Efficient algorithms in quantum query complexity

by

Robin Kothari

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

Waterloo, Ontario, Canada, 2014

Author's Declaration

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

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

Abstract

In this thesis we provide new upper and lower bounds on the quantum query complexity of a diverse set of problems. Specifically, we study quantum algorithms for Hamiltonian simulation, matrix multiplication, oracle identification, and graph-property recognition.

For the Hamiltonian simulation problem, we provide a quantum algorithm with query complexity sublogarithmic in the inverse error, an exponential improvement over previous methods. Our algorithm is based on a new quantum algorithm for implementing unitary matrices that can be written as linear combinations of efficiently implementable unitary gates. This algorithm uses a new form of "oblivious amplitude amplification" that can be applied even though the reflection about the input state is unavailable.

In the oracle identification problem, we are given oracle access to an unknown N-bit string x promised to belong to a known set of size M, and our task is to identify x. We present the first quantum algorithm for the problem that is optimal in its dependence on N and M. Our algorithm is based on ideas from classical learning theory and a new composition theorem for solutions of the filtered γ_2-norm semidefinite program.

We then study the quantum query complexity of matrix multiplication and related problems over rings, semirings, and the Boolean semiring in particular. Our main result is an output-sensitive algorithm for Boolean matrix multiplication that multiplies two $n \times n$ Boolean matrices with query complexity $O(n\sqrt{\ell})$, where ℓ is the sparsity of the output matrix. The algorithm is based on a reduction to the graph collision problem and a new algorithm for graph collision.

Finally, we study the quantum query complexity of minor-closed graph properties and show that most minor-closed properties—those that cannot be characterized by a finite set of forbidden subgraphs—have quantum query complexity $\Theta(n^{3/2})$ and those that do have such a characterization can be solved strictly faster, with $o(n^{3/2})$ queries. Our lower bound is based on a detailed analysis of the structure of minor-closed properties with respect to forbidden topological minors and forbidden subgraphs. Our algorithms are a novel application of the quantum walk search framework and give improved upper bounds for several subgraph-finding problems.

Acknowledgements

I would like to start by thanking Andrew Childs, who has been my supervisor since I started my master's degree. I thank him for being an outstanding supervisor, who worked closely with me when I needed his help, and let me work independently when I felt I could. At every point in time, he was the supervisor I needed right then. I would also like to thank John Watrous for being a great co-supervisor. He was always available to chat and share his ideas, advice, and wisdom.

Next I would like to thank my thesis committee members—Peter Høyer, Ashwin Nayak, Richard Cleve, John Watrous, and Andrew Childs—for reading my thesis and providing excellent feedback. I also thank Andrew, Ashwin, John, and Richard for broadening my knowledge of the field through the many quantum computing courses they taught.

I thank all my coauthors and collaborators for many enjoyable brainstorming sessions and fruitful discussions, my friends and colleagues at IQC and UWaterloo for many quantum and non-quantum discussions, and my friends in Waterloo and elsewhere for their support and friendship. Lastly, I would like to thank Alison for her love and support over the last couple of years, and my mother for a lifetime of love and support.

Dedication

This thesis is dedicated to my father.

Table of Contents

List of Figures

Chapter 1

Introduction

Many of the important questions of theoretical computer science concern the complexity of solving computational problems on powerful models of computation like Turing machines or Boolean circuits. These questions range from comparing the relative power of resources, such as whether untrusted certificates help polynomial-time Turing machines (the P vs NP problem) or whether exploiting quantum mechanics helps polynomial-time Turing machines (the BPP vs BQP problem), to understanding the complexity of specific problems, such as matrix multiplication, linear programming, integer factorization, graph isomorphism, etc.

Unfortunately, precisely because these models are so powerful, we are currently unable to prove good lower bounds for these models. For example, it is consistent with current knowledge that the NP-complete problem 3SAT can be solved in linear time. More surprisingly, it is also possible that all problems in NEXP, the class of problems that can be solved in exponential time on a nondeterministic Turing machine, can be computed by nonuniform polynomial-size Boolean circuits.

A natural course of action when faced with such difficulty is to work with an easier model of computation, following George Pólya's advice [Pól04, p. xxi]:

> "If you can't solve a problem, then there is an easier problem you can't solve: find it."

On the one hand, this model should be simple so we can prove good bounds, but on the other hand, it should be complex enough that we may ask similar questions about the power of certificates, quantum computing, etc.

The model of query complexity inhabits this Goldilocks zone, between models about which we can prove too little and models about which we can prove too much.

1.1 Query complexity

The primary object of study in this thesis is the model of query complexity, which is also called decision tree complexity, especially in the classical (i.e., not quantum) literature. (See [BdW02] for an excellent survey of query complexity and for formal definitions of the models we consider.)

Just like in the Turing machine model,[1] we often focus on the complexity of computing Boolean functions, although more general problems can be studied. Consider the problem of computing an n-bit Boolean function $f : \{0,1\}^n \to \{0,1\}$ on an input $x \in \{0,1\}^n$. Informally, query complexity studies following question: How many bits of the input x need to be read to determine $f(x)$?

Deterministic query complexity

We can formalize this question in the following way. We imagine that an algorithm wants to compute $f(x)$, but does not have direct access to the input x. Instead, it must query an oracle or black box to learn the bits of x. On being queried with an index $i \in [n]$, where $[n] := \{1, 2, \ldots, n\}$, the oracle responds with the value of x_i, the i^{th} bit of x. In between queries, the algorithm can perform arbitrary (deterministic) operations, which may depend on previously queried input bits. Finally, the algorithm must output a value in $\{0,1\}$. The algorithm is said to compute f if it outputs $f(x)$ on input x. The query complexity of an algorithm is the maximum number of queries made by it over all inputs x. The query complexity of a function f is the query complexity of the best algorithm (i.e., the one with minimum query complexity) that computes f.

Let $D(f)$ denote the deterministic query complexity of a function $f : \{0,1\}^n \to \{0,1\}$. Note that this is a number between 0 and n, since the input has only n bits.

Let us consider some examples. If f does not depend on x, i.e., f is a constant function, then $D(f) = 0$, and these are the only functions with $D(f) = 0$. On the other hand, consider the XOR function, defined as $\mathsf{XOR}_n(x) = \bigoplus_{i=1}^n x_i$, which computes the parity of all input bits. Any algorithm that makes fewer than n queries must fail on some input, since changing the value of any bit that has not been queried changes the output of the function and thus $D(\mathsf{XOR}_n) = n$. Another basic function is the OR function, defined as $\mathsf{OR}_n(x) = \bigvee_{i=1}^n x_i$. This function seems easier than the XOR function, since the output is not as sensitive to changing the value of an input bit. Indeed, if the algorithm has already queried a bit that is 1, the answer is now determined to be 1 and the other input bits are irrelevant. Despite this, the OR function is not easier in the worst case and $D(\mathsf{OR}_n) = n$. To see this, imagine a deterministic algorithm makes $n-1$ queries and sees all zeros in the $n-1$ bits it has queried. Now the algorithm is forced to output an answer based on this information, but the OR function depends crucially on the last bit that has not been queried, and thus the algorithm must answer incorrectly on some input.

[1] I will use Turing machines as the archetype of a well-motivated and powerful model to which I compare query complexity, but other models like Boolean circuits or RAM machines would work just as well.

Other query complexity models

Just like for Turing machines, we can study randomized, quantum, or nondeterministic variants of the model. The (bounded-error) randomized query complexity of a function f, denoted $R(f)$, is the query complexity of the best randomized algorithm that outputs $f(x)$ on input x with probability at least $2/3$. For example, $R(\mathsf{OR}_n) \leq \frac{2}{3}n$, as demonstrated by the algorithm that outputs the OR of $\frac{2}{3}n$ uniformly random bits of the input. If $\mathsf{OR}(x) = 0$, this algorithm always outputs 0, and if $\mathsf{OR}(x) = 1$, then with probability at least $2/3$ the algorithm will see a 1 in the $\frac{2}{3}n$ randomly chosen bits. However, the XOR function remains as hard and $R(\mathsf{XOR}_n) = n$. Any randomized algorithm that makes fewer than n queries can at best guess the answer with probability $1/2$, since the answer completely depends on the last bit that has not been seen.

Many of the important open problems of complexity theory have analogous questions in query complexity. Some of these questions can be answered and some remain open. For example, in the query complexity model $\mathsf{P} \neq \mathsf{NP}$ and $\mathsf{P} = \mathsf{BPP}$, which are widely believed to be true for Turing machines.[2] On the other hand, in query complexity $\mathsf{P} = \mathsf{NP} \cap \mathsf{coNP}$, which is believed to be false for Turing machines. Several simple questions also remain open: Can the randomized bounded-error (Monte Carlo) complexity ever be asymptotically smaller than the randomized zero-error (Las Vegas) complexity? What is best asymptotic separation between $D(f)$ and $R(f)$?

Further motivation

One reason to study query complexity is that it is a simple, but nontrivial model of computation that we can reason about: a stepping stone to understanding more complex models. But besides just being a simpler model, query complexity also shares many conceptual ideas and algorithmic techniques with Turing machine models, making it possible to import tools and techniques from one model to the other, from standard algorithmic techniques like divide-and-conquer, repeated squaring, success probability amplification by majority vote, random walks, etc. to more specialized techniques like color coding [AYZ95], which we use in Chapter 5, and deterministic coin tossing [CV86], used in several Hamiltonian simulation algorithms [BACS07, CK11b].

Another reason to study query complexity is to prove lower bounds on restricted classes of algorithms in the standard Turing machine model. For example, the well-known result that any comparison-based sorting algorithm must make $\Omega(n \log n)$ comparisons to sort n items fits right into the model of query complexity. Similarly, we can ask about the complexity of solving $\mathsf{3SAT}$ if the algorithm is constrained to use the formula as a black box, i.e., it is only allowed to evaluate it at inputs of its choice. When restricted to such algorithms, it can be shown that solving $\mathsf{3SAT}$ requires exponential time. This is also essentially a query complexity problem. This shows that any potential polynomial-time algorithm for $\mathsf{3SAT}$ must use the structure of the Boolean formula and not merely its input–output relationship.

[2] Complexity classes can be defined in the query model by thinking of $\log n$ as the input size. Thus $f \in \mathsf{P}$ if $D(f) = \mathrm{poly}(\log(n))$, $f \in \mathsf{BPP}$ if $R(f) = \mathrm{poly}(\log(n))$, etc. E.g., see [Juk12, Ch. 14], [AB09, Ch. 12], or [Ver98].

Quantum query complexity

Having motivated the study of query complexity in general, we now move on to quantum query complexity. Some of the great breakthroughs of quantum algorithms have been conceived in this model (e.g., Grover's algorithm [Gro96]). Shor's factoring algorithm [Sho97] also essentially solves a query problem exponentially faster than any classical algorithm. Query complexity is a model of computation where quantum computers are provably better than classical computers.

Let us define the quantum query model more formally. As before, the quantum algorithm has oracle access to an input string $x \in \{0,1\}^n$. Since the map $i \mapsto x_i$ is not unitary, quantum algorithms have access to a unitary version of this oracle, which acts as $Q_x|i,b\rangle = |i, b \oplus x_i\rangle$. A quantum algorithm with query complexity T is specified by $T+1$ unitaries $U_0, U_1, \ldots, U_T$ acting on $m \geq \log n + 1$ qubits. This quantum algorithm is said to perform the unitary $V_x = U_T Q_x U_{T-1} Q_x \ldots U_1 Q_x U_0$, where Q_x represents the tensor product of Q_x and the identity matrix when U_i acts on more qubits than Q_x. We say this quantum algorithm computes a function $f(x)$ with error ϵ if upon measuring the first qubit of $V_x|0^m\rangle$, the probability of obtaining $f(x)$ is at least $1-\epsilon$. The bounded-error quantum query complexity of f, denoted $Q(f)$, is the quantum query complexity of the best quantum algorithm that computes f with error $1/3$.

Many of the reasons to study classical query complexity carry over to quantum query complexity. For example, the search lower bound [BBBV97] shows that any potential polynomial-time quantum algorithm for 3SAT must use the structure of the Boolean formula and not merely its input–output relationship.

Quantum query complexity also possesses intuitive structural properties that are not known to hold even for randomized query complexity. For example, consider the direct sum property of general (not necessarily Boolean) functions. Let D and E be finite sets and consider a (not necessarily total) function f from $\mathcal{D} \subseteq D^n$ to E. In the examples so far, $D = E = \{0,1\}$ and $\mathcal{D} = \{0,1\}^n$. Let $f^k : \mathcal{D}^k \to E^k$ be the function that computes f on k independent inputs, i.e., $f^k(y^1, \ldots, y^k) = (f(y^1), \ldots, f(y^k))$, where $y^i \in \mathcal{D}$.

For deterministic query complexity, a perfect direct sum theorem holds, i.e., $D(f^k) = kD(f)$. In this case the upper bound is obvious since we can compute k independent copies of a function using an optimal algorithm for the function k times. The lower bound was shown by [JKS10]. On the other hand, for randomized query complexity the lower bound still holds up to constants [JKS10], i.e., $R(f^k) = \Omega(kR(f))$, but the upper bound is not known to hold. The obstacle is that a randomized algorithm that computes f with bounded error no longer computes k copies of f with bounded error. The probability that all outputs are correct may be much smaller than a constant. Using a standard error reduction argument, which boosts the success probability of a function by running it $O(\log k)$ times and taking the majority of the outcomes, we obtain an algorithm for f that has error probability $1/\operatorname{poly}(k)$, which can then be used to compute f^k with bounded error. However, this only shows the weaker result that $R(f^k) = O(kR(f)\log k)$. In fact, in certain settings it can be shown that this $\log k$ factor cannot be removed [FRPU94].

Surprisingly, a direct sum result holds for quantum query complexity, up to a multiplicative constant [LMR+11]. We utilize this theorem several times in this thesis to prove lower bounds and remove log factors from upper bounds.

Theorem 1.1 (Direct sum theorem)**.** *Let $f : \mathcal{D} \to E$ be a function and let $f^k : \mathcal{D}^k \to E^k$ denote the problem of computing f on k independent inputs. Then $Q(f^k) = \Theta(kQ(f))$.*

Another surprising property of quantum query complexity that holds for deterministic query complexity, but is not known to hold for randomized query complexity, is a Boolean function composition theorem. If $g : \mathcal{D} \to \{0,1\}$ and $f : \{0,1\}^n \to E$ are functions, where $\mathcal{D} \subseteq D^m$, let $f \circ g : \mathcal{D}^n \to E$ denote the composed function defined as

$$(f \circ g)(x) = f(g(x_1, \ldots, x_m), \ldots, g(x_{(n-1)m+1}, \ldots, x_{nm})), \tag{1.1}$$

where $x \in \mathcal{D}^n$ and $x_i \in D$ for all $i \in [nm]$. Then it is known that deterministic query complexity satisfies $D(f \circ g) = D(f)D(g)$ [Mon13]. Remarkably, the same result holds for quantum query complexity, up to a multiplicative constant [LMR+11].

Theorem 1.2 (Boolean function composition)**.** *If f and g are functions where the input of f is Boolean and the output of g is Boolean, then $Q(f \circ g) = \Theta(Q(f)Q(g))$.*

We will also use this theorem several times in this thesis to prove upper and lower bounds. As before, an analogous theorem is not known to hold for randomized query complexity.

These examples show that quantum query complexity behaves more like deterministic query complexity than randomized query complexity does, and some of our intuition from designing deterministic algorithms can be carried over to designing quantum algorithms. It seems from the perspective of query complexity, quantum mechanics is a more elegant description of our universe than just adding randomness to a deterministic description.

Measuring non-query resources

While we only study query complexity in this thesis, other non-query resources may also be studied in this model, such as time complexity, gate complexity, space complexity, etc. It is desirable to minimize these non-query resources if the aim is to implement these algorithms on a quantum computer. In this thesis, we will only discuss these measures in passing and so we do not define them formally. Informally, time complexity usually refers to the running time of the quantum algorithm when run on a model with quantum random access memory. Gate complexity refers to the total number of arbitrary 1- and 2-qubit gates needed to implement the non-query unitary operations, restricted to use only such gates. An algorithm is time efficient (or gate efficient) if its time complexity (or gate complexity) is close to its query complexity.

1.2 Overview

We study the quantum query complexity of a diverse set of problems in this thesis, proving new upper and lower bounds using several different techniques. The classical (i.e., deterministic or randomized) query complexities of problems considered in this thesis are usually uninteresting and easy to characterize.

Since this thesis studies the quantum query complexity of several different problems, it has been written keeping in mind that some readers may only wish to read about a particular problem or technique, without reading through the entire thesis. To accommodate such readers, the chapters are written so as to be independent of each other, unless they explicitly use new results from another chapter (e.g., Chapter 4 uses a new result from Chapter 3). Each chapter also begins with a short summary, to aid readers who have directly skipped to a particular chapter, and a citation to published work on which the chapter is based, if any. Consequently, the section on preliminaries (Section 1.3) only contains ideas that appear throughout the thesis and are not specific to any one problem.

I have also assumed that the reader has at least as much background in quantum computation as a first-year graduate student in computer science who has taken an introductory course in quantum computing. A great resource for attaining this background is the book by Nielsen and Chuang [NC00]. For example, I assume the reader is familiar with standard notation and terminology in quantum computing, Grover's algorithm, query complexity, basic ideas in classical (deterministic and randomized) algorithms, and elementary graph theory. Other resources for gaining familiarity with this material include the book by Kaye, Laflamme, and Mosca [KLM06] for quantum computing; the book by Jukna [Juk12] and the survey by Buhrman and de Wolf [BdW02] for query complexity; the book by Arora and Barak [AB09] for query complexity and randomized algorithms; the book by Manber [Man89] for deterministic algorithms; the book by Motwani and Raghavan [MR95] for randomized algorithms; and the book by Diestel [Die05] for graph theory.

Problems studied

We now briefly describe the problems studied in this thesis, highlighting the techniques used to solve them.

In Chapter 2 we study the sparse Hamiltonian simulation problem and the problem of simulating the continuous- and fractional-query models of query complexity. In the sparse Hamiltonian simulation problem we are given oracle access to the entries of a sparse Hamiltonian H and we wish to approximately implement the unitary operator e^{-iHt}, which represents time evolution due to H for time t. We describe a new algorithm that achieves optimal scaling in terms of the error parameter and almost optimal scaling in terms of the other parameters of interest. The

continuous- and fractional-query models are models of computation that are potentially more powerful than the standard model of quantum query complexity. We show that these models are not significantly more powerful by simulating them in the standard model with little overhead due to simulation.

Our algorithms follow from a new quantum algorithm for implementing unitary matrices that can be written as linear combinations of efficiently implementable unitary gates. This algorithm uses a new form of "oblivious amplitude amplification" that can be applied even though the reflection about the input state is unavailable. The lower bound constructs a Hamiltonian whose simulation yields an unbounded-error quantum algorithm for the parity function, which is as hard as computing parity with bounded error.

This chapter is partly based on the following paper:

[BCC+14] Dominic W. Berry, Andrew M. Childs, Richard Cleve, Robin Kothari, and Rolando D. Somma. Exponential improvement in precision for simulating sparse Hamiltonians. In *Proceedings of the 46th ACM Symposium on Theory of Computing (STOC 2014)*, pages 283–292, 2014.

In Chapter 3 we give an optimal quantum algorithm for the oracle identification problem. In the oracle identification problem, we are given oracle access to an unknown N-bit string x promised to belong to a known set of size M and our task is to identify x. We present a quantum algorithm for the problem that is optimal in its dependence on N and M. Our algorithm considerably simplifies and improves the previous best algorithm due to Ambainis et al. [AIK+07]. Our algorithm also has applications in quantum learning theory, where it improves the complexity of exact learning with quantum membership queries, resolving a conjecture of Hunziker et al. [HMP+10].

Our algorithm is based on ideas from classical learning theory and a new composition theorem for solutions of the filtered γ_2-norm semidefinite program, which characterizes quantum query complexity. Our algorithm also yields an improved upper bound for the problem of exact learning using quantum membership queries. Our composition theorem is quite general and allows us to compose quantum algorithms with input-dependent query complexities without incurring a logarithmic overhead for error reduction. We use the composition theorem in the next chapter to remove all log factors from the optimal quantum algorithm for Boolean matrix multiplication.

This chapter is based on the following paper:

[Kot14] Robin Kothari. An optimal quantum algorithm for the oracle identification problem. In *Proceedings of the 31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014)*, volume 25 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 482–493, 2014.

In Chapter 4 we study the quantum query complexity of matrix multiplication and related problems. We survey and extend known results on the quantum query complexity of matrix multiplication, matrix product verification, and related problems over rings, semirings and the Boolean semiring in particular. We also study relationships between these problems and other problems studied in quantum query complexity, such as the triangle finding problem and the graph collision problem.

Our main result is an output-sensitive quantum algorithm for Boolean matrix multiplication that multiplies two $n \times n$ Boolean matrices with query complexity $O(n\sqrt{\ell})$, where ℓ is the number of nonzero entries in the output matrix. Our algorithm is based on the observation that Boolean matrix multiplication can be reduced to several instances of the graph collision problem and that these instances of graph collision correspond to dense graphs. For these instances we develop a quantum algorithm that is more efficient than known algorithms for the general graph collision problem. We also prove a matching lower bound for Boolean matrix multiplication for all $\ell \leq \epsilon n^2$, for any constant $\epsilon < 1$.

Section 4.4 of this chapter contains results from the following paper:

[JKM12] Stacey Jeffery, Robin Kothari, and Frédéric Magniez. Improving Quantum Query Complexity of Boolean Matrix Multiplication Using Graph Collision. In *Automata, Languages, and Programming*, volume 7391 of *Lecture Notes in Computer Science*, pages 522–532. Springer, 2012.

In Chapter 5 we study the quantum query complexity of minor-closed graph properties of n-vertex graphs. We show that most minor-closed properties—those that cannot be characterized by a finite set of forbidden subgraphs—have quantum query complexity $\Theta(n^{3/2})$ and those that do have such a characterization can be solved strictly faster, with $o(n^{3/2})$ queries, where the input size is $\Theta(n^2)$.

Our algorithms are a novel application of the quantum walk search framework and give improved upper bounds for several subgraph-finding problems. Our lower bound is based on a detailed analysis of the structure of minor-closed properties with respect to forbidden topological minors and forbidden subgraphs.

This chapter is based on the following paper:

[CK12] Andrew M. Childs and Robin Kothari. Quantum query complexity of minor-closed graph properties. *SIAM Journal on Computing*, 41(6):1426–1450, 2012.

The thesis ends with concluding remarks in Chapter 6.

1.3 Preliminaries and notation

In this thesis we will occasionally describe oracles only by their classical behavior and assume that quantum algorithms have access to the standard unitary version of the oracle. For example, when the oracle holds a bit string x, classical algorithms are given access to an oracle that outputs x_i on input i, while quantum algorithms have access to a unitary that maps $|i, b\rangle$ to $|i, b \oplus x_i\rangle$ for $b \in \{0, 1\}$. When the oracle holds a string x where each $x_i \in \{0, 1, \ldots, m-1\}$, we have access to a unitary that maps $|i, b\rangle$ to $|i, (b+x_i) \bmod m\rangle$ for $b \in \{0, 1, \ldots, m-1\}$. If the oracle is supposed to contain real or complex numbers, we assume the oracle contains finite precision approximations.

We will often implicitly use the fact that if we want to perform a product of unitaries $U_m U_{m-1} \ldots U_2 U_1$, but instead perform unitaries that are ϵ-close to these, then the total error caused by this is at most ϵm. This is justified by the following theorem [NC00, eq. (4.63)].

Theorem 1.3 (Subadditivity of error in implementing unitaries)**.** *If U_i and V_i are unitary matrices for $i \in [m]$, then*

$$\|U_m U_{m-1} \ldots U_2 U_1 - V_m V_{m-1} \ldots V_2 V_1\| \leq \sum_{i=1}^{m} \|U_i - V_i\|. \tag{1.2}$$

Throughout the thesis, when proving asymptotic statements we will assume that large integers are powers of 2, unless this assumption is not valid for the situation under consideration. For example, if we are trying to show that $Q(f) = O(\sqrt{n})$ and it is clear that the quantum query complexity of f is monotonically increasing in n, it is sufficient to prove this claim for all n that are powers of 2.

The function $\log n$ denotes the base-2 logarithm of n. For any positive integer n, we define $[n] := \{1, 2, \ldots, n\}$. We use the notation $\tilde{O}$ to indicate that we are suppressing polylog factors. More precisely, $f(n) = \tilde{O}(g(n))$ means $f(n) = \mathrm{O}(g(n) \log^k n)$ for some constant k.

We will use $\|A\|$ to denote the spectral norm of an operator A. If unspecified, we quantify the distance between operators U and V with the function $\|U - V\|$ and the distance between states $|\psi\rangle$ and $|\phi\rangle$ with the function $\||\psi\rangle - |\phi\rangle\|$.

