-sized circuit, we have $M = \operatorname{poly}(n)$. We add $\lceil \log M \rceil$ additional qubits to act as a 'clock' register, which is used to control the application of the individual unitaries, and define the unitary operator

$$W := \sum_{l=0}^{M-1} |l+1\rangle\langle l| \otimes U_l,$$

where addition is taken to be modulo M. It is straightforward to check that

$$W^M = \sum_{l=0}^{M-1} |l\rangle\langle l| \otimes U_{l+M}...U_{l+2}U_{l+1}.$$

Then we have

$$\mathrm{Tr}(W^M) = \sum_{l=0}^{M-1} \mathrm{Tr}(|l\rangle\langle l|) \cdot \mathrm{Tr}(U_{l+M}...U_{l+2}U_{l+1}) = \sum_{l=0}^{M-1} \mathrm{Tr}(U_M...U_2U_1) = M\,\mathrm{Tr}(U),$$

where the second step follows from invariance of the trace under cyclic permutations.

W is log-local with $m = \operatorname{poly}(n)$ terms, since each clock operator $|l+1\rangle\langle l|$ acts on

$\lceil \log M \rceil$ qubits, and each of the unitaries U_l act on at most $O(1)$ qubits each. Define the Hermitian matrix

$$A := \frac{1}{2}(W + W^\dagger).$$

Then the trace of A^M gives the real part of the trace of $\frac{2W^M}{2^M}$, since A^M equals $1/2^M(W^M + W^{\dagger M})$ plus some other powers of W and $W^\dagger$ that are traceless (since the clock unitaries can only have a trace if they return the clock state back to its initial state, which takes at least M applications of W), and therefore do not contribute to the trace of A^M.

W is a $2^{n+\lceil \log M \rceil} \times 2^{n+\lceil \log M \rceil}$ unitary matrix, and so we have $\|A\| \leq 1$. Thus, given the ability to estimate the normalised trace of A^p up to accuracy $\left(\frac{\|A\|}{2}\right)^p \epsilon$, we can estimate the value of $\mathrm{Re}[\mathrm{Tr}(U)]/2^n$ up to accuracy $1/\operatorname{poly}(n)$, which is the level of accuracy required for the class DQC1. To see this, we observe that, taking $p = M$ and assuming (without loss of generality) that M is a power of 2 (which also means that $|A|^M = A^M$),

$$\frac{\mathrm{Tr}(A^M)}{2^{n+\log M}} \pm \frac{\epsilon}{2^M} = \frac{2\,\mathrm{Re}(\mathrm{Tr}(W^M))}{2^M 2^{n+\log M}} \pm \frac{\epsilon}{2^M} = \frac{2M\,\mathrm{Re}(\mathrm{Tr}(U))}{M 2^M 2^n} \pm \frac{\epsilon}{2^M}.$$

Multiplying by 2^M, we obtain $\frac{\mathrm{Re}(\mathrm{Tr}(U))}{2^n} \pm \epsilon$. We can also obtain $\frac{\mathrm{Im}(\mathrm{Tr}(U))}{2^n} \pm \epsilon$ by choosing $A = \frac{1}{2i}(W - W^\dagger)$ and following the same argument as above. Combining these, we obtain $\frac{\mathrm{Tr}(U)}{2^n} \pm \epsilon$, which is precisely the quantity that is DQC1-hard to compute. This is sufficient to show that the problem of estimating $\mathrm{Tr}(A^p)/2^n$ up to accuracy $\left(\frac{\|A\|}{2}\right)^p \epsilon$ for a log-local n-qubit Hamiltonian is hard for the class DQC1.

Note that we were not able to use standard techniques from the Hamiltonian complexity literature to make this construction work for k-local Hamiltonians with constant k [72, 74]. These techniques involve the introduction of a larger clock space that is then acted upon by k-local Hamiltonians. A term is then added to the Hamiltonian to 'penalise' invalid clock states and prevent them from contributing to the ground state energy. In our case, we care about the entire space on which the Hamiltonian acts and not just the subspace containing the valid clock states, and therefore the invalid clock states contribute to the trace of A^M in a non-trivial way.

4.3 Estimating $\mathrm{Tr}(|A|^p)/2^n$ is in DQC1

Here we show that the problem of estimating $\mathrm{Tr}(|A|^p)/2^n$ for a log-local Hamiltonian A, up to reasonable error, is in DQC1. In precise terms, we are given a k-local n-qubit Hamiltonian A, with $k = O(\log n)$ and the problem is to estimate $\mathrm{Tr}(|A|^p)/2^n$ up to error $\epsilon\|A\|^p$, for some integer $p = O(\mathrm{poly}(n))$ and accuracy $\epsilon = \Omega(1/\mathrm{poly}(n))$. Our approach is to construct a unitary U such that the normalised trace of U approximates the normalised trace of $|A|^p$. We show that this construction can be performed in polynomial time (that is, the unitary U takes $\mathrm{poly}(n, p, 1/\epsilon)$ time to implement). Using this approach, we can use the DQC1 model to compute the normalised trace of the matrix $|A|^p$, hence showing that this problem is contained in DQC1. We will use the following corollary of Lemma 6:

Corollary 3. *For a log-local Hamiltonian A, and any* log*-space polynomial-time computable function $f : I \to \mathbb{R}$ (where I contains the spectrum of A) that is Lipschitz continuous with constant K' (i.e. $|f(x) - f(y)| \leq K'|x - y|$ for all $x, y \in I$), there exists a* DQC1 *algorithm to estimate $\mathrm{Tr}(f(A))/2^n = \sum_j f(\lambda_j)/2^n$ up to additive accuracy $\epsilon(K' + f_{\max})$, where λ_j denote the eigenvalues of A, $\epsilon = \Omega(1/\mathrm{poly}(n))$, and $f_{\max}$ is the supremum of $|f|$ on the interval I.*

The proof of Lemma 6 (and hence the above corollary) is split into roughly three parts. The first part, in Section 4.3.1, describes how the algorithm works, via a description of the unitary that is constructed from the input matrix. Following this, Section 4.3.2 discusses the accuracy and failure probability of the algorithm, and finally, Section 4.3.3 shows that the number of ancilla qubits required (and therefore the number of pure qubits needed) to implement the algorithm is at most $O(\log n)$.

4.3.1 Constructing the Unitary

We are given a log-local Hamiltonian A with eigenvectors $|\psi_j\rangle$ and corresponding eigenvalues λ_j. The basic idea is to construct a unitary U whose eigenvalues correspond to the eigenvalues of A in a useful way. In particular, we construct a polynomial-sized circuit whose associated unitary has eigenvalues λ'_j such that $\lambda'_j = f'(\lambda_j)$, for some function f' that depends on f.

The first step is to use Hamiltonian simulation to implement the unitary e^{iA}, which has eigenvalues $e^{i\lambda_j}$ for each eigenvector $|\psi_j\rangle$ of A. Section 4.3.4 discusses the time complexity of this part of the circuit. Then the circuit performs the following sequence of operations,

which we will describe in terms of their effects on an eigenvector $|\psi_j\rangle$ of A and an arbitrary single qubit state of the form $\alpha|0\rangle + \beta|1\rangle$. We use $|0\ldots0\rangle$ to denote an arbitrarily large ancilla register (with each qubit initialised to 0), and for now we will assume that both the phase estimation and Hamiltonian simulation parts of the circuit work perfectly.

1. Apply phase estimation on e^{iA} with the input $|\psi_j\rangle$, to obtain an estimate of the eigenvalue λ_j:

$$|\psi_j\rangle(\alpha|0\rangle + \beta|1\rangle)|0\ldots0\rangle \mapsto |\psi_j\rangle(\alpha|0\rangle + \beta|1\rangle)|\lambda_j\rangle$$

2. Perform controlled phase rotations, where the phase depends on a function f of λ_j contained in the 3rd register (for example, $f(x) = x^p$):

$$|\psi_j\rangle(\alpha|0\rangle + \beta|1\rangle)|\lambda_j\rangle \mapsto |\psi_j\rangle(\alpha e^{i\arccos(f(\lambda_j))}|0\rangle + \beta e^{-i\arccos(f(\lambda_j))}|1\rangle)|\lambda_j\rangle$$

3. Undo the phase estimation to uncompute the value in the 3rd register:

$$\mapsto |\psi_j\rangle(\alpha e^{i\arccos(f(\lambda_j))}|0\rangle + \beta e^{-i\arccos(f(\lambda_j))}|1\rangle)|0\ldots0\rangle$$

This gives us a unitary U that performs the mapping

$$|\psi_j\rangle(\alpha|0\rangle + \beta|1\rangle)|0\ldots0\rangle \mapsto (\alpha e^{+i\arccos(f(\lambda_j))}|\psi_j\rangle|0\rangle + \beta e^{-i\arccos(f(\lambda_j))}|\psi_j\rangle|1\rangle)|0\ldots0\rangle$$

for each eigenvector $|\psi_j\rangle$ of A. Therefore, for each eigenvalue λ_j of A, U has two corresponding eigenvalues $e^{\pm i\arccos(f(\lambda_j))}$.

By using the results described in Section 4.1.1, we can compute the trace of a sub-matrix of U in the one clean qubit model, provided that the number of ancilla qubits used is $O(\log n)$ (we check that this is indeed the case at the end of this section). In particular, we compute the trace of U', the sub-matrix of U obtained by fixing the ancilla qubits

(except the one explicitly mentioned above) to $|0\rangle$. Then the trace of U' is

$$\begin{aligned}\mathrm{Tr}(U') &= \sum_j e^{\pm i \arccos(f(\lambda_j))} = \sum_j \cos(\pm \arccos(f(\lambda_j))) + i \sin(\pm \arccos(f(\lambda_j))) \\ &= \sum_j 2\cos(\arccos(f(\lambda_j))) + i\sin(\arccos(f(\lambda_j))) - i\sin(\arccos(f(\lambda_j))) \\ &= \sum_j 2f(\lambda_j).\end{aligned}$$

4.3.2 Error Analysis

Errors can arise in three places. Firstly, we will have some error in the Hamiltonian simulation part of the circuit. Secondly, there will be errors in estimating eigenvalues by using the phase estimation routine. And finally, there will be some error in the estimation of the normalised trace of U from using the one clean qubit model. Here we consider the effect of all three sources of error, and show that we can estimate $\frac{1}{2^n}\sum_j f(\lambda_j)$ with additive error at most $\epsilon(K+1)$, for any $\epsilon = \Omega(1/\operatorname{poly}(n))$, where K is the Lipschitz constant of f. The analysis in this section is analogous to that of [63], in that we use the same method for estimating an eigenvalue of A via simulation of e^{iA}, but uses different methods to bound the errors introduced by phase estimation and Hamiltonian simulation.

Error from Hamiltonian Simulation

We begin by considering the error that arises in the circuit from Hamiltonian simulation. We assume that the Hamiltonian simulation step implements a unitary V that approximates e^{iA} in the sense that $||V - e^{iA}|| \leq \delta$, so that the eigenvalues of V and e^{iA} can differ by at most δ. For now, we will assume that the phase estimation routine works perfectly (i.e. introduces no error). This part of the circuit outputs an estimate for an eigenvalue of A in the range $[-\pi, \pi)$. Denote by λ_j and μ_j the output of the phase estimation routine when it is run using e^{iA} and V, respectively. We have

$$\left| e^{i\lambda_j} - e^{i\mu_j} \right| \leq \delta$$

by the bound on the error of the Hamiltonian simulation, where we can assume $|\mu_j - \lambda_j| \leq \pi$, by adding multiples of 2π to λ_j if necessary. The left hand side can be written as

$$\begin{aligned}
\left|1 - e^{i(\mu_j - \lambda_j)}\right| &= \left|e^{i\frac{(\mu_j-\lambda_j)}{2}}\left(e^{-i\frac{(\mu_j-\lambda_j)}{2}} - e^{i\frac{(\mu_j-\lambda_j)}{2}}\right)\right| \\
&= \left|e^{-i\frac{(\mu_j-\lambda_j)}{2}} - e^{i\frac{(\mu_j-\lambda_j)}{2}}\right| \\
&= 2\left|\sin\left(\frac{\mu_j - \lambda_j}{2}\right)\right| = 2\sin\left|\frac{\mu_j - \lambda_j}{2}\right| \qquad \text{(since } |\mu_j - \lambda_j| \leq 2\pi\text{)}.
\end{aligned}$$

We will use the inequality $(2/\pi)\theta \leq \sin\theta$ for $0 \leq \theta \leq \pi/2$. Therefore, we have that

$$(4/\pi)\frac{|\mu_j - \lambda_j|}{2} \leq 2\sin\left|\frac{\mu_j - \lambda_j}{2}\right| \leq \delta$$

and hence

$$|\mu_j - \lambda_j| \leq \pi\delta/2.$$

To see how this affects the accuracy of the algorithm, we consider the difference in the trace of U' when using V in place of e^{iA}.

$$\begin{aligned}
2\left|\sum_j f(\lambda_j) - \sum_j f(\mu_j)\right| &\leq 2\sum_j |f(\lambda_j) - f(\mu_j)| \\
&\leq 2\sum_j K\,|\lambda_j - \mu_j| \qquad \text{by the Lipschitz condition} \\
&\leq 2\sum_j K\pi\delta/2 = 2^n K\pi\delta.
\end{aligned}$$

Choosing the simulation accuracy to be $\delta \leq \epsilon/(2\pi)$, this contributes an error term of $2^n\epsilon K/2$. Thus, we have

$$2\left|\sum_j f(\lambda_j) - \sum_j f(\mu_j)\right| \leq 2^n\epsilon K/2. \tag{4.1}$$

Error from Phase Estimation

Here we consider the error that arises from using the phase estimation routine to estimate the eigenvalues μ_j of the unitary V from the previous sub-section. The phase estimation

routine requires the addition of a ancilla qubits, which are used to control the application of powers of V on an n-qubit register. The lth ancilla qubit is used to control the application of the unitary V^{2^l}, so that we apply the controlled gate

$$W_l := |0\rangle\langle 0|_l \otimes I + |1\rangle\langle 1|_l \otimes V^{2^l}$$

where the subscript l denotes that the projector acts on the lth ancilla/control qubit (and as the identity everywhere else). Let $W := W_1 W_2 \cdots W_a$. Then the phase estimation routine consists of applying Hadamard gates to all of the control qubits, applying W, and then applying the inverse quantum Fourier transform to the control qubits.

If we apply phase estimation to an eigenvector of V with eigenvalue $e^{i2\pi\theta}$, and measure the control register, we obtain some output $x \in \{0, 1, ..., 2^a - 1\}$ such that

$$\Pr(|\theta - x/2^a| < \eta) > 1 - \varphi \tag{4.2}$$

for $\varphi, \eta > 0$. To obtain this level of accuracy and probability of failure, it is sufficient [92] to set

$$a = \lceil \log(1/\eta) \rceil + \lceil \log(2 + (1/(2\varphi))) \rceil. \tag{4.3}$$

Let ϕ be defined as follows:

$$\phi(x) := \begin{cases} x2\pi/2^a & \text{if } x \leq 2^{a-1} \\ x2\pi/2^a - 2\pi & \text{otherwise} \end{cases}$$

Then let $\phi(x_j)$ be our estimate of the eigenvalue μ_j corresponding to the eigenvector $|\psi_j\rangle$ of V, which, by the definition of ϕ above, lies in the interval $[-\pi, \pi)$. By Equation (4.2), if we apply phase estimation to an eigenvector $|\psi_j\rangle$ of V with corresponding eigenvalue $e^{i\mu_j}$, and measure, we have

$$\Pr(|\mu_j - \phi(x_j)| < 2\pi\eta) > 1 - \varphi \tag{4.4}$$

where the extra factor of 2π results from rescaling the value of x_j by 2π.

In our case, we do not measure the control register, and therefore we do not collapse the superposition over eigenvalues that phase estimation produces. Here we consider the effect that this has on the output of the algorithm, and simultaneously bound the error introduced by this part of the circuit. When phase estimation does not work perfectly, the

algorithm consists of the following steps, implementing a unitary $\tilde{U}$:

1. Apply phase estimation on $V \approx e^{iA}$ with the input $|\psi_j\rangle$, to obtain a superposition over k corresponding to estimates $\phi(k)$ of the eigenvalue $\mu_j = 2\pi\theta_j$:

$$|\psi_j\rangle (\alpha |0\rangle + \beta |1\rangle) |0 \dots 0\rangle \mapsto |\psi_j\rangle (\alpha |0\rangle + \beta |1\rangle) \sum_k \gamma_{k|j} |k\rangle$$

   where $\gamma_{k|j} = \frac{1}{N} \sum_{a \in \{0,1\}^n} e^{2\pi i a(\theta_j - k/N)}$, $N = 2^a$.

2. Perform controlled phase rotations:

$$|\psi_j\rangle (\alpha |0\rangle + \beta |1\rangle) \sum_k \gamma_{k|j} |k\rangle \mapsto |\psi_j\rangle \sum_k \gamma_{k|j} (\alpha e^{i \arccos(f(\phi(k)))} |0\rangle + \beta e^{-i \arccos(f(\phi(k)))} |1\rangle) |k\rangle$$

3. Undo the phase estimation to uncompute the value in the 3rd register. To undo phase estimation we: a) apply the QFT to the register containing the k's, b) apply controlled powers of the unitary $V^\dagger \approx e^{-iA}$, and c) apply Hadamard gates to all qubits in the third register.

   (a) Apply the QFT:

$$|\psi_j\rangle \frac{1}{\sqrt{N}} \sum_k \gamma_{k|j} (\alpha e^{i \arccos(f(\phi(k)))} |0\rangle + \beta e^{-i \arccos(f(\phi(k)))} |1\rangle) \sum_{w \in \{0,1\}^a} e^{i\phi(k)w/N} |w\rangle .$$

   (b) Apply the controlled (on the third register) $V^\dagger$ gates:

$$|\psi_j\rangle \frac{1}{\sqrt{N}} \sum_k \gamma_{k|j} (\alpha e^{i \arccos(f(\phi(k)))} |0\rangle + \beta e^{-i \arccos(f(\phi(k)))} |1\rangle) \sum_{w \in \{0,1\}^a} e^{i\phi(k)w/N} e^{-i\mu_j w} |w\rangle .$$

   (c) Apply Hadamard gates to each of the ancilla qubits:

$$|\psi_j\rangle \frac{1}{N} \sum_k \gamma_{k|j} (\alpha e^{i \arccos(f(\phi(k)))} |0\rangle + \beta e^{-i \arccos(f(\phi((k)))} |1\rangle) \sum_{w \in \{0,1\}^a} \sum_{w \in \{0,1\}^a} e^{i\phi(k)w/N} e^{-i\mu_j w} (-1)^{w \cdot x} |x\rangle .$$

This means that $\tilde{U}$ performs the mapping

$$|\psi_j\rangle (\alpha |0\rangle + \beta |1\rangle) |0 \dots 0\rangle$$

$$\mapsto |\psi_j\rangle \frac{1}{N} \sum_k \gamma_{k|j} (\alpha e^{i \arccos(f(\phi(k)))} |0\rangle + \beta e^{-i \arccos(f(\phi(k)))} |1\rangle) \sum_x \left(\sum_w e^{i\phi(k)w/N} e^{-i\mu_j w} (-1)^{w \cdot x} \right) |x\rangle$$

for each eigenvector $|\psi_j\rangle$ of V.

Let $\{|\psi_j\rangle |b\rangle |\phi\rangle,\ b \in \{0,1\}\}$ be a basis for the tensor product of the three registers. By design, the only states that contribute to the trace of U' are those of the form $|\psi_j\rangle |b\rangle |0 \ldots 0\rangle$. Hence, we can consider the trace of $\tilde{U'}$ – the submatrix of $\tilde{U}$ in which the third register is in the state $|0 \ldots 0\rangle$) – which is given by:

$$\begin{aligned}
\mathrm{Tr}(\tilde{U'}) &= \sum_j \langle \psi_j | \langle 0 | \left(|\psi_j\rangle \frac{1}{N} \sum_k \gamma_{k|j} \sum_w e^{2\pi i w k/N} e^{-2\pi i \theta_j w} e^{i \arccos(f(\phi(k)))} |0\rangle \right) \\
&+ \sum_j \langle \psi_j | \langle 1 | \left(|\psi_j\rangle \frac{1}{N} \sum_k \gamma_{k|j} \sum_w e^{2\pi i w k/N} e^{-2\pi i \theta_j w} e^{-i \arccos(f(\phi(k)))} |1\rangle \right) \\
&= \frac{1}{N} \sum_{j,k} \gamma_{k|j} \sum_w e^{2\pi i w k/N} e^{-2\pi i \theta_j w} \left(e^{i \arccos(f(\phi(k)))} + e^{-i \arccos(f(\phi(k)))} \right) \\
&= \frac{1}{N} \sum_{j,k} \gamma_{k|j} 2 f(\phi(k)) \sum_w e^{2\pi i w (k/N - \theta_j)} \\
&= 2 \sum_{j,k} \left| \gamma_{k|j} \right|^2 f(\phi(k)) \\
&= 2 \sum_k f(\phi(k)) \sum_j \left| \gamma_{k|j} \right|^2 .
\end{aligned}$$

Suppose that $\theta_j = z_j/N$ for some z_j – that is, each $\mu_j = 2\pi\theta_j$ eigenvalue of V can be represented precisely by an n-bit rational number z_j/N. Then $\gamma_{k|j} = \delta_{k,z_j}$, and so $\mathrm{Tr}(\tilde{U'}) = 2\sum_j f(\mu_j)$. This corresponds to the case in which phase estimation works perfectly; in reality, we will not be able to express all eigenvalues precisely as n-bit rational numbers. Instead, suppose that $\theta_j = \tilde{z}_j/N + \delta_j$, where $\tilde{z}_j/N$ is the closest n-bit approximation of θ_j,

and so $0 \leq \delta_j \leq 1/(2N)$. The difference between the trace in the two cases is given by

$$\begin{aligned} 2\left|\sum_j f(\mu_j) - \sum_j \sum_k \left|\gamma_{k|j}\right|^2 f(\phi(k))\right| &\leq 2\sum_j \left|f(\mu_j) - \sum_k \left|\gamma_{k|j}\right|^2 f(\phi(k))\right| \\ &= 2\sum_j \left|\sum_k \left|\gamma_{k|j}\right|^2 (f(\mu_j) - f(\phi(k)))\right| \\ &\leq 2\sum_j \sum_k \left|\gamma_{k|j}\right|^2 |f(\mu_j) - f(\phi(k))|, \end{aligned}$$

where the second step follows because $\sum_k \left|\gamma_{k|j}\right|^2 = 1$. The coefficient $\left|\gamma_{k|j}\right|^2$ is precisely the probability of measuring $\phi(k)$ on the ancilla register when the true eigenvalue is μ_j. By the promises of phase estimation (Equation (4.4)), with probability $\leq \varphi$ we have $|\mu_j - \phi(k)| > 2\pi\eta$, in which case $|f(\mu_j) - f(\phi(k))| \leq 2f_{\max}$; and with probability $\geq 1 - \varphi$ we have $|\mu_j - \phi(k)| \leq 2\pi\eta$, in which case $|f(\mu_j) - f(\phi(k))| \leq 2\pi K\eta$. Hence, the error from this part of the circuit is bounded above by

$$2\left|\sum_j f(\mu_j) - \sum_j \sum_k \left|\gamma_{k|j}\right|^2 f(\phi(k))\right| \leq 4\sum_j (\pi K\eta + \varphi f_{\max}) = 2^{n+2}(\pi K\eta + \varphi f_{\max}).$$

Choosing $\eta < \epsilon/(8\pi)$ and $\varphi < \epsilon/8$, and assuming that $f_{\max} \leq 1$ (as stated earlier), this becomes

$$2\left|\sum_j f(\mu_j) - \sum_j \sum_k \left|\gamma_{k|j}\right|^2 f(\phi(k))\right| \leq 2^n \frac{1}{2}\epsilon(K+1). \tag{4.5}$$

Now we consider how this contributes to the overall error. As before, let λ_j denote the eigenvalues of e^{iA}. Then the error of the algorithm, taking into account both the Hamiltonian simulation and phase estimation steps, is

$$2\left|\sum_j f(\lambda_j) - \sum_j \sum_k \left|\gamma_{k|j}\right|^2 f(k)\right| \leq 2\left|\sum_j f(\lambda_j) - \sum_j f(\mu_j)\right| + 2\left|\sum_j f(\mu_j) - \sum_j \sum_k \left|\gamma_{k|j}\right|^2 f(k)\right|$$

where the first term on the right corresponds to the error from the Hamiltonian simulation part of the circuit (i.e. the difference between the trace of the circuit when using V instead of e^{iA}), and the second term corresponds to the error introduced by phase estimation. A

bound on the first term is given by Equation (4.1), and the second term is bounded via Equation (4.5). Therefore, the difference in the trace of U' in the case where Hamiltonian simulation and phase estimation both work perfectly, and when they do not, is bounded by

$$2\left|\sum_j f(\lambda_j) - \sum_j \sum_k \left|\gamma_{k|j}\right|^2 f(k)\right| \leq 2^n \epsilon (K + 1/2) \tag{4.6}$$

Error from estimating $\mathrm{Tr}(U')/2^n$ in the DQC1 model

The one clean qubit model can estimate the normalised trace of a $2^n \times 2^n$ sub-matrix of a $2^{n+O(\log n)} \times 2^{n+O(\log n)}$ unitary matrix (implemented by a $\mathrm{poly}(n)$-sized circuit) up to accuracy $\zeta = \Omega(1/\mathrm{poly}(n))$. Therefore, using the one clean qubit model to estimate the trace of U' will introduce an extra error term ζ. Let $\widetilde{\mathrm{Tr}(U')}/2^n$ be the output from the one clean qubit algorithm. Then choosing $\zeta = \epsilon/2$, and using the bound from Equation (4.6), we have

$$\left|\frac{2}{2^n}\sum_j f(\lambda_j) - \widetilde{\mathrm{Tr}(U')}/2^n\right| \leq \epsilon(K+1). \tag{4.7}$$

Hence, we can estimate $\frac{1}{2^n}\sum_j f(\lambda_j)$ in polynomial time with accuracy $\epsilon(K+1)$ for any $\epsilon = \Omega(1/\mathrm{poly}(n))$.

4.3.3 How many clean qubits are needed?

Here we consider how many clean qubits are required to implement the circuit described in Section 4.3.1 up to the desired accuracy. Any time the circuit uses ancilla qubits, these qubits will generally need to be initialised in the all-zeros state – that is, they must be under our control, and be 'clean'. As discussed in Section 4.1.1, we can use $O(\log n)$ clean qubits without changing the model of computation. In this section we argue that the implementation of the circuit described above requires no more than $O(\log n)$ ancilla qubits.

The two main parts of the circuit are the phase estimation routine, and Hamiltonian simulation. The rest of the circuit consists of more basic operations that require only a constant number of ancilla qubits (provided that the function f we choose is sufficiently easy to compute).

To achieve the accuracy stated in the previous section, we showed that the phase

estimation part of the circuit must be able to achieve an additive accuracy of $\frac{1}{2}\epsilon(K+1)$, for which it only requires $O(\log n)$ ancilla qubits. Hence, the number of clean qubits required to implement the phase estimation part of the circuit is $O(\log n)$.

In order to implement the simulation of the Hamiltonian A, we can use the technique of Trotterisation [82]. This requires no more than a constant number of ancilla qubits, and, since we assume that we are given the Hamiltonian directly as a set of m individual Hamiltonians that each act on $O(\log n)$ qubits, there are no ancilla qubits required to 'load' the input into the system, which would be the case if we considered the case where the input Hamiltonian is specified by an oracle (it is precisely for this reason that we define the problem in terms of a log-local Hamiltonian rather than a sparse Hamiltonian). In our case, we can run a polynomial-time classical algorithm to compute the quantum circuit required to implement the unitary e^{iA}, given such a description of A. This is discussed more fully in the following section.

4.3.4 Simulating log-local Hamiltonians

We are required to implement the unitary e^{iA} for some log-local Hamiltonian A. We are limited to using at most $O(\log n)$ ancilla qubits, which rules out the more advanced Hamiltonian simulation techniques that are based on quantum walks (e.g. [30]). Instead, we use the vanilla version of Hamiltonian simulation, which is based on the Lie-Trotter product formula [82].

We are given a log-local n-qubit Hamiltonian A, and wish to implement a unitary operator that approximates e^{iAt} for some value of t, up to a specified accuracy δ (in the operator norm). That is, we want to construct, in classical polynomial time, a quantum circuit that implements a unitary operator V such that

$$\|V - e^{iAt}\| \leq \delta.$$

It is straightforward to check that the standard techniques, which are usually presented for $O(1)$-local Hamiltonians, indeed work for log-local Hamiltonians and allow us to simulate e^{iAt} up to accuracy δ in time

$$O(\text{poly}(m, n, \tau, 1/\delta)),$$

where $\tau = t\|A\|$, using a circuit that can be computed by a polynomial-time classical algorithm, and we verify this here. The time complexity could be improved by the use of more

complicated simulation techniques [29], but we do not consider this here.

Using the Lie-Trotter product formula, we have that for any Hermitian matrices $H_1, ..., H_m$ satisfying $\|H_j\| \leq \zeta$ for all j,

$$e^{iH_1}e^{iH_2}\cdots e^{iH_m} = e^{i(H_1+H_2+...+H_m)} + O(m^3\zeta^2),$$

where the term $O(m^3\zeta^2)$ is used to denote some matrix E such that $\|E\| = O(m^3\zeta^2)$. Applying this to the matrices $H_j t/p$ for arbitrary t and some large integer p, we have

$$\left|\left|e^{iH_1t/p}e^{iH_2t/p}\cdots e^{iH_mt/p} - e^{i(H_1+H_2+...+H_m)t/p}\right|\right| = O\left(m^3\left(\frac{t\zeta}{p}\right)^2\right).$$

Let $p \geq Cm^3t^2\zeta^2/\delta$ for some constant C. Then

$$\left|\left|e^{iH_1t/p}e^{iH_2t/p}\cdots e^{iH_mt/p} - e^{i(H_1+H_2+...+H_m)t/p}\right|\right| \leq \delta/p,$$

and therefore

$$\left|\left|\left(e^{iH_1t/p}e^{iH_2t/p}\cdots e^{iH_mt/p}\right)^p - e^{i(H_1+H_2+...+H_m)t}\right|\right| \leq \delta.$$

Thus, to approximate e^{iHt} up to accuracy δ, it suffices to be able to implement the individual unitaries $e^{iH_jt/p}$ for $j \in [m]$, and $p = O(m^3t^2\zeta^2/\delta)$. If $\zeta \leq \|H\|$, and each individual unitary takes at most time T to implement, then we can approximate e^{iHt} up to accuracy δ in time $O(Tm^4\tau^2/\delta)$, where $\tau = t\|H\|$.

An arbitrary unitary operation on k qubits may be decomposed into a sequence of $O(k^2 2^{2k})$ one- and two-qubit gates [92]. In order to implement such a unitary up to accuracy ϵ using some universal gate set, we must implement each individual gate up to an accuracy of $O(\epsilon/(k^2 2^{2k}))$, which, by the Solovay-Kitaev theorem [44], can be achieved by using $O(\text{polylog}((k^2 2^{2k})/\epsilon))$ gates from a universal gate set. Furthermore, the precise circuit implementing these unitaries can be computed classically in polynomial time [44]. Then the entire unitary may be implemented up to accuracy ϵ using a circuit of size $O(\text{poly}(k, n, 1/\epsilon))$.

In our case, the unitaries that we want to implement act non-trivially on $O(\log n)$ qubits. Since there are m individual unitaries, and we apply each of them p times, we must be able to implement each one to an accuracy $\epsilon = \delta/(mp)$ in order to implement the entire unitary e^{iHt} up to accuracy δ.

By the above arguments, we can implement each unitary $e^{iH_j t/p}$ up to accuracy $\delta/(mp)$ in time that is polynomial in m, p, n and $1/\delta$. Hence, we find that we can simulate e^{iHt} up to accuracy δ in time

$$O(\text{poly}(m, n, \tau, 1/\delta)).$$

In the circuit described in Section 4.3.1, we set $t = 1$, and require that $\delta = O(1/\text{poly}(n))$. Thus, the time taken to implement the Hamiltonian simulation part of the circuit will be $O(\text{poly}(n))$.

4.3.5 Proof that computing $\text{Tr}(|A|^p)/2^n$ is in DQC1

The proof of Theorem 9, which states that the problem of estimating $\text{Tr}(|A|^p)/2^n$ up to error $\epsilon\|A\|^p$ is in DQC1 for $p, 1/\epsilon = \text{poly}(n)$, follows almost immediately from Lemma 6. The same proof also applies to the problem of estimating $\text{Tr}(A^p)/2^n$. It is straightforward to check[3] that, on the interval $[-b, b]$, both $f(x) = x^p$ and $f(x) = |x|^p$ are Lipschitz continuous with Lipschitz constant $K = pb^{p-1}$. Furthermore, $f_{\max} = b^p$ for both functions. In our case we can take $b = \|A\|$, since f is a function of the eigenvalues of A. Putting these values into Corollary 3, and replacing ϵ with $\frac{\epsilon}{p/\|A\|+1}$, we obtain an estimate of $\frac{\text{Tr}(|A|^p)}{2^n}$ up to accuracy $\epsilon\|A\|^p$. Furthermore, this estimate can be obtained in DQC1 in time that is polynomial in n and inverse polynomial in ϵ.

[3] A real-valued function $f(x) : \mathbb{R} \to \mathbb{R}$ is Lipschitz continuous with constant K if for all x_1, x_2 in the domain of f, we have $|f(x_1) - f(x_2)| \leq K|x_1 - x_2|$. To show that this holds for $f(x) = x^p$, we note that it is everywhere differentiable, and so it suffices to bound the absolute value of the derivative: $|f'(x)| = |px^{p-1}| \leq pb^{p-1}$ for all $x \in I$. For the case $g(x) = |x|^p$, we note that $g(x) = f(|x|)$. This is the composition of f with the modulus function $h(x) = |x|$. The latter is Lipschitz continuous with Lipschitz constant 1, by the reverse triangle inequality: $||x| - |y|| \leq |x - y|$ for all $x, y \in \mathbb{R}$. The composition of two Lipschitz continuous functions with Lipschitz constants K, K' is also Lipschitz continuous with constant KK'. Hence, $g(x)$ is Lipschitz continuous over I with Lipschitz constant pb^{p-1}.

4.4 Classical Algorithms

We now turn to classical algorithms for estimating Schatten p-norms, in an effort to understand the cases in which the quantum algorithms from the previous sections gain an advantage. We describe a classical algorithm for diagonal entry estimation, which is the problem of estimating an entry on the diagonal of the matrix A^p, up to reasonable error. Given the ability to estimate the diagonal entries of a matrix, we are able to estimate the normalised trace of that matrix.

We first present an algorithm for the special case where A contains only $0, 1$ entries, and then in Section 4.4.1 discuss how it can be extended to work for arbitrary real matrices. In the first case, the matrix A defines an unweighted, undirected graph with N vertices. The value of $(A^p)_{jj}$ is equivalent to the number of distinct walks (i.e. traversals around the graph that may traverse any edge more than once, or not at all) of length p starting and ending at vertex j.

We begin by observing that $(A^p)_{jj}$ can be re-interpreted as the total number of walks of length p leaving j multiplied by the probability that such a walk ends at vertex j. We can obtain an estimate of the latter by performing a number of random walks of length p, beginning at vertex j, and counting how many of them return to vertex j on the final step.

In order to obtain an estimate of the total number of walks of length p leaving a given vertex, we can do the following: given an upper bound d on the degree of the graph, we generate a number of sequences of p integers chosen independently and uniformly at random from the range $[0, d]$. Any given sequence provides a 'candidate' walk of length p on the graph, which may or may not be realisable on the graph defined by A. Given a candidate walk of the form $(n_0, n_1, ..., n_p)$, we test whether or not it is realisable by starting a walk at vertex j, and then moving to the n_0th neighbour of j. We then move to the n_1th neighbour of that vertex, and so on. If, at any step i of the walk, a vertex does not have a neighbour n_i, we terminate the process and conclude that the candidate is not realisable.

If we tried all d^p possible candidate walks from vertex j, then by counting the number of successes we would know the exact value of the number of walks of length p that leave vertex j; however, this would require $O(d^p)$ walks to be performed. If instead we sample from the set of all possible walks by generating a number of sequences at random, we can obtain a close estimate of the true number of walks. Below is the full algorithm for diagonal entry estimation. We assume that we are given some bound d on the degree of the graph,

and that we wish to estimate $(A^p)_{jj}$.

1. Estimate the total number of walks of length p leaving vertex j:
   (a) Define variables X_i for $i \in [k]$, for some value of k to be determined later.
   (b) For $i = 1$ to k:
      i. Generate a sequence $(n_0, n_1, ..., n_p)$, where each $n_l \in [d]$.
      ii. Attempt to follow the walk defined by the sequence.
      iii. If the walk was successful, set $X_i = 1$, otherwise set $X_i = 0$.
   (c) Then $\overline{X} = \frac{d^p}{k}(X_1 + X_2 + ... + X_k)$ provides an estimate of the total number of walks of length p leaving vertex j.
2. Estimate the probability that a given walk returns to vertex j:
   (a) Define variables Y_i for $i \in [k']$, for some value of k' to be determined later.
   (b) For $i = 1$ to k':
      i. Perform a random walk of length p starting at vertex j.
      ii. If the walk returns to vertex j (as its final step), then set $Y_i = 1$, otherwise set it to 0.
   (c) Then $\overline{Y} = \frac{1}{k'}(Y_1 + Y_2 + ... + Y_{k'})$ gives an estimate of the probability that a given walk returns to vertex j.
3. Multiplying the two values together gives us our desired estimate: $(\tilde{A^p})_{jj} = \overline{X} \cdot \overline{Y}$.

To analyse the accuracy of this estimation, we will look at the errors in the two estimates $\overline{X}$ and $\overline{Y}$.

In both steps, we are essentially aiming to estimate the success probability of some Bernoulli process: in step 1 we aim to estimate the probability with which a randomly generated sequence of 'moves' succeeds in generating a valid walk around the graph, and in step 2 we are estimating the probability that a given (valid) walk of length p succeeds in returning to its starting vertex on the final step of the walk. In both cases, we can estimate the appropriate probability up any desired accuracy ϵ by choosing the number of samples (k in step 1, and k' in step 2) to be inverse polynomial in ϵ.

We use Hoeffding's inequality to bound the accuracy of both estimates. For step 1, we absorb the factor of d^p into the random variables X_i, and use the general form of the

bound:

$$\Pr\left[|\overline{X} - \mathbb{E}[\overline{X}]| \geq \epsilon d^p\right] \leq 2e^{-2\epsilon^2 k}.$$

And for step 2, we have

$$\Pr\left[|\overline{Y} - \mathbb{E}[\overline{Y}]| \geq \epsilon'\right] \leq 2e^{-2\epsilon'^2 k'}.$$

Therefore, by setting $\epsilon, \epsilon' = 1/\operatorname{poly}(n)$, we need $k, k' = \operatorname{poly}(n)$, we can estimate $(A^p)_{jj}$ up to additive error that is at most $d^p(\epsilon + \epsilon' + \epsilon\epsilon') = d^p\delta$ for $\delta = 1/\operatorname{poly}(n)$, with a constant probability of failure.

4.4.1 Extension to real matrices

In this section we extend the diagonal entry estimation algorithm of the previous section to work for arbitrary real matrices. Recall that this algorithm works for matrices with $0, 1$ entries by interpreting the input matrix as the adjacency matrix for an unweighted, undirected graph. More general (symmetric) matrices may be viewed as undirected graphs with (perhaps negatively) weighted edges, and a similar interpretation of the value of $(A^p)_{jj}$ holds in these cases.

In the case of general matrices, the value of $(A^p)_{jj}$ depends not only on the number of closed walks (i.e. those that return to their start vertex) leaving vertex j, but also on the 'weight' of those walks. Let $\mathcal{C}_p^j$ be the set of all *closed* walks of length p leaving vertex j, and $E(\omega)$ be the set of edges that make up a given walk ω.

Then we have

$$(A^p)_{jj} = \sum_{c \in \mathcal{C}_p^j} \prod_{e \in E(c)} \text{weight}(e).$$

In order to estimate this quantity, we proceed similarly to the above case.

Let us denote the set of all (not necessarily closed) walks of length p originating at vertex j by $\mathcal{W}_p^j$, and write $W_p = |\mathcal{W}_p^j|$. Then we can re-write the above quantity as

$$(A^p)_{jj} = W_p \, \mathbb{E}_{\omega \in \mathcal{W}_p^j} \left[\prod_{e \in E(\omega)} \text{weight}(e) \right],$$

by using the same reasoning as before – i.e. that the jth diagonal entry of A^p is given by the total number of walks of length p leaving vertex j multiplied by the expected 'weight' of each walk, where we assign a weight of 0 if the walk does not return to vertex j.

We can estimate the expectation on the right by sampling from the set of closed walks of length p originating at vertex j. This can be done by performing random walks of length p starting at vertex j, and recording the total weights of those walks that return to vertex j. This is easily incorporated into the existing algorithm: we set the variable Y_i to 0 if the ith walk does not return to vertex j, and otherwise we set it to the total weight of the walk (i.e. the product over the weights of the edges of the walk). W_p can be estimated as before, up to error ϵd^p. The error in estimating the expectation value depends upon the largest total weight of a closed walk in the graph. This is smaller than or equal to $(\|A\|_{\max})^p$, where $\|A\|_{\max}$ is the maximum absolute size of an entry in A. A bound on the accuracy of estimating the expectation value is once again given by Hoeffding's inequality:

$$\Pr[|\overline{Y} - \mathbb{E}[\overline{Y}]| \geq \epsilon' \|A\|_{\max}^p] \leq 2e^{-2\epsilon'^2 k'}.$$

Multiplying the two estimates together, we obtain an estimate of $(A^p)_{jj}$ up to accuracy $\delta d^p \|A\|_{\max}^p$ with constant probability.

4.4.2 Estimating $\mathrm{Tr}(A^p)/N$ Classically

We can use the classical version of diagonal entry estimation to estimate the normalised trace of a matrix. More precisely, we obtain the empirical mean of $(A^p)_{jj}$ over a sample of values of j chosen uniformly at random. To see that the mean value of $(A^p)_{jj}$ for $j \in [N]$ does indeed give us the desired value, we observe that

$$\mathbb{E}_j[(A^p)_{jj}] = \frac{1}{N}\sum_{j=0}^{N-1}(A^p)_{jj} = \frac{\mathrm{Tr}(A^p)}{N}.$$

Let the output of the diagonal entry estimation algorithm be $(\widetilde{A^p})_{jj}$ (which is an estimate of $(A^p)_{jj}$ up to additive error $\delta d^p \|A\|_{\max}^p$). Then let $\overline{(\widetilde{A^p})_{jj}}$ be the mean value of the variable $(\widetilde{A^p})_{jj}$ after sampling k times for randomly chosen values of j. The value of $(\widetilde{A^p})_{jj}$ is bounded in the interval $[-(d\|A\|_{\max})^p, (d\|A\|_{\max})^p]$. Then by Hoeffding's inequality:

$$\Pr\left[\left|\overline{(\widetilde{A^p})_{jj}} - \mathbb{E}[(\widetilde{A^p})_{jj}]\right| \geq \delta d^p \|A\|_{\max}^p\right] \leq 2\exp\left(\frac{-\delta^2}{2}k\right).$$

Thus, choosing k to be inverse polynomial in δ allows us to obtain an estimate of $\mathbb{E}[(A^p)_{jj}] = \mathrm{Tr}(A^p)/N$ up to error $\delta d^p \|A\|_{\max}^p$ with high constant probability. Note that for $0, 1$ and

$-1, 0, +1$ matrices, $\|A\|_{\max} = 1$ and therefore the accuracy of the estimation in this case is just δd^p.