Chapter 2

Hamiltonian simulation and continuous queries

Chapter summary: We provide a quantum algorithm for simulating the dynamics of sparse Hamiltonians with complexity sublogarithmic in the inverse error, an exponential improvement over previous methods. Specifically, we show that a d-sparse Hamiltonian H can be simulated for time t with precision ϵ using $O\left(\tau \frac{\log(\tau/\epsilon)}{\log\log(\tau/\epsilon)}\right)$ queries, where $\tau = d^2\|H\|_{\max}t$. Unlike previous approaches based on product formulas, the query complexity is independent of the number of qubits acted on. We also show that our algorithm is optimal as a function of the error.

Our algorithm is based on a new quantum algorithm for implementing a linear combination of unitary matrices and a new form of "oblivious amplitude amplification" that can be applied even though the reflection about the input state is unavailable. Using this algorithm we also significantly improve the simulation of the continuous- and fractional-query models using discrete quantum queries, showing that the former models are not much more powerful than the discrete model even for very small error.

This chapter is based on the following paper:

[BCC+14] Dominic W. Berry, Andrew M. Childs, Richard Cleve, Robin Kothari, and Rolando D. Somma. Exponential improvement in precision for simulating sparse Hamiltonians. In *Proceedings of the 46th ACM Symposium on Theory of Computing (STOC 2014)*, pages 283–292, 2014.

2.1 Introduction

In this chapter we study two a priori unrelated problems: the sparse Hamiltonian simulation problem and the problem of simulating the continuous- or fractional-query model of quantum query complexity in the standard (discrete-query) model.

We solve both problems using a new quantum algorithm for implementing unitary matrices that can be expressed as a linear combination of efficiently implementable unitary gates. We call this the Linear Combination of Unitaries (LCU) algorithm. Although this chapter is based on [BCC+14] and our algorithms are conceptually similar to those in [BCC+14], our approach is different as we solve both problems using the LCU algorithm. Let us start by introducing the two problems and our techniques.

2.1.1 Hamiltonian simulation

Motivation

The simulation of quantum systems is a major potential application of quantum computers. Indeed, the problem of simulating Hamiltonian dynamics was the original motivation for the idea of quantum computation [Fey82]. The first quantum algorithm for simulating Hamiltonians was due to Lloyd, who provided an efficient algorithm for simulating local Hamiltonians [Llo96]. A local Hamiltonian is a sum of terms that act nontrivially on a constant number of qubits. Here we consider the more general problem of simulating sparse Hamiltonians, a generalization of local Hamiltonians. This problem naturally fits into the query complexity model and has been widely studied [AT03, Chi04, BACS07]. Sparse Hamiltonian simulation has also proved to be a useful subroutine in the design of new quantum algorithms [CCD+03, CCJY09, HHL09].

Problem description

In the Hamiltonian simulation problem, we are given a Hamiltonian H (a $2^n \times 2^n$ Hermitian matrix) acting on n qubits, a time $t > 0$, and an error parameter $\epsilon > 0$. Our task is to construct a circuit that performs the unitary operation e^{-iHt} with error at most ϵ. For a quantum system with Hamiltonian H, the state of the system at time t, $|\psi(t)\rangle$, satisfies $|\psi(t)\rangle = e^{-iHt}|\psi(0)\rangle$.

More formally, we consider the problem of simulating a d-sparse Hamiltonian, a Hamiltonian with at most d nonzero entries in any row or column, in the model of query complexity where we are given access to the d-sparse Hamiltonian H via a black box that accepts a row index $i \in [2^n]$ and a number $j \in [d]$, and returns the position and value of the j^{th} nonzero entry of H in the i^{th} row. More precisely since the entries are complex numbers, the oracle provides finite precision approximations of the real and imaginary parts. Our task is to construct a circuit U, such that $\|U - e^{-iHt}\| \leq \epsilon$, using as few queries to the black box as possible.

History

The first efficient algorithm for sparse Hamiltonian simulation was due to Aharonov and Ta-Shma [AT03]. The key idea (also applied in [CCD+03]) is to use edge coloring to decompose the Hamiltonian H into a sum of Hamiltonians $\sum_{j=1}^{m} H_j$, where each H_j is easy to simulate. These terms are then recombined using the Lie product formula, which states that

$$e^{-iHt} \approx \left(e^{-iH_1 t/r} e^{-iH_2 t/r} \cdots e^{-iH_m t/r}\right)^r \tag{2.1}$$

for large r. This method gives query complexity $\mathrm{poly}(n,d)(\|H\|t)^2/\epsilon$, where $\|\cdot\|$ denotes the spectral norm. This was later improved using high-order product formulas and more efficient decompositions of the Hamiltonian [Chi04, BACS07, CK11b]. The best algorithm of this type [CK11b] has query complexity $O(d^2(d+\log^* n)\|H\|t(d\|H\|t/\epsilon)^\delta)$, where $\delta > 0$ can be arbitrarily small.[1] This complexity is only slightly superlinear in t and increases as an arbitrarily small power of $1/\epsilon$. An alternative Hamiltonian simulation method based on a quantum walk [Chi10, BC12] is incomparable to these methods. That method has query complexity $O(d\|H\|_{\max} t/\sqrt{\epsilon})$, where $\|H\|_{\max}$ denotes the largest entry of H in absolute value, so its performance is better in terms of $\|H\|_{\max} t$ and d but significantly worse in terms of ϵ.

We note that all these algorithms have exponential or nearly exponential dependence on $\log(1/\epsilon)$, the number of bits needed to specify ϵ.

Our results

In this chapter, we present a dramatically improved algorithm that shows the following.

Theorem 2.1 (Sparse Hamiltonian simulation)**.** *A d-sparse Hamiltonian H can be simulated for time t with error at most ϵ using $O\left(\tau \frac{\log(\tau/\epsilon)}{\log\log(\tau/\epsilon)}\right)$ queries, where $\tau := d^2\|H\|_{\max} t \geq 1$.*

Our algorithm has exponentially improved dependence on $1/\epsilon$ compared to all previous algorithms. Our algorithm strictly improves upon all previous approaches based on product formulas (e.g., [Llo96, AT03, Chi04, BACS07, CK11b]), as our algorithm has no dependence on n, and improved dependence on d and t. The Hamiltonian simulation method based on a quantum walk [Chi10, BC12] is incomparable as its performance is better in terms of $\|H\|_{\max} t$ and d but significantly worse in terms of ϵ. Thus, while suboptimal for (say) constant-precision simulation, the results of Theorem 2.1 currently give the best known Hamiltonian simulations as a function of ϵ.

The performance of our algorithm is optimal or nearly optimal as a function of some of its parameters. A lower bound of $\Omega(\|H\|_{\max} t)$ follows from the no-fast-forwarding theorem of [BACS07], showing that our algorithm's dependence on $\|H\|_{\max} t$ is almost optimal. However,

[1] Here $\log^*$ denotes the iterated logarithm function, the number of times the logarithm function must be applied to obtain a result at most 1, i.e., $\log^* n = 0$ if $n \leq 1$ and $\log^* n = 1 + \log^*(\log n)$ if $n > 1$.

prior to our work, there was no known ϵ-dependent lower bound, not even one ruling out algorithms with no dependence on ϵ. We show that, surprisingly, our dependence on ϵ in Theorem 2.1 is optimal.

Theorem 2.2 (ϵ-dependent lower bound for Hamiltonian simulation)**.** *For any $\epsilon > 0$, there exists a 2-sparse Hamiltonian H with $\|H\|_{\max} < 1$ such that simulating H with precision ϵ for constant time requires $\Omega\left(\frac{\log(1/\epsilon)}{\log\log(1/\epsilon)}\right)$ queries.*

2.1.2 Continuous-query model

Problem description

To introduce the notion of continuous queries, recall that in the usual model of quantum query complexity, we wish to solve a problem whose input $x \in \{0,1\}^N$ is given by an oracle (or black box) that can be queried to learn the bits of x. The measure of complexity, called the query complexity, is the number of times we query the oracle. More precisely, we are given access to a unitary gate Q_x whose action on the basis states $|j\rangle|b\rangle$ is

$$Q_x|j\rangle|b\rangle = (-1)^{bx_j}|j\rangle|b\rangle \tag{2.2}$$

for all $j \in [N] := \{1, 2, \ldots, N\}$ and $b \in \{0,1\}$.[2] A quantum query algorithm is a quantum circuit consisting of arbitrary x-independent unitaries and Q_x gates. The query complexity of such an algorithm is the total number of Q_x gates used in the circuit.

The query model is often used to study the complexity of evaluating a classical function of x. However, it is also natural to consider more general tasks. In order of increasing generality, such tasks include state generation [AMRR11], state conversion [LMR+11], and implementing unitary operations [BC12]. Here we focus on the last of these tasks, where for each possible input x we must perform some unitary operation U_x. Considering this task leads to a strong notion of simulation: to simulate a given algorithm in the sense of unitary implementation, one must reproduce the entire correct output state for every possible input state, rather than simply (say) evaluating some predicate in one bit of the output with a fixed input state.

Since quantum mechanics is fundamentally described by the continuous dynamics of the Schrödinger equation, it is natural to ask if the query model can be made less discrete. In particular, instead of using the gate Q_x for unit cost, what if we can make half a query for half the cost? This perspective is motivated by the idea that if Q_x is performed by a Hamiltonian running for unit time, we can stop the evolution after half the time to obtain half a query. In general we could run this Hamiltonian for time $\alpha \in (0,1]$ at cost α. This fractional-query model is at least as powerful as the standard (discrete-query) model. More formally, we define the model as follows.

[2] This unitary is equivalent to one that maps $|j,b\rangle$ to $|j, b \oplus x_j\rangle$. They are related by a Hadamard transform performed on the second register.

Definition 2.1 (Fractional-query model)**.** For $x \in \{0,1\}^N$ and $\alpha \in [0,1]$, let Q_x^α be the operator that acts as

$$Q_x^\alpha |j\rangle|b\rangle = e^{-i\pi\alpha b x_j}|j\rangle|b\rangle \tag{2.3}$$

for all $j \in [N]$ and $b \in \{0,1\}$. An algorithm in the fractional-query model is a sequence of unitary gates $U_m Q^{\alpha_m} U_{m-1} \cdots U_1 Q^{\alpha_1} U_0$, where U_l are arbitrary unitaries and $\alpha_l \in (0,1]$ for all $l \in [m]$. The fractional-query complexity of this algorithm is $\sum_{l=1}^m \alpha_l$ and the total number of fractional-query gates used is m.

This idea can be taken further by taking the limit as the sizes of the fractional queries approach zero to obtain a continuous variant of the model, called the continuous-query model [FG98]. In this model, we have access to a query Hamiltonian H_x acting as $H_x|j\rangle|b\rangle = \pi b x_j |j\rangle|b\rangle$. Unlike the fractional- and discrete-query models, this is not a circuit-based model of computation. In this model we are allowed to evolve for time T according to the Hamiltonian given by $H_x + H_D(t)$ for an arbitrary time-dependent driving Hamiltonian $H_D(t)$, at cost T.

Definition 2.2 (Continuous-query model)**.** Let H_x act as $H_x|j\rangle|b\rangle = \pi b x_j |j\rangle|b\rangle$ for all $j \in [N]$ and $b \in \{0,1\}$. An algorithm in the continuous-query model is specified by an arbitrary x-independent driving Hamiltonian $H_D(t)$ for $t \in [0,T]$. The algorithm implements the unitary operation $U(T)$ obtained by solving the Schrödinger equation

$$i\frac{\mathrm{d}}{\mathrm{d}t}U(t) = \big(H_x + H_D(t)\big)U(t) \tag{2.4}$$

with $U(0) = \mathbb{1}$. The continuous-query complexity of this algorithm is T, the total evolution time.

Because $e^{-i\alpha H_x} = Q_x^\alpha$, running the Hamiltonian H_x with no driving Hamiltonian for time $T = \alpha$ is equivalent to an α-fractional query.

History

While initial work on the continuous-query model focused on finding analogues of known algorithms [FG98, Moc07], it has also been studied with the aim of proving lower bounds on the discrete-query model [Moc07]. Furthermore, the model has led to the discovery of new quantum algorithms. In particular, Farhi, Goldstone, and Gutmann [FGG08] discovered an algorithm with continuous-query complexity $O(\sqrt{n})$ for evaluating a balanced binary NAND tree with n leaves, which is optimal. This result was later converted to the discrete-query model with the same query complexity [CCJY09, ACR+10].

A similar conversion can be performed for any algorithm with a sufficiently well-behaved driving Hamiltonian [Chi10]. However, this leaves open the question of whether continuous-query algorithms can be generically converted to discrete-query algorithms with the same query

complexity. This was almost resolved by Cleve, Gottesman, Mosca, Somma, and Yonge-Mallo [CGM+09], who gave an algorithm that approximates a T-query continuous-query algorithm to bounded error with $O\left(T\frac{\log T}{\log\log T}\right)$ discrete queries. However, to approximate a continuous-query algorithm to precision ϵ, the algorithm of [CGM+09] uses $O\left(\frac{1}{\epsilon}\frac{T\log T}{\log\log T}\right)$ queries. Ideally we would like the dependence on ϵ to be polylogarithmic, instead of polynomial, in $1/\epsilon$. For example, such behavior would be desirable when using a fractional-query algorithm as a subroutine.

For the problem of evaluating a classical function of a black-box input, an approach based on the general adversary bound and the filtered γ_2 norm shows that the continuous-query complexity is at most a constant factor smaller than the discrete-query complexity for a bounded-error simulation [LMR+11]. However, it remains unclear whether the unitary implemented by a continuous-query algorithm can be simulated (even with bounded error) using $O(T)$ queries. Such a result does hold for state conversion, but its dependence on error is quadratic [LMR+11].

Our results

We present a significantly improved and dramatically simplified simulation of the continuous- and fractional-query models. In Section 2.4, we show the following.

Theorem 2.3 (Continuous-query simulation)**.** *An algorithm with continuous- or fractional-query complexity $T \geq 1$ can be simulated with error at most ϵ with $O\left(T\frac{\log(T/\epsilon)}{\log\log(T/\epsilon)}\right)$ queries.*

Since the continuous-query model is at least as powerful as the discrete-query model, a discrete simulation must use $\Omega(T)$ queries, showing our dependence on T is close to optimal. As in Hamiltonian simulation, a lower bound of $\Omega\left(\frac{\log(1/\epsilon)}{\log\log(1/\epsilon)}\right)$ queries for a continuous-query algorithm with $T = O(1)$ can be shown [BCC+14], so the dependence of our simulation on ϵ is optimal.

2.1.3 High-level overview of techniques

Both the upper bounds in this chapter (Theorem 2.1 and Theorem 2.3) are based on the algorithm for implementing linear combinations of unitary matrices, which we abbreviate to LCU algorithm.

LCU algorithm

The LCU algorithm is a new quantum algorithm for implementing any unitary V that can be written as a linear combination of efficiently implementable unitary matrices U_i, i.e., $V = \sum_i a_i U_i$. If each U_i requires at most q queries to implement, then the LCU algorithm allows us to implement V perfectly on a quantum computer using $O(aq)$ queries, where $a := \|\vec{a}\|_1 = \sum_i |a_i|$ and $\vec{a} := (a_1, a_2, \ldots)$. The algorithm also works when only an approximation to V is known in terms of a linear combination of unitaries. This is described in Section 2.2.

Oblivious amplitude amplification

The LCU algorithm works by first implementing a probabilistic version of V, which is a unitary W that implements V coherently with (say) constant probability. From such a map W we show how to construct an algorithm for V using only a constant number of uses of W. This step requires us to perform amplitude amplification in a situation where we do not know how to reflect about the input state, a primitive needed in standard amplitude amplification. Even though this reflection is not available, we show that a different reflection suffices to reproduce the behavior of amplitude amplification. We call this technique oblivious amplitude amplification, as it is a form of amplitude amplification that works without knowledge of the input state. We show this in Section 2.2.

Hamiltonian simulation algorithm

The reduction from Hamiltonian simulation to implementing a linear combination of unitaries is conceptually easy. Suppose our Hamiltonian was itself a linear combination of unitaries, i.e., $H = \sum_j c_j U_j$, where U_j is unitary for all j. Then the map we wish to implement, e^{-iHt}, is already a linear combination of unitaries:

$$e^{-iHt} = \sum_{l=0}^{\infty} \frac{1}{l!}(-iHt)^l = \sum_{l=0}^{\infty} \frac{1}{l!}\Big(-i \sum_j c_j U_j t\Big)^l. \tag{2.5}$$

When expanded out, each term is a scalar multiplied by a product of unitary operators. Although this is an infinite sum, a good approximation can be obtained by truncating the sum.

This shows that the LCU algorithm can handle the situation where H is itself a linear combination of unitaries. However, what we have is an arbitrary d-sparse H, and not one that is the linear combination of unitaries. This is resolved in two stages: First, we decompose H into a sum of 1-sparse Hamiltonians, a popular strategy in Hamiltonian simulation algorithms. After this, we show that a 1-sparse Hamiltonian can be decomposed into a linear combination of unitaries. We describe this in more detail in Section 2.3.

Hamiltonian simulation lower bound

The main idea behind the lower bound is the construction of a Hamiltonian whose exact simulation for any time $t > 0$ allows us to compute the parity of an n-bit string with unbounded error, which is as hard as computing parity exactly, requiring $\Omega(n)$ queries [BBC+01, FGGS98]. Therefore, exactly simulating a sparse Hamiltonian for any time $t > 0$ can be made arbitrarily hard, independent of t. Since we want to prove a lower bound for a simulation with error, we must have an accurate enough simulation to allow this reduction to work. If the simulation error is smaller than the bias of our unbounded-error algorithm, then the unbounded-error algorithm will still compute parity and will therefore require $\Omega(n)$ queries. We show this result in Section 2.3.

Continuous-query simulation algorithm

We start by showing that the continuous-query model and fractional-query models are equivalent. Thus we can instead simulate the fractional-query model, which is almost already in a form that can be simulated by the LCU algorithm. Consider the unitary V performed by a fractional-query algorithm with fractional-query complexity at most 1. V has the following form for some $m \in \mathbb{N}$:

$$V = W_m Q^{\alpha_m} W_{m-1} Q^{\alpha_{m-1}} \cdots W_1 Q^{\alpha_1} W_0, \tag{2.6}$$

where for all $i \in [m]$, W_i are arbitrary unitaries, $\alpha_i \in (0, 1]$, and $\sum_i \alpha_i \leq 1$. While it may not be obvious from the definition of Q^α, Q^α can be expressed as a linear combination of $\mathbb{1}$ and Q (see (2.67)). Thus V can be written as a linear combination of unitaries. However, in some of these terms Q appears a large number of times, and these terms would be costly to implement. But these terms also have low weight in the superposition and we show that deleting all such expensive terms does not affect the unitary V much. We show this in Section 2.4.

2.2 Linear Combination of Unitaries (LCU) algorithm

We begin by introducing the LCU algorithm, a new quantum algorithm for implementing any unitary that can be expressed as a linear combination of efficiently implementable unitary gates.

Suppose we wish to implement a unitary V that can be written as a linear combination of unitary gates U_i, i.e., $V = \sum a_i U_i$, where the unitaries U_i are considered easy to perform in the model under consideration (e.g., query complexity or gate complexity). Formally we mean the set of unitaries U_i can be efficiently implemented in superposition in the sense that the map $U := \sum_i |i\rangle\langle i| \otimes U_i$ is efficiently implementable. The unitary U maps $|i\rangle|\psi\rangle$ to $|i\rangle U_i|\psi\rangle$. In the query complexity model, it is easy to show that if every U_i can be implemented with at most q queries then U can also be implemented with q queries. In other models this may not be the case, and hence we state our results in terms of the number of uses of U.

We will also assume that the coefficients a_i are real and positive without loss of generality, since the phase of a_i can be absorbed into U_i.

In addition to the map U, we will need a unitary A that maps the state $|0^m\rangle$ to $\frac{1}{\sqrt{a}} \sum_i \sqrt{a_i}|i\rangle$, where $a := \|\vec{a}\|_1 = \sum_i |a_i|$. In our applications, the matrices U_i will depend on the oracle and therefore cost queries to perform, but the values of a_i are known and thus implementing A costs no queries. However, we state our results in terms of number of uses of U and A for generality.

We first show an exact version of the LCU algorithm, which is a procedure that exactly implements V using the maps U and A (and their inverses) $O(a)$ times. We then extend our algorithm to work in the case where V can only be approximated by a linear combination of unitary gates. In this case the LCU algorithm only approximately performs V.

This section is structured as follows. In Section 2.2.1 we define what we call (for some $p \geq 0$) a p-implementation, a kind of probabilistic implementation of an operator (see Definition 2.3). We then show how any operator that is expressible as a linear combination of unitaries can be p-implemented. Using oblivious amplitude amplification, described in Section 2.2.2, we show in Section 2.2.3 that a p-implementation W of a unitary operator V can be converted to a perfect implementation of V using $O(1/\sqrt{p})$ uses of W. In Section 2.2.4 we prove a more robust version of oblivious amplitude amplification to handle the case where V is only close to a unitary. In Section 2.2.5, we show how to implement V approximately using $O(1/\sqrt{p})$ uses of W, even when V is only close to a unitary.

Finally in Section 2.2.6 we state a useful corollary of the results of this section, which will be used in a black box manner for the rest of this chapter. Readers may skip to Section 2.2.6 if they only wish to use the LCU algorithm as a black box.

2.2.1 A p-implementation of any linear combination of unitaries

We start by defining what it means to p-implement an operator V.

Definition 2.3 (p-implementation)**.** Let V be an operator acting on n qubits and $p \geq 0$. We say a unitary W acting on $m+n$ qubits *p-implements* V if for all n-qubit states $|\psi\rangle$,

$$W|0^m\rangle|\psi\rangle = \sqrt{p}|0^m\rangle V|\psi\rangle + |\Phi^\perp\rangle, \tag{2.7}$$

where the unnormalized state $|\Phi^\perp\rangle$ satisfies $(|0^m\rangle\langle 0^m| \otimes \mathbb{1})|\Phi^\perp\rangle = 0$.

In this thesis we only use this informally to build intuition and do not study it in detail. A p-implementation W of a (possibly nonunitary) map V is a probabilistic implementation of V in the sense that after performing W we can measure whether the first register is $|0^m\rangle$ and if so, we have exactly performed V. The probability of obtaining this result is $p\|V|\psi\rangle\|^2$. When V is unitary, this probability is p. This notion is more intuitive when V is unitary, since in this case p cannot be larger than 1 and p measures how well W implements V. In particular $p = 1$ corresponds to a perfect implementation of V. On the other hand, when V is nonunitary, p can be larger than 1.

Such implementations have been studied before in several contexts, often paired with some technique for recovering the original input state in case we obtain the unfavorable measurement outcome. In the quantum circuit synthesis community, this is known as the Repeat-Until-Success paradigm (see [PS13] and the references therein).

This problem also arose in the previous best simulation of the fractional- and continuous-query models using discrete queries [CGM+09, BCG14]. In these approaches they p-implement a unitary V for some constant p and show how to recover the original input state from $|\Phi^\perp\rangle$.

However, this recovery procedure is itself a probabilistic procedure, i.e., it is only known how to p-implement the recovery map. Thus with some probability the recovery fails again, and now two recovery operations are needed, and so on. Thus this recovery stage required a complicated fault-correction procedure that is difficult to analyze and considerably harder to make efficient in terms of gate complexity. Indeed, the original algorithm of [CGM+09] was not gate efficient and was made gate efficient by a later paper [BCG14].

Let us now show that any (not necessarily unitary) operator V that can be expressed as a linear combination of unitaries can be p-implemented by a simple quantum circuit.

Lemma 2.1 (p-implementation of V)**.** *Let $V = \sum_i a_i U_i$ be a linear combination of unitary matrices U_i with $a_i > 0$ for all i. Let A be a unitary matrix that maps $|0^m\rangle$ to $\frac{1}{\sqrt{a}}\sum_i \sqrt{a_i}|i\rangle$, where $a := \|\vec{a}\|_1 = \sum_i a_i$. Let $U := \sum_i |i\rangle\langle i| \otimes U_i$ and $p := 1/a^2$. Then $W := A^\dagger U A$ satisfies*

$$W|0^m\rangle|\psi\rangle = \sqrt{p}|0^m\rangle V|\psi\rangle + |\Phi^\perp\rangle \tag{2.8}$$

for all states $|\psi\rangle$, where the unnormalized state $|\Phi^\perp\rangle$, which depends on $|\psi\rangle$, satisfies $\Pi|\Phi^\perp\rangle = 0$, where $\Pi = |0^m\rangle\langle 0^m| \otimes \mathbb{1}$.