4.5 Quantum vs. Classical

We will now compare the complexities of the (BQP) quantum and classical algorithms for computing $\mathrm{Tr}(A^p)$, for random $N \times N$ matrices. Recall that the quantum algorithm has an accuracy of $\epsilon \|A\|^p$, and that the classical algorithm has an accuracy of $\epsilon d^p \|A\|_{\max}^p$, where $p = \mathrm{polylog}(N)$.

In the event that $\|A\| \ll d\|A\|_{\max}$, the quantum algorithm achieves an improvement in accuracy over the classical algorithm. However, since the quantum algorithm requires the matrix A to be sparse, we must restrict our attention to only sparse matrices that have this property. With this in mind, we will begin by considering a general model for random graphs, and introduce some results that relate the degrees of the vertices of the graph to the eigenvalues of the adjacency matrix. Following this, we will consider how these results apply to sparse graphs.

4.5.1 Random Graphs

We consider a general model for unweighted (directed) random graphs (see e.g. [39]), in which each vertex v is associated with a fixed weight w_v. Then a random graph G is constructed by assigning an edge independently to each pair of vertices (i, j) with probability $\frac{w_i w_j}{\sum_i w_i}$, such that the expected degree of vertex v is given by w_v. Denote by $d = \max_v w_v$ the maximum expected degree, and by $\tilde{d}$ the value

$$\tilde{d} := \frac{\sum_{i=1}^N w_i^2}{\sum_{i=1}^N w_i}.$$

Then we have the following results from [39]:

Theorem 12. *If $\tilde{d} > \sqrt{d} \ln N$, then as $N \to \infty$ the magnitude of the largest eigenvalue (in absolute value) of a random graph $G(w)$ is almost surely $(1 + o(1))\tilde{d}$.*

Theorem 13. *If $\sqrt{d} > \tilde{d} \ln^2 N$, then as $N \to \infty$ the magnitude of the largest eigenvalue (in absolute value) of a random graph $G(w)$ is almost surely $(1 + o(1))\sqrt{d}$*

Intuitively, $\|A\|$ is (asymptotically) the maximum of $\sqrt{d}$ and $\tilde{d}$ if the two values $\sqrt{d}$ and $\tilde{d}$ are far apart (i.e. by a power of $\log N$).

4.5.2 Restriction to Sparse Graphs

We are interested in sparse graphs – i.e. those in which the degree of every vertex is $O(\text{polylog}(N))$. If we use the random graph model above, and set $d = \Theta(\log^2 N)$, then if we allow all vertices to have an expected degree similar to d, then by Theorem 12, $\|A\| = (1 + o(1))d$ almost surely, and the accuracies of both the classical and quantum algorithms are the same. Therefore, we are only going to see an advantage when we restrict the number of vertices that are allowed to have degree close to the maximum (which will be $O(\text{polylog}(N))$ by necessity). In general, in an effort to make $\|A\| = o(d)$, we should only allow at most $O(\log N)$ vertices to have degree close to the maximum, and the others must have asymptotically smaller (e.g. constant) degree. A class of graphs that satisfies this requirement is the class of power law graphs.

A distribution on power-law graphs is given in [39] for which $d, \bar{d}$ and β are parameters that can be varied freely. In graphs of this type, the number of vertices with degree k is proportional to $k^{-\beta}$, and d is the maximum expected degree of a vertex in the graph, while $\bar{d}$ is the average degree. We have the following results, also from [39]:

1. For $\beta > 3$ and $d > \bar{d}^2 \log^{3+\epsilon} N$, the largest eigenvalue of the graph is almost surely $(1 + o(1))\sqrt{d}$, for some $\epsilon = O(1)$, and where $\bar{d}$ denotes the average degree.
2. For $2.5 < \beta < 3$ and $d > \bar{d}^{(\beta-2)/(\beta-2.5)} \log^{3/(\beta-2.5)} N$, the largest eigenvalue of the graph is almost surely $(1 + o(1))\sqrt{d}$.
3. For $2 < \beta < 2.5$ and $m > \log^{3/2.5-\beta} N$, the largest eigenvalue is almost surely $(1 + o(1))\tilde{d}$.

Note that in all of the above, the bounds still apply when the graph is sparse (i.e. $d = O(\text{polylog}\, N)$). Hence, for power law graphs with exponent $\beta > 2.5$, we almost always get a quadratic improvement in accuracy over the classical algorithm. As the exponent decreases, so does the advantage gained by the quantum algorithm.

Some interesting subclasses of power law graphs have exponents between 2 and 2.5. For example, 'internet graphs' have exponents between 2.1 and 2.4, and the 'Hollywood' graph

has exponent ≈ 2.3 [48]. In these cases, we might expect some quantum improvement over a classical approach, but not the full square root improvement.

Chapter 5

Post-selected classical query complexity

The work in this chapter is available as the pre-print "Post-selected classical query complexity", `arXiv:1804.10010`. The content of this chapter is mostly identical to the content of the pre-print, although a couple of the more minor results here are new – in particular, the exponential separation between post-selected quantum and classical query complexities described in the pre-print has been superseded by a stronger *unbounded* separation, which follows from a new observation relating rational degrees of Boolean functions to their non-deterministic degrees. Some content has also been modified following feedback from anonymous reviewers (mostly for clarity), and we include a lengthier introduction here to better put our results into context.

5.1 Introduction

Post-selection is the (hypothetical) power to discard all the runs of a computation in which a particular event does not occur [5]. For instance, suppose that we have a Boolean formula, and we wish to find an assignment of values to the variables that makes the formula evaluate to true. If we are guaranteed that such an assignment exists, then post-selection allows us to easily find it: we just guess a random assignment of values to the variables, and then post-select on the event that the formula evaluates to true. Thus, an ordinarily difficult problem becomes trivial when we have the power to post-select. We might then ask the following question: given the power of post-selection, which problems become easy to solve,

and which remain difficult?

The complexity class that captures this model of computation is PostBPP [56, 2]: bounded-error probabilistic polynomial time (BPP) with post-selection. More precisely, PostBPP is the class of languages L for which there exist two polynomial-time randomised classical algorithms $\mathcal{A}$ and $\mathcal{B}$ such that for all x,

$$\begin{aligned} &\text{If } x \in L \qquad \Pr[\mathcal{A}(x) = 1 | \mathcal{B}(x) = 1] \geq 2/3 \\ &\text{If } x \notin L \qquad \Pr[\mathcal{A}(x) = 1 | \mathcal{B}(x) = 1] \leq 1/3 \end{aligned}$$

and $\Pr[\mathcal{B}(x) = 1] > 0$, where the probabilities are taken over the random bits given to the algorithm. We can think of the algorithm $\mathcal{B}$ as the 'post-selector', which tells us whether or not to accept the output of $\mathcal{A}$. An alternative characterisation of the class is in terms of computational path lengths: here, in contrast to the class BPP, we allow the computational paths of a randomised algorithm to have different lengths, so that an input is accepted if there are at least twice as many possible accepting paths than rejecting paths, and is rejected if there are at least twice as many possible rejecting paths than accepting paths. In this view we no longer use post-selection, and instead design our algorithms so that the number of paths that return the correct answer sufficiently overwhelm the number of paths that return the incorrect answer. For this reason PostBPP is sometimes called $\mathsf{BPP}_{\mathsf{path}}$.

It is known that PostBPP contains MA and $\mathsf{P}^{\mathsf{NP}[\log]}$ (P with an NP oracle that can be called a logarithmic number of times), and is contained in PP, $\mathsf{BPP}^{\mathsf{NP}}$, and the third level of the polynomial hierarchy[1] [56]. Hence, PostBPP lies somewhere between the first and third levels of the polynomial hierarchy. Its quantum counterpart, PostBQP, was shown by Aaronson [5] to be equal to the class PP. Toda [118] showed that any problem in the polynomial hierarchy (PH) can be reduced to solving a polynomial number of problems in PP, and hence that $\mathsf{PH} \subseteq \mathsf{P}^{\mathsf{PP}}$. Therefore, quantum post-selection appears to be more powerful than classical post-selection, but the precise difference in computational power is not entirely clear. Indeed, if PostBPP = PostBQP, then the polynomial hierarchy collapses – something that is believed to be extremely unlikely. Conversely, the non-collapse of the polynomial hierarchy would imply that the computational powers of classical post-selection and quantum post-selection are far apart.

This observation forms the basis of many results in the area of quantum computational supremacy. In particular, if it were possible to efficiently simulate a quantum com-

[1] For a description of the polynomial hierarchy, see e.g. [3]

puter using a classical computer, then this would imply that a classical computer with post-selection could efficiently simulate a quantum computer with post-selection (i.e. that PostBPP = PostBQP) which, as noted above, is generally believed to be unlikely to be true. Moreover, many non-universal models of quantum computation become universal once post-selection is allowed [35], and so even an efficient classical simulation of one of these restricted models of quantum computing would lead to a collapse of the polynomial hierarchy [58]. Therefore, the study of both classical and quantum computation with post-selection might yield interesting insights into the differences between classical and quantum computation in general.

The idea of post-selection has been studied in the quantum information setting for some time, where it has been used as a tool for exploring the foundations of quantum mechanics, particularly in combination with the ideas of weak measurements [11, 84, 78] and pre- and post-selected systems [110, 12]. Only fairly recently has it received interest in the context of computation, where the ideas of post-selection and quantum computation have been combined to study the effects that alterations of quantum mechanics have on computational power. One interesting line of research in which post-selection has recently played a role is in the study of computation in models with 'non-deterministic causal order': see for instance the work of Araujo et al. [16], and also the work of Silva et al. [111] which demonstrates a connection between 'process matrices', which play a central role in Ref. [16], and pre- and post-selected quantum states.

In this chapter we focus very much on the computer science perspective, and study the query analogue of the class PostBPP. In the query model, we count only the number of times that an algorithm needs to access the bits of the input string to be able to compute some (usually Boolean) function, and not the total computation time. The query analogues of complexity classes can often shed light on the differences between various models of computation, and in this setting it is often possible to prove quite strong results. For instance, it is possible to rigorously prove separations between the power of quantum and classical computation in the query model [7]; however, we do not know how to prove such strong results in the context of circuit complexity, where many important questions remain open.

Recently, the query complexity of post-selected quantum computation was studied by Mahadev and de Wolf [86]: they showed that the post-selected quantum query complexity (PostQ) of a Boolean function is equal, up to a constant factor of 2, to the minimum

degree of a rational function that approximates it. The link between post-selection and rational approximation was first pointed out by Aaronson [5], and was made rigorous in [86]. Here, we study the classical analogue of this query model, and find that the classical post-selected query complexity (which we name PostR) is related in a similar way to the degrees of rational functions that have only positive coefficients.

We remark that the lower bounds developed in this chapter carry over to the communication complexity setting (see Section 5.3.2 for a definition of this model). Post-selected classical communication complexity, which we write $\mathsf{PostR}^{\mathsf{CC}}$, has been studied previously: as "approximate majority covers" in [75], and "zero-communication protocols" in [49], where the term "extended discrepancy" was coined for the dual characterisation. Göös et al. [53] proved a so-called 'simulation theorem', showing that lower bounds in certain query models can yield lower bounds in their communication analogues. Our characterisation of post-selected classical query complexity in terms of the degrees of rational functions can therefore also be used to derive lower bounds in the communication complexity setting.

5.1.1 Organisation

Section 5.2 provides some definitions that will be useful throughout the chapter. In Section 5.3 we describe our main results, and then in Section 5.4 we prove them. In Section 5.5, we give the proof of a lower bound on the post-selected classical query complexity of approximating the Majority function, which allows us to separate quantum and classical post-selected query complexities. In Section 5.8 we generalise the lower bound to arbitrary symmetric functions. Finally, in Section 5.6 we describe an efficient post-selected query algorithm for approximate counting.

5.2 Definitions

We begin by introducing and defining some concepts that will be helpful in understanding our results. We write $|x|$ to denote the Hamming weight of the bit-string x. When we refer to a 'query' of an n-bit input string $x \in \{0,1\}^n$, we mean that each query returns a the value of a single specified bit of x, and that these queries can of course be made to many bits of x in superposition. For an overview of complexity measures for Boolean functions and their relationships to each other, see for example the survey paper by Buhrman and

de Wolf [36].

5.2.1 Polynomial approximations

In this work we will concentrate on N-variate polynomials in which the input variables take on values from $\{0,1\}$. Such an N-variate polynomial is a function $P : \{0,1\}^N \to \mathbb{R}$, which can be written as $P(x) = \sum_{S \subseteq [N]} \alpha_S X_S$, where $X_S = \prod_{i \in S} x_i$, and the α_i are real coefficients. The *degree* of P is $\deg(P) = \max\{|S| : \alpha_S \neq 0\}$. Since $x_i^k = x_i$ for $k \geq 1$, we can restrict our attention to *multilinear polynomials*, where the degree of each variable is at most 1 (and therefore the degree of any polynomial is at most N).

For $\epsilon \in [0, 1/2)$, we say that a polynomial P ϵ-approximates a function $f : D \subseteq \{0,1\}^N \to \mathbb{R}$, if $|f(x) - P(x)| \leq \epsilon$ for all inputs $x \in D$. The ϵ-approximate degree of f, abbreviated $\deg_\epsilon(f)$, is the minimum degree over all polynomials that ϵ-approximate f. The exact degree of f is usually written as $\deg(f)$, and the 'approximate degree' of f, for some constant ϵ bounded away from $1/2$ (usually ϵ is chosen to be $1/3$) is written as $\widetilde{\deg}(f)$.

Non-deterministic polynomials

We will also have reason to consider *non-deterministic polynomials*. A non-deterministic polynomial for a Boolean function f is a polynomial $P : \{0,1\}^N \to \mathbb{R}$ which is non-zero iff $f(x) = 1$. The non-deterministic degree of a function f is the smallest degree of a non-deterministic polynomial for f, and is written $\mathsf{ndeg}(f)$.

Nonnegative literal polynomials

As well as non-deterministic polynomials, we will briefly consider *nonnegative literal polynomials*, introduced by Kaniewski et al. in [69]. These are $2N$-variate polynomials over the variables $\{x_i, (1 - x_i) : i \in [N]\}$ with only nonnegative coefficients. For a function $f : \{0,1\}^N \to \mathbb{R}^+$, let the nonnegative literal degree for 'yes'-instances, $\mathsf{ldeg}_1^+(f)$, be the minimum degree over nonnegative literal polynomials that *equal* $f(x)$ for all $x \in \{0,1\}^N$. Similarly, let the nonnegative literal degree for 'no'-instances, $\mathsf{ldeg}_0^+(f)$ be the minimum degree over nonnegative literal polynomials that equal $1 - f(x)$ for all $x \in \{0,1\}^N$. Finally, define $\mathsf{ldeg}^+(f) = \max\{\mathsf{ldeg}_0^+(f), \mathsf{ldeg}_1^+(f)\}$. The reason for this slightly non-standard definition of polynomial degree should become apparent later in this section.

5.2.2 Rational functions

A *rational function*, or *rational polynomial*, $R = P/Q$ is the ratio of two N-variate polynomials P and Q, where Q is defined to be non-zero everywhere to avoid division by zero. The degree of R is defined to be the maximum of the degrees P and Q. As before, we say that a rational function ϵ-approximates a function f if $|P(x)/Q(x) - f(x)| \leq \epsilon$ for all $x \in D$. For $x \notin D$, we allow $P(x)/Q(x)$ to take any value. The ϵ-approximate rational degree $\mathsf{rdeg}(f)$ of f is the minimum degree over all rational functions that ϵ-approximate f. In this work, we will consider $2N$-variate rational functions over the variables $\{x_i, (1-x_i) : i \in [N]\}$ in which all of the coefficients of P and Q are nonnegative, and call such a function a rational function with positive coefficients (note that these are not as general as rational functions with arbitrary coefficients – indeed, if f is a rational function with positive coefficients then $f(x) \geq 0$ for $x \in [0,1]^n$). We denote by $\mathsf{rdeg}^+_\epsilon(f)$ the minimum degree over all rational functions with positive coefficients that ϵ-approximate f.

5.2.3 Post-selected classical query algorithms

Here we define the complexity measure PostR_ϵ: the query complexity of post-selected (randomised) classical computation with error ϵ. First we precisely define what we mean by a classical post-selected query algorithm.

Definition 3. *A classical post-selected query algorithm $\mathcal{A}$ consists of a probability distribution over deterministic decision trees*[2] *that can output $0, 1$, or $\perp$ ('don't know'). Given an input x, $\mathcal{A}$ chooses a decision tree from its distribution (independently of x), and outputs the answer of that decision tree on input x. For a Boolean function f, we require that the output of $\mathcal{A}$ on x satisfies*

$$\Pr[\mathcal{A}(x) = f(x) | \mathcal{A}(x) \neq \perp] \geq 1 - \epsilon \qquad \textit{and} \qquad \Pr[\mathcal{A}(x) \neq \perp] > 0$$

for some choice of $\epsilon \in [0, 1/2)$.

Note that this definition is similar to the one given for $\mathsf{PostBPP}$ in Section 5.1, in which we consider a separate algorithm $\mathcal{B}$ to be the 'post-selector'. It is easy to see that the two views are equivalent – we can consider a single algorithm/decision tree that outputs $0, 1$,

[2] A (deterministic) decision tree is an adaptive algorithm for computing Boolean functions that chooses which input variable to query next based on the answers to the previous queries.

or $\perp$; or two algorithms/decision trees that each output 0 or 1.

Now we can define the complexity measure PostR_ϵ.

Definition 4. *The ϵ-approximate post-selected classical query complexity of a Boolean function f, written $\mathsf{PostR}_\epsilon(f)$, is the minimum query complexity over all classical post-selected query algorithms that ϵ-approximate f.*

One might ask what happens if we are allowed multiple rounds of post-selection: that is, what if we allow a post-selected query algorithm to be used as a subroutine inside another post-selected query algorithm? In this case, every Boolean function can be computed up to constant bounded error using only 1 query to the input: we can construct a post-selected query algorithm that decides, up to arbitrarily small one-sided error, whether some 'guessed' string y equals the input x. Then, using this as a subroutine, we can guess a string y and post-select on it being equal to x. Then we output $f(y)$, which will be correct with bounded error (of, say, $1/3$), and requires only a single query to x (made by the single call to the subroutine). A more rigorous proof of this observation is given in Section 5.7.

5.2.4 Certificate complexity and non-deterministic query complexity

Let $f : D \subseteq \{0,1\}^N \to \{0,1\}$ be an N-bit Boolean function. Then we have the following definitions:

Definition 5 (Certificate complexity)**.** *A b-certificate for an input x is an assignment $C_b : S \to \{0,1\}$ to some set $S \subseteq [N]$ of variables, such that $f(x) = b$ whenever an input x is consistent with C_b. The size of the certificate C_b is $|S|$. The b-certificate complexity $\mathsf{C}_b^x(f)$ of f on input x is the minimal size of a b-certificate that is consistent with x (where $b = f(x)$). The 1-certificate complexity is $\mathsf{C}_1(f) = \max_{x:f(x)=1} C_1^x(f)$. The 0-certificate complexity $\mathsf{C}_0(f)$ is defined analogously. Finally, we define the certificate complexity of f to be $\mathsf{C}(f) = \max\{\mathsf{C}_0(f), \mathsf{C}_1(f)\}$.*

Definition 6 (Non-deterministic query algorithms)**.** *A non-deterministic query algorithm for b-instances is a randomised algorithm whose acceptance probability is positive if $f(x) = b$, and zero if $f(x) = 1 - b$. Then the non-deterministic query complexity on input x, $\mathsf{N}_{f(x)}(f)$, is the minimum number of queries required by a classical algorithm to achieve the above behaviour on input x. The non-deterministic query complexity for 1-instances*

is $\mathsf{N}_1(f) = \max_{x:f(x)=1} \mathsf{N}_1(f)$*, and the non-deterministic query complexity for* 0*-instances,* $\mathsf{N}_0(f)$*, is defined analogously. Finally, the non-deterministic query complexity of* f *is defined to be* $\mathsf{N}(f) = \max\{\mathsf{N}_0(f), \mathsf{N}_1(f)\}$.

If we replace the classical query algorithm with a quantum query algorithm, we obtain the quantum counterparts of the above complexity measures $\mathsf{NQ}_1, \mathsf{NQ}_0$ *and* NQ.

The following results from [124] and [125] will prove useful later on:

Lemma 7 ([124], Proposition 1)**.** *For any Boolean function* f*,*

$$\mathsf{C}_b(f) = \mathsf{N}_b(f)$$

for $b \in \{0,1\}$*. Therefore,*

$$\mathsf{C}(f) = \mathsf{N}(f).$$

Lemma 8 ([125], Theorem 2.3)**.** *For any Boolean function* f*,*

$$\mathsf{NQ}(f) = \mathsf{ndeg}(f) \leq N(f).$$

Finally, we have the following useful result:

Lemma 9 ([124])**.** *Let* f *be a non-constant symmetric Boolean function on* N *variables. Suppose* f *achieves value* 0 *on* z *Hamming weights* $k_1, \ldots, k_z$*. Then* $\mathsf{ndeg}(f) \leq z$.

Proof. Since $|x| = \sum_j x_j$, then $(|x| - k_1)(|x| - k_2) \cdots (|x| - k_z)$ is a non-deterministic polynomial for f with degree at most z. □

We note here a formal connection between non-deterministic polynomials and rational functions, which, taking into account the results of Mahadev and de Wolf [86], demonstrates a connection between non-deterministic quantum query complexity and post-selected quantum query complexity.

Lemma 10. *Let* $p(x)$ *be a non-deterministic polynomial for a Boolean function* $f : \{0,1\}^n \to \{0,1\}$ *such that* $p(x) = 0$ *when* $f(x) = 0$*, and* $|p(x)| > s(x)$ *when* $f(x) = 1$*, for some (possibly constant) function* s*. Let* $t : \{0,1\}^n \to \mathbb{R}$ *be a degree-*d *polynomial that exactly represents* s*. Then* $r(x) = \frac{p(x)}{p(x)+s(x)\cdot\epsilon/2}$ *is a rational function of degree at most* $\max(\mathsf{ndeg}(f), d)$ *that* ϵ*-approximates* f.

Proof. The degree of r is immediate from the definition. We focus on correctness here. When $f(x) = 0$, $r(x) = 0$. When $f(x) = 1$, we have

$$r(x) = \frac{p(x)}{p(x) + s(x) \cdot \epsilon/2} = \frac{p(x) + s(x) \cdot \epsilon/2 - s(x) \cdot \epsilon/2}{p(x) + s(x) \cdot \epsilon/2} = 1 - \frac{\epsilon}{2\frac{p(x)}{s(x)} + \epsilon}.$$

In the case that $p(x) < 0$, we have that $p(x) \leq -s(x)$, and so we can bound

$$r(x) \leq 1 + \frac{\epsilon}{2 - \epsilon} \leq 1 + \epsilon$$

for all $0 < \epsilon < 1$.

In the case that $p(x) > 0$, then $p(x) \geq s(x)$, and so we can bound

$$r(x) \geq 1 - \frac{\epsilon}{2 + \epsilon} \geq 1 - \epsilon$$

for all $0 < \epsilon < 1$. Combining both cases means that for all x, $|r(x) - f(x)| \leq \epsilon$, and therefore that r ϵ-approximates f. □

We observe that this gives the following corollary:

Corollary 4. *Let f be a non-constant symmetric Boolean function that takes the value 0 on exactly z Hamming weights $k_1, \ldots, k_z$. Then for $\epsilon \in (0, 1]$, $\mathsf{rdeg}_\epsilon(f) \leq \mathsf{ndeg}(f) \leq z$.*

Proof. Let $p(x) = (|x| - k_1)(|x| - k_2) \cdots (|x| - k_z)$ be a non-deterministic polynomial for f, where we can write $|x| = \sum_{i \in [n]} x_i$. Since $|x|$ and all k_i are integers, the minimum absolute value for any $(|x| - k_i)$ is 1. In the wording of Lemma 10, we have that when $f(x) = 1$, $|p(x)| \geq 1$. Then by Lemmas 9 and 10, $\mathsf{rdeg}_\epsilon(f) \leq \mathsf{ndeg}(f) \leq z$. □

In [69], Kaniewski et al. show that the minimal degree of (literal) polynomials with positive coefficients (i.e. those described in Section 5.2.1) exactly characterises the model of 'classical query complexity in expectation': in this model, we consider (classical) query algorithms that, on input x, output a nonnegative random variable whose *expectation* equals $f(x)$. For a function $f : \{0,1\}^N \to \mathbb{R}^+$, $\mathsf{RE}(f)$ is the smallest number of queries that an algorithm needs to make to obtain the above behaviour for all $x \in \{0,1\}^N$. In particular, and using the notation introduced in this section, they show:

Lemma 11 ([69], Theorem 9)**.** *Let $f : \{0,1\}^N \to \mathbb{R}^+$. Then $\mathsf{RE}(f) = \mathsf{ldeg}_1^+(f)$.*

In the special case that f is a Boolean function, a query algorithm that computes f in expectation is (a slightly more constrained version of) a non-deterministic algorithm for yes-instances of f. Combining Lemma 7 with a straightforward generalisation of this result, we have, for any Boolean function f,

Corollary 5. $\mathsf{C}(f) = \mathsf{N}(f) \leq \mathsf{ldeg}^{+}(f)$.

5.3 Overview of Results

Our main result is a tight connection between post-selected classical query algorithms and rational functions with positive coefficients. This result is essentially the classical analogue of the one shown by Mahadev and de Wolf in [86]. In particular, we show

Theorem 14. *For any Boolean function $f : D \subseteq \{0,1\}^N \to \{0,1\}$, and $\epsilon \in (0, 1/2)$ we have*

$$\mathsf{rdeg}^{+}_{\epsilon}(f) \leq \mathsf{PostR}_{\epsilon}(f) \leq 2\mathsf{rdeg}^{+}_{\epsilon}(f).$$

That is, the post-selected classical query complexity of a Boolean function is essentially equivalent to the minimal degree of a rational function with positive coefficients that approximates that function. This can be compared to the quantum case

Theorem 15 (Mahadev & de Wolf [86])**.** *For any Boolean function $f : D \subseteq \{0,1\}^N \to \{0,1\}$, and $\epsilon \in (0, 1/2)$ we have*

$$\frac{1}{2}\mathsf{rdeg}_{\epsilon}(f) \leq \mathsf{PostQ}_{\epsilon}(f) \leq \mathsf{rdeg}_{\epsilon}(f).$$

in which there is also a tight relation between the two complexity measures. This can be contrasted to the conventional case (i.e. without post-selection), where the approximate degree $\widetilde{\deg}(f)$ of a function is only known to be polynomially related to the randomised query complexity $\mathsf{R}(f)$, and where polynomial gaps have indeed been shown to exist [7]. Recent work by Arunachalam et al. [18] has shown that a refinement of the notion of approximate degree tightly characterises the quantum query complexity of functions, but it is not completely clear how natural these refinements are, or whether they will yield new quantum query algorithms for interesting functions.

Our next result concerns the special case of exact approximation:

Theorem 16. *For any Boolean function $f : D \subseteq \{0,1\}^N \to \{0,1\}$ we have*

$$\mathsf{PostR}_0(f) = \mathsf{N}(f) = \mathsf{C}(f).$$

Combining this result with Corollary 5, we have an upper bound on the complexity of zero-error post-selected classical query algorithms:

Corollary 6. $\mathsf{PostR}_0(f) \leq \mathsf{Ideg}_+(f)$.

There is a long-standing open question, attributed to Fortnow, but given by Nisan and Szegedy in [93], which asks: is there a polynomial relation between the *exact* rational degree of a (total) Boolean function f and its usual polynomial degree? In [86], Mahadev and de Wolf recast this question in the context of post-selection algorithms: can we efficiently simulate an exact quantum algorithm with post-selection by a bounded-error quantum algorithm without post-selection?

By considering exact *classical* algorithms with post-selection, it is tempting to say that we have resolved this question in the case where the rational approximation can have only positive coefficients, but, unfortunately, the results of Theorem 14 break down when $\epsilon = 0$. What we *can* state is the following:

Corollary 7. *Given a degree-d rational function with only positive coefficients that exactly represents a Boolean function f, it is possible to construct a post-selected classical query algorithm $\mathcal{A}$ with query complexity at most d, such that when $f(x) = 1$, $\mathcal{A}(x) = 1$ with certainty, and when $f(x) = 0$, $\Pr[\mathcal{A}(x) = 0] \geq \frac{1}{1+\delta}$ for arbitrary $\delta > 0$.*

That is, we can obtain an algorithm with arbitrarily small one-sided error, but not an exact algorithm.

5.3.1 Separations

It is often instructive to find explicit functions that exhibit a gap between complexity measures. Here we seek functions that separate post-selected classical query complexity from various other complexity measures. In our case, the OR function[3] provides a number of useful separations. A post-selection algorithm for the N-bit OR function is as follows:

[3] Defined as $OR(x) = \begin{cases} 0 & \text{if } x = 00\ldots0 \\ 1 & \text{o.w.} \end{cases}$

- Choose a bit x_i, $i \in [N]$ uniformly at random from the input.
- If $x_i = 1$, return 1.
- Else return 0 with probability $1/(2N)$.
- Else return $\perp$ ('don't know').

We post-select on not seeing $\perp$ as the outcome. In the case that the input is the all-zero string, then the algorithm can only ever return 0, conditioned on it not returning $\perp$. In the case that at least one bit is equal to 1, the probability that the algorithm returns 1 is $\frac{\Pr[\text{return } 1]}{\Pr[\text{not return} \perp]} = \frac{|x|/N}{|x|/N+(1-|x|/N)\cdot 1/(2N)} \geq 2/3$. Therefore we can compute the OR function up to (one-sided) error $1/3$ using a single query to the input, and hence $\mathsf{PostR}_{1/3}(OR) = 1$. By combining this with existing results, we obtain the following separations:

- An unbounded separation from bounded-error quantum query complexity (Q_2), since $\mathsf{Q}_2(OR) = \Theta(\sqrt{N})$ due to Grover's algorithm [54].
- An unbounded separation from *exact* post-selected classical query complexity: since $\mathsf{C}_1(OR) = 1$ and $\mathsf{C}_0(OR) = N$, then $\mathsf{C}(OR) = N$ and so by Theorem 16, $\mathsf{PostR}_0(OR) = N$.
- An unbounded separation from quantum certificate complexity QC[4], since $\mathsf{QC}(OR) = \Omega(\sqrt{N})$, as shown by Aaronson [4]. This separation is interesting, since QC is the query analogue of the class QMA, and it is not clear whether $\mathsf{PostBPP}$ is more powerful than QMA.

Hence, post-selection makes the OR function trivial to compute up to bounded error, and shows that post-selected algorithms (both quantum and classical) can be much more powerful than classical, quantum, and non-deterministic (or exact post-selected) query algorithms. The OR function is an example of a problem that is easy for both quantum and classical post-selection. Indeed, a simple degree-1 rational function for estimating the OR function up to bounded error is

$$P_{OR}(x) = \frac{\sum_{i=1}^{N} x_i}{\epsilon + \sum_{i=1}^{N} x_i},$$

for some small and fixed constant $\epsilon > 0$. This polynomial has only positive coefficients, and so by Theorem 14 allows for an approximation by a classical post-selected query

[4] See [4] for a definition of quantum certificate complexity.

algorithm. In order to separate post-selected classical query complexity and post-selected quantum query complexity, we must find a function that allows for a low-degree rational approximation (and hence a small post-selected quantum query complexity, Theorem 15), but does not allow for a low-degree approximation by rational functions with only positive coefficients (and hence a large post-selected classical query complexity, Theorem 14).

To this end, in Section 5.5 we prove a $\Omega(N)$ lower bound on the post-selected classical query complexity of approximating the Majority function, defined on N-bit strings as:

$$MAJ_N(x) = \begin{cases} 1 & \text{if } |x| > N/2 \\ 0 & \text{if } |x| \leq N/2 \end{cases}.$$

This gives an exponential separation between quantum and classical post-selected query complexities. In particular, we show

Theorem 17. $\mathsf{PostR}_{1/3}(MAJ_N) = \Theta(N)$.

Since $\mathsf{PostQ}(MAJ) = \Theta(\log N)$ [86] and $\mathsf{PostR}(MAJ) = \Theta(N)$, this result shows that post-selected quantum computation can be much more powerful than post-selected classical computation in the query model.

Following this, we generalise the lower bound to any symmetric Boolean function f, which allows us to obtain an *unbounded* separation between quantum and classical post-selected query complexities. Write $f_k = f(x)$ for $|x| = k$, and define

$$\Gamma(f) = \min\{|2k - N + 1| : f_k \neq f_{k+1}, 0 \leq k \leq N-1\},$$

Then we show:

Theorem 18.

$$\mathsf{PostR}(f) \geq \frac{1}{8}\left(N - \Gamma(f)\right),$$

for all non-constant symmetric Boolean functions f.

This result is similar to a result of Paturi [95]:

Theorem 19 (Paturi)**.** *If f is a non-constant symmetric Boolean function on $\{0,1\}^N$, then $\widetilde{\deg}(f) = \Theta(\sqrt{N(N - \Gamma(f))})$.*

The lower bound from Theorem 18 above can be written in a similar form:

$$\mathsf{rdeg}^{+}(f) = \Omega(N - \Gamma(f))$$

for any (non-constant) symmetric function f.

We can use this characterisation to separate non-deterministic quantum and post-selected classical query complexities, which in turn separates the latter from post-selected quantum query complexity. Consider the N-bit function

$$f(x) = \begin{cases} 0 & \text{if } |x| = \lceil N/2 \rceil \\ 1 & \text{o.w.} \end{cases}$$

By Lemma 9 from Section 5.2.4, we know that $\mathsf{ndeg}(f) \leq 1$, since f is 0 on only one Hamming weight $|x| = \lceil N/2 \rceil$. On the other hand, we have $\Gamma(f) = O(1)$ and so by Theorem 18, $\mathsf{PostR}(f) = \Omega(N)$. This gives an unbounded separation between non-deterministic quantum query complexity $\mathsf{NQ}(f) = \mathsf{ndeg}(f)$ and post-selected classical query complexity $\mathsf{PostR}(f)$ by Theorem 15. By combining this with Corollary 4, which states that $\mathsf{rdeg}(f) \leq \mathsf{ndeg}(f)$ for any symmetric Boolean function f, we obtain an unbounded separation between post-selected quantum query complexity and post-selected classical query complexity, further emphasising the difference in computational power between classical and quantum post-selection.

5.3.2 Relation to communication complexity

Our results have some interesting connections to the communication complexity model. In this model, we consider two parties, Alice and Bob, who together want to compute some function $f : D \to \{0,1\}$, where $D \subseteq X \times Y$. Alice receives input $x \in X$, Bob receives input $y \in Y$, with $(x, y) \in D$. Typically, we choose $X = Y = \{0,1\}^n$. As the value $f(x, y)$ will generally depend on both x and y, some amount of communication between Alice and Bob is required in order for them to be able to compute $f(x, y)$. If $D = X \times Y$, then the function f is called *total*, otherwise it is called *partial*. The communication complexity of f is then the minimal number of bits that must be communicated in order for Alice and Bob to compute f.

In the introduction, we mentioned a method for obtaining lower bounds in commu-

nication complexity from lower bounds in query complexity, via the 'simulation theorem' of Göös et al. [53]. More precisely, they show that any $\mathsf{PostR}^{\mathsf{CC}}$ protocol for the composed function $f \circ g^N$ (where g is a particular small two-party function, often called a 'gadget'[5]) can be be converted into a corresponding query algorithm for f, which implies that $\mathsf{PostR}^{\mathsf{CC}}(f \circ g^N) \geq \Omega(\mathsf{PostR}(f) \cdot \log N)$. The authors prove analogous results for other models of computation, in which the lower bound becomes an equality (up to constant factors). Interestingly, for post-selection only a lower bound was shown, and so we cannot use our results to completely characterise the communication analogue of PostR.

In Section 5.5, we prove a $\Omega(N)$ lower bound on the post-selected classical query complexity of approximating the Majority function. In [75], Klauck gives an analogous $\Omega(N)$ lower bound on the post-selected classical *communication complexity* of approximating the Majority function. We note that this lower bound could also be obtained by combining the lower bound presented in this work with the fact that lower bounds on query complexity imply lower bounds on communication complexity in the presence of post-selection.

The reader familiar with communication complexity might wonder why Klauck's lower bound doesn't immediately imply the query lower bound, since Alice and Bob can run a communication protocol in which they just simulate the query algorithm, and hence the communication complexity is upper bounded by the query complexity. However, in order to simulate (post-selected) query algorithms in this way we require that the two parties have access to shared randomness, whereas Klauck's lower bound assumes that the two parties only have access to private randomness. This is a somewhat necessary assumption – if we allow for shared randomness, then all Boolean functions have $O(1)$ communication complexity [53] unless we incorporate an extra charge of $\log(1/\alpha)$ into the communication cost, where α is the probability that the query algorithm does not return 'don't know'.

Independent of the above, Klauck [75] gives an $O(\log N)$ *upper bound* for approximating the Majority function in the communication complexity analogue of the class PP. This upper bound is analogous to the one from [86] for the query complexity analogue of the class $\mathsf{PostBQP} = \mathsf{PP}$, and hence we suspect that Sherstov's [107] lower bound on the rational polynomial degree of the Majority function also carries over into the communication complexity setting.

[5] In this case, g is chosen to be

- $g(x, y) := \langle x, y \rangle \mod 2$, where $x, y \in \{0, 1\}^b$
- The block length $b = b(N)$ satisfies $b(N) \geq 100 \log N$.

5.3.3 Approximate counting using post-selection

It is possible to characterise PostBPP in terms of approximate counting problems (see e.g. [94]). This observation provides an alternative perspective on quantum supremacy results [58], as well as giving us a 'post-selection free' way to define the complexity classes related to PostBPP. Approximate counting was introduced by Sipser [112] and Stockmeyer [113, 114] as the problem of estimating the number of satisfying assignments to a given circuit or formula. In the query complexity setting, we can view this as the problem of estimating the Hamming weight $|x|$ of a bit-string x. In this context, the 'weak' approximate counting problem is defined to be the problem of estimating $|x|$ up to a factor of 2, and 'strong' approximate counting to be the problem of estimating $|x|$ up to a factor of $(1+1/p)$ for some 'accuracy parameter' $p \geq 1$.

In Appendix 5.6 we show that there exist efficient post-selected classical query algorithms for both versions of the approximate counting problem. By Theorem 14, this implies that there exist low-degree rational functions with positive coefficients for approximating the Hamming weight of a bit-string. In particular, we show the following:

Theorem 20. *Given an N-bit string x and an integer $p \geq 1$, there exists a post-selected classical query algorithm that outputs $\widetilde{|x|}$ such that*

$$\frac{1}{(1+1/p)}|x| \leq \widetilde{|x|} \leq (1+1/p)|x|$$

with probability $\geq 1-\epsilon$, by making at most $O(p \log N \log(\log N/\epsilon))$ queries to the input.

5.3.4 Techniques

The proof of our main result (Theorem 14) – namely that any degree-d rational function with positive coefficients can be approximated by a $O(d)$-query post-selected classical query algorithm – uses the observation that we can treat such rational functions as probability distributions over sets of monomials. Hence, given a rational function, we can sample from the set of monomials and use the probability amplification powers of post-selection to accurately approximate its value. On the other hand, if the coefficients of the rational functions can be negative, or take complex values, then the view that the coefficients represent probabilities can be replaced by the view that they are 'amplitudes' that can interact in complicated ways to produce more complex behaviour. This view is used in [86], in which post-selected *quantum* algorithms are used to approximate rational functions by

constructing quantum states whose amplitudes are proportional to the coefficients of the monomials of the rational functions. Hence, the only difference between classical and quantum post-selected query algorithms is in the use of complex amplitudes over conventional probabilities – an intuition that fits nicely with our understanding of quantum mechanics.

In order to prove the lower bounds in Section 5.5, we essentially bound how fast a low-degree (univariate) rational function can grow around a certain threshold. Similarly to other lower bounds on the degrees of polynomials that approximate threshold functions (e.g. [95]), we find that the nearer this threshold is to the 'middle', the faster the rational function needs to grow, and hence the larger its degree needs to be. Crucially, without the presence of negative coefficients, there can be no 'cancelling out' between the monomials of the polynomials that form the rational function, and therefore their degrees must be large enough to allow for rapid growth from 0 to 1 around the threshold value.

Many of the other results in the chapter involve the construction of post-selected query algorithms. Each algorithm follows the same general structure:

- Query some input bits.
- Based on these bits (for example, we might run some deterministic algorithm with these bits as input), either:
  - Accept.
  - Reject with some fixed small probability.
- Otherwise return 'don't know'.
- Finally, post-select on not seeing 'don't know'.

The power of post-selection lies in the ability to decide to return 'don't know' rather than just accepting or rejecting, which allows us to drastically amplify the success probability of the underlying algorithm.

An interesting phenomenon of success probability amplification in post-selected query algorithms was observed by Sherstov [107], in the context of rational approximations to Boolean functions.

Lemma 12. *[Sherstov [107]] Denote by $R(f,d)$ the smallest error achievable by a degree-d rational approximation to the Boolean function $f : X \subseteq \mathbb{R}^N \to \{0,1\}$. Then for all integers $d > 1$ and reals $t > 2$,*

$$R(f,td) \leq R(f,d)^{t/2}.$$

Informally, this result says the following: given a rational function that ϵ-approximates a function f, we can obtain a new rational function that ϵ^2-approximates f by increasing the degree by at most a constant factor of 4. We can then repeatedly apply this procedure to amplify the probability of success further. Sherstov describes how to construct the new rational function without introducing any negative coefficients, so by Theorem 14 we would expect to see a similar amplification property for post-selected classical (and quantum) query algorithms. Indeed, we implicitly make use of this observation in our proof of Theorem 14.

5.4 Proofs of the main results

5.4.1 Exact post-selected query complexity = certificate complexity

Here we show that when post-selected query algorithms are not allowed to make mistakes, they are equivalent to non-deterministic query algorithms.

Theorem 16. *For any Boolean function $f : D \subseteq \{0,1\}^N \to \{0,1\}$ we have*

$$\mathsf{PostR}_0(f) = \mathsf{N}(f) = \mathsf{C}(f).$$

This result follows from the following upper and lower bounds, and the fact that $\mathsf{N}(f) = \mathsf{C}(f)$ [124].

- $\mathsf{PostR}_0(f) \leq \mathsf{C}(f)$,
- $\mathsf{PostR}_0(f) \geq \max\{\mathsf{N}_0(f), \mathsf{N}_1(f)\} = \mathsf{N}(f)$.

$\mathsf{PostR}_0(f) \leq \mathsf{C}(f)$

To prove this bound, we construct a PostR_0 algorithm $\mathcal{A}$ that can compute any Boolean function f using at most C queries to the input. Given an n-bit input $x \in \{0,1\}^N$, we compute $f(x) \in \{0,1\}$ as follows:

- Choose a random certificate (which can be either a 1-certificate or a 0-certificate) C.
- If x is consistent with C, then output 0 if C is a 0-certificate, or 1 if C is a 1-certificate.
- Otherwise, output $\perp$.

Then we post-select on not seeing $\perp$. The query complexity follows from the fact that the algorithm only checks one certificate, whose size will be at most $\mathsf{C}(f)$. To prove correctness, consider the case where $f(x) = 0$. In this case, at least one of the 0-certificates will be consistent with x, and none of the 1-certificates will be. Let the number of consistent 0-certificates be c. Then

$$\Pr[\mathcal{A}(x) = 0 | \mathcal{A}(x) \neq \perp] = \frac{\Pr[\mathcal{A}(x) = 0]}{\Pr[\mathcal{A}(x) \neq \perp]} = \frac{c/2^N}{c/2^N} = 1.$$

Where the value of the denominator follows from the fact that only a 0-certificate can be consistent with x and cause the algorithm to not return $\perp$. A similar argument holds in the case that $f(x) = 1$. Hence, the algorithm will return the value of $f(x)$ with certainty in all cases.