Proof. This can be shown by straightforward computation.

$$W|0^m\rangle|\psi\rangle = A^\dagger U A|0^m\rangle|\psi\rangle \tag{2.9}$$

$$= A^\dagger U \left(\frac{1}{\sqrt{a}}\sum_i \sqrt{a_i}|i\rangle|\psi\rangle\right) = A^\dagger \left(\frac{1}{\sqrt{a}}\sum_i \sqrt{a_i}|i\rangle U_i|\psi\rangle\right) \tag{2.10}$$

$$= \Pi A^\dagger \left(\frac{1}{\sqrt{a}}\sum_i \sqrt{a_i}|i\rangle U_i|\psi\rangle\right) + (\mathbb{1}-\Pi)A^\dagger \left(\frac{1}{\sqrt{a}}\sum_i \sqrt{a_i}|i\rangle U_i|\psi\rangle\right) \tag{2.11}$$

$$= (|0^m\rangle \otimes \mathbb{1}) \left(\frac{1}{\sqrt{a}}\sum_i \sqrt{a_i}\langle i| \otimes \mathbb{1}\right) \left(\frac{1}{\sqrt{a}}\sum_i \sqrt{a_i}|i\rangle U_i|\psi\rangle\right) + |\Phi^\perp\rangle \tag{2.12}$$

$$= \frac{1}{a}|0^m\rangle V|\psi\rangle + |\Phi^\perp\rangle = \sqrt{p}|0^m\rangle V|\psi\rangle + |\Phi^\perp\rangle, \tag{2.13}$$

where $|\Phi^\perp\rangle := (\mathbb{1}-\Pi)A^\dagger\left(\frac{1}{\sqrt{a}}\sum_i \sqrt{a_i}|i\rangle U_i|\psi\rangle\right)$ satisfies $\Pi|\Phi^\perp\rangle = 0$ because $\Pi(\mathbb{1}-\Pi) = 0$. □

Given a p-implementation of the operator we wish to implement, previous approaches measured the first register and attempted to recover the input state from $|\Phi^\perp\rangle$ in case of the unfavorable outcome. This is often difficult to analyze and requires understanding the set of states $|\Phi^\perp\rangle$ for every input state $|\psi\rangle$. We avoid these difficulties by introducing a general tool we call oblivious amplitude amplification.

2.2.2 Oblivious amplitude amplification

Given a unitary W that p-implements a unitary V, oblivious amplitude amplification uses a version of amplitude amplification and provides a p'-implementation of V, where $p' > p$. In particular, as in amplitude amplification, if p is known, we can exactly perform V.

This technique applies to any p-implementation of a unitary and can therefore be applied in any context in which a p-implementation is available. Oblivious amplitude amplification has already found use in quantum circuit synthesis [PS13, WR14]. It can also be used to improve the performance of the previous best algorithms for simulating the continuous- and fractional-query models as shown in [BCC+14]. In this chapter we match the performance of this improved algorithm, although we use the LCU algorithm instead.

In standard amplitude amplification, to amplify the "good" part of a state, we need to be able to reflect about the state itself and the projector onto the good subspace. While the latter is easy in our application, we cannot reflect about the unknown input state. Nevertheless, we show the following.

Lemma 2.2 (Oblivious amplitude amplification)**.** *Let W and V be unitary matrices on $n+m$ qubits and n qubits, respectively, and let $\theta \in (0, \pi/2)$. Suppose that for any n-qubit state $|\psi\rangle$,*

$$W|0^m\rangle|\psi\rangle = \sin(\theta)|0^m\rangle V|\psi\rangle + \cos(\theta)|\Phi^\perp\rangle, \tag{2.14}$$

where $|\Phi^\perp\rangle$ is an $(n+m)$-qubit state that depends on $|\psi\rangle$ and satisfies $\Pi|\Phi^\perp\rangle = 0$, where $\Pi = |0^m\rangle\langle 0^m| \otimes \mathbb{1}$. Let $R := 2\Pi - \mathbb{1}$ and $S := -WRW^\dagger R$. Then for any $t \in \mathbb{Z}$,

$$S^t W|0^m\rangle|\psi\rangle = \sin\big((2t+1)\theta\big)|0^m\rangle V|\psi\rangle + \cos\big((2t+1)\theta\big)|\Phi^\perp\rangle. \tag{2.15}$$

Note that $WRW^\dagger$ is not the reflection about the initial state, so Lemma 2.2 does not follow from amplitude amplification alone. However, in the context described in the lemma, it suffices to use a different reflection.

The motivation for oblivious amplitude amplification comes from work of Marriott and Watrous on in-place amplification of QMA [MW05] (see also related work using amplitude amplification to obtain a quadratic improvement [NWZ09]). It is also related to Watrous' Quantum Rewinding Lemma [Wat09], but our situation is slightly different and also we use amplitude amplification, which gives us a perfect implementation of V when p is known.

Specifically, Lemma 2.2 follows from the following technical lemma, which shows that during amplitude amplification the state remains in a certain 2-dimensional subspace in which it is possible to perform the appropriate reflections.

Lemma 2.3 (2D Subspace Lemma)**.** *Let W and V be unitary matrices on $n+m$ qubits and n qubits, respectively, and let $p \in (0,1)$. Suppose that for any n-qubit state $|\psi\rangle$,*

$$W|0^m\rangle|\psi\rangle = \sqrt{p}|0^m\rangle V|\psi\rangle + \sqrt{1-p}|\Phi^\perp\rangle, \tag{2.16}$$

where $|\Phi^\perp\rangle$ is an $(n+m)$-qubit state that depends on $|\psi\rangle$ and satisfies $\Pi|\Phi^\perp\rangle = 0$, where $\Pi = |0^m\rangle\langle 0^m| \otimes \mathbb{1}$. Then the state $|\Psi^\perp\rangle$, defined by the equation

$$W|\Psi^\perp\rangle := \sqrt{1-p}|0^m\rangle V|\psi\rangle - \sqrt{p}|\Phi^\perp\rangle \tag{2.17}$$

satisfies $\Pi|\Psi^\perp\rangle = 0$.

Proof. For any n-qubit state $|\psi\rangle$, let $|\Psi\rangle := |0^m\rangle|\psi\rangle$ and $|\Phi\rangle := |0^m\rangle V|\psi\rangle$. Then the following equations hold for all $|\psi\rangle$.

$$W|\Psi\rangle = \sqrt{p}|\Phi\rangle + \sqrt{1-p}|\Phi^\perp\rangle \tag{2.18}$$

$$W|\Psi^\perp\rangle = \sqrt{1-p}|\Phi\rangle - \sqrt{p}|\Phi^\perp\rangle, \tag{2.19}$$

where $\Pi|\Phi^\perp\rangle = 0$. It follows from these equations that $\langle\Psi|\Psi^\perp\rangle = 0$. The lemma asserts that not only is $|\Psi^\perp\rangle$ orthogonal to $|\Psi\rangle$, but also $\Pi|\Psi^\perp\rangle = 0$.

To show this, consider the operator

$$Q := (\langle 0^m| \otimes \mathbb{1})W^\dagger \Pi W(|0^m\rangle \otimes \mathbb{1}). \tag{2.20}$$

For any state $|\psi\rangle$,

$$\langle\psi|Q|\psi\rangle = \|\Pi W|0^m\rangle|\psi\rangle\|^2 = \|\Pi(\sqrt{p}|\Phi\rangle + \sqrt{1-p}|\Phi^\perp\rangle)\|^2 = \|\sqrt{p}|\Phi\rangle\|^2 = p. \tag{2.21}$$

In particular, this holds for a basis of eigenvectors of Q, and so $Q = p\mathbb{1}$.

Thus for any $|\psi\rangle$, we have

$$p|\psi\rangle = Q|\psi\rangle = (\langle 0^m| \otimes \mathbb{1})W^\dagger \Pi W(|0^m\rangle \otimes \mathbb{1})|\psi\rangle = (\langle 0^m| \otimes \mathbb{1})W^\dagger \Pi W|\Psi\rangle = \sqrt{p}(\langle 0^m| \otimes \mathbb{1})W^\dagger|\Phi\rangle. \tag{2.22}$$

From (2.18) and (2.19) we get $W^\dagger|\Phi\rangle = \sqrt{p}|\Psi\rangle + \sqrt{1-p}|\Psi^\perp\rangle$. Plugging this into the previous equation we get

$$p|\psi\rangle = \sqrt{p}(\langle 0^m| \otimes \mathbb{1})(\sqrt{p}|\Psi\rangle + \sqrt{1-p}|\Psi^\perp\rangle) = p|\psi\rangle + \sqrt{p(1-p)}(\langle 0^m| \otimes \mathbb{1})|\Psi^\perp\rangle. \tag{2.23}$$

This gives us $\sqrt{p(1-p)}(\langle 0^m| \otimes \mathbb{1})|\Psi^\perp\rangle = 0$. Since $p \in (0,1)$, this implies $\Pi|\Psi^\perp\rangle = 0$. □

Note that this fact can also be viewed as a consequence of Jordan's Lemma [Jor75]. Since this alternate proof may be more illuminating for some readers, we provide a short proof sketch.

Jordan's Lemma guarantees that the space of $(n+m)$-qubit states can be decomposed into a direct sum of 1- and 2-dimensional subspaces that are invariant under the projectors Π and $W^\dagger \Pi W$. In this decomposition, Π and $W^\dagger \Pi W$ are rank-1 projectors when restricted to the 2-dimensional subspaces. Let $|0^m\rangle|\psi_i\rangle$ denote the eigenvalue-1 eigenvector of Π within the i^{th} 2-dimensional subspace S_i. Since S_i is invariant under $W^\dagger \Pi W$, the state $W^\dagger \Pi W |0^m\rangle|\psi_i\rangle = \sqrt{p} W^\dagger |0^m\rangle V|\psi_i\rangle$ belongs to S_i. Thus $|\Psi_i^\perp\rangle := W^\dagger(\sqrt{1-p}|0^m\rangle V|\psi_i\rangle - \sqrt{p}|\Phi_i^\perp\rangle)$ is in S_i, since it is a linear combination of $|0^m\rangle|\psi_i\rangle = W^\dagger(\sqrt{p}|0^m\rangle V|\psi_i\rangle + \sqrt{1-p}|\Phi_i^\perp\rangle)$ and $W^\dagger \Pi W|0^m\rangle|\psi_i\rangle$. However, $|\Psi_i^\perp\rangle$ is orthogonal to $|0^m\rangle|\psi_i\rangle$ and is therefore an eigenvalue-0 eigenvector of Π, since Π is a rank-1 projector in S_i. Thus the lemma is true when restricted to the states $|\psi_i\rangle$, as opposed to all n-qubit states $|\psi\rangle$.

We now show that a general state $|\psi\rangle$ can be written as a linear combination of the states $|\psi_i\rangle$, which completes the proof. We claim that the number of states $|\psi_i\rangle$ is exactly 2^n, and that these states therefore form a basis for the set of states on n qubits, since these states belong to orthogonal subspaces. There are at most 2^n such states since Π has rank 2^n and each such state is an eigenvalue-1 eigenvector of Π. There also must be at least 2^n states $|\psi_i\rangle$ that belong to these 2-dimensional subspaces, since otherwise there would be a state $|0^m\rangle|\psi\rangle$ that is in a 1-dimensional subspace, i.e., is invariant under both Π and $W^\dagger \Pi W$. This is not possible because $W^\dagger \Pi W$ acting on $|0^m\rangle|\psi\rangle$ yields $\sqrt{p} W^\dagger |0^m\rangle V|\psi\rangle$, which is a subnormalized state since $0 < p < 1$. Finally, since there are 2^n orthogonal vectors $|\psi_i\rangle$, an arbitrary state $|\psi\rangle$ can be written as a linear combination of $|\psi_i\rangle$, and the result follows.

With the help of this 2D subspace lemma (Lemma 2.3), we can prove the oblivious amplitude amplification lemma (Lemma 2.2):

Proof of Lemma 2.2. Since Lemma 2.3 shows that the evolution occurs within a two-dimensional subspace (or its image under W), the remaining analysis is essentially the same as in standard amplitude amplification. For any $|\psi\rangle$, we define $|\Psi\rangle := |0^m\rangle|\psi\rangle$ and $|\Phi\rangle := |0^m\rangle V|\psi\rangle$, so that

$$W|\Psi\rangle = \sin(\theta)|\Phi\rangle + \cos(\theta)|\Phi^\perp\rangle, \tag{2.24}$$

where $\theta \in (0, \pi/2)$ is such that $\sqrt{p} = \sin(\theta)$. We also define $|\Psi^\perp\rangle$ through the equation

$$W|\Psi^\perp\rangle := \cos(\theta)|\Phi\rangle - \sin(\theta)|\Phi^\perp\rangle. \tag{2.25}$$

By Lemma 2.3, we know that $\Pi|\Psi^\perp\rangle = 0$. Using these two equations, we have

$$W^\dagger|\Phi\rangle = \sin(\theta)|\Psi\rangle + \cos(\theta)|\Psi^\perp\rangle \tag{2.26}$$

$$W^\dagger|\Phi^\perp\rangle = \cos(\theta)|\Psi\rangle - \sin(\theta)|\Psi^\perp\rangle. \tag{2.27}$$

Then a straightforward calculation gives

$$\begin{aligned} S|\Phi\rangle &= -WRW^\dagger|\Phi\rangle \\ &= -WR(\sin(\theta)|\Psi\rangle + \cos(\theta)|\Psi^\perp\rangle) \\ &= -W(\sin(\theta)|\Psi\rangle - \cos(\theta)|\Psi^\perp\rangle) \\ &= \big(\cos^2(\theta) - \sin^2(\theta)\big)|\Phi\rangle - 2\cos(\theta)\sin(\theta)|\Phi^\perp\rangle \\ &= \cos(2\theta)|\Phi\rangle - \sin(2\theta)|\Phi^\perp\rangle. \end{aligned} \tag{2.28}$$

Similarly,

$$\begin{aligned} S|\Phi^\perp\rangle &= WRW^\dagger|\Phi^\perp\rangle \\ &= WR(\cos(\theta)|\Psi\rangle - \sin(\theta)|\Psi^\perp\rangle) \\ &= W(\cos(\theta)|\Psi\rangle + \sin(\theta)|\Psi^\perp\rangle) \\ &= 2\cos(\theta)\sin(\theta)|\Phi\rangle + \big(\cos^2(\theta) - \sin^2(\theta)\big)|\Phi^\perp\rangle \\ &= \sin(2\theta)|\Phi\rangle + \cos(2\theta)|\Phi^\perp\rangle. \end{aligned} \tag{2.29}$$

Thus we see that S acts as a rotation by 2θ in the subspace $\mathrm{span}\{|\Phi\rangle, |\Phi^\perp\rangle\}$, and the result follows. □

2.2.3 Exact LCU algorithm

We can now show how to perfectly implement a unitary V that is a linear combination of unitaries, since we have a p-implementation of V from Lemma 2.1 and a procedure for converting a p-implementation into a better implementation from Lemma 2.2.

Theorem 2.4 (Exact LCU algorithm)**.** *Let V be a unitary matrix such that $V = \sum_i a_i U_i$ is a linear combination of unitary matrices U_i with $a_i > 0$ for all i. Let A be a unitary matrix that maps $|0^m\rangle$ to $\frac{1}{\sqrt{a}}\sum_i \sqrt{a_i}|i\rangle$, where $a := \|\vec{a}\|_1 = \sum_i a_i$. Then there exists a quantum algorithm that performs the map V exactly with $O(a)$ uses of A, $U := \sum_i |i\rangle\langle i| \otimes U_i$, and their inverses.*

Proof of Theorem 2.4. From Lemma 2.1 we know that $W = A^\dagger U A$ satisfies

$$W(|0^m\rangle|\psi\rangle) = \sin(\theta)|0^m\rangle V|\psi\rangle + \cos(\theta)|\Phi^\perp\rangle \tag{2.30}$$

for all states $|\psi\rangle$, where $\sin(\theta) = 1/a$ and $\theta \in (0, \pi/2)$. From Lemma 2.2 we know that given this map, if $R := 2\Pi - \mathbb{1}$ and $S := -WRW^\dagger R$, then for any $t \in \mathbb{Z}$,

$$S^t W|0^m\rangle|\psi\rangle = \sin\big((2t+1)\theta\big)|0^m\rangle V|\psi\rangle + \cos\big((2t+1)\theta\big)|\Phi^\perp\rangle. \tag{2.31}$$

If $\frac{\pi/2}{\theta}$ is an odd integer, then we are done; choosing t such that $2t+1 = \frac{\pi/2}{\theta}$ gives us a perfect implementation of V as $S^t W|0^m\rangle|\psi\rangle = |0^m\rangle V|\psi\rangle$. In this implementation, we only use the operators A and U (and their inverses) $O(t)$ times, which is $O(1/\theta) = O(1/\sin(\theta)) = O(a)$.

Now we show that this was essentially without loss of generality. Consider the case where $\frac{\pi/2}{\theta}$ is not an odd integer and let $2t+1$ be the smallest odd integer larger than $\frac{\pi/2}{\theta}$. Let $\theta' < \theta$ be such that $(2t+1)\theta' = \pi/2$. We can now perturb the map W so that it $\sin(\theta')$-implements V instead of $\sin(\theta)$-implementing V, which will reduce this to the previously considered case. Let Z be any unitary that performs the map

$$|0\rangle \mapsto \left(\frac{\sin(\theta')}{\sin(\theta)}\right)|0\rangle + \sqrt{1 - \left(\frac{\sin(\theta')}{\sin(\theta)}\right)^2}\,|1\rangle, \tag{2.32}$$

which is a valid map because $\sin(\theta')/\sin(\theta) < 1$. Now $Z \otimes W$ can play the role of W, since

$$(Z \otimes W)|0^{m+1}\rangle|\psi\rangle = \sin(\theta')|0^{m+1}\rangle V|\psi\rangle + \cos(\theta')|\Phi'^{\perp}\rangle, \tag{2.33}$$

which satisfies the conditions of the lemma. □

2.2.4 Approximate oblivious amplitude amplification

To extend our results to the case where the operator we wish to implement is only close to being unitary, we will need a stronger version of oblivious amplitude amplification. This result is technical and uninterested readers may directly jump to the section summary (Section 2.2.6).

Let $\widetilde{V}$ be an operator that is close to a unitary V and can be represented as a linear combination of unitaries. We want to implement V, but we only know a representation of $\widetilde{V}$ in terms of unitaries. We show how this suffices if V and $\widetilde{V}$ are close.

The assumption of the oblivious amplitude amplification lemma (Lemma 2.2) is that we have available a map W that acts as follows on all n-qubit states $|\psi\rangle$:

$$W|0^m\rangle|\psi\rangle = \sqrt{p}|0^m\rangle V|\psi\rangle + \sqrt{1-p}|\Phi^{\perp}\rangle, \tag{2.34}$$

where $\Pi|\Phi^{\perp}\rangle = 0$ and V is unitary. We need to relax this assumption to allow a unitary $\widetilde{W}$ that acts as follows on all n-qubits states $|\psi\rangle$:

$$\widetilde{W}|0^m\rangle|\psi\rangle = \sqrt{p}|0^m\rangle\widetilde{V}|\psi\rangle + \sqrt{1-p}|\widetilde{\Phi}^{\perp}\rangle, \tag{2.35}$$

where $\Pi|\widetilde{\Phi}^{\perp}\rangle = 0$ and $\widetilde{V}$ is not necessarily unitary.

Unlike the case where V is unitary, the states $\widetilde{V}|\psi\rangle$ and $|\widetilde{\Phi}^{\perp}\rangle$ are not necessarily normalized because $\widetilde{V}$ is not necessarily unitary.

To prove this approximate version, we can go through the proofs of Lemma 2.2 and Lemma 2.3 and prove approximate analogues of all statements. Alternately, we can reduce this case to the exact case by showing that for any $\widetilde{V}$, there exists a unitary W that p-implements a unitary V such that W is close to $\widetilde{W}$ and V is close to $\widetilde{V}$.

Note that this statement does not follow from the fact that V and $\widetilde{V}$ are close, because we need an operator W that is close to $\widetilde{W}$ on the entire vector space, including states that are not of the form $|0^m\rangle|\psi\rangle$, for example. An obvious idea would be to take the unitary $\widetilde{W}$ and modify its action on the $|0^m\rangle|\psi\rangle$ subspace so that it p-implements V correctly. However, this may not result in a unitary operator. If we think of $\widetilde{W}$ as a matrix, this is like modifying the entries in the first few columns of $\widetilde{W}$ slightly and expecting $\widetilde{W}$ to remain unitary. In general this will not be the case and could require a large number of other entries to be changed to make the matrix unitary again.

Lemma 2.4. *Let $\widetilde{W}$ be a unitary matrix on $n+m$ qubits and $\widetilde{V}$ be a matrix on n qubits that is δ-close to a unitary matrix in spectral norm, where $\delta < 1$. Additionally, let $p \in (0,1)$ be such that $p(1+\delta)^2 < 1$. Suppose that for any n-qubit state $|\psi\rangle$,*

$$\widetilde{W}|0^m\rangle|\psi\rangle = \sqrt{p}|0^m\rangle\widetilde{V}|\psi\rangle + |\widetilde{\Phi}^{\perp}\rangle, \tag{2.36}$$

where $|\widetilde{\Phi}^{\perp}\rangle$ is an unnormalized $(n+m)$-qubit state that depends on $|\psi\rangle$ and satisfies $\Pi|\Phi^{\perp}\rangle = 0$, where $\Pi = |0^m\rangle\langle 0^m| \otimes \mathbb{1}$. Then there exist unitary matrices W and V on $n+m$ qubits and n qubits respectively, such that $\|W - \widetilde{W}\| = O(\sqrt{\delta})$, $\|V - \widetilde{V}\| \leq \delta$, and for all $|\psi\rangle$

$$W|0^m\rangle|\psi\rangle = \sqrt{p}|0^m\rangle V|\psi\rangle + \sqrt{1-p}|\Phi^{\perp}\rangle, \tag{2.37}$$

where $|\Phi^{\perp}\rangle$ is an $(n+m)$-qubit state that depends on $|\psi\rangle$ and satisfies $\Pi|\Phi^{\perp}\rangle = 0$.

Proof. As in the proof of Lemma 2.3, we consider the operator

$$Q := (\langle 0^m| \otimes \mathbb{1})\widetilde{W}^{\dagger}\Pi\widetilde{W}(|0^m\rangle \otimes \mathbb{1}). \tag{2.38}$$

Since this is a Hermitian operator, it has an orthonormal set of eigenvectors. Let these be denoted $|\psi_i\rangle$ and the corresponding eigenvalues be p_i, i.e., $Q|\psi_i\rangle = p_i|\psi_i\rangle$. For any state $|\psi_i\rangle$,

$$p_i = \langle\psi_i|Q|\psi_i\rangle = \|\Pi\widetilde{W}|0^m\rangle|\psi_i\rangle\|^2 = \|\sqrt{p}|0^m\rangle\widetilde{V}|\psi_i\rangle\|^2 = p\|\widetilde{V}|\psi_i\rangle\|^2, \tag{2.39}$$

which gives us the following upper and lower bounds on p_i since $\widetilde{V}$ is δ-close to a unitary:

$$p_i \leq p(1+\delta)^2 < 1 \quad \text{and} \quad p_i \geq p(1-\delta)^2 > 0. \tag{2.40}$$

For all i, we define $|\phi_i\rangle := \widetilde{V}|\psi_i\rangle/\|\widetilde{V}|\psi_i\rangle\|$ and thus $\widetilde{V}|\psi_i\rangle = \sqrt{p_i/p}|\phi_i\rangle$. For any $i \neq j$, we have

$$0 = \langle\psi_i|Q|\psi_j\rangle = p\langle\psi_j|\widetilde{V}^{\dagger}\widetilde{V}|\psi_j\rangle = \sqrt{p_i p_j}\langle\phi_i|\phi_j\rangle, \tag{2.41}$$

which gives $\langle\phi_i|\phi_j\rangle = 0$, since $p_i \neq 0$ for all i. We see that $\widetilde{V}$ almost behaves like a unitary, since it maps an orthonormal basis to an orthogonal basis, such that the norms of the output vectors are close to 1. Based on this we can define a unitary V that maps $|\psi_i\rangle$ to $|\phi_i\rangle$. Now since $V|\psi_i\rangle = |\phi_i\rangle$ and $\widetilde{V}|\psi_i\rangle = \sqrt{p_i/p}|\phi_i\rangle$, we have $V^\dagger\widetilde{V}|\psi_i\rangle = \sqrt{p_i/p}|\psi_i\rangle$, which gives $(V^\dagger\widetilde{V} - \mathbb{1})|\psi_i\rangle = (\sqrt{p_i/p} - 1)|\psi_i\rangle$. Thus $\|V^\dagger\widetilde{V} - \mathbb{1}\| \leq \max_i |\sqrt{p_i/p} - 1|$, which is at most δ using the upper and lower bounds on p_i. Thus $\|\widetilde{V} - V\| = \|V^\dagger\widetilde{V} - \mathbb{1}\| \leq \delta$.