$\mathsf{PostR}_0(f) \geq \max\{\mathsf{N}_0(f), \mathsf{N}_1(f)\} = \mathsf{N}(f)$

For the other direction, suppose that we are given the description of a PostR_0 algorithm for computing some boolean function f. Such an algorithm is defined by a probability distribution over a set of decision trees that each output $0, 1$, or $\perp$ for some input x. If we replace the $\perp$ output with 0, then we obtain a non-deterministic algorithm for 'yes' instances: any tree chosen from the distribution will either output $f(x)$ or 0, thus the probability of the algorithm accepting is non-zero if and only if $f(x) = 1$. Likewise, if we replace the $\perp$ output with 1, then we obtain a non-deterministic algorithm for 'no' instances. In both cases, the decision trees themselves have not been changed, and therefore the query complexity of the non-deterministic algorithm is the maximum of the query complexity of any of the decision trees, which is just the query complexity of the post-selection algorithm. This is enough to show that $\mathsf{PostR}_0(f) \leq \max\{\mathsf{N}_0(f), \mathsf{N}_1(f)\} = \mathsf{N}(f)$ for all f.

By Lemma 7, we have $\mathsf{N}(f) = \mathsf{C}(f)$ (i.e. non-deterministic query complexity is the same as certificate complexity) [124], and so $\mathsf{PostR}_0(f) = \mathsf{N}(f) = \mathsf{C}(f)$. This result is perhaps

a little surprising – classical post-selection provides no advantage over non-deterministic query algorithms when it is not allowed to make mistakes.

5.4.2 Bounded-error post-selected query algorithms are more powerful than non-deterministic query algorithms

5.4.3 $\mathsf{PostR}_{1/3}(f) \leq \min\{\mathsf{C}_0(f), \mathsf{C}_1(f)\}$

Here we show that classical post-selection can have a (much) smaller query complexity than non-deterministic query algorithms when it is allowed to make mistakes. In particular, we consider the case where the post-selection algorithm must return $f(x)$ with probability $\geq 2/3$.

Given some input $x \in \{0,1\}^N$, we show that it is possible to compute $f(x) \in \{0,1\}$ up to bounded-error using at most $\min\{\mathsf{C}_0(f), \mathsf{C}_1(f)\}$ queries with classical post-selection. First we show that $f(x)$ can be computed using at most C_1 queries, using the following post-selection algorithm:

- Choose a random 1-Certificate C.
- If x is consistent with C, then output 1.
- Else output 0 with probability $1/2^{N+1}$.
- Otherwise, output $\perp$.

Then we post-select on not obtaining $\perp$. The algorithm only makes use of a single 1-Certificate, and hence makes at most $\mathsf{C}_1(f)$ queries to the input.

To show correctness, first suppose that $f(x) = 1$. Then there must exist at least one 1-Certificate that is consistent with x. Since there are at most 2^N 1-Certificates for f, then the probability that at least one randomly chosen 1-Certificate is consistent with x is $\geq 1/2^N$. Then the probability of seeing 1, post-selecting on not seeing $\perp$, is given by

$$\begin{aligned} \Pr[\text{see } 1 | \text{not see } \perp] &= \frac{\Pr[\text{see } 1]}{\Pr[\text{not see } \perp]} \\ &\geq \frac{\frac{1}{2^N}}{\left(1 - \frac{1}{2^N}\right)\frac{1}{2^{N+1}} + \frac{1}{2^N}} \\ &> \frac{1}{\frac{1}{2} + 1} = \frac{2}{3}. \end{aligned}$$

On the other hand, suppose that $f(x) = 0$. Then there is no 1-Certificate C that is consistent with x. So, regardless of our choice of C, the algorithm always either returns 0 with probability $1/2^{N+1}$, or otherwise returns $\perp$. Hence, in this case we have

$$\begin{aligned} \Pr[\text{see } 0 | \text{not see } \perp] &= \frac{\Pr[\text{see } 0]}{\Pr[\text{not see } \perp]} \\ &= \frac{1/2^{N+1}}{1/2^{N+1}} = 1. \end{aligned}$$

So, when $f(x) = 1$, the algorithm returns 1 with probability at least $2/3$. If $f(x) = 0$, the algorithm returns 0 with certainty. It does this by making at most $C_1(f)$ queries to x, where $C_1(f)$ is the maximum size of a 1-certificate for f. This implies that $\mathsf{PostR}_{1/3}(f) \leq \mathsf{C}_1(f)$.

We can use a similar algorithm to show that $\mathsf{PostR}_{1/3}(f) \leq \mathsf{C}_0(f)$. By a similar proof to the previous case, if $f(x) = 0$, we return 0 with probability $> 2/3$. If $f(x) = 1$, we return 1 with certainty. Hence, $\mathsf{PostR}_{1/3}(f) \leq \mathsf{C}_0(f)$.

Combining these cases, we have that $\mathsf{PostR}_{1/3}(f) \leq \min\{\mathsf{C}_0(f), \mathsf{C}_1(f)\} \leq \mathsf{C}(f)$, with equality between the two terms on the right only when $\mathsf{C}_0(f) = \mathsf{C}_1(f)$.

5.4.4 Query complexity with classical post-selection ≈ degree of rational approximation with positive coefficients

Here we prove our main result – that the complexity of post-selected classical query algorithms is tightly characterised by the minimal degree of rational functions with positive coefficients.

Theorem 14. *For any Boolean function $f : D \subseteq \{0,1\}^N \to \{0,1\}$, and $\epsilon \in (0, 1/2)$ we have*

$$\mathsf{rdeg}^+_\epsilon(f) \leq \mathsf{PostR}_\epsilon(f) \leq 2\mathsf{rdeg}^+_\epsilon(f).$$

To prove this, we begin by showing that for any d-query classical post-selected query algorithm on the variables $\{x_1, \ldots, x_N\}$, there is a corresponding degree-d rational function over the variables $\{x_1, \ldots, x_N\} \cup \{(1-x_1), \ldots, (1-x_N)\}$ that has only positive coefficients.

Lemma 13. *For all Boolean functions f, $\mathsf{rdeg}^+_\epsilon(f) \leq \mathsf{PostR}_\epsilon(f)$*

Proof. Consider a classical post-selected query algorithm with complexity PostR_ϵ that computes some boolean function $f : \{0,1\}^N \to \{0,1\}$ with bounded error ϵ. We take the view that this algorithm consists of a probability distribution σ over a number of deterministic decision trees. Each decision tree i computes two functions $g_i(x)$ and $h_i(x)$, where $x = x_1, \ldots, x_N$. The first, g, is the tree's (proposed) answer to the Boolean function, and the second, h, is the post-selection function. Thus, the output of the decision tree is $g_i(x) \cdot h_i(x)$. Each decision tree must compute *both* of these functions using at most PostR_ϵ queries to the input. Denote the decision tree complexities of g_i and h_i by $\mathsf{D}(g_i)$ and $\mathsf{D}(h_i)$, respectively. Associated to each boolean function g_i and h_i are (unique) multivariate polynomials p_i and q_i [93], such that $\deg(p_i) + \deg(q_i) \leq \mathsf{PostR}_\epsilon(f)$. The acceptance probability of the algorithm is given by

$$\Pr_i[g_i(x) \cdot h_i(x) = 1 | h_i(x) = 1] = \frac{\Pr[g_i(x) = 1 \wedge h_i(x) = 1]}{\Pr_i[h_i(x) = 1]} = \frac{\Pr_i[p_i(x) = 1 \wedge q_i(x) = 1]}{\Pr_i[q_i(x) = 1]}.$$

The latter term is just a rational function $P \cdot Q/Q$, where $P(x) = \sum_i \sigma(i) p_i(x)$ and $Q(x) = \sum_i \sigma(i) q_i(x)$. Since g_i and h_i are total Boolean functions, the associated polynomials p_i and q_i can be written as polynomials in the variables $\{x_1, \ldots, x_N\} \cup \{(1-x_1), \ldots, (1-x_N)\}$, and only positive coefficients. To see this, consider some depth-1 decision tree, which queries the input bit x_i, and then outputs $b \in \{0,1\}$ if $x_i = 1$, or $1 - b$ otherwise. Such a decision tree can be (exactly) represented by the degree-1 polynomial $d(x) = bx_i + (1-b)(1-x_i)$. Now consider a depth-$T$ decision tree t, which begins by querying the input bit x_t. To obtain the polynomial that represents this tree, we can start at the root and write the corresponding polynomial as $d_t(x) = x_t d_l(x) + (1 - x_t) d_r(x)$, where $d_l(x)$ and $d_r(x)$ are the polynomials corresponding to the left and right sub-trees of t, respectively. In this way, the polynomial can be defined recursively in T steps, where at each stage the polynomial increases in degree by at most 1. When we reach a leaf of the tree, then the depth-1 case is used to determine the coefficient of each monomial in the polynomial. It follows that the resulting polynomial can be written as a degree-T polynomial in the variables $\{x_1, \ldots, x_N\} \cup \{(1 - x_1), \ldots, (1 - x_N)\}$, and only positive coefficients.

Hence, we have that P and Q are both polynomials with positive coefficients, and therefore $P \cdot Q/Q$ is a rational function with positive coefficients that ϵ-approximates f and has degree $\mathsf{rdeg}^+_\epsilon(f) = \max_i\{\deg(p_i) + \deg(q_i), \deg(q_i)\} = \mathsf{D}(g_i) + \mathsf{D}(h_i) \leq \mathsf{PostR}_\epsilon(f)$. □

To illustrate, consider the following PostR algorithm for approximating the AND func-

tion[6] up to bounded error 1/3:

- Choose a variable x_i uniformly at random from x.
- If $x_i = 0$, return 0.
- Else with probability $\frac{1}{2N}$, return 1.
- Else return $\perp$.

As usual, we post-select on not seeing $\perp$. Clearly, this algorithm makes one query to the input, and so we would expect the corresponding rational function to have degree 1. In this algorithm, each decision tree is of the following form:

- Query variable x_i, where i is fixed in advance.
- If $x_i = 0$, return 0.
- Else, either return 1 or $\perp$.

We require that for each i, there are $2N-1$ trees that output $\perp$ in the final step, and only one that outputs 1. For each i, if we choose a tree uniformly at random, the probability that it returns 1 is $\Pr[\text{see } 1] = \frac{1}{2N}x_i$. The probability that it doesn't return $\perp$ is $\Pr[\text{not see } \perp] = \frac{1}{2N}x_i + (1-x_i)$. Hence, choosing i also uniformly at random, the probability that the post-selection algorithm returns 1 is:

$$\Pr[\text{see } 1|\text{not see } \perp] = \frac{\Pr[\text{see } 1]}{\Pr[\text{not see } \perp]} = \frac{\frac{1}{N}\sum_i \frac{1}{2N}x_i}{\frac{1}{N}\sum_i \frac{1}{2N}x_i + (1-x_i)} = \frac{\frac{1}{2N}\sum_i x_i}{\frac{1}{2N}\sum_i x_i + \sum_i (1-x_i)},$$

which is a degree-1 rational function in $\{x_1, \ldots, x_N\} \cup \{(1-x_1), \ldots, (1-x_N)\}$. To check that this polynomial does indeed approximate the AND function, consider the hardest case to distinguish, when $|x| = N-1$. In this case, we have

$$\Pr[\text{see } 1|\text{not see } \perp] = \frac{\frac{1}{2N}(N-1)}{\frac{1}{2N}(N-1)+1} = \frac{N-1}{N-1+2N} \leq 1/3.$$

On the other hand, when $|x| = N$, we have $\Pr[\text{see } 1|\text{not see } \perp] = 1$, and so this polynomial does indeed approximate the AND function up to one-sided error of 1/3.

[6] $AND(x) = \begin{cases} 1 & \text{if } x = 11\ldots1 \\ 0 & \text{o.w.} \end{cases}$.

The next step in our proof is to show that a post-selected query algorithm can accurately determine the value of rational functions with positive coefficients.

Lemma 14. *For all Boolean functions f, $\mathsf{PostR}_\epsilon(f) \leq 2\mathsf{rdeg}^+_\epsilon(f)$.*

Proof. Consider an n-bit Boolean function f and a rational function P/Q with degree $d = \mathsf{rdeg}_\epsilon(f)$ and positive coefficients that ϵ-approximates f. That is, $|P(x)/Q(x) - f(x)| \leq \epsilon$ for all $x \in \{0,1\}^n$. In particular, for each x

- If $f(x) = 1$,
$$1 - \epsilon \leq \frac{P(x)}{Q(x)} \leq 1 + \epsilon$$
and hence
$$(1-\epsilon)Q(x) \leq P(x) \leq (1+\epsilon)Q(x).$$
- If $f(x) = 0$,
$$0 \leq \frac{P(x)}{Q(x)} \leq \epsilon$$
and hence
$$0 \leq P(x) \leq \epsilon Q(x).$$

To approximate f with bounded error ϵ, all we need to be able to do is determine, up to some reasonable level of error, which of these cases is true. Write $P(x) = \sum_{S\subseteq[n]} \alpha_S X_S$ and $Q(x) = \sum_{S\subseteq[n]} \beta_S Y_S$, and let $\gamma = \sum_S \alpha_S + \beta_S$. We will use the notation $M \in P$ (resp. $M \in Q$) to denote the event that the monomial M belongs to the polynomial P (resp. Q).

Consider the following post-selection query algorithm:

- Choose k monomials $M_1, M_2, \ldots, M_k$ (independently, and with replacement) from P or Q with probabilities $p(X_S) = \alpha_S/\gamma$, $p(Y_S) = \beta_S/\gamma$.
- If *all* k monomials evaluate to 1, then:
  - If all k monomials belong to P, i.e $M_i \in P\ \forall i \in [k]$, return 1.
  - Else if all k monomials belong to Q, i.e. $M_i \in Q\ \forall i \in [k]$, return 0 with probability r.

– Else, return $\perp$.

- Else, return $\perp$.

Once again, we post-select on not seeing $\perp$. The query complexity of this algorithm is at most kd. To see correctness, consider the case $f(x) = 1$. The probability of the algorithm returning 1, conditioned on not seeing $\perp$, is

$$\begin{aligned}
\Pr[\text{see } 1 | \text{not see } \perp] &= \frac{\prod_{i=1}^{k} \Pr[M_i = 1] \cdot \Pr[M_i \in P | M_i = 1]}{\prod_{i=1}^{k} \Pr[M_i = 1] \cdot \Pr[M_i \in P | M_i = 1] + r \cdot \prod_{i=1}^{k} \Pr[M_i = 1] \cdot \Pr[M_i \in Q | M_i = 1]} \\
&= \frac{\prod_{i=1}^{k} \Pr[M_i \in P | M_i = 1]}{\prod_{i=1}^{k} \Pr[M_i \in P | M_i = 1] + r \cdot \prod_{i=1}^{k} \Pr[M_i \in Q | M_i = 1]} \\
&= \frac{\frac{1}{\gamma^k} \left(\sum_{S:X_S=1} \alpha_S \right)^k}{\frac{1}{\gamma^k} \left(\sum_{S:X_S=1} \alpha_S \right)^k + \frac{r}{\gamma^k} \left(\sum_{S:Y_S=1} \beta_S \right)^k} \\
&= \frac{\left(\sum_S \alpha_S X_S \right)^k}{\left(\sum_S \alpha_S X_S \right)^k + r \cdot \left(\sum_S \beta_S Y_S \right)^k} \\
&= \frac{P(x)^k}{P(x)^k + r \cdot Q(x)^k}
\end{aligned}$$

But since $f(x) = 1$, we have $(1 - \epsilon)Q(x) \leq P(x) \leq (1 + \epsilon)Q(x)$, and so

$$\frac{P(x)^k}{P(x)^k + r \cdot Q(x)^k} \geq \frac{P(x)^k}{P(x)^k + \frac{r}{(1-\epsilon)^k} P(x)^k} = \frac{(1-\epsilon)^k}{(1-\epsilon)^k + r}.$$

We can perform the equivalent calculation for the case $f(x) = 0$:

$$\begin{aligned}
\Pr[\text{see } 0 | \text{not see } \perp] &= \frac{\prod_{i=1}^{k} \Pr[M_i = 1] \cdot r \cdot \Pr[M_i \in Q | M_i = 1]}{\prod_{i=1}^{k} \Pr[M_i = 1] \cdot \Pr[M_i \in P | M_i = 1] + r \cdot \prod_{i=1}^{k} \Pr[M_i = 1] \cdot \Pr[M_i \in Q | M_i = 1]} \\
&= \frac{r \cdot \prod_{i=1}^{k} \Pr[M_i \in P | M_i = 1]}{\prod_{i=1}^{k} \Pr[M_i \in P | M_i = 1] + r \cdot \prod_{i=1}^{k} \Pr[M_i \in Q | M_i = 1]} \\
&= \frac{\frac{r}{\gamma^k} \left(\sum_{S:Y_S=1} \beta_S \right)^k}{\frac{1}{\gamma^k} \left(\sum_{S:X_S=1} \alpha_S \right)^k + \frac{r}{\gamma^k} \left(\sum_{S:Y_S=1} \beta_S \right)^k} \\
&= \frac{r \cdot \left(\sum_S \beta_S Y_S \right)^k}{\left(\sum_S \alpha_S X_S \right)^k + r \cdot \left(\sum_S \beta_S Y_S \right)^k} \\
&= \frac{r \cdot Q(x)^k}{P(x)^k + r \cdot Q(x)^k}
\end{aligned}$$

In the case $f(x) = 0$, $0 \leq P(x) \leq \epsilon Q(x)$, and so

$$\begin{aligned}
\frac{r \cdot Q(x)^k}{P(x)^k + r \cdot Q(x)^k} &\geq \frac{r \cdot Q(x)^k}{\epsilon^k Q(x)^k + r \cdot Q(x)^k} \\
&= \frac{r}{\epsilon^k + r}.
\end{aligned}$$

Choosing $r = \sqrt{\epsilon^k (1-\epsilon)^k}$ maximises the difference between $\frac{(1-\epsilon)^k}{(1-\epsilon)^k + r}$ and $1 - \frac{r}{\epsilon^k + r}$. With this choice of r, the probability of failure (i.e. that the algorithm returns 0 when $f(x) = 1$ and 1 when $f(x) = 0$) is

$$p_{\text{fail}} \leq \frac{\sqrt{\epsilon^k}}{\sqrt{\epsilon^k} + \sqrt{(1-\epsilon)^k}}.$$

Choosing $k = 2$, we have

$$p_{\text{fail}} \leq \frac{\epsilon}{\epsilon + (1-\epsilon)} = \epsilon.$$

Therefore, by choosing $k = 2$ and $r = \epsilon(1-\epsilon)$, we obtain a post-selected classical query algorithm that ϵ-approximates f, and has degree at most $2\mathsf{rdeg}^+_\epsilon(f)$, implying that $\mathsf{PostR}_\epsilon \leq 2\mathsf{rdeg}^+_\epsilon$. Note that the above algorithm still works for polynomials over the variables $\{x_i, \ldots, x_N\} \cup \{(1-x_i), \ldots, (1-x_N)\}$, since all we have to do to determine if a monomial $x_i, \ldots, x_d(1-x_{i'}), \ldots, (1-x_{d'})$ evaluates to 1 is check that all $x_i, \ldots, x_d = 1$ and all $x_{i'}, \ldots, x_{d'} = 0$. □

Following the first version of this work, Ronald de Wolf pointed out that there is an alternative proof for Lemma 14 that uses the relationship between polynomials with positive coefficients and the model of query complexity in expectation introduced in Section 5.2.4:

Proof (Lemma 14). Consider an n-bit Boolean function f, and suppose that there exists a rational function P/Q with degree $d = \mathsf{rdeg}^+_\epsilon(f)$ and positive coefficients that ϵ-approximates f. Then P and Q are both degree-d nonnegative literal polynomials such that $Q(x) > 0$ and $|P(x)/Q(x) - f(x)| \leq \epsilon$ for all x. By Lemma 11, there exist classical d-query algorithms A and B whose *expected* outputs equal $P(x)$ and $Q(x)$, respectively. Without loss of generality, we can assume that such algorithms only output either 0 or some large fixed number m (if the algorithms instead output a lower value m', we can replace them with algorithms that output m with probability m'/m, and 0 otherwise, preserving the expected output value). By definition, we have $\Pr[A(x) = m] = P(x)/m$ and $\Pr[B(x) = m] = Q(x)/m$. Consider the following post-selected query algorithm $\mathcal{A}$:

- Run B on input x. If $B(x) = 0$, return $\perp$.
- Otherwise, run A on input x. If $A(x) = m$, return 1, otherwise return 0 (if A is not run, we assume that it just outputs 1).
- Post-select on not obtaining $\perp$.

Note that $\Pr[B(x) = m] = Q(x)/m$, which is non-zero everywhere since $Q(x)$ is non-zero everywhere by definition, and therefore post-selecting on the event that $B(x) = m$ is a valid move. The probability that the algorithm returns 1 is

$$\Pr[\mathcal{A}(x) = 1 | \mathcal{A}(x) \neq \perp] = \frac{\Pr[A(x) = B(x) = m]}{\Pr[B(x) = m]} = \frac{\Pr[A(x) = m]}{\Pr[B(x) = m]} = \frac{P(x)/m}{Q(x)/m} = \frac{P(x)}{Q(x)},$$

which is close to $f(x)$ by assumption. □

5.5 Lower bound on the Majority function

In this section we prove a $\Omega(N)$ lower bound on the bounded-error post-selected classical query complexity of the Majority function. We consider the problem of estimating the Majority function up to error ϵ, whose post-selected *quantum* query complexity is $\Theta(\log(N/\log(1/\epsilon))\log(1/\epsilon))$ (where the lower bound was proved by Sherstov [107], and

the corresponding upper bound by Mahadev and de Wolf [86]). Here we prove a lower bound of $\Omega\left(N\left(1-\sqrt{\frac{\epsilon}{1-\epsilon}}\right)\right)$, by using the relationship between post-selected classical query complexity and the degree of rational functions with positive coefficients.