Returning to the question of constructing a map W, we define $|\Phi_i^\perp\rangle$ to be the normalized state that satisfies

$$\widetilde{W}|0^m\rangle|\psi_i\rangle = \sqrt{p_i}|0^m\rangle|\phi_i\rangle + \sqrt{1-p_i}|\Phi_i^\perp\rangle, \tag{2.42}$$

where $|\Phi_i^\perp\rangle$ satisfies $\Pi|\Phi_i^\perp\rangle = 0$. Since $\{|\psi_i\rangle\}$ and $\{|\phi_i\rangle\}$ are orthonormal sets, $\{|\Phi_i^\perp\rangle\}$ is also an orthonormal set. Furthermore, since every vector in $\{|0^m\rangle|\phi_i\rangle\}$ is orthogonal to every vector in $\{|\Phi_i^\perp\rangle\}$, the set containing all states of the form $|0^m\rangle|\phi_i\rangle$ and $|\Phi_i^\perp\rangle$ is an orthonormal set of vectors. Let this subspace of the entire space $n+m$-qubit states be called S and let $S^\perp$ be its orthogonal complement.

Based on this, we define a unitary R, which is the unique unitary that acts as

$$R(\sqrt{p_i}|0^m\rangle|\phi_i\rangle + \sqrt{1-p_i}|\Phi_i^\perp\rangle) = \sqrt{p}|0^m\rangle|\phi_i\rangle + \sqrt{1-p}|\Phi_i^\perp\rangle \tag{2.43}$$

$$R(\sqrt{1-p_i}|0^m\rangle|\phi_i\rangle - \sqrt{p_i}|\Phi_i^\perp\rangle) = \sqrt{1-p}|0^m\rangle|\phi_i\rangle - \sqrt{p}|\Phi_i^\perp\rangle \tag{2.44}$$

for all i, and acts as the identity on $S^\perp$. R is a indeed unitary because it is unitary within each subspace $\text{span}\{|0^m\rangle|\phi_i\rangle, |\Phi_i^\perp\rangle\}$ and on $S^\perp$.

Finally note that $W = R\widetilde{W}$ is the map we want since

$$R\widetilde{W}(|0^m\rangle|\psi_i\rangle) = \sqrt{p}|0^m\rangle|\phi_i\rangle + \sqrt{1-p}|\Phi_i^\perp\rangle = \sqrt{p}|0^m\rangle V|\psi_i\rangle + \sqrt{1-p}|\Phi_i^\perp\rangle, \tag{2.45}$$

and therefore by linearity, $W|0^m\rangle|\psi\rangle = \sqrt{p}|0^m\rangle V|\psi\rangle + \sqrt{1-p}|\Phi^\perp\rangle$ for any $|\psi\rangle$.

Lastly, we need to show that W is close to $\widetilde{W}$. Since $\|W - \widetilde{W}\| = \|R - \mathbb{1}\|$, we need to compute $\|R - \mathbb{1}\|$. R is the direct sum of 2×2 blocks, and hence $\|R - \mathbb{1}\|$ is the maximum eigenvalue of any of these blocks, which gives

$$\|R - \mathbb{1}\| = \max_i \left\| \begin{pmatrix} \sqrt{p_i} - \sqrt{p} & \sqrt{1-p_i} - \sqrt{1-p} \\ \sqrt{1-p_i} - \sqrt{1-p} & -(\sqrt{p_i} - \sqrt{p}) \end{pmatrix} \right\|. \tag{2.46}$$

For a fixed i, the matrix is Hermitian with trace 0, therefore its eigenvalues are $\lambda > 0$ and $-\lambda$. Since the determinant is the product of the eigenvalues, we have

$$\lambda^2 = 2\big(1 - \sqrt{p_i p} - \sqrt{(1-p_i)(1-p)}\big) \leq 2\left(1 - \min\{p_i, p\} - \min\{1-p_i, 1-p\}\right) \leq 2|p_i - p|, \tag{2.47}$$

where we have used the fact that $\min\{x, y\} \leq x$ and $\min\{x, y\} \leq y$. Using the relationships between p_i and p from (2.40), it follows that $p_i - p \leq p(2\delta + \delta^2) \leq p(3\delta) < 3\delta$ and $p - p_i \leq p(2\delta - \delta^2) \leq p(2\delta) < 2\delta$. Thus $\lambda^2 \leq 2|p_i - p| < 6\delta$. This gives $\lambda < \sqrt{6\delta}$ and therefore $\|W - \widetilde{W}\| = \|R - \mathbb{1}\| < \sqrt{6\delta} = O(\sqrt{\delta})$. □

2.2.5 Approximate LCU algorithm

In Section 2.2.3 we considered the problem of implementing a unitary V that can be written in the form $V = \sum_i a_i U_i$, where U_i are unitaries considered easy to perform. Our results, however, require a generalization of this situation where we only have an approximation for V in terms of unitaries. We show how to extend our LCU algorithm to this situation using Lemma 2.4.

We consider the problem of implementing a matrix $\widetilde{V}$ that is not necessarily unitary, but satisfies $\|V - \widetilde{V}\| \leq \delta$ for some unitary V, and can be expressed as a linear combination of unitaries U_i as before, i.e., $\widetilde{V} = \sum_i a_i U_i$. In this case we would like to show that we can approximately implement $\widetilde{V}$ using $O(a)$ uses of the maps A and U as before, where $a := \|\vec{a}\|_1 = \sum_i a_i$. The accuracy of our implementation will be $O(a\sqrt{\delta})$. The analogue of Theorem 2.4 in this situation is the following:

Theorem 2.5 (Approximate LCU algorithm)**.** *Let $\widetilde{V}$ be a matrix that is δ-close to some unitary in spectral norm, such that $\widetilde{V} = \sum_i a_i U_i$ is a linear combination of unitary matrices U_i with $a_i > 0$ for all i. Let A be a unitary matrix that maps $|0^m\rangle$ to $\frac{1}{\sqrt{a}} \sum_i \sqrt{a_i}|i\rangle$, where $a := \|\vec{a}\|_1 = \sum_i a_i$. Then there exists a quantum algorithm that performs the map $\widetilde{V}$ with error $O(a\sqrt{\delta})$ and makes $O(a)$ uses of A, $U := \sum_i |i\rangle\langle i| \otimes U_i$, and their inverses.*

Proof. We will follow the same steps as in Theorem 2.4. First, from Lemma 2.1 we have that $\widetilde{W} = A^\dagger U A$ satisfies

$$\widetilde{W}|0^m\rangle|\psi\rangle = \sqrt{p}|0^m\rangle\widetilde{V}|\psi\rangle + \sqrt{1-p}|\widetilde{\Phi}^\perp\rangle \tag{2.48}$$

for all states $|\psi\rangle$, where the state $|\widetilde{\Phi}^\perp\rangle$, which depends on $|\psi\rangle$, satisfies $\Pi|\widetilde{\Phi}^\perp\rangle = 0$, where $\Pi = |0^m\rangle\langle 0^m| \otimes \mathbb{1}$.

From Lemma 2.4, we know that there exist unitary matrices W and V on $n+m$ qubits and n qubits respectively, such that $\|W - \widetilde{W}\| = O(\sqrt{\delta})$, $\|V - \widetilde{V}\| \leq \delta$, and

$$W|0^m\rangle|\psi\rangle = \sqrt{p}|0^m\rangle V|\psi\rangle + \sqrt{1-p}|\Phi^\perp\rangle, \tag{2.49}$$

where $|\Phi^\perp\rangle$ is an $(n+m)$-qubit state that depends on $|\psi\rangle$ and satisfies $\Pi|\Phi^\perp\rangle = 0$.

We wish to implement $\widetilde{V}$, but instead let us implement V. We know that W p-implements V for $p = 1/a^2$, and that $\|W - \widetilde{W}\| = O(\sqrt{\delta})$, where $\widetilde{W} = A^\dagger U A$. Therefore using Lemma 2.2, we have a perfect implementation of V that makes $O(a)$ uses of W. But since we only have available the map $\widetilde{W} = A^\dagger U A$, if we use this instead of W, the implemented unitary will be distance $O(a\sqrt{\delta})$ from V, by the subadditivity of error in implementing unitaries (Theorem 1.3). Since V is δ-close to $\widetilde{V}$, we have an implementation of $\widetilde{V}$ with error $O(a\sqrt{\delta} + \delta) = O(a\sqrt{\delta})$. □

2.2.6 Summary

In summary, we have a procedure for implementing a map that can be approximately represented as a linear combination of unitaries that are easy to perform. Theorem 2.5 is the most general statement of our result.

For both the applications we consider, we will only use the following corollary of Theorem 2.5, which is specific to the query complexity model, assumes $a = \|\vec{a}\|_1$ is a constant, and has no constraint on a_i being real as the phase of a_i can be subsumed into the unitary U_i.

Corollary 2.1. *Let $\widetilde{V}$ be a matrix that is δ-close to some unitary in spectral norm, such that $\widetilde{V} = \sum_i a_i U_i$ is a linear combination of unitary matrices U_i, where $\|\vec{a}\|_1 = O(1)$ and any unitary U_i requires at most q queries to perform. Then the map $\widetilde{V}$ can be performed with error $O(\sqrt{\delta})$ using $O(q)$ queries.*

2.3 Hamiltonian simulation

We now apply the results of the previous section to give improved algorithms for simulating sparse Hamiltonians. To see how these are related, suppose we wish to implement a Hamiltonian H that is the sum of m unitaries U_j for time $t = 1/m$. We do not require that U_j be Hermitian as well, although that will be the case in our application. Then the operator we need to implement is

$$V := e^{-iH/m} = e^{-i(\sum_j U_j)/m}. \tag{2.50}$$

Expanding the definition of the matrix exponential we get

$$V = \sum_{l=0}^{\infty} \frac{1}{l!\, m^l} \Big(-i \sum_j U_j\Big)^l = \mathbb{1} - \frac{i}{m}\Big(\sum_j U_j\Big) - \frac{1}{2m^2}(U_1^2 + 2U_1U_2 + U_2^2 + \ldots) + \ldots. \tag{2.51}$$

Notice that each term in the linear combination is a unitary matrix. However, since this is an infinite sum, let us delete the terms in the sum after $l = k$ and call the resulting matrix $\widetilde{V}$.

$$\widetilde{V} := \sum_{l=0}^{k} \frac{1}{l!\, m^l} \Big(-i \sum_j U_j\Big)^l. \tag{2.52}$$

Note that when $\widetilde{V}$ is expressed as a linear combination of unitaries, $a = \|\vec{a}\|_1 < e = O(1)$. Indeed, this was the reason for choosing $t = 1/m$.

The error in this approximation, $\delta := \|V - \widetilde{V}\|$, is at most

$$\begin{aligned}\delta &= \left\| \sum_{l=k+1}^{\infty} \frac{1}{l!\, m^l} \Big(-i \sum_{j=1}^{m} U_j\Big)^l \right\| \leq \sum_{l=k+1}^{\infty} \frac{1}{l!\, m^l} \Big(\Big\|\sum_{j=1}^{m} U_j\Big\|\Big)^l \\ &\leq \sum_{l=k+1}^{\infty} \frac{1}{l!} = \frac{1}{k!} \sum_{l=k+1}^{\infty} \frac{k!}{l!} \leq \frac{1}{k!} \sum_{l=1}^{\infty} \frac{1}{(k+1)^l} < \frac{1}{k!}. \end{aligned} \tag{2.53}$$

If implementing any U_j has unit cost, then $\widetilde{V}$ is a linear combination of unitaries where each unitary may cost up to k queries to implement, since the most expensive term in the linear combination will be the product of k matrices U_j. Corollary 2.1 now gives us a simulation of H with error $O(\sqrt{\delta})$ with cost $O(k)$. If we want error at most ϵ in the final simulation, we need to take k large enough. Solving $\sqrt{1/k!} = O(\epsilon)$ yields $k = \Omega\Big(\frac{\log(1/\epsilon)}{\log\log(1/\epsilon)}\Big)$. To see this, observe that if we choose $k = c\Big(\frac{\log(1/\epsilon)}{\log\log(1/\epsilon)}\Big)$, for some constant c, then $k \log k$ is asymptotically $c \log(1/\epsilon)$ up to lower order terms. Choosing c large enough will ensure that $\log(k!)$ is much larger than $2\log(1/\epsilon)$, which is equivalent to $1/k!$ being smaller than ϵ^2.

Applying the LCU algorithm (Corollary 2.1) gives us the following lemma.

Lemma 2.5. *Let $H = \sum_{j=1}^{m} U_j$ be a Hamiltonian, where each U_j is unitary for all $j \in [m]$ and costs $O(1)$ queries to implement. Then the unitary $e^{-iH/m}$ can be implemented up to error ϵ with query complexity $O\left(\frac{\log(1/\epsilon)}{\log\log(1/\epsilon)}\right)$. Thus for $t \geq 1/m$, the unitary e^{-iHt} can be implemented up to error ϵ with query complexity $O\left(mt \frac{\log(mt/\epsilon)}{\log\log(mt/\epsilon)}\right)$.*

The last part of the lemma follows from the fact that $e^{-iHt} = (e^{-iH/m})^{mt}$, and thus composing mt implementations of the operator $e^{-iH/m}$ with error at most ϵ/mt in each implementation gives an implementation of e^{-iHt} up to error ϵ by Theorem 1.3.

This handles the case when we have a decomposition of H as a sum of unitaries. While this was a simple scenario, having a decomposition in terms of unitaries is not unrealistic. For example, H could be a weighted sum of tensor products of Pauli operators, which are unitary, or H may the be the sum of local terms, which can be represented as a linear combination of unitaries of constant dimension, which are easy to implement. Note that any Hamiltonian can be represented as a linear combination of unitaries. In fact any Hermitian matrix can be written as a linear combination of 2 unitaries. To show this, note that by conjugating the Hermitian matrix by a unitary that diagonalizes it and multiplying by a scalar, it suffices to prove this claim for diagonal Hermitian matrices with eigenvalues between -1 and $+1$. The fact that any real number in $[-1, +1]$ can be written as the average of two complex numbers of unit modulus completes the

proof. The problem, of course, is that these unitaries need to be efficiently implementable and the coefficients in the linear combination should have a small value of $\|\vec{a}\|_1$.

To handle arbitrary sparse Hamiltonians, we decompose them into a linear combination of unitary Hamiltonians. To do this we first decompose a d-sparse Hamiltonian into a sum of d^2 1-sparse Hamiltonians, which have at most 1 nonzero entry in any row or column, and then decompose 1-sparse Hamiltonians into a linear combination of unitary Hamiltonians.

Decomposing a sparse Hamiltonian as a sum of 1-sparse Hamiltonians

The step of decomposing a sparse Hamiltonian as a sum of 1-sparse Hamiltonians appears in almost every result on Hamiltonian simulation. Known results decompose a d-sparse Hamiltonian H into a sum of $O(d^2)$ 1-sparse Hamiltonians [BACS07], but simulating one query to a 1-sparse Hamiltonian requires $O(\log^* n)$ queries to the oracle for H. This suffices for our purposes, but would add a $\log^* n$ factor to the complexity.

We improve upon previous results and present a simplified decomposition theorem that decomposes a d-sparse Hamiltonian into d^2 Hamiltonians that are 1-sparse where a query to the individual 1-sparse Hamiltonians can be performed using $O(1)$ queries to the original Hamiltonian, removing the $\log^* n$ factor. This factor originally comes from a distributed vertex coloring algorithm. It also considerably simplifies the decomposition argument (even as compared to substantially less efficient simulations, such as in [AT03]).

Lemma 2.6. *If H is a d-sparse Hamiltonian, there exists a decomposition $H = \sum_{j=1}^{d^2} H_j$ where each H_j is 1-sparse and a query to any H_j can be simulated with $O(1)$ queries to H.*

Proof. The new ingredient in our proof is to assume that the graph of H is bipartite. (Here the *graph of H* has a vertex for each basis state and an edge between two vertices if the corresponding entry of H is nonzero.) This is without loss of generality because we can simulate the Hamiltonian $\sigma_x \otimes H$ instead, which is indeed bipartite and has the same sparsity as H. From a simulation of $\sigma_x \otimes H$, we can recover a simulation of H using the identity $e^{-i(\sigma_x \otimes H)t}|+\rangle|\psi\rangle = |+\rangle e^{-iHt}|\psi\rangle$.

Now we decompose a bipartite d-sparse Hamiltonian into a sum of d^2 terms. To do this, we give an edge coloring of the graph of H (i.e., an assignment of colors to the edges so that no two edges incident on the same vertex have the same color). Given such a coloring with d^2 colors, the Hamiltonian H_j formed by only considering edges with color j is 1-sparse.

We use the following simple coloring. For any pair of adjacent vertices u and v, let $r(u, v)$ denote the rank of v in u's neighbor list, i.e., the position occupied by v in a sorted list of u's neighbors. This is a number between 1 and d. Let the color of the edge (u, v), where u comes from the left part of the bipartition and v comes from the right, be the ordered pair $(r(u, v), r(v, u))$. This is a valid coloring since if (u, v) and (u, w) have the same color, then in particular the first

component of the ordered pair is the same, so $r(u,v) = r(u,w) \Rightarrow v = w$. A similar argument handles the case where the common vertex is on the right.

Given a color (a,b), it is easy to simulate queries to the Hamiltonian corresponding to that color. To compute the nonzero entries of the i^{th} row for this color, depending on whether i is in the left or right partition, we merely have to determine the neighbor of i that has rank a or b, respectively, in its neighbor list, and check if the pair (a,b) is consistent with their respective rankings, which can be obtained from the oracle with $O(1)$ queries. □

Observe that the simple trick of making the Hamiltonian bipartite suffices to remove the $O(\log^* n)$ term present in previous decompositions of this form. This trick is quite general and can be applied to remove the $O(\log^* n)$ factor wherever such a factor appears in a known Hamiltonian simulation algorithm (e.g., [BACS07, CK11b, WBHS11]).

Decomposing a 1-sparse Hamiltonian into a linear combination of unitaries

We now decompose a 1-sparse Hamiltonian G into a sum of $O(\|G\|_{\max}/\gamma)$ unitary Hamiltonians G_j up to error $O(\gamma)$.

Lemma 2.7. *For any 1-sparse Hamiltonian G and precision $\gamma > 0$, there exist $O(\|G\|_{\max}/\gamma)$ unitary Hamiltonians G_j with eigenvalues ± 1 such that $\|G - \gamma \sum_j G_j\|_{\max} \leq 3\gamma$.*

Proof. First we decompose the Hamiltonian G as $G = G_X + iG_Y + G_Z$, where G_X contains the off-diagonal real terms, iG_Y contains the off-diagonal imaginary terms, and G_Z contains the on-diagonal real terms. Next, for each of G_P for $P \in \{X,Y,Z\}$, we construct an approximation $\tilde{G}_P$ with each entry rounded off to the closest multiple of 2γ. Since each entry of $\tilde{G}_P$ is at most γ away from the corresponding entry in G_P, we have $\|G_P - \tilde{G}_P\|_{\max} \leq \gamma$. Denoting $\tilde{G} = \tilde{G}_X + i\tilde{G}_Y + \tilde{G}_Z$, this implies $\|G - \tilde{G}\|_{\max} \leq 3\gamma$.

Next, we take $C^P := \tilde{G}_P/\gamma$, so $\|C^P\|_{\max} = \lceil \|G_P\|_{\max}/\gamma \rceil \leq \lceil \|G\|_{\max}/\gamma \rceil$. We can then decompose each 1-sparse matrix C^P into $\|C^P\|_{\max}$ Hermitian matrices, each of which is 1-sparse and has entries from $\{-2,0,2\}$. If C^P_{jk} is $2p$, then the first $|p|$ matrices in the decomposition have a 2 for $p > 0$ (or -2 if $p < 0$) at the (j,k) entry, and the rest have 0. More explicitly, we define

$$C^{P,\ell}_{jk} := \begin{cases} 2 & \text{if } C^P_{jk} \geq 2\ell > 0 \\ -2 & \text{if } C^P_{jk} \leq -2\ell < 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.54}$$

for $P \in \{X,Y,Z\}$ and $\ell \in [\|C^P\|_{\max}]$. This gives a decomposition into at most $3\lceil \|G\|_{\max}/\gamma \rceil$ Hermitian matrices with eigenvalues in $\{-2,0,2\}$.

To obtain matrices with eigenvalues ± 1, we perform one more step to remove the 0 eigenvalues. We divide each $C^{P,\ell}$ into two copies, $C^{P,\ell,+}$ and $C^{P,\ell,-}$. For any column where $C^{P,\ell}$ is all zero, the corresponding diagonal element of $C^{P,\ell,+}$ is $+1$ (if $P \in \{X, Z\}$) or $+i$ (if $P = Y$) and the diagonal element of $C^{P,\ell,-}$ is -1 (if $P \in \{X, Z\}$) or $-i$ (if $P = Y$). Otherwise, we let $C^{P,\ell,+}_{jk} = C^{P,\ell,-}_{jk} = C^{P,\ell}_{jk}/2$. Thus $C^{P,\ell} = C^{P,\ell,+} + C^{P,\ell,-}$. Moreover, each column of $C^{P,\ell,\pm}$ has exactly one nonzero entry, which is ± 1 (or $\pm i$ on the diagonal of $C^{Y,\ell,\pm}$).

This gives a decomposition $\tilde{G}/\gamma = \sum_{\ell,\pm}(C^{X,\ell,\pm} + iC^{Y,\ell,\pm} + C^{Z,\ell,\pm})$ in which each term has eigenvalues ± 1. The decomposition contains at most $6\lceil \|G\|_{\max}/\gamma \rceil = O(\|G\|_{\max}/\gamma)$ terms. □

Putting it all together

Lemma 2.6 decomposes our Hamiltonian H into d^2 Hamiltonians that are 1-sparse. We further decompose H using Lemma 2.7 into a sum of $m = O(d^2\|H\|_{\max}/\gamma)$ unitary Hamiltonians G_j such that $\|H - \gamma \sum_{j=1}^m G_j\|_{\max} \leq 3\gamma d^2$, since each 1-sparse Hamiltonian is approximated with precision 3γ and there are d^2 approximations in this sum. To upper bound the simulation error, we have

$$\left\| e^{-iHt} - e^{-i\gamma \sum_j G_j t} \right\| \leq \left\| (H - \gamma \sum_{j=1}^m G_j)t \right\| \leq 3\gamma d^3 t, \tag{2.55}$$

where we used the fact that $\|e^{iA} - e^{iB}\| \leq \|A - B\|$ and $\|A\| \leq d\|A\|_{\max}$ for a d-sparse matrix A. For unitaries A and B, $\|e^{iA} - e^{iB}\| \leq \|A - B\|$ can be proved by observing that

$$\|e^{iA} - e^{iB}\| = \|(e^{iA/n})^n - (e^{iB/n})^n\| \leq n\|e^{iA/n} - e^{iB/n}\| \leq \|A - B\| + O(1/n), \tag{2.56}$$

where the first inequality uses subadditivity of error (Theorem 1.3) and the second inequality follows by Taylor expansion. Since the statement is true for all n, the claim follows.

Choosing $\gamma = \epsilon/3d^3t$ gives the required precision. We now invoke Lemma 2.5 with $H = \sum_{j=1}^m G_j$ where $m = O(d^2\|H\|_{\max}/\gamma)$ evolved for time γt to get the following theorem.

Theorem 2.1 (Sparse Hamiltonian simulation)**.** *A d-sparse Hamiltonian H can be simulated for time t with error at most ϵ using $O\left(\tau \frac{\log(\tau/\epsilon)}{\log\log(\tau/\epsilon)}\right)$ queries, where $\tau := d^2\|H\|_{\max}t \geq 1$.*

Note that the value of γ does not appear in the final expression and therefore plays no role in determining the query complexity. Thus the error in approximating H by a linear combination of unitaries can be made arbitrarily small and is not the dominant source of error—the truncation of the Taylor series for e^x is the dominant source of error.

Lower bound

We now show that in general any sparse Hamiltonian simulation method must use $\Omega\left(\frac{\log(1/\epsilon)}{\log\log(1/\epsilon)}\right)$ discrete queries to obtain error at most ϵ, and so the dependence of the query complexity in Theorem 2.1 on ϵ is tight up to constant factors. To show this, we use ideas from the proof of the no-fast-forwarding theorem [BACS07, Theorem 3], which says that generic Hamiltonians cannot be simulated in time sublinear in the evolution time. The Hamiltonian used in the proof of that theorem has the property that simulating it for time $t = \pi n/2$ determines the parity of n bits exactly. We observe that simulating this Hamiltonian (with sufficiently high precision) for any time $t > 0$ gives an unbounded-error algorithm for the parity of n bits, which also requires $\Omega(n)$ queries [FGGS98, BBC+01].

Theorem 2.2 (ϵ-dependent lower bound for Hamiltonian simulation)**.** *For any $\epsilon > 0$, there exists a 2-sparse Hamiltonian H with $\|H\|_{\max} < 1$ such that simulating H with precision ϵ for constant time requires $\Omega\left(\frac{\log(1/\epsilon)}{\log\log(1/\epsilon)}\right)$ queries.*

Proof. To construct the Hamiltonian, we begin with a simpler Hamiltonian H' that acts on vectors $|i\rangle$ with $i \in \{0, 1, \ldots, n\}$. The nonzero matrix entries of H' are $\langle i\,|H'|\,i+1\rangle = \langle i+1\,|H'|\,i\rangle = \sqrt{(n-i)(i+1)}/n$ for $i \in \{0, 1, \ldots, n-1\}$ [CDEL04]. We have $\|H'\|_{\max} < 1$, and simulating H' for $t = \pi n/2$ starting with the state $|0\rangle$ gives the state $|n\rangle$ (i.e., $e^{-iH'\pi n/2}|0\rangle = |n\rangle$). More generally, for $t \in [0, \pi n/2]$, we claim that $|\langle n|e^{iH't}|0\rangle| = |\sin(t/n)|^n$.

To see this, consider the Hamiltonian $\bar{X} := \sum_{j=1}^{n} X^{(j)}$, where $X := \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$ and $X^{(j)}$ denotes the single qubit operator X acting on the j^{th} qubit. Since $e^{-iXt} = \cos(t)\mathbb{1} - i\sin(t)X$, we have $|\langle 11\ldots1|e^{-i\bar{X}t}|00\ldots0\rangle| = |\sin(t)|^n$. Defining $|\mathrm{wt}_k\rangle := \binom{n}{k}^{-1/2} \sum_{|x|=k} |x\rangle$, we have

$$\bar{X}|\mathrm{wt}_k\rangle = \sqrt{(n-k+1)k}|\mathrm{wt}_{k-1}\rangle + \sqrt{(n-k)(k+1)}|\mathrm{wt}_{k+1}\rangle. \tag{2.57}$$

This is precisely the behavior of nH' with $|k\rangle$ playing the role of $|\mathrm{wt}_k\rangle$, so the claim follows.

Now, as in [BACS07], consider a Hamiltonian H generated from an n-bit string $x_0x_1\ldots x_{n-1}$. H acts on vertices $|i, j\rangle$ with $i \in \{0, \ldots, n\}$ and $j \in \{0, 1\}$. The nonzero matrix entries of this Hamiltonian are

$$\langle i, j\,|H|\,i+1, j \oplus x_i\rangle = \langle i+1, j \oplus x_i\,|H|\,i, j\rangle = \sqrt{(n-i)(i+1)}/n \tag{2.58}$$

for all i and j. By construction, $|0, 0\rangle$ is connected to either $|i, 0\rangle$ or $|i, 1\rangle$ (but not both) for any i; it is connected to $|i, j\rangle$ if and only if $j = x_0 \oplus x_1 \oplus \cdots \oplus x_{i-1}$. Thus $|0, 0\rangle$ is connected to either $|n, 0\rangle$ or $|n, 1\rangle$, and determining which is the case determines the parity of x. The graph of this Hamiltonian contains two disjoint paths, one containing $|0, 0\rangle$ and $|n, \mathsf{XOR}(x)\rangle$ and the other containing $|0, 1\rangle$ and $|n, 1 \oplus \mathsf{XOR}(x)\rangle$. Restricted to the connected component of

$|0,0\rangle$, this Hamiltonian is the same as H'. Thus, starting with the state $|0,0\rangle$ and simulating H for time t gives $|\langle n, \mathsf{XOR}(x)|e^{-iHt}|0,0\rangle| = |\sin(t/n)|^n$. Furthermore, for any t, we have $\langle n, 1 \oplus \mathsf{XOR}(x)|e^{-iHt}|0,0\rangle = 0$ since the two states lie in disconnected components.

Simulating this Hamiltonian exactly for any time $t > 0$ starting with the state $|0,0\rangle$ yields an unbounded-error algorithm for computing the parity of x, as follows. First we measure $e^{-iHt}|0,0\rangle$ in the computational basis. We know that for any $t > 0$, the state $e^{-iHt}|0,0\rangle$ has some nonzero overlap on $|n, \mathsf{XOR}(x)\rangle$ and zero overlap on $|n, 1 \oplus \mathsf{XOR}(x)\rangle$. If the first register is not n, we output 0 or 1 with equal probability. If the first register is n, we output the value of the second register. This is an unbounded-error algorithm for the parity of x, and thus requires $\Omega(n)$ queries.

Since the unbounded-error query complexity of parity is $\Omega(n)$ [FGGS98, BBC+01], this shows that exactly simulating H for any time $t > 0$ needs $\Omega(n)$ queries. However, even if we only have an approximate simulation, the previous algorithm still works as long as the error in the output state is smaller than the bias $|\langle n, \mathsf{XOR}(x)|e^{-iHt}|0,0\rangle|^2$. If we ensure that the overlap is larger than ϵ by a constant factor, then even with error ϵ, the overlap on that state will be larger than ϵ. On the other hand, the overlap on $|n, 1 \oplus \mathsf{XOR}(x)\rangle$ is at most ϵ, since the output state is ϵ close to the ideal output state which has no overlap.

To achieve an overlap larger than ϵ, we need $|\sin(t/n)|^{2n}$ to be larger than ϵ for constant t. More concretely, choose $t = 1$ so we need to find the largest n that satisfies $|\sin(1/n)|^{2n} > \epsilon$. Asymptotically $\sin(1/n) \sim 1/n$, and thus it suffices to choose an n for which n^{2n} is much smaller than $1/\epsilon$, which is equivalent to $2n \log n < \log(1/\epsilon)$. Choosing $n = c\left(\frac{\log(1/\epsilon)}{\log\log(1/\epsilon)}\right)$, for some constant $c > 0$, satisfies this since $n \log n \sim c \log(1/\epsilon)$ up to lower order terms.

This gives the required lower bound of $\Omega(n) = \Omega\left(\frac{\log(1/\epsilon)}{\log\log(1/\epsilon)}\right)$ queries. □

2.4 Continuous- and fractional-query simulation

In this section we present a dramatically simplified and improved simulation of the continuous- or fractional-query model (introduced in Section 2.1.2) in the conventional discrete-query model. The main result of this section is Theorem 2.3, which states that any T-query algorithm in the continuous- or fractional-query model can be simulated in the standard (discrete-query) model within error ϵ using $O\left(T\frac{\log(T/\epsilon)}{\log\log(T/\epsilon)}\right)$ queries.

Equivalence of the continuous- and fractional-query models

We begin by recalling the equivalence of the continuous- and fractional-query models for any error $\epsilon > 0$. An explicit simulation of the continuous-query model by the fractional-query model was provided by [CGM+09, Section II.A]; the proof is a straightforward application of a result of

[HR90]. The other direction is apparently folklore (e.g., both directions are implicitly assumed in [Moc07]); we provide their proof for completeness.

Theorem 2.6 (Equivalence of continuous- and fractional-query models)**.** *For any $\epsilon > 0$, any algorithm with continuous-query complexity T can be implemented with fractional-query complexity T with error at most ϵ and conversely, any algorithm with fractional-query complexity T can be implemented with continuous-query complexity T with error at most ϵ.*

Proof. We wish to implement the unitary $U(T)$ satisfying the Schrödinger equation (2.4) with $U(0) = \mathbb{1}$. To refer to the solutions of this equation for arbitrary Hamiltonians and time intervals, we define $U_H(t_2, t_1)$ to be the solution of the Schrödinger equation with Hamiltonian H from time t_1 to time t_2 where $U(t_1) = \mathbb{1}$. In this notation, $U(T) = U_{H_x+H_D}(T, 0)$.

Let m be an integer and $\theta = T/m$. We have

$$U_{H_x+H_D}(T,0) = U_{H_x+H_D}(m\theta,(m-1)\theta)\cdots U_{H_x+H_D}(2\theta,\theta)U_{H_x+H_D}(\theta,0). \tag{2.59}$$

If we can approximate each of these m terms, we can use the subadditivity of error in implementing unitaries (Theorem 1.3) to obtain an approximation of $U(T)$.

Reference [HR90] shows that for small θ, the evolution according to Hamiltonians A and B over an interval of length θ approximates the evolution according to $A+B$ over the same interval. Specifically, from [HR90, eq. A8b] we have

$$\|U_{A+B}((j+1)\theta, j\theta) - U_A((j+1)\theta, j\theta)U_B((j+1)\theta, j\theta)\| \le \int_{j\theta}^{(j+1)\theta} \mathrm{d}v \int_{j\theta}^{v} \mathrm{d}u \, \|[A(u), B(v)]\|, \tag{2.60}$$

where $[A, B] := AB - BA$. In our application, $A(t) = H_D(t)$ and $B = H_x$. Since $\|H_x\| = 1$, the right-hand side is at most

$$2\int_{j\theta}^{(j+1)\theta} \mathrm{d}v \int_{j\theta}^{v} \mathrm{d}u \, \|H_D(u)\| \le 2\int_{j\theta}^{(j+1)\theta} \mathrm{d}v \int_{j\theta}^{(j+1)\theta} \mathrm{d}u \, \|H_D(u)\| = 2\theta \int_{j\theta}^{(j+1)\theta} \|H_D(u)\| \, \mathrm{d}u. \tag{2.61}$$

By subadditivity, the error in implementing $U(T)$ is at most

$$2\theta \sum_{j=0}^{m-1} \int_{j\theta}^{(j+1)\theta} \|H_D(u)\| \, \mathrm{d}u = 2\theta \int_0^T \|H_D(u)\| \mathrm{d}u = 2\frac{T}{m} \int_0^T \|H_D(u)\| \mathrm{d}u, \tag{2.62}$$

which can be made smaller than ϵ by choosing a large enough m, which proves this direction of the equivalence.

For the other direction, consider a fractional-query algorithm

$$U_{\mathrm{fq}} := U_m Q_x^{\alpha_m} U_{m-1} \cdots Q_x^{\alpha_2} U_1 Q_x^{\alpha_1} U_0 \tag{2.63}$$

where $\alpha_i \in (0,1]$ for all $i \in [m]$, with complexity $T = \sum_{i=1}^m \alpha_i$. Let $A_i := \sum_{j=1}^i \alpha_j$ for all $i \in [m]$ and let $U_j =: e^{-iH_D^{(j)}}$ for all $j \in \{0,1,\ldots,m\}$. Consider the piecewise constant Hamiltonian

$$H(t) = H_x + \frac{1}{\epsilon_1}\left(\delta_{t\in[0,\epsilon_1]} H_D^{(0)} + \sum_{i=1}^m \delta_{t\in[A_i-\epsilon_1,A_i]} H_D^{(i)}\right), \tag{2.64}$$

where δ_P is 0 if P is false and 1 if P is true. Provided $\epsilon_1 \leq \min\{\alpha_1/2, \alpha_2, \ldots, \alpha_m\}$, evolving with $H(t)$ from $t = 0$ to T implements a unitary close to our fractional-query algorithm. More precisely, it implements

$$\begin{aligned} U(T) = e^{-i(H_D^{(m)}+\epsilon_1 H_x)} e^{-i(\alpha_m-\epsilon_1)H_x} e^{-i(H_D^{(m-1)}+\epsilon_1 H_x)} \cdots \\ e^{-i(\alpha_2-\epsilon_1)H_x} e^{-i(H_D^{(1)}+\epsilon_1 H_x)} e^{-i(\alpha_1-2\epsilon_1)H_x} e^{-i(H_D^{0}+\epsilon_1 H_x)}, \end{aligned} \tag{2.65}$$

which satisfies $\|U(T) - U_{\text{fq}}\| = O(m\epsilon_1)$. This follows from the fact that each exponential in (2.65) approximates the corresponding unitary of (2.63) with error $O(\epsilon_1)$ (e.g., $\|e^{-i(H_D^{(m)}+\epsilon_1 H_x)} - U_m\| = O(\epsilon_1)$ and $\|e^{-i(\alpha_m-\epsilon_1)H_x} - Q_x^{\alpha_m}\| = O(\epsilon_1)$) and the subadditivity of error when implementing unitaries (Theorem 1.3). The fact that each exponential has error $O(\epsilon_1)$ follows from the inequality $\|e^{iA} - e^{iB}\| \leq \|A - B\|$, which we proved in (2.56).

This simulation has continuous-query complexity T. Its error can be made less than ϵ by choosing ϵ_1 sufficiently small (in particular, it suffices to take some $\epsilon_1 = \Theta(\epsilon/m)$). □

Since the two models are equivalent, it suffices to convert a fractional-query algorithm to a discrete-query algorithm.

Simulating the fractional-query model with discrete queries

We start with a fractional-query algorithm that makes at most 1 query. The result for multiple queries (Theorem 2.3) follows straightforwardly.

Lemma 2.8. *An algorithm with fractional-query complexity at most* 1 *can be simulated with error at most* ϵ *with* $O\left(\frac{\log(1/\epsilon)}{\log\log(1/\epsilon)}\right)$ *queries.*

Proof. Consider the unitary V performed by a fractional-query algorithm with fractional-query complexity at most $T \leq 1$. V has the following form for some positive integer m:

$$V = W_m Q^{\alpha_m} W_{m-1} Q^{\alpha_{m-1}} \cdots W_1 Q^{\alpha_1} W_0, \tag{2.66}$$

where for all $i \in [m]$, W_i are arbitrary oracle-independent unitaries, $\alpha_i \in (0,1]$, and $\sum_i \alpha_i = T \leq 1$. Let Q be the unitary representing a discrete oracle query. Recall that if the discrete-query

operator Q acts as $Q|j\rangle|b\rangle = (-1)^{bx_j}|j\rangle|b\rangle$, then we have defined Q^α as $Q^\alpha|j\rangle|b\rangle = e^{-i\pi\alpha b x_j}|j\rangle|b\rangle$. From the definition, it follows that

$$Q^\alpha = \frac{1}{2}(\mathbb{1}+Q) + e^{-i\pi\alpha}\frac{1}{2}(\mathbb{1}-Q) = e^{-\pi\alpha/2}(\cos(\pi\alpha/2)\mathbb{1} + i\sin(\pi\alpha/2)Q). \tag{2.67}$$

Let us define $c_i := \cos(\pi\alpha_i/2)$ and $s_i := \sin(\pi\alpha_i/2)$. In terms of these, we can write V as

$$V = e^{-i\pi T/2}W_m(c_m\mathbb{1} + is_mQ)W_{m-1}(c_{m-1}\mathbb{1} + is_{m-1}Q)\cdots W_1(c_1\mathbb{1} + is_1Q)W_0. \tag{2.68}$$

It is clear that this is a linear combination of unitaries. For this linear combination, the 1-norm of the coefficients, $\|\vec{a}\|_1$, is $\prod_i(c_i + s_i)$, which we can upper bound as follows:

$$\prod_i(c_i+s_i) = \prod_i\sqrt{1+\sin(\pi\alpha_i)} \leq \sqrt{\prod_i(1+\pi\alpha_i)} \leq \sqrt{\prod_i e^{\pi\alpha_i}} \leq e^{\pi T/2} \leq e^{\pi/2}, \tag{2.69}$$

where we have used the inequalities $\sin x \leq x$ and $1 + x \leq e^x$ for all $x \geq 0$.

Although V is a linear combination of unitaries with constant $\|\vec{a}\|_1$, some terms in the linear combination are expensive to implement. For example, there is a term that involves m products of Q, which would require m queries to implement. However, this term has very little weight in the linear combination. This suggests that for some k we can truncate the expression for V by deleting all terms that have more than k occurrences of Q and the resulting operator will still be close to V. Let this approximation to V be denoted $\widetilde{V}$. The error in this approximation, $\|V - \widetilde{V}\|$ is the sum of all the terms in $\prod_i(c_i + s_i)$ that contain more than k factors of s_i.

We can succinctly represent this error term by defining m independent random variables X_i with $\Pr(X_i = 0) = \frac{c_i}{c_i+s_i}$ and $\Pr(X_i = 1) = \frac{s_i}{c_i+s_i}$. Then

$$\|V - \widetilde{V}\| \leq \left(\prod_i(c_i+s_i)\right)\Pr\left(\sum_i X_i > k\right) \leq e^{\pi/2}\Pr\left(\sum_i X_i > k\right), \tag{2.70}$$

where we have used (2.69) to obtain the last inequality. We can bound the probability that $\sum_i X_i > k$ using the Chernoff bound (see for example [MR95, Theorem 4.1]), which says that for any $\delta > 0$,

$$\Pr\left(\sum_i X_i > (1+\delta)\mu\right) < \frac{e^{\delta\mu}}{(1+\delta)^{(1+\delta)\mu}}, \tag{2.71}$$

where $\mu := \sum_i \Pr(X_i = 1) = \sum_i \frac{s_i}{c_i+s_i}$. Since $\alpha_i \geq 0$ and $\sum_i \alpha_i = T \leq 1$, we have $\mu \geq 0$ and

$$\mu = \sum_i \frac{s_i}{c_i+s_i} \leq \sum_i s_i = \sum_i \sin(\pi\alpha_i/2) \leq \sum_i \pi\alpha_i/2 = \pi T/2, \tag{2.72}$$

where we used the facts that $\sin x \leq x$ for all $x > 0$ and $\sin\theta + \cos\theta \geq 1$ for all $\theta \in [0, \pi/2]$. Setting $k = (1+\delta)\mu$, we get

$$\Pr\left(\sum_i X_i > k\right) < \frac{e^{k-\mu}}{(1+\delta)^k} = \frac{e^{k-\mu}\mu^k}{k^k} < \frac{(e\mu)^k}{k^k}. \tag{2.73}$$

Since $\mu = O(1)$, we can choose some $k = \Omega\left(\frac{\log(1/\epsilon)}{\log\log(1/\epsilon)}\right)$ to achieve $\|V - \widetilde{V}\| = O(\epsilon^2)$.

Finally, $\widetilde{V}$ is a linear combination of unitaries, each of which cost at most k discrete queries, with $\|\vec{a}\|_1 = O(1)$ and $\|V - \widetilde{V}\| = O(\epsilon^2)$. The result now follows from Corollary 2.1. □

Since we can simulate a fractional-query algorithm that makes at most 1 query in the discrete-query model with error at most ϵ using $O\left(\frac{\log(1/\epsilon)}{\log\log(1/\epsilon)}\right)$ queries, we can simulate a T query fractional-query algorithm by dividing it into T pieces, each of which makes at most 1 fractional query, and simulating each piece with error at most ϵ/T. The final error in simulating the T-query fractional-query algorithm will be at most ϵ by the subadditivity of error (Theorem 1.3). This gives us the main result of the section.

Theorem 2.3 (Continuous-query simulation)**.** *An algorithm with continuous- or fractional-query complexity $T \geq 1$ can be simulated with error at most ϵ with $O\left(T\frac{\log(T/\epsilon)}{\log\log(T/\epsilon)}\right)$ queries.*

As in Hamiltonian simulation, it can be shown that this dependence on ϵ is optimal [BCC+14].

2.5 Open problems

While our algorithm for continuous-query simulation is optimal as a function of ϵ alone, it is suboptimal as a function of T, and it is unclear what tradeoffs might exist between these two parameters.

Open Problem 2.1. Given a continuous- or fractional-query algorithm with query complexity T, what is the optimal query complexity of simulating it in the discrete-query model up to error ϵ, as a function of T and ϵ? We know this can be done with $O\left(T\frac{\log(T/\epsilon)}{\log\log(T/\epsilon)}\right)$ queries (Theorem 2.3) and that $\Omega\left(T + \frac{\log(1/\epsilon)}{\log\log(1/\epsilon)}\right)$ queries are necessary [BCC+14]. In particular, can we at least perform a bounded-error simulation with $O(T)$ queries?

In the context of sparse Hamiltonian simulation, the quantum walk-based simulation of [Chi10, BC12] achieves linear dependence on t, whereas our upper bound is superlinear in t. However,

the dependence on ϵ is significantly worse in the walk-based approach. It would be desirable to combine the benefits of these two approaches into a single algorithm. Another open question is to better understand the dependence of our sparse Hamiltonian simulation method on the sparsity d. While we use $d^{2+o(1)}$ queries, the method of [BC12] uses only $O(d)$ queries. Could the performance of the simulation be improved by a different decomposition of the Hamiltonian? These questions are encompassed by the following.

Open Problem 2.2. What is the optimal query complexity of sparse Hamiltonian simulation in terms of all parameters of interest? We know upper bounds of $O(d\|H\|_{\max}t/\sqrt{\epsilon})$ [BC12] and $O\left(\tau\frac{\log(\tau/\epsilon)}{\log\log(\tau/\epsilon)}\right)$ queries, where $\tau := d^2\|H\|_{\max}t$ (Theorem 2.1). The best lower bounds we know are $\Omega(\|H\|_{\max}t)$ [BACS07] and $\Omega\left(\frac{\log(1/\epsilon)}{\log\log(1/\epsilon)}\right)$ (Theorem 2.2).

Chapter 3

Oracle identification

Chapter summary: In the oracle identification problem, we are given oracle access to an unknown N-bit string x promised to belong to a known set of size M and our task is to identify x. We present the first quantum algorithm for the problem that is optimal in its dependence on N and M. Our algorithm considerably simplifies and improves the previous best algorithm due to Ambainis et al. [AIK+07]. Our algorithm also has applications in quantum learning theory, where it improves the complexity of exact learning with quantum membership queries, resolving a conjecture of Hunziker et al. [HMP+10].

Our algorithm is based on ideas from classical learning theory and a new composition theorem for solutions of the filtered γ_2-norm semidefinite program, which characterizes quantum query complexity. Our composition theorem is quite general and allows us to compose quantum algorithms with input-dependent query complexities without incurring a logarithmic overhead for error reduction. Our composition theorem is used in the next chapter to remove all log factors from the nearly optimal quantum algorithm for Boolean matrix multiplication.

This chapter is based on the following paper:

[Kot14] Robin Kothari. An optimal quantum algorithm for the oracle identification problem. In *Proceedings of the 31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014)*, volume 25 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 482–493, 2014.

3.1 Introduction

In this chapter we study the query complexity of the oracle identification problem, the very basic problem of completely determining a string given oracle access to it.

Problem description

In the oracle identification problem, we are given an oracle for an unknown N-bit string x, which is promised to belong to a known set $\mathcal{C} \subseteq \{0,1\}^N$, and our task is to identify x while minimizing the number of oracle queries. For a set $\mathcal{C}$, we denote this problem $\mathsf{OIP}(\mathcal{C})$. In other words, $\mathsf{OIP}(\mathcal{C})$ is the problem of computing the identity function $f(x) = x$, promised that $x \in \mathcal{C}$. As usual, classical algorithms are given access to an oracle that outputs x_i on input i, while quantum algorithms have access to a unitary Q_x that maps $|i,b\rangle$ to $|i,b \oplus x_i\rangle$ for $b \in \{0,1\}$.

For example, let $\mathcal{C} := \{0,1\}^N$. Then the classical (deterministic or bounded-error) query complexity of $\mathsf{OIP}(\mathcal{C})$ is N, since every bit needs to be queried to completely learn x, even with bounded error. A surprising result of van Dam shows that $Q(\mathsf{OIP}(\mathcal{C})) = N/2 + O(\sqrt{N})$ [vD98], which is optimal up to lower order terms [ABSdW13]. As another example, consider the set $\mathcal{C}_1 = \{x : |x| = 1\}$, where $|x|$ denotes the Hamming weight of x. This corresponds to the search problem with 1 marked item and thus $Q(\mathsf{OIP}(\mathcal{C}_1)) = \Theta(\sqrt{N})$ [BBBV97, Gro96], while the classical query complexity is $\Theta(n)$.

History and motivation

Since oracle identification is a very general problem, it has been studied in several different contexts. Ambainis et al. [AIK+04, AIK+07] studied this problem from the perspective of finding algorithms that are optimal in terms of N and the size of the set $\mathcal{C}$. This is the version of the problem we will focus on in this chapter.

The problem has also been studied in quantum machine learning [SG04, AS05, HMP+10], where it is known as the problem of exact learning using quantum membership queries. In the machine learning literature the focus is on finding optimal algorithms for $\mathsf{OIP}(\mathcal{C})$ in terms of combinatorial parameters of the set $\mathcal{C}$. We also address this version of the problem in this chapter.

Several well-known problems are special cases of oracle identification, e.g., the ordered search problem, the search problem with one marked element [Gro96], the Bernstein–Vazirani problem [BV97], the oracle interrogation problem [vD98], and hidden shift problems [vDHI06]. For some applications, generic oracle identification algorithms are almost as good as algorithms tailored to the specific application, as observed in [CKOR13]. Consequently, the main result of this chapter improves some of the upper bounds in [CKOR13].

The problem has also been studied in the context of post-quantum cryptography from the perspective of designing quantum-secure message authentication codes [BZ13]. There the problem was studied over a larger alphabet without any constraint on the set $\mathcal{C}$ (i.e., $\mathcal{C}$ is the set of all strings of length N over the chosen alphabet). Boneh and Zhandry [BZ13] provided an optimal algorithm and matching lower bound for this case.