The main tool that we will use to prove the lower bound is the well-known symmetrization technique, due to Minsky and Papert [87]. Let $f:\{0,1\}^N \to \{0,1\}$ be a symmetric (i.e. invariant under permutations of the bits of the input string) Boolean function on N variables, and let $P:\{0,1\}^N \to \{0,1\}$ be a degree-d polynomial that ϵ-approximates f. Let S_n be the set of all $n!$ permutations over n elements. Then the symmetrization of P is the average over all permutations of the input:

$$P^{\text{sym}}(X) = \frac{\sum_{\pi \in S_n} P(\pi(X))}{n!}.$$

Since f is a symmetric function, then P^{sym} still ϵ-approximates f, and has degree at most d. The following result allows us to reduce a multivariate polynomial approximating a symmetric function to a univariate polynomial:

Lemma 15. *[Minsky and Papert] There exists a univariate polynomial $p:\mathbb{R}\to\mathbb{R}$,*

$$p(k) = \mathop{\mathbb{E}}_{|X|=k}[p(X)]$$

of degree at most d, such that $p(|X|) = P^{sym}(X)$ for all $X \in \{0,1\}^N$, and hence p ϵ-approximates f.

Our first step will be to explicitly write down the form of the univariate polynomials that correspond to multivariate polynomials in the $2N$ variables $\{x_1,\ldots,x_N\} \cup \{(1-x_1),\ldots,(1-x_N)\}$.

Claim 1. *Let $P:\{0,1\}^{2N} \to \{0,1\}$ be a degree-d polynomial in the variables $\{x_1,\ldots,x_N\} \cup \{(1-x_1),\ldots,(1-x_N)\}$, which is necessarily of the form*

$$P(x) = \sum_{\substack{S,T\subseteq[N] \\ 1\leq|S|+|T|\leq d}} a_{S,T} \prod_{i\in S} x_i \prod_{j\in T}(1-x_j).$$

Then there exists a univariate polynomial p of degree at most d, of the form

$$p(k) = \sum_{\substack{S,T\subseteq[N]\\ 1\leq|S|+|T|\leq d}} a_{S,T}\frac{(N-|S|-|T|)!}{N!}k(k-1)\dots(k-|S|+1)\cdot(N-k)(N-k-1)\dots(N-k-|T|+1)$$

such that, if P ϵ-approximates a symmetric Boolean function f, then p also ϵ-approximates f, and depends only on the Hamming weight of the input string.

Proof. In the usual way, we take the expectation of P over permutations of the input bits, or equivalently, over all bit strings with the same Hamming weight as the input. That is, we consider

$$\begin{aligned} p(k) &= \mathop{\mathbb{E}}_{|x|=k}[P(x)] \\ &= \mathop{\mathbb{E}}_{|x|=k}\left[\sum_{\substack{S,T\subseteq[N]\\ 1\leq|S|+|T|\leq d}} a_{S,T}\prod_{i\in S}x_i\prod_{j\in T}(1-x_j)\right] \\ &= \sum_{\substack{S,T\subseteq[N]\\ 1\leq|S|+|T|\leq d}} a_{S,T}\mathop{\mathbb{E}}_{|x|=k}\left[\prod_{i\in S}x_i\prod_{j\in T}(1-x_j)\right]. \end{aligned}$$

The expectation over strings of Hamming weight k can be written as

$$\begin{aligned} \mathop{\mathbb{E}}_{|x|=k}\left[\prod_{i\in S}x_i\prod_{j\in T}(1-x_j)\right] &= \binom{N-|S|-|T|}{k-|S|}\Big/\binom{N}{k} \\ &= \frac{(N-|S|-|T|)!}{(k-|S|)!(N-k-|T|)!}\cdot\frac{k!(N-k)!}{N!} \\ &= \frac{(N-|S|-|T|)!}{N!}\cdot\frac{k!}{(k-|S|)!}\cdot\frac{(N-k)!}{(N-k-|T|)!} \end{aligned}$$

$$= \frac{(N-|S|-|T|)!}{N!}k(k-1)\dots(k-|S|+1)\cdot(N-k)(N-k-1)\dots(N-k-|T|+1)$$

which is a degree-$(|S|+|T|)$ polynomial in k. Hence, $p(k)$ is a degree-d polynomial in k. □

We remark that the coefficients $a_{S,T}$ are preserved in the transformation from P to the univariate polynomial p. Therefore if P is a polynomial with nonnegative coefficients, then

p has the property that all of the coefficients $a_{S,T}$ are nonnegative – a fact that will be useful later on.

Finally, we check that the symmetrization technique can still be used when working with rational functions.

Lemma 16. *Let $R(x) = P(x)/Q(x)$ be a degree d rational function that ϵ-approximates a symmetric N-bit Boolean function $f : \{0,1\}^N \to \{0,1\}$. Then there exist univariate polynomials p and q of degrees at most d such that*

$$\left| f(x) - \frac{p(|x|)}{q(|x|)} \right| \leq \epsilon,$$

where $p(k) = \mathbb{E}_{|x|=k}[P(x)]$ and $q(k) = \mathbb{E}_{|x|=k}[Q(x)]$.

Proof. Since $P(x)/Q(x)$ ϵ-approximates $f(x)$, we have

$$f(x) - \epsilon \leq P(x)/Q(x) \leq f(x) + \epsilon$$

and since $Q(x)$ is positive for all x,

$$Q(x)(f(x) - \epsilon) \leq P(x) \leq Q(x)(f(x) + \epsilon).$$

We take the expectation over permutations of the input bits (which is equivalent to taking the expectation over strings with identical Hamming weights). By Lemma 15, there exist univariate polynomials p and q such that $\mathbb{E}_\sigma[P(\sigma(x))] = p(|x|)$ and $\mathbb{E}_\sigma[Q(\sigma(x))] = q(|x|)$, where the expectation is taken over permutations $\sigma \in S_N$. Then since f is invariant under permutations of the input bits,

$$\begin{aligned}
& \mathbb{E}_\sigma[Q(\sigma(x))(f(\sigma(x)) - \epsilon)] \leq && \mathbb{E}_\sigma[P(\sigma(x))] && \leq \mathbb{E}_\sigma[Q(\sigma(x))(f(\sigma(x)) + \epsilon)] \\
\iff & f(x)\,\mathbb{E}_\sigma[Q(\sigma(x))] - \epsilon\,\mathbb{E}_\sigma[Q(\sigma(x))] \leq && \mathbb{E}_\sigma[P(\sigma(x))] && \leq f(x)\,\mathbb{E}_\sigma[Q(\sigma(x))] + \epsilon\,\mathbb{E}_\sigma[Q(\sigma(x))] \\
\iff & q(|x|)f(x) - q(|x|)\epsilon \leq && p(|x|) && \leq q(|x|)f(x) + q(|x|)\epsilon \\
\iff & f(x) - \epsilon \leq && p(|x|)/q(|x|) && \leq f(x) + \epsilon
\end{aligned}$$

which proves the result. □

Note that if P and Q are polynomials with nonnegative coefficients in the variables $\{x_0,\ldots,x_N\}\cup\{(1-x_0),\ldots,(1-x_N)\}$ (i.e. precisely those polynomials that correspond to the acceptance probabilities of post-selected classical query algorithms), then p and q are both of the form described in Claim 1, where each coefficient $a_{S,T}$ is nonnegative.

We are now ready to prove the lower bound.

Theorem 21. $\mathsf{PostR}_\epsilon(MAJ_N)\geq\frac{N}{2}\left(1-\sqrt{\frac{\epsilon}{1-\epsilon}}\,(1+o(1))\right)$.

Proof. To prove the lower bound, we prove a lower bound on the degree of rational functions with positive coefficients that approximate the Majority function, and then use Theorem 14 to obtain a lower bound on the post-selected query complexity. To begin, consider a degree-d rational function $P(x)/Q(x)$, with nonnegative coefficients, that approximates the Majority function up to error ϵ. Then by Lemma 16, there exist univariate polynomials p and q such that $p(|x|)/q(|x|)$ ϵ-approximates the Majority function. Since $MAJ(x)=1$ when $|x|>N/2$ and $MAJ(x)=0$ when $|x|\leq N/2$, we have the inequalities

$$(1-\epsilon)q\left(\frac{N}{2}+1\right)\leq p\left(\frac{N}{2}+1\right)\leq(1+\epsilon)q\left(\frac{N}{2}+1\right) \tag{5.1}$$

and

$$0\leq p\left(\frac{N}{2}\right)\leq\epsilon q\left(\frac{N}{2}\right). \tag{5.2}$$

From Claim 1, we can write

$$q\left(\frac{N}{2}\right)=\sum_{\substack{S,T\subseteq[N]\\1\leq|S|+|T|\leq d}}\gamma_{S,T}b_{S,T}\frac{N}{2}\left(\frac{N}{2}-1\right)\ldots\left(\frac{N}{2}-|S|+1\right)\cdot\frac{N}{2}\left(\frac{N}{2}-1\right)\ldots\left(\frac{N}{2}-|T|+1\right)$$

and

$$q\left(\frac{N}{2}+1\right)=\sum_{\substack{S,T\subseteq[N]\\1\leq|S|+|T|\leq d}}\gamma_{S,T}b_{S,T}\left(\tfrac{N}{2}+1\right)\tfrac{N}{2}\left(\tfrac{N}{2}-1\right)\ldots\left(\tfrac{N}{2}-|S|+2\right)\cdot\left(\tfrac{N}{2}-1\right)\ldots\left(\tfrac{N}{2}-|T|+1\right)\left(\tfrac{N}{2}-|T|\right).$$

Defining $\Delta_{S,T}:=\frac{N}{2}\left(\frac{N}{2}-1\right)\ldots\left(\frac{N}{2}-|S|+1\right)\cdot\frac{N}{2}\left(\frac{N}{2}-1\right)\ldots\left(\frac{N}{2}-|T|+1\right)$, these become

$$q\left(\frac{N}{2}\right)=\sum_{\substack{S,T\subseteq[N]\\1\leq|S|+|T|\leq d}}\gamma_{S,T}b_{S,T}\Delta_{S,T}$$

and

$$q\left(\frac{N}{2}+1\right) = \sum_{\substack{S,T\subseteq[N]\\ 1\leq|S|+|T|\leq d}} \gamma_{S,T} b_{S,T} \Delta_{S,T} \cdot \frac{\left(\frac{N}{2}+1\right)\left(\frac{N}{2}-|T|\right)}{\frac{N}{2}\left(\frac{N}{2}-|S|+1\right)}.$$

Our approach will be to bound the value of the term on the right for different values of d. Since $|S|+|T| \leq d$, we have the following lower bound

$$\begin{aligned}\frac{\left(\frac{N}{2}+1\right)\left(\frac{N}{2}-|T|\right)}{\frac{N}{2}\left(\frac{N}{2}-|S|+1\right)} &\geq \frac{\left(\frac{N}{2}+1\right)\left(\frac{N}{2}-d\right)}{\frac{N}{2}\left(\frac{N}{2}+1\right)} \\ &= \frac{N-2d}{N}.\end{aligned}$$

Since $\gamma_{S,T}, b_{S,T}, \Delta_{S,T} \geq 0$ for all choices of S and T, this implies that

$$q\left(\frac{N}{2}\right) \leq \frac{N}{N-2d} \cdot q\left(\frac{N}{2}+1\right). \tag{5.3}$$

Combining this with inequalities (5.1) and (5.2), we have

$$p\left(\frac{N}{2}\right) \leq \epsilon \cdot q\left(\frac{N}{2}\right) \leq \frac{\epsilon N}{N-2d} \cdot q\left(\frac{N}{2}+1\right) \leq \frac{\epsilon N}{(1-\epsilon)(N-2d)} \cdot p\left(\frac{N}{2}+1\right)$$

and hence

$$p\left(\frac{N}{2}\right) \leq \frac{\epsilon N}{(1-\epsilon)(N-2d)} \cdot p\left(\frac{N}{2}+1\right). \tag{5.4}$$

Similarly, we can write

$$p\left(\frac{N}{2}\right) = \sum_{\substack{S,T\subseteq[N]\\ 1\leq|S|+|T|\leq d}} \gamma_{S,T} a_{S,T} \Delta S, T$$

and

$$\frac{\epsilon}{(1-\epsilon)(N-2d)} \cdot p\left(\frac{N}{2}+1\right) = \sum_{\substack{S,T\subseteq[N]\\ 1\leq|S|+|T|\leq d}} \gamma_{S,T} a_{S,T} \Delta S, T \cdot \frac{\epsilon}{(1-\epsilon)(N-2d)} \frac{\left(\frac{N}{2}+1\right)\left(\frac{N}{2}-|T|\right)}{\frac{N}{2}\left(\frac{N}{2}-|S|+1\right)}.$$

This time we want to *upper bound* the value of the term on the right. Again using the fact

that $|S| + |T| \leq d$, we have

$$\begin{aligned}\frac{\epsilon N}{(1-\epsilon)(N-2d)} \frac{\left(\frac{N}{2}+1\right)\left(\frac{N}{2}-|T|\right)}{\frac{N}{2}\left(\frac{N}{2}-|S|+1\right)} &\leq \frac{\epsilon N\left(\frac{N}{2}+1\right)\frac{N}{2}}{(1-\epsilon)(N-2d)\frac{N}{2}\left(\frac{N}{2}-d+1\right)} \\ &= \frac{\epsilon N\left(\frac{N}{2}+1\right)}{(1-\epsilon)(N-2d)\left(\frac{N}{2}-d+1\right)}.\end{aligned}$$

We consider the values of d for which this quantity is strictly smaller than 1:

$$\begin{aligned}\frac{\epsilon N\left(\frac{N}{2}+1\right)}{(1-\epsilon)(N-2d)\left(\frac{N}{2}-d+1\right)} &< 1 \\ \frac{\epsilon}{(1-\epsilon)} N\left(\frac{N}{2}+1\right) &< (N-2d)\left(\frac{N}{2}-d+1\right) \\ 0 &< 2d^2 - 2(N+1)d + \left(1 - \frac{\epsilon}{(1-\epsilon)}\right) N(\frac{N}{2}+1).\end{aligned}$$

It is straightforward to check that this inequality is satisfied when

$$2d < (N+1) - \sqrt{\frac{\epsilon}{1-\epsilon}(N^2+2N)+1}.$$

Once again, using the fact that $\gamma_{S,T}, b_{S,T}, \Delta_{S,T} \geq 0$ for all choices of S and T, we find that if d satisfies the above constraint, then inequality (5.4) cannot be satisfied. Therefore, to ϵ-approximate the Majority function, any rational function with positive coefficients must have degree at least $\frac{1}{2}\left((N+1) - \sqrt{\frac{\epsilon}{1-\epsilon}(N^2+2N)+1}\right)$. In particular, if we take $\epsilon = 1/3$, then the degree d must satisfy

$$d \geq \frac{N}{8}$$

and so $\mathsf{PostR}_{1/3}(MAJ) = \Omega(N)$. More generally, we have $\mathsf{PostR}_{\epsilon}(MAJ_N) \geq \frac{N}{2}\left(1 - \sqrt{\frac{\epsilon}{1-\epsilon}}(1+o(1))\right)$. □

It turns out that this proof can be generalised in a straightforward manner to encompass *any* symmetric function. In Section 5.8, we prove the following:

Theorem 22. *Let $f : \{0,1\}^N \to \{0,1\}$ be any non-constant symmetric function on N*

bits. Write $f_k = f(x)$ *for* $|x| = k$*, define*

$$\Gamma(f) = \min\{|2k - N + 1| : f_k \neq f_{k+1}, 0 \leq k \leq N - 1,$$

and let T *be the value of* k *for which this quantity is minimised. Then*

$$\mathsf{PostR}(f) \geq \frac{1}{8}\left(N - \Gamma(f)\right),$$

and more generally for $\epsilon \in [0, 1/2]$*,*

$$\mathsf{PostR}_\epsilon(f_T) \geq \frac{1}{2}(N+1) \pm \sqrt{(N+1)^2 - 4\left(\frac{\epsilon}{1-\epsilon}\right)(N-T)(T+1)}.$$

Intuitively, $\Gamma(f)$ is a measure of how close to the middle a 'step change' (from 0 to 1, or vice versa) occurs. For instance, $\Gamma(OR) = N - 1$ and $\Gamma(MAJ) = 1$.

This lower bound shows we obtain no improvement (other than constant factors) over the decision tree complexity of the Majority function by adding post-selection. This is in contrast to the quantum case, in which adding post-selection allows Majority to be computed using exponentially fewer queries than in the non-post-selected case.

5.5.1 Post-selection algorithm for Majority

Here we describe a classical post-selected query algorithm for computing Majority up to bounded error 1/3, which uses $N/2+1$ queries to the input. This matches the lower bound up to a constant factor. Given an N-bit input x, the algorithm is as follows:

- Choose a subset of bits of size $\frac{N}{2} + 1$.
- If all bits are 1, then return 1.
- Else with probability r return 0.
- Else return $\perp$.

We post-select on not seeing $\perp$. To see that this algorithm works, consider an input such that $|x| = k$. Then the probability of the algorithm returning 1 is

$$\begin{aligned} p_1 := \Pr[\text{see } 1 | \text{not see } \perp] &= \frac{\binom{k}{\frac{N}{2}+1} \Big/ \binom{N}{\frac{N}{2}+1}}{\binom{k}{\frac{N}{2}+1} \Big/ \binom{N}{\frac{N}{2}+1} + \left(1 - \binom{k}{\frac{N}{2}+1} \Big/ \binom{N}{\frac{N}{2}+1}\right) \cdot r} \\ &= \frac{1}{1 - r + r \cdot \binom{N}{\frac{N}{2}+1} \Big/ \binom{k}{\frac{N}{2}+1}} \\ &\geq \frac{1}{1 + r \cdot \binom{N}{\frac{N}{2}+1} \Big/ \binom{k}{\frac{N}{2}+1}}. \end{aligned}$$

When $f(x) = 1$, the hardest case to decide is when $k = \frac{N}{2} + 1$, and therefore

$$\begin{aligned} p_1 &\geq \frac{1}{1 + r \cdot \binom{N}{\frac{N}{2}+1} \Big/ \binom{\frac{N}{2}+1}{\frac{N}{2}+1}} \\ &= \frac{1}{1 + r \cdot \binom{N}{\frac{N}{2}+1}}. \end{aligned}$$

Choosing $r = \frac{1}{2\binom{N}{\frac{N}{2}+1}}$, we have $p_1 \geq 2/3$.

In the case that $f(x) = 0$, we have $k \leq \frac{N}{2}$, and therefore $p_1 = 0$ always. Hence, this algorithm gives us bounded one-sided error of $1/3$.

Since the error is one-sided, we can take r to be arbitrarily small, taking the error arbitrarily close to zero without making any extra queries to the input. In this way, we obtain an algorithm for approximating the Majority function using at most $\frac{N}{2} + 1$ queries for *any* error $\epsilon \in (0, 1/2)$.

5.6 Approximate counting using post-selection

Here we show that there exist efficient post-selected classical query algorithms for both versions of the approximate counting problem. By Theorem 14, this implies that there exist low-degree rational functions with positive coefficients for approximating the Hamming weight of a bit-string.

Theorem 23. *Given an N-bit string x and an integer $p \geq 1$, there exists a post-selected classical query algorithm that outputs $\widetilde{|x|}$ such that*

$$\frac{1}{(1+1/p)}|x| \leq \widetilde{|x|} \leq (1+1/p)|x|$$

with probability $\geq 1-\epsilon$, by making at most $O(p \log N \log(\log N/\epsilon))$ queries to the input.

Proof. Consider the following algorithm, $\mathcal{V}(A,x)$, where A is a 'guess' at the value of $|x|$:

- Choose k input bits uniformly at random, with replacement.
- If $x_i = 1$ for all chosen bits, return 1.
- Else if $x_i = 0$ for all chosen bits, then with probability r, return 0.
- Else return $\perp$.

As usual, we post-select on not seeing $\perp$. Clearly, only k queries are required. The probability of the algorithm outputting 1 is

$$\begin{aligned} p_1 := \Pr[\mathcal{V}(A,x)=1 | \mathcal{V}(A,x) \neq \perp] &= \frac{\Pr[\mathcal{V}(A,x)=1]}{\Pr[\mathcal{V}(A,x)\neq\perp]} \\ &= \frac{(|x|/N)^k}{(|x|/N)^k + (1-|x|/N)^k r} \\ &= \frac{1}{1+r\frac{N^k}{|x|^k}(1-|x|/N)^k}. \end{aligned}$$

Our approach will be to guess a value $A = 2^i$ for some $0 \leq i \leq \lceil \log_2 N \rceil$. Then by choosing r appropriately, the algorithm will output 1 with high probability if our guess is at least a factor of 2 below the true value of $|x|$, and will output 0 with high probability if our guess is larger than the true value of $|x|$ by a factor of 2. We will choose $r = 2A^k/N^k$, so that

$$p_1 = \frac{1}{1+\frac{2A^k}{|x|^k}\left(1-\frac{|x|}{N}\right)^k}.$$

Consider a guess A for which $A < |x|/2$. In this case,

$$p_1 > \frac{1}{1+\frac{2}{2^k}\left(1-\frac{|x|}{N}\right)^k} \geq \frac{1}{1+\frac{2}{2^k}\frac{(N-1)^k}{N^k}}.$$

This value is exponentially (in k) close to 1. Now consider a guess A for which $A > 2|x|$. In this case,

$$p_1 < \frac{1}{1 + 2 \cdot 2^k \left(1 - \frac{|x|}{N}\right)^k}.$$

Since $A > 2|x|$, we can assume that $|x| \leq N/2$ (since we won't guess a value of A that is greater than N), and thus

$$p_1 < \frac{1}{1 + \frac{2 \cdot 2^k}{2^k}} = \frac{1}{3}.$$

By taking $k = 2$, when $A \leq |x|/2$, $p_1 \geq 2/3$, and when $A \geq 2|x|$, $p_1 \leq 1/3$. (Interestingly, choosing a larger value of k doesn't help us, since it can only increase the probability that the algorithm returns 1 when $A \leq |x|/2$, but not the probability that the algorithm returns 0 when $A \geq 2|x|$).

As we double our guess A, we will at some point encounter a run of the algorithm that returns 0 with probability greater than 2/3. At this point, we can stop our search, and output A as our estimate for $|x|$. This estimate will be within a factor of 4 of the true value of $|x|$, with high probability.

The overall algorithm will be:

1. For $i = 0$ to $\lceil \log_2 N \rceil$:
   (a) Set $A = \min\{2^i, N\}$, and run $\mathcal{V}(A, x)$ $O(\log(\log N/\epsilon))$ times.
   (b) If the majority of runs return 0, then we return A as our guess at $|x|$.
   (c) Otherwise, continue.
2. If $i = \lceil \log_2 N \rceil$, then return N as our guess for $|x|$.

By repeating the algorithm $\mathcal{V}$ $O(\log(\log N/\epsilon))$ times in step 1(a) and taking the majority, we amplify the acceptance/rejection probability of the algorithm: if $A \geq 2|x|$, the majority answer will be 0 with probability $\geq 1 - \frac{\epsilon}{\log N}$, and if $A < |x|/2$, the majority answer will be 1 with probability $\geq 1 - \frac{\epsilon}{\log N}$. For intermediate values of A, the algorithm might return 0 or 1 with probabilities close to 1/2. In this case, if the algorithm returns 1, we continue as normal. Else, if it returns 0, then we will be at most a factor of 1/2 away from the true value of $|x|$.

In the worst case, when $|x| > N/2$, then at most $m \leq \lceil \log_2 N \rceil$ rounds of the algorithm will be performed. In each round, the algorithm erroneously returns 1 with probability at most $\epsilon / \log N$. Hence, by the union bound, the probability that the algorithm fails in at least one of these rounds is at most $m\epsilon / \log N = \epsilon$. Otherwise, the probability that the algorithm doesn't return 0 in the first round that it guesses an $A \geq 2|x|$ is at most $\epsilon / \log N < \epsilon$. Thus, the algorithm fails with probability at most ϵ.

If the algorithm correctly halts after the first guess in which A is larger than $2|x|$, then our estimate will be at most $4|x|$. Hence we obtain an estimate $\widetilde{|x|}$ of $|x|$ such that

$$\frac{|x|}{2} \leq \widetilde{|x|} \leq 4|x|$$

with probability at least $1 - \epsilon$, making at most $O(\log N \log(\log N/\epsilon)$ queries to the input. We can improve the accuracy so that the estimate is within a factor of 2 by increasing the number of queries by a constant factor. Hence, this algorithm gives a solution to the 'weak' approximate counting problem.

We can use Stockmeyer's trick[7] [114] to obtain an algorithm that can produce an estimate of $|x|$ such that

$$\frac{1}{(1+1/p)}|x| \leq \widetilde{|x|} \leq (1+1/p)|x|$$

for arbitrary $p \geq 1$ by repeating the above algorithm $O(p)$ times. This is a solution to the 'strong' approximate counting problem, and we can obtain such a solution using post-selection by making at most $O(p \log N \log(\log N/\epsilon)$ queries to the input. □

Note that this algorithm, as stated, is not strictly a PostR algorithm. Rather, it can be viewed as a randomised classical algorithm with access to a PostR 'oracle', which it can call some polynomial number of times. However, in order to work the algorithm only needs to make non-adaptive calls to this oracle, since we don't have to stop the algorithm early if the majority of runs return 0 in step b above. Instead, we can run the algorithm for all $\lceil \log_2 N \rceil$ values of A, and do some post-processing to extract the right approxima-

[7] This roughly works as follows: assuming we have the ability to estimate $|x|$ for some input string $x = x_1, x_2, \ldots, x_n$ up to a constant factor of 2 in time $t(n)$, then we can estimate the value $|x'|$ for $x' := x_1 \wedge x_1, x_1 \wedge x_2, \ldots, x_n \wedge x_1, x_n \wedge x_2, \ldots, x_n \wedge x_n$ up to a constant factor of 2 in time $t(n^2)$ (since the length of x' is the square of the length of x). But since $|x'| = |x|^2$, then we have obtained an estimate of $|x|$ up to a factor of $\sqrt{2}$. An m-fold version of this process will allow us to obtain an estimate of $|x|$ up to a factor of $1 + O\left(\frac{1}{m}\right)$ in time $t(n^m)$. In our case, $t(n) = O(\log n)$, and so $t(n^m) = O(m \log n)$.

tion. Since the calls to the post-selection sub-routines are non-adaptive, we can replace the multiple post-selection steps with a single, 'global' post-selection step. In this way, we can construct a true PostR algorithm, which only makes a single use of post-selection as a final step.

Finally, we remark that this algorithm is almost optimal, since any algorithm that makes substantially fewer queries to the input would be able to violate the lower bound from Theorem 21. In particular, by choosing $p \geq \frac{1}{\sqrt{1+2/N}-1} \approx N$, and $\epsilon = 1/3$ above, we obtain an algorithm for approximating the Majority function up to accuracy $1/3$. Hence, any sub-linear dependence on p is ruled out by the Majority lower bound.

5.7 Post-selection algorithms with post-selection sub-routines

In this section, we show that if one is allowed to use a post-selection algorithm as a sub-routine inside another post-selection algorithm, then all Boolean functions can be computed up to bounded error using only a single query.

The algorithm works as follows: Guess a bit-string y, post-select on it being the same as the input x (using a single query), and then return $f(y)$. We first construct a verifier $V(x, y)$, which will accept with certainty if $y = x$, and reject with a probability arbitrarily close to 1 when $x \neq y$. It works as follows:

- Choose an index $i \in [n]$ uniformly at random.
- If $x_i \neq y_i$, return 0.
- Else, return 1 with probability $1/K$, for some K to be determined later.
- Else, return $\perp$.
- Post-select on not seeing $\perp$.

This procedure requires a single query to x. To see correctness, first consider the case in which $y = x$. Then, the probability of seeing 1 is

$$\begin{aligned} \Pr[\text{see 1}|\text{not see } \perp] &= \frac{\Pr[\text{see 1}]}{\Pr[\text{not see } \perp]} \\ &= \frac{\Pr[x_i = y_i] \cdot \frac{1}{K}}{\Pr[x_i \neq y_i] + \Pr[x_i = y_i] \cdot \frac{1}{K}} = 1. \end{aligned}$$

Now suppose that $y \neq x$. The hardest case to distinguish is when y differs from x on only a single bit:

$$\begin{aligned}
\Pr[\text{see } 0 | \text{not see } \perp] &= \frac{\Pr[\text{see } 0]}{\Pr[\text{not see } \perp]} \\
&= \frac{\Pr[x_i \neq y_i]}{\Pr[x_i \neq y_i] + \Pr[x_i = y_i] \cdot \frac{1}{K}} \\
&= \frac{\frac{1}{n}}{\frac{1}{n} + \left(1 - \frac{1}{n}\right) \cdot \frac{1}{K}} \\
&\geq \frac{\frac{1}{n}}{\frac{1}{n} + \frac{1}{K}} = \frac{1}{1 + \frac{n}{K}}.
\end{aligned}$$

We can choose K to be arbitrarily large in order to make this probability as close to 1 as needed.

Now that we have our verifier, we can use it to find a y such that $y = x$. The procedure is similar:

- Choose a y uniformly at random.
- If $V(x, y) = 1$, return $f(y)$.
- Else, return $\perp$.
- Post-select on not seeing $\perp$.

Since each use of $V(x, y)$ requires a single query to x, this procedure again only requires a single query to the input. Let A be the number of inputs in $f^{-1}(1)$, $N := 2^n$, and define $\epsilon := 1 - \frac{1}{1+\frac{n}{K}}$.

Suppose that $f(x) = 1$. The probability of seeing 1 is the probability that we choose a $y \in f^{-1}(1)$ multiplied by the probability that the verifier accepts such a bit-string. If $x = y$, this happens with certainty, and if $x \neq y$, this happens with probability at most ϵ. Thus,

$$\frac{1}{N} \leq p_1 \leq \frac{A}{N}\left(\frac{1}{A} + \frac{\epsilon(A-1)}{A}\right) = \frac{1 + \epsilon A - \epsilon}{N}.$$

On the other hand, the probability of seeing the 'wrong' answer, 0, is given by the probability of choosing a $y \in f^{-1}(0)$ multiplied by the probability that the verifier accepts,

which happens with probability at most ϵ (since $x \neq y$ in the case that $f(x) \neq f(y)$):

$$p_0 \leq \frac{\epsilon(N-A)}{N}.$$

Finally,

$$\begin{aligned}
\Pr[\text{see } 1 | \text{not see } \perp] &= \frac{\Pr[\text{see } 1]}{\Pr[\text{not see } \perp]} \\
&= \frac{p_1}{p_1 + p_0} \\
&\geq \frac{\frac{1}{N}}{\frac{1+\epsilon A - \epsilon}{N} + \frac{\epsilon(N-A)}{N}} \\
&\geq \frac{1}{1+\epsilon N}.
\end{aligned}$$

By choosing $\epsilon = \frac{1}{2N}$, and hence $K = 2nN$, we can ensure that this probability is greater than $2/3$. An identical argument follows in the case that $f(x) = 0$.

Note that we can't 'flatten' this algorithm to obtain one that produces the same result, but only performs one post-selection step.

5.8 Generalisation of the lower bound to arbitrary symmetric functions

Here we prove the more general version of the lower bound from Theorem 22, restated here for convenience:

Theorem 22. *Let $f : \{0,1\}^N \to \{0,1\}$ be any non-constant symmetric function on N bits. Write $f_k = f(x)$ for $|x| = k$, define*

$$\Gamma(f) = \min\{|2k - N + 1| : f_k \neq f_{k+1}, 0 \leq k \leq N-1,$$

and let T be the value of k for which this quantity is minimised. Then

$$\mathsf{PostR}(f) \geq \frac{1}{8}\left(N - \Gamma(f)\right),$$

and more generally for $\epsilon \in [0, 1/2]$,

$$\mathsf{PostR}_\epsilon(f_T) \geq \frac{1}{2}(N+1) \pm \sqrt{(N+1)^2 - 4\left(\frac{\epsilon}{1-\epsilon}\right)(N-T)(T+1)}.$$

Proof. Since $f_T \neq f_{T+1}$, there must be a step-change in the value of $f(x)$ at the point $|x| = T$. Without loss of generality, we can assume that this change is such that $f(T) = 0$ and $f(T+1) = 1$. Then, assuming that there exists some (univariate) rational function $p(x)/q(x)$ that ϵ-approximates f, we have

$$q\left(T\right) = \sum_{\substack{S,T \subseteq [N] \\ 1 \leq |S|+|T| \leq d}} \gamma_{S,T} b_{S,T} \Delta_{S,T}$$

and

$$q\left(T+1\right) = \sum_{\substack{S,T \subseteq [N] \\ 1 \leq |S|+|T| \leq d}} \gamma_{S,T} b_{S,T} \Delta_{S,T} \cdot \frac{\left(T+1\right)\left(N-T-|T|\right)}{\left(N-T\right)\left(T-|S|+1\right)}.$$

where

$$\Delta_{S,T} := \frac{N}{2}\left(\frac{N}{2}-1\right)\ldots\left(\frac{N}{2}-|S|+1\right) \cdot \frac{N}{2}\left(\frac{N}{2}-1\right)\ldots\left(\frac{N}{2}-|T|+1\right).$$

In the following, we can assume that $|T| \leq N-T$ and $|S| \leq T+1$ (since any monomial with $|S|$ or $|T|$ larger would necessarily equal zero and not contribute to the polynomials considered above). We can lower bound the term on the right using the fact that $|S|+|T| \leq d$:

$$\frac{\left(T+1\right)\left(N-T-|T|\right)}{\left(N-T\right)\left(T-|S|+1\right)} \geq \frac{\left(T+1\right)\left(N-T-d\right)}{\left(N-T\right)\left(T+1\right)} = \frac{\left(N-T-d\right)}{\left(N-T\right)}.$$

Hence,

$$q(T) \leq \frac{N-T}{N-T-d} \cdot q(T+1)$$

and so

$$p(T) \leq \epsilon q(T) \leq \frac{\epsilon(N-T)}{N-T-d} q(T+1) \leq \frac{\epsilon(N-T)}{(1-\epsilon)(N-T-d)} p(T+1). \tag{5.5}$$

The next step is to upper bound the value of $\frac{\epsilon(N-T)}{(1-\epsilon)(N-T-d)} \cdot \frac{(T+1)(N-T-|T|)}{(N-T)(T-|S|+1)}$. Using $|S|+|T| \leq$

d, we have

$$\frac{\epsilon(N-T)}{(1-\epsilon)(N-T-d)} \cdot \frac{(T+1)(N-T-|T|)}{(N-T)(T-|S|+1)} \leq \frac{\epsilon(N-T)}{(1-\epsilon)(N-T-d)} \cdot \frac{(T+1)(N-T)}{(N-T)(T-d+1)}$$

$$= \frac{\epsilon(N-T)}{(1-\epsilon)(N-T-d)} \cdot \frac{(T+1)}{(T-d+1)}.$$

We consider the values of d for which this quantity is strictly smaller than 1:

$$\begin{aligned} \frac{\epsilon(N-T)}{(1-\epsilon)(N-T-d)} \cdot \frac{(T+1)}{(T-d+1)} &< 1 \\ \frac{\epsilon(N-T)(T+1)}{(1-\epsilon)} &< (N-T)(T+1) - (T+1)d - (N-T)d + d^2 \\ 0 &< d^2 - (N+1)d + (N-T)(T+1)\left(1 - \frac{\epsilon}{1-\epsilon}\right) \end{aligned}$$

Writing $\alpha := \left(1 - \frac{\epsilon}{1-\epsilon}\right)$, the roots of the inequality are given by:

$$2d = (N+1) \pm \sqrt{(N+1)^2 - 4\alpha(N-T)(T+1)}$$

We can obtain a lower bound by taking the root $(N+1) - \sqrt{(N+1)^2 - 4\alpha(N-T)(T+1)}$. In this case, we have the lower bound

$$2d \geq (N+1) - \sqrt{(N+1)^2 - 4\alpha(T+1)(N+1) + 4\alpha(T+1)^2}.$$

If this bound is violated then inequality (5.5) cannot be satisfied, and hence no rational approximation can exist with degree smaller than this lower bound.

In particular, if we take $\epsilon = 1/3$, we have

$$\begin{aligned} 2d &\geq N + 1 - \sqrt{(N+1)^2 - 2(N-T)(T+1)} \\ &= N + 1 - \sqrt{(N-T)^2 + (T+1)^2}. \end{aligned}$$

It is straightforward to check that a suitable lower bound is

$$d \geq \frac{N}{8} - \frac{1}{4}\left|\frac{N}{2} - T\right| = \frac{1}{8}(N - \Gamma(f)).$$

□

As we pointed out in Section 5.3.1, this result is quite similar to a result of Paturi [95], and characterises the rational degree (and hence the post-selected classical query complexity) of symmetric functions.

Chapter 6

Concluding remarks

The thesis began by asking a number of related questions:

- What sorts of problems can we expect to obtain efficient (in terms of both time, and space) quantum algorithms for? And is there a general-purpose method for designing efficient quantum algorithms from existing classical ones?
- What happens when we restrict the power of a quantum computer?
- How do classical and quantum computation compare when we add power to both?
- Which property of quantum physics allows quantum computers access to more computational power than classical ones?

Each chapter focuses on roughly one of these questions at a time. In Chapter 2, we introduced the notion of a span program – a classical tool that has proven to be exceedingly useful in the design of efficient quantum query algorithms for a variety of problems, especially those relating to graph properties. Depending on the structure of the problem, in some cases these query algorithms can give rise to time efficient implementations, which also have the important property of being space efficient. We discussed some interesting recent work that has fully characterised the complexity of span-program based quantum algorithms for the problem of s-t connectivity in terms of two natural graph properties – effective resistance and effective capacitance. An obvious next step is to characterise the complexity of span-program based algorithms for other graph theoretic problems besides s-t connectivity, like, for instance, subgraph containment.

The work in Chapter 3 complemented the material introduced in Chapter 2 by considering span-program based quantum algorithms for detecting cycles in graphs, and deciding whether a graph is bipartite. The algorithms were found to be optimal, both in terms of time and space complexity, and were designed using a reduction from the original problems to that of s-t connectivity. This showed that quantum algorithms for these problems can be both time and space efficient, whereas the analogous classical result is not known (the logarithmic-space classical algorithms for these problems do not have optimal time complexity). An interesting related question is: can we also solve these (or related) problems in logarithmic space on a quantum computer for *directed* graphs? Since even the simple problem of st-connectivity on directed graphs is known to be NL-complete (where NL refers to non-deterministic classical log-space), then it would be very surprising if it could be solved by a quantum computer efficiently using only logarithmic space.

Following publication, our work was improved upon by DeLorenzo et al. [45], who obtained simpler algorithms by making use of a similar reduction, combined with an improved analysis of the span program for s-t connectivity that arose from the work of Jeffery and Kimmel [67], and later Jarret et al. [64]. These more recent results demonstrate the usefulness of both span programs in designing quantum algorithms, and the characterisation of span program witness sizes in terms of natural properties of computational problems. Such characterisations can indicate which types of problems are good candidates for quantum speedups. For instance, in the case that we have considered here, the characterisation of span program witness sizes in terms of the properties of effective resistance and capacitance of a graph allowed [67] to obtain more efficient quantum algorithms when the input had a certain structure – something that went unnoticed until these characterisations were found. This suggests that further work along these lines might yield more efficient quantum algorithms for a variety of problems when the input is promised to have a particular underlying structure.

In Chapter 4 we turned our attention to the next question, and considered what sorts of problems a restricted quantum computer can solve efficiently. We focussed on a non-universal model of quantum computation inspired by NMR computing: the one clean qubit model of computation, whose associated complexity class is known as DQC1. In this chapter, we showed that a natural mathematical problem, that of estimating Schatten p-norms of matrices up to a suitable level of accuracy, is contained in DQC1, and that the problem of estimating them up to a higher level of accuracy is DQC1-hard. These results suggest that

even restricted, non-universal quantum computers can efficiently solve problems that we do not know how to efficiently solve classically, which suggests that we do not necessarily need large-scale fault tolerant quantum computers to be able to perform useful computations.

The work in this chapter also relates somewhat to the final question asked above. It is known that the amount of entanglement present in DQC1 computations is severely limited; indeed, for certain types of (DQC1-complete) circuits, there is never entanglement between the single clean qubit and the maximally mixed register [98], and in general the amount of entanglement (as measured by the multiplicative negativity) present in the computer is always upper-bounded by a constant independent of the number of qubits [41]. Such results have prompted very credible suggestions that entanglement is not necessary for (mixed-state) quantum computers to achieve significant speedups over classical ones, with alternative proposals often focussing on other correlations that are present in mixed-state computation [43]. However, a recent result by Yoganathan and the current author [126] shows that entanglement is indeed necessary for the one clean qubit model to outperform classical computers, since without it, the computation can be classically simulated[1]. A natural open question is whether this result can be extended to the general case: can mixed-state quantum computers be classically simulated if there is never any entanglement present in the computation?

Finally, Chapter 5 considers what happens when we add power to computational models. In particular, it investigates the difference between quantum and classical computation when we add the hypothetical power of post-selection, in the query setting. Existing computational complexity results suggest that quantum computers augmented with post-selection are much more powerful than their classical counterparts augmented with that same power, and here we sought to make this separation rigorous by studying it in the query complexity setting, where it is often possible to prove separations between query complexity measures. Our main result, as it relates to quantum computing, is that the post-selected quantum and classical query complexities of a function can be significantly different – indeed, we obtained an unbounded separation between the two for a particular (total) Boolean function. More generally, we found that post-selected classical query algo-

[1] More precisely, this result considers a particular type of circuit in the one clean qubit model, in which the clean qubit is used only to control the application of a circuit on the other qubits. This restriction is well motivated, however, since such circuits can be shown to be complete for the class DQC1 – meaning that any problem that can be solved using the one clean qubit model can be solved by a circuit with this structure.

rithms are completely characterised by non-negative rational functions, a result analogous to that of Mahadev and de Wolf [86]. The characterisation of the query complexity of a model of computation in terms of degrees of polynomials follows a vein of research in this direction [20, 19], which has allowed, amongst other things, a proof of the optimality of Grover's algorithm for unstructured search [19, 54, 93]. A natural next step is to study the query complexities of quantum and classical computation when other restrictions or powers are added. For instance, in [126] it is conjectured that separable mixed state quantum computation is equivalent to classical computation when post-selection is added to both – that is, that $\mathsf{PostBQP}_{\mathsf{sep}} = \mathsf{PostBPP}$. It may be interesting to study such a conjecture in the query complexity setting, in which it might be possible to prove the equivalence and obtain some guidance as to whether it is indeed true.

References

[1] Complexity zoo. https://complexityzoo.uwaterloo.ca/Complexity_Zoo. Accessed: 11-01-2019.

[2] Complexity zoo: $\mathsf{BPP}_{\mathsf{path}}$. https://complexityzoo.uwaterloo.ca/Complexity_Zoo:B#bpppath. Accessed: 11-01-2018.

[3] Complexity zoo: The polynomial hierarchy, PH. https://complexityzoo.uwaterloo.ca/Complexity_Zoo:P#ph. Accessed: 11-01-2019.

[4] S. Aaronson. Quantum certificate complexity. In *Proceedings of the 18th IEEE Annual Conference on Computational Complexity (CCC), 2003*, pages 171–178. IEEE, 2003. arXiv:quant-ph/0210020.

[5] S. Aaronson. Quantum computing, postselection, and probabilistic polynomial-time. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 461, pages 3473–3482, 2005. arXiv:quant-ph/0412187.

[6] S. Aaronson. PDQP/qpoly=ALL. arXiv:1805.08577, 2018.