We primarily study the version of the problem studied by Ambainis et al., although we will later discuss how our algorithm yields improved upper bounds in the context of quantum machine learning. Ambainis et al. [AIK+04, AIK+07] studied the oracle identification problem in terms of N and $M := |\mathcal{C}|$. They exhibited algorithms whose query complexity is close to optimal in its dependence on N and M.

For a given N and M, we say an oracle identification algorithm is optimal in terms of N and M if it solves all N-bit oracle identification problems with $|\mathcal{C}| = M$ making at most Q queries and there exists some N-bit oracle identification problem with $|\mathcal{C}| = M$ that requires $\Omega(Q)$ queries. This does not, however, mean that the algorithm is optimal for each set $\mathcal{C}$ individually, since these two parameters do not completely determine the query complexity of the problem. For example, all oracle identification problems with $M = N$ can be solved with $O(\sqrt{N})$ queries, and this is optimal since this class includes the search problem with one marked item ($\mathcal{C}_1$ above). However there exists a set $\mathcal{C}$ of size $M = N$ with query complexity $\Theta(\log N)$, such as the set of all strings with arbitrary entries in the first $\log N$ bits and zeros elsewhere. There even exists a set $\mathcal{C}$, derived from the Bernstein–Vazirani problem [BV97], with $M = N$ with quantum query complexity equal to 1. (This set has classical query complexity $\Theta(\log N)$.)

Let $\mathsf{OIP}(M, N)$ denote the set of oracle identification problems with $\mathcal{C} \subseteq \{0,1\}^N$ and $|\mathcal{C}| = M$. Let the query complexity of $\mathsf{OIP}(M, N)$ be the maximum query complexity of any problem in this set. Our goal is to characterize $Q(\mathsf{OIP}(M, N))$.

Known results

The classical query complexity of $\mathsf{OIP}(M, N)$ is easy to characterize:

Proposition 3.1. *The classical (deterministic or bounded-error) query complexity of* $\mathsf{OIP}(M, N)$ *is* $\Theta(\min\{M, N\})$.

For $M \leq N$, the upper bound follows from the observation that we can always eliminate at least one potential string in $\mathcal{C}$ with one query. For the lower bound, consider any subset of $\mathcal{C}_1$ of size M. For $M > N$, the lower bound follows from any set $\mathcal{C} \supseteq \mathcal{C}_1$ and the upper bound is trivial since any query problem can be solved with N queries.

In the quantum case, the $M \leq N$ case is fully understood. For a lower bound, we consider (as before) any subset of $\mathcal{C}_1$ of size M, which is as hard as the search problem on M bits and requires $\Omega(\sqrt{M})$ queries. For an upper bound, we can reduce this to the case of $M = N$ by selecting M

bits such that the strings in $\mathcal{C}$ are distinct when restricted to these bits. A proof that such a set of M bits always exists appears in [CKOR13, Theorem 11]. Thus $Q(\mathsf{OIP}(M,N)) \leq Q(\mathsf{OIP}(M,M))$, which is $O(\sqrt{M})$ [AIK+04, Theorem 3]. In summary, we have the following.

Proposition 3.2. *For $M \leq N$, $Q(\mathsf{OIP}(M,N)) = \Theta(\sqrt{M})$.*

For the other regime, where $M > N$, the best known lower and upper bounds are the following, from [AIK+04, Theorem 2] and [AIK+07, Theorem 2] respectively.

Theorem 3.1 ([AIK+04, AIK+07])**.** *If $N < M \leq 2^{N^d}$ for some constant $d < 1$, then*

$$Q(\mathsf{OIP}(M,N)) = O(\sqrt{N \log M / \log N}) \tag{3.1}$$

and for all $M > N$, $Q(\mathsf{OIP}(M,N)) = \Omega(\sqrt{N \log M / \log N})$.

When M gets closer to 2^N, their algorithm no longer gives nontrivial upper bounds. For example, if $M \geq 2^{N/\log N}$, their algorithm makes $O(N)$ queries. While not stated explicitly, an improved algorithm follows from the techniques of [AIN+09, Theorem 6], but the improved algorithm also does not yield a nontrivial upper bound when $M \geq 2^{N/\log^3 N}$.

Ambainis et al. left open two problems, in increasing order of difficulty. Quoting [AIK+07, Section 5]:

> A challenging question is whether or not there exists an OIP algorithm whose upper bound is $o(N)$ for $M > 2^{N/\log N}$, say, for $M = 2^{N/\log\log N}$. Even more challenging is to design an OIP algorithm which is optimal in the whole range of M.

Our results

In this chapter we resolve both open problems by completely characterizing the quantum query complexity of the oracle identification problem in the range $N < M \leq 2^N$. Furthermore, we present a single algorithm that is optimal in the entire range of M, including $M \leq N$. Our main result is the following:

Theorem 3.2. *For $N < M \leq 2^N$, $Q(\mathsf{OIP}(M,N)) = \Theta\left(\sqrt{\frac{N \log M}{\log(N/\log M)+1}}\right)$.*

The lower bound follows from the ideas in [AIK+04], but needs additional calculation. The lower bound also appears in an unpublished manuscript [AIN+09, Remark 1], but we provide a proof in Section 3.2 for completeness. The $+1$ term in the denominator is relevant only when M gets close to 2^N; it ensures that the complexity is $\Theta(N)$ in that regime.

The main result is the algorithm, which is quite different from and simpler than that of [AIK+07]. It is also optimal in the full range of M as it makes $O\left(\sqrt{\frac{N \log M}{\log(N/\log M)+1}}\right)$ queries when $M \geq N$ and $O(\sqrt{M})$ queries when $M \leq N$. We now present a high-level overview.

High-level overview of our techniques

Our upper bound has two main ingredients: First, we use ideas from classical learning theory, where the oracle identification problem is studied as the problem of exact learning with membership queries [Ang88]. In particular, our quantum algorithm is based on Hegedűs' implementation of the halving algorithm [Heg95]. Hegedűs characterizes the number of queries needed to solve the classical oracle identification problem in terms of the "extended teaching dimension" of $\mathcal{C}$. While we do not use that notion, we borrow some of the main ideas of the algorithm.

Our algorithm is based on some simple ideas. Say we know that the oracle string x belongs to a set S. We can construct from S a string s, known as the "majority string," which is 1 at position i if at least half the strings in S are 1 at position i and 0 otherwise. Importantly, for any i, the set of strings in S that disagree with s at position i is at most half the size of S. Now we search for a disagreement between x and s using Grover's algorithm. If the algorithm finds no disagreement, then $x = s$. If it does, we have reduced the size of S by a factor of 2. This gives an algorithm with query complexity $O(\sqrt{N} \log M)$, which is suboptimal. We improve the algorithm by taking advantage of two facts: first, that Grover's algorithm can find a disagreement faster if there are many disagreements to be found, and second, that there exists an order in which to find disagreements that reduces the size of S as much as possible in each iteration. The existence of such an order was shown by Hegedűs [Heg95]. Our algorithm is fully explained in Section 3.3.

The second ingredient of our upper bound is a general composition theorem for solutions of the filtered γ_2-norm semidefinite program (SDP) introduced by Lee et al. [LMR+11] that preserves input-dependent query complexities. We need such a result to resolve the following problem: Our algorithm consists of k bounded-error quantum algorithms that must be run sequentially because each algorithm requires as input the output of the previous algorithm. Let the query complexities of the algorithms be $Q_1(x), Q_2(x), \ldots, Q_k(x)$ on input x. If these were exact algorithms, we could merely run them one after the other, giving one algorithm's output to the next as input, to obtain an algorithm with worst-case query complexity $O(\max_x \sum_i Q_i(x))$. However, since these are bounded-error algorithms, we cannot guarantee that all k algorithms will give the correct output with high probability. One option is to apply standard error reduction, but this would yield an algorithm that makes $O(\max_x \sum_i Q_i(x) \log k)$ queries in the worst case. Instead we prove a general composition theorem for the filtered γ_2-norm SDP that gives us an algorithm that makes $O(\max_x \sum_i Q_i(x))$ queries, as if the algorithms had no error. A similar result is known for worst-case query complexity, but that gives a suboptimal upper bound of $O(\sum_i \max_x Q_i(x))$ queries in this case. Our composition theorem is proved in Section 3.4.

Other applications

The oracle identification problem was also studied by Atıcı and Servedio [AS05], who studied algorithms that are optimal for a given set $\mathcal{C}$. The query complexity of their algorithm depends

on a combinatorial parameter of $\mathcal{C}$, $\hat{\gamma}^{\mathcal{C}}$, which satisfies $2 \leq 1/\hat{\gamma}^{\mathcal{C}} \leq N+1$. They prove $Q(\mathsf{OIP}(\mathcal{C})) = O(\sqrt{1/\hat{\gamma}^{\mathcal{C}}} \log M \log\log M)$. Our algorithm for oracle identification, without modification, makes fewer queries than this bound. Our algorithm's query complexity is $O\left(\sqrt{\frac{1/\hat{\gamma}^{\mathcal{C}}}{\log 1/\hat{\gamma}^{\mathcal{C}}}} \log M\right)$, which resolves a conjecture of Hunziker et al. [HMP+10]. We prove this in Section 3.5.

Our composition theorem can also be used to remove unneeded log factors from existing quantum query algorithms. In the next chapter we show how to improve the query complexity of our algorithm for Boolean matrix multiplication from $\tilde{O}(n\sqrt{\ell})$, where n is the size of the matrices and ℓ is the sparsity of the output, to $O(n\sqrt{\ell})$. We conclude with some discussion and open problems in Section 3.6.

3.2 Oracle identification lower bound

The main result, Theorem 3.2, naturally has two parts. In this section we prove the lower bound.

Theorem 3.3. *For any $N < M \leq 2^N$, $Q(\mathsf{OIP}(M,N)) = \Omega\left(\sqrt{\frac{N \log M}{\log(N/\log M)+1}}\right)$.*

To simplify our lower bounds, we first show the result for $M \geq 2^{N/2}$. In this case, we can consider the set of all N-bit strings with zeros in the last $N/2$ bits. This set has size $2^{N/2}$, and oracle identification on this set is equivalent to identifying an arbitrary $N/2$-bit string, which requires $\Theta(N)$ queries [FGGS98, BBC+01]. Thus the theorem is true when $M \geq 2^{N/2}$.

Now consider the set $\mathcal{C}_k$, the set of all N-bit strings with Hamming weight k. We will show that the oracle identification problem on this set provides the lower bound claimed in Theorem 3.3. We claim that $Q(\mathsf{OIP}(\mathcal{C}_k)) = \Omega(\sqrt{Nk})$ when $k \leq N/2$, and that $|\mathcal{C}_k| = \binom{N}{k}$ is at most M when $k = \frac{1}{10}\frac{\log M}{\log(N/\log M)+1}$ and $M < 2^{N/2}$. The lower bound follows from these two claims.

Lemma 3.1. *Let $\mathcal{C}_k = \{x \in \{0,1\}^N : |x| = k\}$ be the set of all N-bit strings with Hamming weight k. Then $Q(\mathsf{OIP}(\mathcal{C}_k)) = \Omega(\sqrt{Nk})$ when $k \leq N/2$.*

Proof. Consider the problem PromiseSearch, in which we have oracle access to a d-bit string promised to contain exactly one 1 and our task is to find it. This problem requires $\Omega(\sqrt{d})$ queries to solve, which can be shown by the adversary method (see [Amb02, Theorem 1]).

Without loss of generality, assume N is an integer multiple of k. (If not, we could choose the largest such integer smaller than N.) Divide the N-bit string into k equal parts of size $N/k \geq 2$. On each part of size N/k, we embed the PromiseSearch problem. Solving the oracle identification problem on this string would solve k independent instances of the PromiseSearch problem on N/k bits, which by the direct sum theorem (Theorem 1.1) requires $\Omega(k\sqrt{N/k}) = \Omega(\sqrt{Nk})$ queries. □

Lemma 3.2. *For any $N < M < 2^{N/2}$, if $k = \frac{1}{10}\frac{\log M}{\log(N/\log M)+1}$ then $\binom{N}{k} \leq M$.*

Proof. We prove this for $k = \frac{1}{10}\frac{\log M}{\log(N/\log M)}$ instead. The claim for $k = \frac{1}{10}\frac{\log M}{\log(N/\log M)+1}$ follows from the monoticity of the function $\binom{N}{k}$ when $k < N/2$.

For this proof, we define $n := \log N$ and $m := \log M$. In this notation, $k = \frac{1}{10}\frac{m}{\log(N/m)}$. Now

$$\binom{N}{k} \leq \left(\frac{Ne}{k}\right)^k = \left(\frac{10Ne\log\left(\frac{N}{m}\right)}{m}\right)^k = \left(\frac{\left(\frac{N}{m}\right)^{10} 10e\log\left(\frac{N}{m}\right)}{\left(\frac{N}{m}\right)^9}\right)^k = M\left(\frac{10e\log\left(\frac{N}{m}\right)}{\left(\frac{N}{m}\right)^9}\right)^k, \quad (3.2)$$

where the last equality follows from the fact that $\left(\frac{N}{m}\right)^{10k} = 2^m = M$. Since $M < 2^{N/2}$, $m < N/2$ and $\left(\frac{N}{m}\right) > 2$, we have

$$\left(\frac{10e\log\left(\frac{N}{m}\right)}{\left(\frac{N}{m}\right)^9}\right) < 1, \quad (3.3)$$

which shows that $\binom{N}{k} \leq M$. □

Observe that the set of hard instances we use for the lower bound is all strings of Hamming weight k for the largest k that satisfies $\binom{N}{k} \leq M$. However, a matching upper bound for this set is easy since we can find all k 1s in a string using $O(\sqrt{Nk})$ queries (e.g., see Lemma 5.6).

3.3 Oracle identification algorithm

In this section we explain the ideas that go into our algorithm and prove its correctness. We also prove the query upper bound assuming we can compose bounded-error quantum algorithms without incurring log factors, which we justify in Section 3.4.

Throughout this section, let $x \in \mathcal{C}$ be the string we are trying to identify. For any set $S \in \{0,1\}^N$, let $\mathsf{MAJ}(S)$ be an N-bit string such that $\mathsf{MAJ}(S)_i$ is 1 if $|\{y \in S : y_i = 1\}| \geq |\{y \in S : y_i = 0\}|$ and 0 otherwise. In words, $\mathsf{MAJ}(S)_i$ is b if the majority of strings in S have bit i equal to b. Note that the string $\mathsf{MAJ}(S)$ need not be a member of S.

3.3.1 Basic halving algorithm

We begin by describing a general learning strategy called the halving algorithm, attributed to Littlestone [Lit88]. Say we currently know that the oracle string x belongs to a known set $S \subseteq \mathcal{C}$. The halving algorithm tests if the oracle string x is equal to $\mathsf{MAJ}(S)$. If it is equal, we have

identified x; if not, we look for a bit at which they disagree. Having found such a bit i, we know that $x_i \neq \mathsf{MAJ}(S)_i$, and we may delete all strings in S that are inconsistent with this. Since at most half the strings in S disagree with $\mathsf{MAJ}(S)$ at any position, we have at least halved the number of potential strings.

To convert this into a quantum algorithm, we need a subroutine that tests if a given string $\mathsf{MAJ}(S)$ is equal to the oracle string x and finds a disagreement otherwise. This can be done by running Grover's algorithm on the bitwise XOR of x and $\mathsf{MAJ}(S)$. This gives us the following simple algorithm.

Algorithm 3.1 Basic halving algorithm

1: $S \leftarrow \mathcal{C}$
2: **repeat**
3: Search for a disagreement between x and $\mathsf{MAJ}(S)$. $O(\sqrt{N})$
4: If a disagreement is found, delete all inconsistent strings from S. If not, let $S \leftarrow \{\mathsf{MAJ}(S)\}$.
5: **until** $|S| = 1$

This algorithm always finds the unknown string x, since S always contains x. The loop can run at most $\log M$ times, since each iteration cuts down the size of S by a factor of 2. Grover's algorithm needs $O(\sqrt{N})$ queries, but it is a bounded-error algorithm. For this section, let us assume that bounded-error algorithms can be treated like exact algorithms and need no error reduction. Assuming this, Algorithm 3.1 makes $O(\sqrt{N} \log M)$ queries.

3.3.2 Improved halving algorithm

Even assuming free error reduction, Algorithm 3.1 is not optimal. Primarily, this is because Grover's algorithm can find an index i such that $x_i \neq \mathsf{MAJ}(S)_i$ faster if there are many such indices to be found, and Algorithm 3.1 does not exploit this fact.

Given an N-bit binary string, we can find a 1 with $O(\sqrt{N/K})$ queries in expectation, where $K > 0$ is the number of 1s in the string [BBHT98]. Alternately, there is a variant of Grover's algorithm that finds the first 1 (from left to right, say) in the string in $O(\sqrt{p})$ queries in expectation where p is the position of the first 1. This follows from the known $O(\sqrt{N})$ query algorithm for finding the first 1 in a string of size N [DHHM06], by running that algorithm on the first 2^k bits, for $k = 1, 2, \ldots, \log N$. Let us refer to this function as the find-first-one function. We can now modify the previous algorithm to look for the first disagreement between x and $\mathsf{MAJ}(S)$ instead of any disagreement. Algorithm 3.2 is our modified algorithm.

As before, the algorithm always finds the unknown string. To analyze the query complexity, let r be the number of times the loop repeats, including the last run of the loop where no disagreement is found. For convenience, we will say that in the last run of the loop, a disagreement was found

Algorithm 3.2 Improved halving algorithm

1: $S \leftarrow \mathcal{C}$
2: **repeat**
3: Search for the first disagreement between x and $\mathsf{MAJ}(S)$. $O(\sqrt{p_i})$
4: If a disagreement is found, delete all inconsistent strings from S. If not, let $S \leftarrow \{\mathsf{MAJ}(S)\}$.
5: **until** $|S| = 1$

at the $(N+1)^{\text{th}}$ bit, i.e., one position after the last bit in the string. Let $p_1, p_2, \ldots, p_r$ be the relative positions of disagreement, where by relative positions we mean that the i^{th} disagreement is found p_i bits after p_{i-1}. For example, this means that the first disagreement was found at bit p_1, the next one was found at bit $p_1 + p_2$ and so on. After the first run of the loop, since a disagreement is found at position p_1, we have learned the first p_1 bits of x; the first $p_1 - 1$ bits agree with $\mathsf{MAJ}(S)$, while bit p_1 disagrees with $\mathsf{MAJ}(S)$. Thus we are left with a set S in which all strings agree on these p_1 bits. For convenience, we can treat S as a set of strings of length $N - p_1$ (instead of length N) and only look for disagreements starting from position p_1 in the next iteration. This explains why the cost of finding the second disagreement scales like $O(\sqrt{p_2})$ instead of $O(\sqrt{p_1 + p_2})$, because we are not going to search over the first p_1 bits. Each iteration reduces the effective length of strings in S by p_i, and since the last disagreement is found at position $N + 1$ by convention, we have $\sum_{i=1}^{r} p_i = N + 1$.

For example, consider the situation where $x = \mathsf{MAJ}(S)$ the first time the loop is run. In this case, no disagreement will be found and therefore $p_1 = N + 1$ and $r = 1$. As another example, let x be a string for which the first disagreement is found at position p, and after updating S by eliminating strings that are inconsistent with the first p bits of x, $x = \mathsf{MAJ}(S)$. In this example $p_1 = p$, $p_2 = N + 1 - p$, and $r = 2$.

To analyze the query complexity, note that as in Algorithm 3.1, the loop can run at most $\log M$ times, thus $r \leq \log M$. Finally, let us assume again that these bounded-error search subroutines are exact. Then this algorithm requires $O(\sum_i \sqrt{p_i})$ queries, which is $O(\sqrt{N \log M})$, by the Cauchy–Schwarz inequality.

3.3.3 Final algorithm

While Algorithm 3.2 is an improvement over Algorithm 3.1, it is still not optimal. One reason is that sometimes a disagreement between the majority string and x may eliminate more than half the possible strings. This observation can be exploited by finding disagreements in such a way as to maximize the reduction in size when a disagreement is found, an idea due to Hegedűs [Heg95].

To understand the basic idea, consider searching for a disagreement between x and $\mathsf{MAJ}(S)$ classically. The most obvious strategy is to check if $x_1 = \mathsf{MAJ}(S)_1$, $x_2 = \mathsf{MAJ}(S)_2$, and so on

until a disagreement is found. This strategy makes more queries if the disagreement is found at a later position. However, we could have chosen to examine the bits in any order. We would like the order to be such that if a disagreement is found at a later position, it cuts down the size of S by a larger factor. Such an ordering would ensure that either we spend very few queries and achieve a factor-2 reduction right away, or we spend more queries but the size of S goes down significantly. Hegedűs shows that there is always a reordering of the bits that achieves this. The following lemma is similar to [Heg95, Lemma 3.2], but we provide a proof for completeness.

Lemma 3.3. *For any $S \subseteq \{0,1\}^N$, there exists a string $s \in \{0,1\}^N$ and a permutation σ on N, such that for any $p \in [N]$, $|S_p| \leq \frac{|S|}{\max\{2,p\}}$, where $S_p := \{y \in S : y_{\sigma(i)} = s_{\sigma(i)}$ for $1 \leq i \leq p-1$ and $y_{\sigma(p)} \neq s_{\sigma(p)}\}$, the set of strings in S that agree with s at $\sigma(1), \ldots, \sigma(p-1)$ and disagree with it at $\sigma(p)$.*

Proof. We will construct the permutation σ and string s greedily, starting with the first position, $\sigma(1)$. We choose this bit to be one that intuitively contains the most information, i.e., a bit for which the fraction of strings that agree with the majority is closest to $1/2$. This choice will make $|S_1|$ as large as possible. More precisely, we choose $\sigma(1)$ to be any j that maximizes $|\{y \in S : y_j \neq \mathsf{MAJ}(S)_j\}|$. Then let $s_{\sigma(1)}$ be $\mathsf{MAJ}(S)_{\sigma(1)}$.

In general, after having chosen $\sigma(1), \ldots, \sigma(k-1)$ and having defined s on those bits, we choose $\sigma(k)$ to be the most informative bit assuming all previous bits have agreed with string s on positions $\sigma(1), \ldots, \sigma(k-1)$. This choice makes $|S_k|$ as large as possible. More precisely, define

$$\bar{S}_p := \{y \in S : y_{\sigma(i)} = s_{\sigma(i)} \text{ for all } 1 \leq i \leq p\}. \tag{3.4}$$

We choose $\sigma(k)$ to be any bit j that maximizes $|\{y \in \bar{S}_{k-1} : y_j \neq \mathsf{MAJ}(\bar{S}_{k-1})_j\}|$. Then let $s_{\sigma(k)}$ be $\mathsf{MAJ}(\bar{S}_{k-1})_{\sigma(k)}$.

This construction ensures that $|S_1| \geq |S_2| \geq \ldots \geq |S_N|$. To show this, note that since $\sigma(k)$ was chosen to maximize $|\{y \in \bar{S}_{k-1} : y_j \neq \mathsf{MAJ}(\bar{S}_{k-1})_j\}|$, we have

$$|S_k| = |\{y \in \bar{S}_{k-1} : y_{\sigma(k)} \neq \mathsf{MAJ}(\bar{S}_{k-1})_{\sigma(k)}\}| \geq |\{y \in \bar{S}_{k-1} : y_{\sigma(k+1)} \neq \mathsf{MAJ}(\bar{S}_{k-1})_{\sigma(k+1)}\}|. \tag{3.5}$$

The size of this set is at least $|\{y \in \bar{S}_k : y_{\sigma(k+1)} \neq \mathsf{MAJ}(\bar{S}_{k-1})_{\sigma(k+1)}\}|$, since $\bar{S}_k \subseteq \bar{S}_{k-1}$. We do not know the value of $\mathsf{MAJ}(\bar{S}_{k-1})_{\sigma(k+1)}$ (e.g., it need not be equal to $s_{\sigma(k+1)}$), but we do know that it is either 0 or 1, which gives

$$\begin{aligned} |S_k| &\geq \min\left\{|\{y \in \bar{S}_k : y_{\sigma(k+1)} \neq 0\}|, |\{y \in \bar{S}_k : y_{\sigma(k+1)} \neq 1\}|\right\} \\ &= \min\left\{|\{y \in \bar{S}_k : y_{\sigma(k+1)} \neq s_{\sigma(k+1)}\}|, |\{y \in \bar{S}_k : y_{\sigma(k+1)} = s_{\sigma(k+1)}\}|\right\} \\ &= \min\{|S_{k+1}|, |\bar{S}_{k+1}|\} = |S_{k+1}|, \end{aligned} \tag{3.6}$$

where the last equality uses $|S_k| \leq |\bar{S}_k|$ for all k.