[7] S. Aaronson, S. Ben-David, and R. Kothari. Separations in query complexity using cheat sheets. In *Proceedings of the 48th annual ACM symposium on Theory of Computing*, pages 863–876. ACM, 2016. arXiv:1511.01937.

[8] S. Aaronson, A. Bouland, J. Fitzsimons, and M. Lee. The space just above BQP. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, pages 271–280. ACM, 2016. arXiv:1412.6507.

[9] D. Aharonov, V. Jones, and Z. Landau. A polynomial quantum algorithm for approximating the Jones polynomial. In *Proceedings of the thirty-eighth an-*

nual ACM symposium on Theory of computing, pages 427–436. ACM, 2006. arXiv:quant-ph/0511096.

[10] D. Aharonov and T. Naveh. Quantum NP-a survey. arXiv:quant-ph/0210077, 2002.

[11] Y. Aharonov, A. Botero, S. Popescu, B. Reznik, and J. Tollaksen. Revisiting Hardy's paradox: counterfactual statements, real measurements, entanglement and weak values. *Physics Letters A*, 301(3-4):130–138, 2002. arXiv:quant-ph/0104062.

[12] Yakir Aharonov and Lev Vaidman. Complete description of a quantum system at a given time. *Journal of Physics A: Mathematical and General*, 24(10):2315, 1991.

[13] R. Aleliunas, R. Karp, R. Lipton, L. Lovasz, and C. Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *Proc. 20th Annual Symposium on Foundations of Computer Science*, pages 218–223, 1979. doi:10.1109/SFCS.1979.34.

[14] A. Ambainis. Quantum walks and their algorithmic applications. *International Journal of Quantum Information*, 1(04):507–518, 2003. arXiv:quant-ph/0403120.

[15] A. Ambainis, K. Iwama, M. Nakanishi, H. Nishimura, R. Raymond, S. Tani, and S. Yamashita. Quantum query complexity of boolean functions with small on-sets. In *International Symposium on Algorithms and Computation*, pages 907–918. Springer, 2008.

[16] M. Araújo, P. Guérin, and Ä. Baumeler. Quantum computation with indefinite causal structures. *Physical Review A*, 96(5):052315, 2017. arXiv:1706.09854.

[17] A. Āriņš. Span-program-based quantum algorithms for graph bipartiteness and connectivity. 2015. arXiv:1510.07825.

[18] S. Arunachalam, J. Briët, and C. Palazuelos. Quantum query algorithms are completely bounded forms. *SIAM Journal on Computing*, 48(3):903–925, 2019. arXiv:1711.07285.

[19] R. Beals, H. Buhrman, R. Cleve, M. Mosca, and R. de Wolf. Quantum lower bounds by polynomials. *Journal of the ACM (JACM)*, 48(4):778–797, 2001. arXiv:quant-ph/9802049.

[20] R. Beigel. The polynomial method in circuit complexity. In *[1993] Proceedings of the Eigth Annual Structure in Complexity Theory Conference*, pages 82–95. IEEE, 1993.

[21] A. Belovs. Span-program-based quantum algorithm for the rank problem. 2011. arXiv:1103.0842.

[22] A. Belovs. Span programs for functions with constant-sized 1-certificates. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 77–84. ACM, 2012. arXiv:1105.4024.

[23] A. Belovs. Span Programs for functions with constant-sized 1-certificates. In *Proc. 44th annual ACM symposium on Theory of computing*, pages 77–84, New York, NY, USA, 2012. ACM. arXiv:1105.4024. doi:10.1145/2213977.2213985.

[24] A. Belovs. Quantum walks and electric networks. 2013. arXiv:1302.3143.

[25] A. Belovs, A. Childs, S. Jeffery, R. Kothari, and F. Magniez. Time-efficient quantum walks for 3-distinctness. In *Proc. 20th Annual European conference on Algorithms. Springer-Verlag*, pages 105–122. Springer, 2013.

[26] A. Belovs and B. Reichardt. Span Programs and quantum algorithms for st-connectivity and claw detection. In *European Symposium on Algorithms (ESA) 2012*, number 7501 in Lecture Notes in Computer Science, pages 193–204. Springer Berlin Heidelberg, 2012. arXiv:1203.2603.

[27] A. Ben-Aroya, O. Regev, and R. de Wolf. A hypercontractive inequality for matrix-valued functions with applications to quantum computing and LDCs. In *FOCS'08*, pages 477–486. IEEE, 2008. arXiv:0705.3806.

[28] E. Bernstein and U. Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26(5):1411–1473, 1997.

[29] D. Berry, G. Ahokas, R. Cleve, and B. Sanders. Efficient quantum algorithms for simulating sparse Hamiltonians. *Communications in Mathematical Physics*, 270(2):359–371, 2007. arXiv:quant-ph/0508139.

[30] D. W Berry, A. Childs, and R. Kothari. Hamiltonian simulation with nearly optimal dependence on all parameters. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 792–809. IEEE, 2015. arXiv:1312.1414.

[31] M. Boyer, G. Brassard, P. Høyer, and A. Tapp. Tight bounds on quantum searching. *Fortschritte der Physik*, 46(4-5):493–506, 1998. arXiv:quant-ph/9605034.

[32] M. Boyer, G. Brassard, P. Høyer, and A. Tapp. Tight bounds on quantum searching. *Fortschritte der Physik: Progress of Physics*, 46(4-5):493–505, 1998.

[33] F. Brandão. Entanglement theory and the quantum simulation of many-body physics. arXiv:0810.0026, 2008.

[34] G. Brassard, M. Mosca, and A. Tapp. Quantum amplitude amplification and estimation. 2002. arXiv: quant-ph/0005055.

[35] M. Bremner, R. Jozsa, and D. Shepherd. Classical simulation of commuting quantum computations implies collapse of the polynomial hierarchy. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 2010. arXiv:1005.1407.

[36] H. Buhrman, R. Cleve, R. de Wolf, and C. Zalka. Bounds for small-error and zero-error quantum algorithms. In *Foundations of Computer Science (FOCS), 1999. 40th Annual Symposium on*, pages 358–368. IEEE, 1999. arXiv:cs/9904019.

[37] S. Chakraborty, A. Gilyén, and S. Jeffery. The power of block-encoded matrix powers: improved regression techniques via faster hamiltonian simulation. arXiv:1804.01973, 2018.

[38] A. Childs and R. Kothari. Quantum query complexity of minor-closed graph properties. *SIAM Journal on Computing*, 41(6):1426–1450, 2012. arXiv:1011.1443. doi:10.1137/110833026.

[39] F. Chung, L. Lu, and V. Vu. Spectra of random graphs with given expected degrees. *Proceedings of the National Academy of Sciences*, 100(11):6313–6318, 2003.

[40] R. Cleve, A. Ekert, C. Macchiavello, and M. Mosca. Quantum algorithms revisited. *Proc. of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 454(1969):339–354, 1998. arXiv:quant-ph/9708016. doi:10.1098/rspa.1998.0164.

[41] A. Datta, S. Flammia, and C. Caves. Entanglement and the power of one qubit. *Physical Review A*, 72(4):042316, 2005. arXiv:quant-ph/0505213.

[42] A. Datta, A. Shaji, and C. Caves. Quantum discord and the power of one qubit. *Physical review letters*, 100(5):050502, 2008. arXiv:0709.0548.

[43] A. Datta and G. Vidal. Role of entanglement and correlations in mixed-state quantum computation. *Physical Review A*, 75(4):042310, 2007. arXiv:quant-ph/0611157.

[44] C. Dawson and M. Nielsen. The Solovay-Kitaev algorithm. *Quantum Information and Computation*, 6(1):81–95, 2006. arXiv:quant-ph/0505030.

[45] K. DeLorenzo, S. Kimmel, and R. Witter. Applications of the quantum algorithm for st-connectivity. In *14th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019. arXiv:1904.05995.

[46] D. Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 400(1818):97–117, 1985.

[47] C. Dürr, M. Heiligman, P. Høyer, and M. Mhalla. Quantum query complexity of some graph problems. In *Automata, Languages and Programming*, pages 481–493. Springer, 2004. arXiv:quant-ph/0401091.

[48] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *ACM SIGCOMM computer communication review*, volume 29, pages 251–262. ACM, 1999.

[49] D. Gavinsky and S. Lovett. En route to the log-rank conjecture: New reductions and equivalent formulations. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 514–524. Springer, 2014.

[50] A. Gilyén, Y. Su, G. Low, and N. Wiebe. Quantum singular value transformation and beyond: exponential improvements for quantum matrix arithmetics. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 193–204. ACM, 2019. arXiv:1806.01838.

[51] V. Giovannetti, S. Lloyd, and L. Maccone. Quantum random access memory. *Physical review letters*, 100(16):160501, 2008. arXiv:0708.1879.

[52] O. Goldreich. On promise problems: A survey. In *Theoretical computer science*, pages 254–290. Springer, 2006.

[53] M. Göös, S. Lovett, R. Meka, T. Watson, and D. Zuckerman. Rectangles are non-negative juntas. *SIAM Journal on Computing*, 45(5):1835–1869, 2016.

[54] L. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th annual ACM symposium on Theory of computing*, pages 212–219. ACM, 1996. `arXiv:quant-ph/9605043`.

[55] I. Gutman. The energy of a graph: old and new results. In *Algebraic combinatorics and applications*, pages 196–211. Springer, 2001.

[56] Y. Han, L. Hemaspaandra, and T. Thierauf. Threshold computation and cryptographic security. *SIAM Journal on Computing*, 26(1):59–78, 1997.

[57] A. Harrow, A. Hassidim, and S. Lloyd. Quantum algorithm for linear systems of equations. *Physical review letters*, 103(15):150502, 2009. `arXiv:0811.3171`.

[58] A. Harrow and A. Montanaro. Quantum computational supremacy. *Nature*, 549(7671):203, 2017.

[59] P. Hayden and A. Winter. Counterexamples to the maximal p-norm multiplicativity conjecture for all $p > 1$. *Communications in mathematical physics*, 284(1):263–280, 2008. `arXiv:0807.4753`.

[60] M. Howard, J. Wallman, V. Veitch, and J. Emerson. Contextuality supplies the 'magic' for quantum computation. *Nature*, 510(7505):351, 2014. `arXiv:1401.4174`.

[61] P. Høyer, M. Mosca, and R. de Wolf. Quantum search on bounded-error inputs. In *Automata, Languages and Programming*, pages 291–299. Springer, 2003. `arXiv:quant-ph/0304052`.

[62] D. Janzing and P. Wocjan. BQP-complete problems concerning mixing properties of classical random walks on sparse graphs. `arXiv:quant-ph/0610235`, 2006.

[63] D. Janzing and P. Wocjan. A Simple PromiseBQP-complete Matrix Problem. *Theory of computing*, 3(1):61–79, 2007.

[64] M. Jarret, S. Jeffery, S. Kimmel, and A. Piedrafita. Quantum algorithms for connectivity and related problems. In *26th Annual European Symposium on Algorithms (ESA 2018)*, volume 112, pages 49:1–49:13, 2018. arXiv:1804.10591.

[65] S. Jeffery. Span programs and quantum space complexity. 2019. arXiv:1908.04232.

[66] S. Jeffery and S. Kimmel. Nand-trees, average choice complexity, and effective resistance. arXiv:1511.02235, 2015.

[67] Stacey Jeffery and Shelby Kimmel. Quantum algorithms for graph connectivity and formula evaluation. *Quantum*, 1:26, August 2017. arXiv:1704.00765. URL: https://doi.org/10.22331/q-2017-08-17-26, doi:10.22331/q-2017-08-17-26.

[68] R. Jozsa and N. Linden. On the role of entanglement in quantum-computational speed-up. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 459(2036):2011–2032, 2003. arXiv:quant-ph/0201143.

[69] J. Kaniewski, T. Lee, and R. de Wolf. Query complexity in expectation. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 761–772. Springer, 2015. arXiv:1411.7280.

[70] A. Karanjai, J. Wallman, and S. Bartlett. Contextuality bounds the efficiency of classical simulation of quantum processes. arXiv:1802.07744, 2018.

[71] M. Karchmer and A. Wigderson. On Span Programs. In *Structure in Complexity Theory Conference*, pages 102–111, 1993.

[72] J. Kempe, A. Kitaev, and O. Regev. The complexity of the local Hamiltonian problem. *SIAM Journal on Computing*, 35(5):1070–1097, 2006.

[73] A. Kitaev. Quantum measurements and the Abelian Stabilizer Problem. 1995. arXiv:quant-ph/9511026.

[74] A. Kitaev, A. Shen, and M. Vyalyi. *Classical and quantum computation*, volume 47. American Mathematical Society Providence, 2002.

[75] H. Klauck. Rectangle size bounds and threshold covers in communication complexity. In *Computational Complexity, 2003. Proceedings. 18th IEEE Annual Conference on*, pages 118–134. IEEE, 2003. arXiv:cs/0208006.

[76] E. Knill and R. Laflamme. Power of one bit of quantum information. *Physical Review Letters*, 81(25):5672, 1998. arXiv:quant-ph/9802037.

[77] E. Knill and R. Laflamme. Quantum computing and quadratically signed weight enumerators. *Information Processing Letters*, 79(4):173–179, 2001. arXiv:quant-ph/9909094.

[78] S. Kocsis, B. Braverman, S. Ravets, M. Stevens, R. Mirin, K. Shalm, and Aephraim M. Steinberg. Observing the average trajectories of single photons in a two-slit interferometer. *Science*, 332(6034):1170–1173, 2011.

[79] François Le Gall. Improved quantum algorithm for triangle finding via combinatorial arguments. In *Proc. 55th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 216–225. IEEE, 2014. arXiv:1407.0085.

[80] T. Lee, R. Mittal, B. Reichardt, R. Špalek, and M. Szegedy. Quantum query complexity of state conversion. In *Proc. 52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 344–353, 2011. arXiv:1011.3020. doi:10.1109/FOCS.2011.75.

[81] X. Li, Y. Shi, and I. Gutman. *Graph energy*. Springer Science & Business Media, 2012.

[82] S. Lloyd. Universal quantum simulators. *Science*, 273(5278):1073, 1996. quant-ph/9703054.

[83] A. Lund, M. J Bremner, and T. Ralph. Quantum sampling problems, Boson Sampling and quantum supremacy. *npj Quantum Information*, 3(1):15, 2017. arXiv:1702.03061.

[84] J. Lundeen, B. Sutherland, A. Patel, C. Stewart, and C. Bamber. Direct measurement of the quantum wavefunction. *Nature*, 474(7350):188, 2011. arXiv:1112.3575.

[85] F. Magniez, A. Nayak, J. Roland, and M. Santha. Search via quantum walk. *SIAM Journal on Computing*, 40(1):142–164, 2011. arXiv:quant-ph/0608026.

[86] U. Mahadev and R. de Wolf. Rational approximations and quantum algorithms with postselection. *Quantum Information & Computation*, 15(3&4):295–307, 2015. arXiv:1401.0912.

[87] M. Minsky and S. Papert. *Perceptrons: An introduction to computational geometry*. 1969.

[88] M. Mitzenmacher and E. Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

[89] A. Montanaro. Quantum speedup of Monte Carlo methods. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 471(2181):20150301, 2015. arXiv:1504.06987.

[90] T. Morimae, K. Fujii, and Joseph F. Fitzsimons. Hardness of classically simulating the one-clean-qubit model. *Physical review letters*, 112(13):130502, 2014. arXiv:1312.2496.

[91] Tomoyuki Morimae. Hardness of classically sampling the one-clean-qubit model with constant total variation distance error. *Physical Review A*, 96(4):040302, 2017. arXiv:1704.03640.

[92] M. Nielsen and I. Chuang. *Quantum computation and quantum information*. Cambridge university press, 2010.

[93] N. Nisan and M. Szegedy. On the degree of Boolean functions as real polynomials. *Computational complexity*, 4(4):301–313, 1994.

[94] R. O'Donnell and A. Say. The weakness of CTC qubits and the power of approximate counting. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 23, page 147, 2016.

[95] R. Paturi. On the degree of polynomials that approximate symmetric boolean functions. In *Proceedings of the 24th annual ACM symposium on Theory of computing*, pages 468–474. ACM, 1992.

[96] D. Perez-Garcia, M. Wolf, D. Petz, and M. Ruskai. Contractivity of positive and trace-preserving maps under L_p norms. *Journal of Mathematical Physics*, 47(8):083506, 2006. arXiv:math-ph/0601063.

[97] S. Piddock. A quantum algorithm for detecting cycles using span programs. Unpublished.

[98] D. Poulin, R. Blume-Kohout, R. Laflamme, and H. Ollivier. Exponential speedup with a single bit of quantum information: Measuring the average fidelity decay. *Physical review letters*, 92(17):177906, 2004. arXiv:quant-ph/0310038.

[99] B. Reichardt. Span programs and quantum query complexity: The general adversary bound is nearly tight for every boolean function. In *Proc. 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 544–551. IEEE, 2009. arXiv:0904.2759.

[100] B. Reichardt. Faster quantum algorithm for evaluating game trees. In *Proc. 22nd annual ACM-SIAM symposium on Discrete Algorithms*, pages 546–559, San Francisco, California, 2011. SIAM. arXiv:0907.1623.

[101] B. Reichardt. Reflections for quantum query algorithms. In *Proc. 22nd annual ACM-SIAM symposium on Discrete Algorithms*, pages 560–569, San Francisco, California, 2011. SIAM. arXiv:1005.1601.

[102] B. Reichardt and R. Špalek. Span-Program-based quantum algorithm for evaluating formulas. In *Proc. 40th annual ACM symposium on Theory of computing*, pages 103–112, New York, NY, USA, 2008. ACM. arXiv:0710.2630.

[103] B.W. Reichardt. Span-Program-based quantum algorithm for evaluating unbalanced formulas. In *Lecture Notes in Computer Science*, number 6745, pages 73–103. Springer Berlin Heidelberg, 2011. arXiv:0907.1622.

[104] N. Robertson and P. Seymour. Graph minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004. doi:10.1016/j.jctb.2004.08.001.

[105] A. Rosenberg. On the time required to recognize properties of graphs: A problem. *ACM SIGACT News*, 5(4):15–16, 1973.

[106] D. Shepherd. Computation with unitaries and one pure qubit. arXiv:quant-ph/0608132, 2006.

[107] A. Sherstov. The intersection of two halfspaces has high threshold degree. In *Foundations of Computer Science (FOCS), 2009. 50th Annual IEEE Symposium on*, pages 343–362. IEEE, 2009. arXiv:0910.1862.

[108] P. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.

[109] P. Shor and S. Jordan. Estimating Jones polynomials is a complete problem for one clean qubit. *Quantum Information & Computation*, 8(8):681–714, 2008. arXiv:0707.2831.

[110] R. Silva, Y. Guryanova, N. Brunner, N. Linden, A. Short, and S. Popescu. Pre-and postselected quantum states: Density matrices, tomography, and kraus operators. *Physical Review A*, 89(1):012121, 2014. arXiv:1308.2089.

[111] R. Silva, Y. Guryanova, A. Short, P. Skrzypczyk, N. Brunner, and S. Popescu. Connecting processes with indefinite causal order and multi-time quantum states. *New Journal of Physics*, 19(10):103022, 2017. arXiv:1701.08638.

[112] M. Sipser. A complexity theoretic approach to randomness. In *Proceedings of the 15th annual ACM symposium on Theory of computing*, pages 330–335. ACM, 1983.

[113] L. Stockmeyer. The complexity of approximate counting. In *Proceedings of the 15th annual ACM symposium on Theory of computing*, pages 118–126. ACM, 1983.

[114] L. Stockmeyer. On approximation algorithms for $\#P$. *SIAM Journal on Computing*, 14(4):849–861, 1985.

[115] S. Subramanian and M. Hsieh. Quantum algorithm for estimating Renyi entropies of quantum states. arXiv:1908.05251, 2019.

[116] X. Sun, A. Yao, and S. Zhang. Graph properties and circular functions: How low can quantum query complexity go? In *Proc. 19th IEEE Annual Conference on Computational Complexity*, pages 286–293. IEEE, 2004.

[117] M. Szegedy. Quantum speed-up of Markov chain based algorithms. In *Proc. 45th Annual IEEE Symposium on Foundations of Computer Science*, pages 32–41, 2004. arXiv:quant-ph/0401053. doi:10.1109/FOCS.2004.53.

[118] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991.

[119] G. Vidal. Efficient classical simulation of slightly entangled quantum computations. *Physical review letters*, 91(14):147902, 2003. arXiv:quant-ph/0301063.

[120] G. Wang. Span-program-based quantum algorithm for tree detection. 2013. arXiv:1309.7713.

[121] J. Watrous. Quantum simulations of classical random walks and undirected graph connectivity. In *Proc. 14th Annual IEEE Conference on Computational Complexity*, pages 180–187. IEEE, 1999. arXiv:cs/9812012.

[122] J. Watrous. Space-bounded quantum complexity. *Journal of Computer and System Sciences*, 59(2):281–326, 1999.

[123] J. Watrous. Quantum computational complexity. In *Encyclopedia of complexity and systems science*, pages 7174–7201. Springer, 2009. arXiv:0804.3401.

[124] R. de Wolf. Characterization of non-deterministic quantum query and quantum communication complexity. In *Computational Complexity, 2000. Proceedings. 15th Annual IEEE Conference on*, pages 271–278. IEEE, 2000. arXiv:cs/0001014.

[125] R. de Wolf. Nondeterministic quantum query and communication complexities. *SIAM Journal on Computing*, 32(3):681–699, 2003. arXiv:cs/0001014.

[126] M. Yoganathan and C. Cade. The one clean qubit model without entanglement is classically simulable. arXiv:1907.08224, 2019.

[127] S. Zhang. On the power of Ambainis lower bounds. *Theoretical Computer Science*, 339(2):241–256, 2005. arXiv:quant-ph/0311060.

[128] Y. Zhu. Quantum query complexity of constant-sized subgraph containment. *International Journal of Quantum Information*, 10(03):1250019, 2012. arXiv:1109.4165. doi:10.1142/S0219749912500190.