Finally, combining $|S_1| + \ldots + |S_p| \leq |S|$ with $|S_1| \geq |S_2| \geq \ldots \geq |S_p|$ gives us $|S_p| \leq |S|/p$. Combining this with $|S_1| \leq |S|/2$, which follows from the definition of S_1, yields the result. □

Algorithm 3.3 Final algorithm

1: $S \leftarrow \mathcal{C}$
2: **repeat**
3: Let σ and s be as in Lemma 3.3. (These depend on the current S.)
4: Search for the first (according to σ) disagreement between x and s. $O(\sqrt{p_i})$
5: If we find a disagreement, delete all inconsistent strings from S. If not, let $S \leftarrow \{s\}$.
6: **until** $|S| = 1$

We can now state our final oracle identification algorithm (Algorithm 3.3).

This algorithm is similar to Algorithm 3.2, except that in each run we define σ and s using Lemma 3.3 based on the current set S and search for the first disagreement between x and s according to σ. Let us also only search for the first disagreement among those bits on which it is possible to have disagreements. For example, if it is never possible to have disagreements beyond the k^{th} bit, then the algorithm should not look for a disagreement beyond the k^{th} bit. This may happen because bits beyond k are equal for all strings in S, or because there is a unique string that agrees with s on the first k bits. We will call these bits noninformative bits. For example, if the only remaining strings are 000 and 111, and we are looking for disagreements with 111, either a disagreement is found at the first bit or not at all. The second and third bits are noninformative. This example shows that a bit i may be noninformative even if it is not the same for all $x \in S$.

With this in mind, let r be the number of times the loop repeats. This means $r-1$ disagreements were found and in the last iteration x agreed with s in all the remaining (informative) bits. Let the position of the first disagreement be p_1. Now that we know the first p_1 bits of x, we have eliminated several strings from S. S now contains at least p_1 noninformative bits, since all strings in S agree on those bits. The next disagreement is found at position p_2 and so on until p_{r-1}. Now the number of remaining informative bits is at most $N - \sum_{i=1}^{r-1} p_i$, since each iteration reveals the value of p_i bits making them noninformative as all strings in the updated set S agree on these bits. In the last iteration, let the total number of informative bits be p_r. The search subroutine in the last iteration will search over p_r bits.

From the definition of p_i, it is clear that $\sum_{i=1}^{r} p_i \leq N$. Unlike the previous analysis, the bound $r \leq \log M$ can be loose, since the size of S may reduce by a larger factor due to Lemma 3.3. Instead, we know that each iteration reduces the set S by a factor of $\max\{2, p_i\}$, which gives us $\prod_{i=1}^{r} \max\{2, p_i\} \leq M$. As before, we will assume the search subroutine is exact, which gives us a query upper bound of $O(\sum_{i=1}^{r} \sqrt{p_i})$, subject to the constraints $\sum_{i=1}^{r} p_i \leq N$ and $\prod_{i=1}^{r} \max\{2, p_i\} \leq M$. We solve this optimization problem below.

Lemma 3.4. *Let $C(M, N)$ be the maximum value attained by $\sum_{i=1}^{r} \sqrt{p_i}$, subject to the constraints*

$\sum_{i=1}^{r} p_i \le N$, $\prod_{i=1}^{r} \max\{2, p_i\} \le M$, $r \in [N]$, *and* $p_i \in [N]$ *for all* $i \in [r]$. *Then*

$$C(M,N) = O\left(\sqrt{\frac{N \log M}{\log(N/\log M) + 1}}\right) \quad \textit{and} \quad C(M,N) = O(\sqrt{M}). \tag{3.7}$$

Proof. First we define two closely related optimization problems and show that their optimum values upper bound $C(M,N)$. Let α_1 and α_2 denote the optimum values of problem 1 and 2 respectively. We will show that $C(M,N) \le \alpha_1 \le \alpha_2$ and then upper bound α_2 using the dual of problem 2. Let $n := \lceil \log N \rceil$ and $m := \lceil \log M \rceil$.

Problem 1 (α_1)

$$\begin{aligned} \text{maximize:} \quad & \sum_{i=1}^{r} \sqrt{q_i} \\ \text{subject to:} \quad & \sum_{i=1}^{r} q_i \le 2N, \\ & \prod_{i=1}^{r} q_i \le M^2, \\ & r \in [N], \\ & 2 \le q_i \le 2N \quad (\text{for } i \in [r]). \end{aligned}$$

Problem 2 (α_2)

$$\begin{aligned} \text{maximize:} \quad & \sum_{k=1}^{n+2} \sqrt{2^k} x_k \\ \text{subject to:} \quad & \sum_{k=1}^{n+2} 2^k x_k \le 4N, \\ & \sum_{k=1}^{n+2} k x_k \le 4m, \\ & x_k \ge 0 \quad (\text{for } k \in [n+2]). \end{aligned}$$

Let $p_1, \ldots, p_r, r$ be an optimal solution of the problem in the statement of the lemma. Thus $C(M,N) = \sum_{i=1}^{r} \sqrt{p_i}$. Define $q_i := 2p_i$, for all $i \in [r]$. This is a feasible solution of problem 1, since $\sum_i p_i \le N \Rightarrow \sum_i q_i \le 2N$, and $\prod_i \max\{2, p_i\} \le M$ gives us $\prod_i 2 \le M$ and $\prod_i p_i \le M$, which together yield $\prod_i 2p_i \le M^2$. Finally $\sum_i \sqrt{p_i} \le \sum_i \sqrt{2p_i}$, which gives us $C(M,N) \le \alpha_1$.

Now let $q_1, \ldots, q_r, r$ be an optimal solution of problem 1. Thus $\alpha_1 = \sum_{i=1}^{r} \sqrt{q_i}$. Define $x_k := |\{i : \lceil \log q_i \rceil = k\}|$. We claim that this is a feasible solution of problem 2. $\sum_i q_i \le 2N \Leftrightarrow \sum_i 2^{\log q_i} \le 2N$, which implies $\sum_i 2^{\lceil \log q_i \rceil} \le 4N$. We can rewrite $\sum_i 2^{\lceil \log q_i \rceil}$ as $\sum_k 2^k x_k$, which gives us $\sum_k 2^k x_k \le 4N$. The next constraint $\prod_i q_i \le M^2$ implies $\sum_i \log q_i \le 2m$. Since each $q_i \ge 2$, the number of terms in this sum is at most $2m$, thus $\sum_i \lceil \log q_i \rceil \le \sum_i (\log q_i + 1) \le 4m$. Again, $\sum_i \lceil \log q_i \rceil$ is the same as $\sum_k k x_k$, which gives us $\sum_k k x_k \le 4m$. Finally $\alpha_1 = \sum_i \sqrt{q_i} = \sum_i \sqrt{2^{\log q_i}} \le \sqrt{2^{\lceil \log q_i \rceil}} \le \sum_k \sqrt{2^k} x_k \le \alpha_2$.

Problem 2 is a linear program, which gives us an easy way to upper bound α_2. For convenience, let $N' = 4N$, $n' = \lceil \log N' \rceil = n + 2$, and $m' = 4m$. Let the optimum values of the following primal and dual linear programs be α and β respectively. Clearly $\alpha_2 = \alpha$. By weak duality of linear programming, we have $\alpha \le \beta$.

$$\begin{array}{lll} & \underline{\text{Primal } (\alpha)} & \\ \text{maximize:} & \displaystyle\sum_{k=1}^{n'} \sqrt{2^k} x_k & \\ \text{subject to:} & \displaystyle\sum_{k=1}^{n'} 2^k x_k \le N', & \\ & \displaystyle\sum_{k=1}^{n'} k x_k \le m', & \\ & x_k \ge 0 \quad (\text{for } k \in [n']). & \end{array} \qquad \begin{array}{lll} & \underline{\text{Dual } (\beta)} & \\ \text{minimize:} & N'y + m'z & \\ \text{subject to:} & 2^k y + kz \ge \sqrt{2^k}, & (\text{for } k \in [n']) \\ & y, z \ge 0. & \end{array}$$

For convenience, define $d := \log(2N'/m') = \log(2N/m)$, which satisfies $d \ge 1$ since $m \le N$. We can use any dual feasible solution to upper bound β. Let $y = \sqrt{\frac{1}{2^d d}}$ and $z = \sqrt{\frac{2^d}{d}}$. Thus $\beta \le N'y + m'z \le 2\sqrt{2}\sqrt{\frac{N'm'}{\log(N'/m')+1}} = O\left(\sqrt{\frac{Nm}{\log(N/m)+1}}\right)$.

Let us check the constraints: Clearly $y, z \ge 0$; the other constraints require that

$$\sqrt{\frac{2^k}{d2^d}} + \sqrt{\frac{k^2 2^d}{2^k d}} \ge 1$$

for all $k \ge 1$ and $d \ge 1$. Using $a + b \ge 2\sqrt{ab}$, the left-hand side of this equation is greater than $2k/d$. Thus the inequality clearly holds for $k \ge d$ (and even $k \ge d/2$).

Now suppose $1 \le k \le d$. Let us show that the second term $\sqrt{\frac{k^2 2^d}{2^k d}}$ is large enough. Since $\frac{k^2}{2^k}$ is concave in this range, the minimum is achieved at either $k = 1$ or $k = d$. For $k = 1$, the second term becomes $\sqrt{2^d/d}$, and for $k = d$, the second term evaluates to $\sqrt{d}$; both of which are at least 1 when $d \ge 1$.

Since the solution is feasible, we get $C(M, N) \le \beta = O\left(\sqrt{\frac{Nm}{\log(N/m)+1}}\right)$. Finally, note that in the problem in the statement of the lemma, $\prod_i \max\{2, p_i\} \le M$ forces $\sum_i p_i \le M$, which also implies $p_i \le M$ and $r \in M$. Thus we may simply substitute N with M to get another valid upper bound. This gives $C(M, N) = O(\sqrt{M})$. □

Thus Algorithm 3.3 achieves the upper bound claimed in Theorem 3.2, under the assumption that our bounded-error subroutines can be treated as exact algorithms. But since they not exact, we could reduce the error with logarithmic overhead. Since the loop repeats at most $\log M$ times, if we ensure that the error in each subroutine is an inverse polynomial in $\log M$, the overall error will be less than a constant. Thus by using standard error reduction, we have an algorithm for the oracle identification problem that makes $O\left(\sqrt{\frac{N \log M}{\log(N/\log M)+1}} \log\log M\right)$ queries.

However, it is usually unnecessary to incur this loss in quantum query algorithms. In the next section we show this is true for Algorithm 3.3, and the $\log \log M$ factor can be removed. Readers uninterested in removing this final log factor may safely skip the next section or jump to the nontechnical summary of this section (Section 3.4.4).

3.4 Removing log factors using the filtered γ_2 norm

While the primary aim is to rigorously establish the query complexity of Algorithm 3.3, in this section we will develop techniques that can be used more generally. Let us begin by describing what we would like to prove.

Algorithm 3.3 essentially consists of a loop repeated $r(x)$ times. We write $r(x)$ to make explicit its dependence on the input x. The loop itself consists of running a variant of Grover's algorithm on x, based on information we have collected thus far about x. Call these algorithms $A_1, A_2, \ldots, A_{r(x)}$. To be clear, A_1 is the algorithm that is run the first time the loop is executed, i.e., it looks for a disagreement under the assumption that $S = \mathcal{C}$. It produces an output $p_1(x)$, which is then used by A_2. A_2 looks for a disagreement assuming a modified set S, which is smaller than $\mathcal{C}$. Let us say that in addition to $p_2(x)$, A_2 also outputs $p_1(x)$. This ensures that the output of A_i completely describes all the information we have collected about x. Thus algorithm A_{i+1} now only needs the output of A_i to work correctly.

Thus we can now view Algorithm 3.3 as a sequential composition of $r(x)$ algorithms, A_1, $A_2, \ldots, A_{r(x)}$. It is a composition in the sense that the output of one is required as the input of the next algorithm. We know that the expected query complexity of A_i on input x is $O(\sqrt{p_i(x)})$. If these algorithms were exact, then running them one after the other would yield an algorithm with expected query complexity $O(\sum_i \sqrt{p_i(x)})$. But since they are bounded error, this straightforward strategy does not work.

Surprisingly, it is known how to sequentially compose quantum algorithms in terms of worst-case query complexity. More precisely, if we have r algorithms $A_1, A_2, \ldots, A_r$ with worst-case query complexities Q_i, then there is a quantum algorithm that solves the composed problem with $O(\sum_i Q_i)$ queries. This is a remarkable property of quantum algorithms, which follows from the work of Lee et al. [LMR+11]. We first discuss this simpler result before moving on to input-dependent query complexities.

3.4.1 Composition theorem for worst-case query complexity

We now show a composition theorem for solutions of the filtered γ_2-norm SDP, which implies a similar result for worst-case quantum query complexity. This follows from the work of Lee et al. [LMR+11], which we generalize in the next section.

Consider functions that map $D \to E$, where $D \subseteq \{0,1\}^N$ and E is some finite set. For any matrix A indexed by elements of D, we define a quantity $\gamma(A)$. (To readers familiar with the notation of [LMR+11], this is the same as their $\gamma_2(A|\Delta)$.)

Definition 3.1. Let A be a square matrix indexed by elements of D. We define $\gamma(A)$ to be the optimum of the following semidefinite program:

$$\gamma(A) := \min_{\{|u_{xj}\rangle, |v_{yj}\rangle\}} \max_{x \in D} \quad c(x) \tag{3.8}$$

$$\text{subject to:} \qquad \forall x \in D, \quad c(x) = \max\left\{ \sum_j \||u_{xj}\rangle\|^2, \sum_j \||v_{xj}\rangle\|^2 \right\} \tag{3.9}$$

$$\forall x, y \in D, \quad \sum_{j : x_j \neq y_j} \langle u_{xj} | v_{yj} \rangle = A_{xy}. \tag{3.10}$$

We call the semidefinite program (SDP) above the filtered γ_2-norm SDP or the $\gamma(A)$ SDP. For a function $f : D \to E$, let F be its Gram matrix, defined as $F_{xy} = 1$ if $f(x) \neq f(y)$ and $F_{xy} = 0$ otherwise. Lee et al. provide an algorithm and matching lower bound to show that $Q(f) = \Theta(\gamma(J - F))$, where J is the all-ones matrix.

More generally, they showed that this SDP can be used to upper bound the quantum query complexity of state conversion. In the state conversion problem, we have to convert a given state $|s_x\rangle$ to $|t_x\rangle$. An explicit description of the states $|s_x\rangle$ and $|t_x\rangle$ is known for all $x \in D$, but we do not know the value of x. Since the query complexity of this task depends only on the Gram matrices of the starting and target states, define S and T by $S_{xy} := \langle s_x | s_y \rangle$ and $T_{xy} := \langle t_x | t_y \rangle$ for all $x, y \in D$. Let $S \mapsto T$ denote the problem of converting states with Gram matrix S to those with Gram matrix T. If F is the Gram matrix of a function f, then $J \mapsto F$ is the function evaluation problem. Lee et al. provide a bounded-error quantum algorithm that solves the state conversion problem with $O(\gamma(S - T))$ queries. This gives $Q(S \mapsto T) = O(\gamma(S - T))$, which generalizes $Q(f) = O(\gamma(J - F))$. Note that unlike in function evaluation, this provides only an upper bound and not a tight characterization.

We now have the tools to prove the composition theorem for the filtered γ_2-norm SDP.

Theorem 3.4 ([LMR+11])**.** *Let $f_0, f_1, \ldots, f_k$ be functions with Gram matrices $F_0, F_1, \ldots, F_k$ respectively. Let $C_1, C_2, \ldots, C_k$ be the optimum value of the SDPs for the state conversion problems $F_0 \mapsto F_1, F_1 \mapsto F_2, \ldots, F_{k-1} \mapsto F_k$, i.e., $\forall i \in [k]$, $C_i = \gamma(F_{i-1} - F_i)$. Then $\gamma(F_0 - F_k) \leq \sum_{i=1}^k C_i$.*

This does not appear explicitly in [LMR+11], but simply follows from the triangle inequality $\gamma(A + B) \leq \gamma(A) + \gamma(B)$ [LMR+11, Lemma A.2]. From this we can also show an analogous theorem for quantum query complexity, which states $Q(F_0 \mapsto F_k) = O(\sum_{i=1}^k Q(F_{i-1} \mapsto F_i))$. We do not prove this claim as we do not need it here.

3.4.2 Composition theorem for input-dependent query complexity

For our application, we require a composition theorem similar to Theorem 3.4, but for input-dependent query complexity. However, it is not even clear what this means a priori, since the value $\gamma(J - F)$ does not contain information about input-dependent complexities. Indeed, the value is a single number and cannot contain such information. However, the SDP does contain this information and we modify this framework to be able to access this.

For example, let f be the find-first-one function discussed in Section 3.3.2, which outputs the smallest i such that $x_i = 1$ and outputs $N + 1$ if $x = 0^N$. There is a quantum algorithm that solves this with $O(\sqrt{f(x)})$ queries in expectation. Furthermore, there is a feasible solution for the $\gamma(J - F)$ SDP with $c(x) = O(\sqrt{f(x)})$, where $c(x)$ is the function that appears in eq. (3.9). This suggests that $c(x)$ gives us information about the x-dependent query complexity. This function $c(x)$ will serve as our input-dependent cost measure.

Definition 3.2 (Cost function)**.** Let A be a square matrix indexed by elements of D. We say $c : D \to \mathbb{R}$ is a feasible cost function for the $\gamma(A)$ SDP if there is a feasible solution of the $\gamma(A)$ SDP with values $c(x)$ in eq. (3.9). Let the set of all feasible cost functions for $\gamma(A)$ be denoted $\Gamma(A)$.

Note that if $c(\cdot)$ is a feasible cost function for $\gamma(J - F)$, then $\max_x c(x)$ is an upper bound on the worst-case cost, $\gamma(J - F)$, which is exactly what we expect from an input-dependent cost. We can now prove an input-dependent analogue of Theorem 3.4 with $c(x)$ playing the role of $\gamma(J - F)$.

Theorem 3.5. *Let $f_0, f_1, \ldots, f_k$ be functions with Gram matrices $F_0, F_1, \ldots, F_k$ respectively. Let $c_1(\cdot), c_2(\cdot), \ldots, c_k(\cdot)$ be feasible cost functions for $\gamma(F_0 - F_1), \gamma(F_1 - F_2), \ldots, \gamma(F_{k-1} - F_k)$, i.e., for all $i \in [k]$, $c_i(\cdot) \in \Gamma(F_{i-1} - F_i)$. Then there is a $c(\cdot) \in \Gamma(F_0 - F_k)$ satisfying $c(x) \leq \sum_i c_i(x)$ for all $x \in D$.*

As in the case of Theorem 3.4, this follows from an analogous triangle inequality.

Lemma 3.5. *Let A and B be square matrices indexed by D. If $c_A(\cdot) \in \Gamma(A)$ and $c_B(\cdot) \in \Gamma(B)$, there exists a $c(\cdot) \in \Gamma(A + B)$ satisfying $c(x) \leq c_A(x) + c_B(x)$ for all $x \in D$.*

Proof. Since $c_A(\cdot) \in \Gamma(A)$ and $c_B(\cdot) \in \Gamma(B)$, there exist vectors that satisfy the following constraints:

$$\sum_{j:x_j \neq y_j} \langle u^A_{xj} | v^A_{yj} \rangle = (A)_{xy} \quad \text{with} \quad c_A(x) = \max\{\sum_j \||u^A_{xj}\rangle\|^2, \sum_j \||v^A_{xj}\rangle\|^2\} \quad \text{and}$$

$$\sum_{j:x_j \neq y_j} \langle u^B_{xj} | v^B_{yj} \rangle = (B)_{xy} \quad \text{with} \quad c_B(x) = \max\{\sum_j \||u^B_{xj}\rangle\|^2, \sum_j \||v^B_{xj}\rangle\|^2\}.$$

Now define $|u_{xj}\rangle := |1\rangle |u^A_{xj}\rangle + |2\rangle |u^B_{xj}\rangle$ and $|v_{xj}\rangle := |1\rangle |v^A_{xj}\rangle + |2\rangle |v^B_{xj}\rangle$. We claim that these vectors are feasible for $\gamma(A+B)$. The constraints are satisfied since

$$\sum_{j:x_j \neq y_j} \langle u_{xj}|v_{yj}\rangle = \sum_{j:x_j \neq y_j} \langle u^A_{xj}|v^A_{yj}\rangle + \sum_{j:x_j \neq y_j} \langle u^B_{xj}|v^B_{yj}\rangle = (A)_{xy} + (B)_{xy} = (A+B)_{xy}.$$

The cost function for this solution, $c(x)$, is $\max\{\sum_j \||u_{xj}\rangle\|^2, \sum_j \||v_{xj}\rangle\|^2\}$, which gives

$$c(x) = \max\{\sum_j \||u^A_{xj}\rangle\|^2 + \||u^B_{xj}\rangle\|^2, \sum_j \||v^A_{xj}\rangle\|^2 + \||v^B_{xj}\rangle\|^2\} \leq c_A(x) + c_B(x),$$

showing that $c(x) \leq c_A(x) + c_B(x)$ for all $x \in D$ as claimed. □

In our applications, we will encounter algorithms that also output their input, i.e., accept as input $f(x)$ and output $(f(x), g(x))$. Note that the Gram matrix of the function $h(x) = (f(x), g(x))$ is merely $H = F \circ G$, defined as $H_{xy} = F_{xy}G_{xy}$.

Such an algorithm can either be thought of as a single quantum algorithm that accepts $f(x) \in E$ as input and outputs $(f(x), g(x))$ or as a collection of algorithms A_e for each $e \in E$, such that algorithm $A_{f(x)}$ requires no input and outputs $(f(x), g(x))$ on oracle input x. These are equivalent viewpoints, since in one direction you can construct the algorithms A_e from A by hardcoding the value of e and in the other direction, we can read the input e and call the appropriate A_e as a subroutine and output $(e, A_e(x))$. Additionally, if the algorithm $A_{f(x)}$ makes $q(x)$ queries on oracle input x, the algorithm A we constructed accepts $f(x)$ as input, outputs $(f(x), g(x))$, and makes $q(x)$ queries on oracle input x. While intuitive for quantum algorithms, we need to establish this rigorously for cost functions.

Theorem 3.6. *Let $f, g : D \to E$ be functions with Gram matrices F and G. For any $e \in E$, let $f^{-1}(e) = \{x : f(x) = e\}$. For every $e \in E$, let $c_e : f^{-1}(e) \to \mathbb{R}$ be a feasible cost function for $\gamma(J - G_e)$, where G_e denotes the matrix G restricted to those x that satisfy $f(x) = e$. Then there exists a $c(\cdot) \in \Gamma(F - F \circ G)$, such that $c(x) = c_{f(x)}(x)$.*

Proof. We build a feasible solution for $\gamma(F - F \circ G)$ out of the feasible solutions for $\gamma(J - G_e)$. We have vectors $\{|u^e_{xj}\rangle, |v^e_{yj}\rangle\}$ for each $e \in E$ that satisfy $\sum_{j:x_j \neq y_j} \langle u^e_{xj}|v^e_{yj}\rangle = (J - G_e)_{xy}$ for all $x, y \in f^{-1}(e)$ and $c_e(x) = \max\{\sum_j \||u^e_{xj}\rangle\|^2, \sum_j \||v^e_{xj}\rangle\|^2\}$.

Let $|u_{xj}\rangle = |f(x)\rangle |u^{f(x)}_{xj}\rangle$ and $|v_{xj}\rangle = |f(x)\rangle |v^{f(x)}_{xj}\rangle$. This is feasible for $\gamma(F - F \circ G)$, since

$$\sum_{j:x_j \neq y_j} \langle u_{xj}|v_{yj}\rangle = \sum_{j:x_j \neq y_j} \langle f(x)|f(y)\rangle \langle u^{f(x)}_{xj}|v^{f(y)}_{yj}\rangle = F_{xy} \circ (J - G_{f(x)})_{xy} = F_{xy} - (F \circ G)_{xy}.$$

Note that when $f(x) \neq f(y)$, the value of $\sum_{j:x_j \neq y_j} \langle u^{f(x)}_{xj}|v^{f(y)}_{yj}\rangle$ is unknown, but this only happens when $F_{xy} = 0$, which makes the term 0. Lastly, the cost function for this solution is $\max\{\sum_j \||u_{xj}\rangle\|^2, \sum_j \||v_{xj}\rangle\|^2\}$, which is $\max\{\sum_j \||u^{f(x)}_{xj}\rangle\|^2, \sum_j \||v^{f(x)}_{xj}\rangle\|^2\} = c_{f(x)}(x)$. □

3.4.3 Algorithm analysis

We can now return to computing the query complexity of Algorithm 3.3. Using the same notation as in the beginning of this section, for any $x \in \mathcal{C}$, we define $r(x)$ to be the number of times the repeat loop is run in Algorithm 3.3 for oracle input x assuming all subroutines have no error. Similarly, let $p_1(x), p_2(x), \ldots p_{r(x)}(x)$ be the relative positions of the first disagreement found in each run of the loop, assuming the algorithms work exactly. Note that $p_1(x), p_2(x), \ldots p_{r(x)}(x)$ together uniquely specify x. Let $r = \max_x r(x)$.

We define r functions $f_1, \ldots, f_r$ as $f_1(x) := p_1(x)$, $f_2(x) := (p_1(x), p_2(x))$, and so on until $f_r(x) := (p_1(x), \ldots, p_r(x))$, where $p_k(x) := 0$ if $k > r(x)$. Thus if P_i are the Gram matrices of the functions p_i, then $F_1 = P_1, F_2 = P_1 \circ P_2, \ldots, F_r = P_1 \circ P_2 \circ \cdots \circ P_r$.

We will now construct a solution for $\gamma(J - F_r)$, using solutions for the intermediate functions f_i. From Theorem 3.5 we know that we only need to construct solutions for $\gamma(J - F_1), \gamma(F_1 - F_2), \ldots, \gamma(F_{r-1} - F_r)$. From Theorem 3.6 we know that instead of constructing a solution for $\gamma(F_k - F_{k+1})$, which is $\gamma(F_k - F_k \circ P_{k+1})$, we can construct several solutions, one for each value of $f_k(x)$. More precisely, let $f_k : D \to E_k$; then we can construct solutions for $\gamma(J - P^e_{k+1})$ for all $e \in E_k$, where P^e_{k+1} is the matrix P_{k+1} restricted to x that satisfy $f_k(x) = e$.

For any k, the problem corresponding to $\gamma(J - P^e_{k+1})$ is just the problem of finding the first disagreement between x and a known string, which is the essentially the find-first-one function. This has a solution with cost function $O(\sqrt{f(x)})$, which in this case is $O(\sqrt{p_{k+1}(x)})$.

Theorem 3.7 (find-first-one function)**.** *Let f be the function that outputs the smallest i such that $x_i = 1$ and outputs $N + 1$ if $x = 0^N$. Let F be its Gram matrix. Then there is a $c(\cdot) \in \Gamma(J - F)$ such that $c(x) = O(\sqrt{f(x)})$.*

Proof. Let $a_k = k^{-1/4}$ and $b_k = 1/a_k = k^{1/4}$. Define $|u_{xj}\rangle = |v_{xj}\rangle$ as the following.

$$|u_{xj}\rangle = |v_{xj}\rangle = \begin{cases} a_j, & \text{if } j < f(x) \\ b_{f(x)}, & \text{if } j = f(x) \\ 0, & \text{if } j > f(x). \end{cases}$$

This is a feasible solution for $\gamma(J - F)$. Since the constraints are symmetric in x and y, there are two cases: either $f(x) < f(y)$ or $f(x) = f(y)$. For the first case, $\sum_{j:x_j \neq y_j} \langle u_{xj}|v_{yj}\rangle = \sum_{j=f(x)} \langle u_{xj}|v_{yj}\rangle = a_{f(x)} b_{f(x)} = 1$, since x and y agree on all positions before $f(x)$. For the second case, $\sum_{j:x_j \neq y_j} \langle u_{xj}|v_{yj}\rangle = 0$, since the only bits that x and y disagree on appear after position $f(x) = f(y)$. To compute the cost function, note that $c(0^N) = \sum_{k=1}^N a_k^2 = O(\sqrt{N}) = O(\sqrt{f(0^N)})$. For all other x, $c(x) = \sum_{k=1}^{f(x)-1} a_k^2 + b_{f(x)}^2 = \sum_{k=1}^{f(x)-1} k^{-1/2} + \sqrt{f(x)} = O(\sqrt{f(x)})$. □

Our function is different from this one in two ways. First, we wish to find the first disagreement with a fixed string s instead of the first 1. This change does not affect the Gram matrix or the SDP. Second, we are looking for a disagreement according to an order σ, not from left to right. This can be fixed by replacing j with $\sigma(j)$ in the definition of the vectors above.

This shows that for any k, there is a feasible cost function for $\gamma(J - P^e_{k+1})$ with cost $c(x) = O(\sqrt{p_{k+1}(x)})$ for any x that satisfies $f_k(x) = e$. Using Theorem 3.6, we get that for any k there is a $c_k(\cdot) \in \Gamma(F_k - F_k \circ P_{k+1})$ with $c_k(x) = O(\sqrt{p_{k+1}(x)})$ for all $x \in D$. Finally, using Theorem 3.5, we have a $c(\cdot) \in \Gamma(J - F_r)$ with cost $c(x) = O(\sum_{i=1}^{r} \sqrt{p_i(x)}) = O(\sum_{i=1}^{r(x)} \sqrt{p_i(x)})$.

Since the function $f_r(x)$ uniquely determines x, we have a feasible cost function for oracle identification with cost $O(\sum_{i=1}^{r(x)} \sqrt{p_i(x)})$, subject to the constraints of Lemma 3.4, which we have already solved. This upper bound for the filtered γ_2 norm can be converted to a quantum query complexity upper bound, which shows that the query complexity of our algorithm is

$$O\left(\sqrt{\frac{N \log M}{\log(N/\log M) + 1}}\right) \quad \text{and} \quad O(\sqrt{M}).$$

Along with the lower bound proved in Section 3.2, this yields the main result.

Theorem 3.2. *For $N < M \leq 2^N$, $Q(\mathsf{OIP}(M, N)) = \Theta\left(\sqrt{\frac{N \log M}{\log(N/\log M)+1}}\right)$.*

3.4.4 Nontechnical summary of techniques

Since this section was particularly technical, let us quickly summarize what has been achieved. We studied the situation where we have k bounded-error algorithms $A_1, A_2, \ldots, A_k$ such that algorithm A_1 accepts as input $f_0(x)$ and outputs $(f_0(x), f_1(x))$, and so on. In general A_i accepts as input $(f_0(x), f_1(x), \ldots, f_{i-1}(x))$ and outputs $(f_0(x), f_1(x), \ldots, f_i(x))$. This situation models what we call a sequential composition of algorithms, where the output of one algorithm is required as the input of the next. We have assumed without loss of generality that each algorithm also outputs its own input, which is why A_1 outputs $(f_0(x), f_1(x))$ and not just $f_1(x)$.

Now we can compute $f_k(x)$ by sequentially composing these algorithms. If the expected query complexity of algorithm A_i on input x is $c_i(x)$, ideally we would expect to compute $f_k(x)$ with worst-case complexity $O(\max_x \sum_i c_i(x))$. However, this straightforward composition does not work since these are bounded-error algorithms. We could reduce their error to an inverse polynomial in k, but this would add an additional $\log k$ factor.

The main result of this section is that it is possible to achieve the worst-case upper bound $O(\max_x \sum_i c_i(x))$ if there exists a feasible solution the filtered γ_2-norm SDP (Definition 3.1) with costs $c_i(x)$ for each algorithm A_i. In our application, we had to show that this is possible for the find-first-one function. In the application to Boolean matrix multiplication in Section 4.4.4, we will need to show this for a related function.

3.5 Application to quantum learning theory

The oracle identification problem has also been studied in quantum learning theory with the aim of characterizing $Q(\mathsf{OIP}(\mathcal{C}))$. The algorithms and lower bounds studied apply to arbitrary sets $\mathcal{C}$, not just to the class of sets of a certain size, as in the previous sections. We show that Algorithm 3.3 also performs well for any set $\mathcal{C}$, outperforming the best known algorithm. The known upper and lower bounds for this problem are in terms of a combinatorial parameter $\hat{\gamma}^{\mathcal{C}}$, defined by Servedio and Gortler. They showed that for any $\mathcal{C}$, $Q(\mathsf{OIP}(\mathcal{C})) = \Omega(\sqrt{1/\hat{\gamma}^{\mathcal{C}}} + \frac{\log M}{\log N})$ [SG04]. Later, Atıcı and Servedio showed that $Q(\mathsf{OIP}(\mathcal{C})) = O(\sqrt{1/\hat{\gamma}^{\mathcal{C}}} \log M \log\log M)$ [AS05]. We show that our algorithm, without modification, performs strictly better than this. To do this, we must first define $\hat{\gamma}^{\mathcal{C}}$.

Definition of the combinatorial parameter $\hat{\gamma}^{\mathcal{C}}$

The definition of this parameter is involved and we break it up into three steps. Let $\mathcal{C} \subseteq \{0,1\}^N$ be a set of N-bit strings. We define

$$\gamma_i^S := \min_{b\in\{0,1\}} \frac{|\{x \in S : x_i = b\}|}{|S|}, \text{ where } i \in [N] \text{ and } S \subseteq \mathcal{C}. \tag{3.11}$$

This parameter is equal to the fraction of strings in S that disagree with the majority string at the i^{th} bit. So if we know that the oracle string x is contained in S and we query bit i, there are two possibilities: If x_i disagrees with the majority string, we have eliminated at least half the strings from S. If x_i agrees with the majority string, we have eliminated $\gamma_i^S|S|$ strings from S. In either case, the remaining set has size at most $(1-\gamma_i^S)|S|$, since $\gamma_i^S \leq 1/2$.

Next we define

$$\gamma^S := \max_{i\in[N]} \gamma_i^S, \text{ where } S \subseteq \mathcal{C}. \tag{3.12}$$

The parameter γ^S is the fraction of strings eliminated from S by querying the best possible i, i.e., the one that eliminates the largest fraction of strings, independent of the value of x_i. In Lemma 3.3 we called this bit the most informative bit.

Lastly, we define

$$\hat{\gamma}^{\mathcal{C}} := \min_{C'\subseteq\mathcal{C}, |C'|\geq 2} \gamma^{C'}. \tag{3.13}$$

Algorithm analysis

Informally, $\hat{\gamma}^{\mathcal{C}}$ is the largest $\alpha \leq 1/2$, such that for any set $S \subseteq \mathcal{C}$, if we know that x belongs to S, there is a bit of x that can be queried such that the size of the set of strings consistent with the answer to this query is at most $(1-\alpha)|S|$, no matter what the oracle responds. This ensures

that if we query the oracle with the permutation of Lemma 3.3, which was chosen to maximize the number of strings eliminated with a query, each query reduces the size of S by a factor of $(1-\hat{\gamma}^{\mathcal{C}})$.

For example, in the first run of the loop in Algorithm 3.3, we search for a disagreement between x and s according to the order of Lemma 3.3, which orders bits by how informative they are, i.e., it picks the bit that maximizes γ^S. When a disagreement is found at position p_1, we have learned the values of the first p_1 bits. Each bit learned decreases the size of the set S by a factor of $(1-\gamma^S)$, where S is the set of strings consistent with all the bits learned so far. But we know that $(1-\gamma^S) \leq (1-\hat{\gamma}^{\mathcal{C}})$, which means we have decreased the size of S by a factor of at least $(1-\hat{\gamma}^{\mathcal{C}})^{p_1}$ by learning p_1 bits.

This adds an extra constraint to the optimization problem appearing in Lemma 3.4 of the form $M\prod_i^r (1-\hat{\gamma}^{\mathcal{C}})^{p_i} \geq 1$, since there will be one $x \in S$ at the end. From this constraint we get $(\sum_i p_i)\log(1-\hat{\gamma}^{\mathcal{C}}) \geq -\log M$. Using $\log(1-\hat{\gamma}^{\mathcal{C}}) \leq -\hat{\gamma}^{\mathcal{C}}$ gives $\sum_i p_i \leq \frac{\log M}{\hat{\gamma}^{\mathcal{C}}}$.

We may now replace the constraint $\sum_i p_i \leq N$ with $\sum_i p_i \leq \frac{\log M}{\hat{\gamma}^{\mathcal{C}}}$ in the optimization problem of Lemma 3.4. This inequality also implies $p_i \leq \frac{\log M}{\hat{\gamma}^{\mathcal{C}}}$ and $r \leq \frac{\log M}{\hat{\gamma}^{\mathcal{C}}}$. Thus we may simply replace all occurrences of N with $\frac{\log M}{\hat{\gamma}^{\mathcal{C}}}$ in Lemma 3.4. This shows that $Q(\mathsf{OIP}(\mathcal{C})) = O\left(\sqrt{\frac{1/\hat{\gamma}^{\mathcal{C}}}{\log 1/\hat{\gamma}^{\mathcal{C}}}}\log M\right)$, which resolves a conjecture of Hunziker et al. [HMP+10, Conjecture 2]. More generally we have the following.

Theorem 3.8. *Algorithm 3.3 provides the following upper bound on the quantum query complexity of* $\mathsf{OIP}(\mathcal{C})$*:*

$$Q(\mathsf{OIP}(\mathcal{C})) = O\left(\min\left\{\sqrt{\frac{1/\hat{\gamma}^{\mathcal{C}}}{\log 1/\hat{\gamma}^{\mathcal{C}}}}\log M, \sqrt{\frac{N\log M}{\log(N/\log M)+1}}\right\}\right). \tag{3.14}$$

This shows that Algorithm 3.3 performs well on any set $\mathcal{C}$, since we know from [SG04] that

$$Q(\mathsf{OIP}(\mathcal{C})) = \Omega\left(\sqrt{1/\hat{\gamma}^{\mathcal{C}}} + \frac{\log M}{\log N}\right). \tag{3.15}$$

From this lower bound we see that Algorithm 3.3 makes

$$O\left(\frac{Q(\mathsf{OIP}(\mathcal{C}))^2}{\sqrt{\log Q(\mathsf{OIP}(\mathcal{C}))}}\log N\right) \tag{3.16}$$

queries, which means for any set $\mathcal{C}$ our algorithm is at most about quadratically worse than the best algorithm for the set $\mathcal{C}$.

3.6 Discussion and open problems

Some readers may wonder if the composition theorem could be avoided by using a standard argument about expected running times (or query complexity), which has the following form: Given k Las Vegas algorithms with expected running times $t_1, \ldots, t_k$, running these algorithms in succession will yield an algorithm with expected running time $\sum_i t_i$ by the linearity of expectation. If we now terminate the algorithm after (say) 5 times its expected running time, then by Markov's inequality we have a bounded-error algorithm with worst-case running time $O(\sum_i q_i)$. However, to use this argument the individual algorithms need to be zero error. If the algorithms are merely bounded error, then the final answer may be incorrect even if one of the k bounded-error algorithms errs. In our applications, oracle identification and Boolean matrix multiplication, we use a subroutine to find the first marked 1 in a string. This algorithm has bounded error since it is too expensive to verify (with zero error) that a given 1 is indeed the first 1 in a string.

Our composition theorem only works for solutions of the filtered γ_2-norm SDP, not for quantum query complexity itself. While this is sufficient for our application, it would be interesting to know if bounded-error quantum algorithms with input-dependent query complexities can be composed in general without incurring log factors.

Open Problem 3.1. Suppose we have k bounded-error quantum algorithms $\{A_i\}$, such that A_1 outputs $f_1(x)$, A_2 accepts as input $f_1(x)$ and outputs $f_2(x)$, and so on, until A_k, which accepts $f_{k-1}(x)$ as input and outputs $f_k(x)$. Let $q_i(x)$ be the expected query complexity of algorithm A_i on input x. From these algorithms can we construct a bounded-error quantum algorithm A that computes $f_k(x)$ with expected query complexity $O(\sum_i q_i(x))$ on input x?

While the query complexity of oracle identification in terms of M and N has been fully characterized, finding an optimal quantum algorithm for $\mathsf{OIP}(\mathcal{C})$ remains open. The corresponding problem for classical query complexity is also open.

Open Problem 3.2. Can we characterize (up to constants) the classical or quantum query complexity of $\mathsf{OIP}(\mathcal{C})$ for every set $\mathcal{C}$?

In the previous section we showed that our algorithm is almost optimal for all sets, since for any set $\mathcal{C}$ it performs at most about quadratically worse than the best quantum algorithm for $\mathcal{C}$. This raises the possibility that our algorithm is optimal for every set $\mathcal{C}$, but this is not the case. There exist sets for which our algorithm performs suboptimally. This follows from the fact that our algorithm can be made fully classical by replacing the search subroutine with a classical search subroutine, which would increase the query complexity by at most a quadratic factor. Thus the square of the query complexity of our algorithm is an upper bound on the classical query complexity of the problem. Consequently, our algorithm is suboptimal for any $\mathsf{OIP}(\mathcal{C})$ for which there is a super-quadratic separation between quantum and classical query complexity, such as the Bernstein–Vazirani problem [BV97] for which we have a 1 vs $\log N$ separation.

Chapter 4

Matrix multiplication

Chapter summary: We study the quantum query complexity of matrix multiplication and related problems. We survey and extend known results on the quantum query complexity of matrix multiplication and matrix product verification over rings, semirings and the Boolean semiring in particular. We also study relationships between these problems and other problems studied in quantum query complexity, such as the triangle problem and the graph collision problem.

Our main result is an output-sensitive quantum algorithm for Boolean matrix multiplication that multiplies two $n \times n$ Boolean matrices with query complexity $O(n\sqrt{\ell})$, where ℓ is the number of nonzero entries in the output matrix. Our algorithm is based on the observation that Boolean matrix multiplication can be reduced to several instances of the graph collision problem and that these instances of graph collision correspond to dense graphs. For these instances we develop a quantum algorithm that is more efficient than known algorithms for the general graph collision problem. We also prove a matching lower bound for Boolean matrix multiplication for all $\ell \leq \epsilon n^2$, for any constant $\epsilon < 1$.

Section 4.4 of this chapter contains results from the following paper:

[JKM12] Stacey Jeffery, Robin Kothari, and Frédéric Magniez. Improving Quantum Query Complexity of Boolean Matrix Multiplication Using Graph Collision. In *Automata, Languages, and Programming*, volume 7391 of *Lecture Notes in Computer Science*, pages 522–532. Springer, 2012.

4.1 Introduction

Motivation

Matrix multiplication is an old and well-studied problem in (classical) algorithms and complexity theory. The problem is to compute the product C of two $n \times n$ matrices A and B, where $C_{ij} := \sum_k A_{ik}B_{kj}$. The straightforward algorithm for matrix multiplication that computes each entry separately using its definition runs in time $O(n^3)$. In 1969, Strassen published an algorithm that runs in time $O(n^{2.807})$ [Str69], showing that the straightforward approch was suboptimal. Since then there have been many improvements to the running time and the complexity of matrix multiplication remains an area of active research.

Surprisingly, the matrix product verification problem can be solved faster. In this problem, we are given three matrices A, B, and C, and we have to check if $C = AB$. In 1979, Freivalds [Fre79] presented an optimal $O(n^2)$ time bounded-error probabilistic algorithm to solve this problem, showing that matrix product verification can be simpler than matrix multiplication.

Since these problems are of fundamental interest, we study them in the quantum setting. We will primarily be interested in the matrix multiplication problem, denoted MM, and the matrix product verification problem, denoted MPV. In the course of their study we will see that these problems have surprising connections to other problems studied in quantum query complexity, such as the triangle problem and the graph collision problem.

While matrices with real number entries are commonly encountered in scientific applications, matrices have been studied over many different algebraic structures, ranging from complex numbers and finite fields to integers and polynomials. These examples motivate the study of matrix multiplication over rings. Another class of matrices that is commonly encountered in theoretical computer science and graph theory is Boolean matrices. The Boolean semiring is the set $\{0, 1\}$ with $\vee$ and $\wedge$ as the addition and multiplication operations respectively. Other examples of semirings are the natural numbers, the nonnegative reals, the tropical semiring, and the $(\max, \min)$-semiring. We will not discuss matrices over algebraic structures more general than semirings (e.g., near-semirings), although it is possible to do so and some results extend to these structures.

In this chapter we primarily study matrix multiplication over arbitrary rings and the Boolean semiring. We choose to study rings in general because we are not aware of any algorithm that exploits the structure of any specific ring (e.g., real or complex numbers). On the other hand, algorithms for Boolean matrix multiplication do exploit properties of the Boolean semiring, making it fruitful to study them separately.

Boolean matrix multiplication also has many applications and surprising connections. The Boolean matrix product has a natural interpretation in terms of graphs: If A and B are Boolean matrices, and G_A and G_B are the graphs corresponding to these adjacency matrices, then the

Boolean matrix product AB is the adjacency matrix of the graph in which two vertices i and j are adjacent if and only if we can go from i to j by taking the first step on graph G_A and the second step on graph G_B. As another example, note that the transitive closure of an n-vertex graph G with adjacency matrix A is $\sum_{i=0}^{n} A^i$. More surprisingly, natural modifications of the Boolean matrix product verification problem are equivalent to the triangle problem and the graph collision problem, two well-studied problems in quantum query complexity.

Definitions

Before we describe the problems we study formally, let us define the algebraic structures we will encounter.

Definition 4.1 (Rings and semirings)**.** A semiring $\mathcal{S} = (S, +, \cdot)$ is a set S with two binary operations $+ : S \times S \to S$ and $\cdot : S \times S \to S$ called addition and multiplication respectively, and two distinguished elements called $0_{\mathcal{S}}$ and $1_{\mathcal{S}}$, such that the following properties hold:

1. $(S, +)$ is a commutative monoid with identity element $0_{\mathcal{S}}$: Addition is associative, commutative, and $\forall a \in S$, $a + 0_{\mathcal{S}} = a$.
2. $(S, \cdot)$ is a monoid with identity element $1_{\mathcal{S}}$: Multiplication is associative and $\forall a \in S$, $a \cdot 1_{\mathcal{S}} = 1_{\mathcal{S}} \cdot a = a$.
3. Multiplication distributes over addition: $\forall a, b, c \in S$, $a \cdot (b + c) = a \cdot b + a \cdot c$ and $(b + c) \cdot a = b \cdot a + c \cdot a$.
4. $0_{\mathcal{S}}$ is the multiplicative annihilator of S: For all $a \in S$, $a \cdot 0_{\mathcal{S}} = 0_{\mathcal{S}} \cdot a = 0_{\mathcal{S}}$.
5. S contains more than one element, which is equivalent to $0_{\mathcal{S}} \neq 1_{\mathcal{S}}$.

A semiring $\mathcal{S} = (S, +, \cdot)$ is a ring if $(S, +)$ is an Abelian group, i.e., if every element in S has an additive inverse.

For example, the set $S = \{0, 1\}$, with $+ = \vee$ (logical OR), and $\cdot = \wedge$ (logical AND), is a semiring known as the Boolean semiring. Integers, real numbers, complex numbers, and finite fields with the usual notions of addition and multiplication are rings. When it is clear from context, we drop the subscript $\mathcal{S}$ and use 0 and 1 for $0_{\mathcal{S}}$ and $1_{\mathcal{S}}$ respectively.

Matrix multiplication is defined over semirings in the natural way. The product of an $m \times n$ matrix A and an $n \times p$ matrix B over $\mathcal{S}$ is an $m \times p$ matrix C, where $C_{ij} := \sum_{k=1}^{n} A_{ik} \cdot B_{kj}$.

Over any semiring, we use $\mathbb{1}$ to denote the identity matrix, which has every diagonal entry equal to 1 and every off-diagonal entry equal to 0, and $\mathbb{0}$ to denote the zero matrix, which has all entries equal to 0. Similarly, we use $\vec{1}$ to denote the all-ones vector and $\vec{0}$ to denote the all-zeros vector.

Problem description

We study the quantum query complexity of matrix multiplication and related problems in the natural query model for this problem in which we have query access to the entries of the input matrices. For example, in the matrix product verification problem we have access to an oracle for A, B, and C and we have to decide if $AB = C$. An oracle for a matrix A accepts as input (i, j) and outputs A_{ij}. As usual, quantum algorithms have access to a unitary version of this oracle.

The primary problems we study are the matrix multiplication problem, denoted MM, and the matrix product verification problem, denoted MPV. We also study the problem of computing matrix–vector products: we use MvM to denote the matrix–vector multiplication problem and MvPV to denote the matrix–vector product verification problem.

For each of these four problems, we also define a simpler version of the problem where the first input matrix is known and part of the problem specification. For example, in the matrix multiplication problem, MM, we are given oracle access to 2 matrices A and B, and we have to output AB. In the version of the problem where A is known, we are given oracle access to B and have to compute AB. We denote this problem MM^A to indicate that the matrix A is part of the specification of the problem itself and not part of the input. A problem that will appear often in reductions is the problem of verifying if the sum of a vector equals a specified value. We refer to this as the vector sum verification problem, denoted vSV.

Since we study these problems for the special case when $\mathcal{S}$ is the Boolean semiring in depth in Section 4.3, we denote the Boolean versions of these problems by appending B before the name of the problem. These problems are defined more formally below.

Definition 4.2. We define the following oracular problems, where $\mathcal{S} = (S, +, \cdot)$ is a semiring, $A, B, C \in S^{n\times n}$, $v, w \in S^n$, and $s \in S$.

- $\mathsf{MM}_{\mathcal{S}}$ (Matrix Multiplication). Input: A, B. Output AB.
- $\mathsf{MPV}_{\mathcal{S}}$ (Matrix Product Verification). Input: A, B, C. Output: Is $AB = C$?
- $\mathsf{MvM}_{\mathcal{S}}$ (Matrix–vector Multiplication). Input: A, v. Output: Av.
- $\mathsf{MvPV}_{\mathcal{S}}$ (Matrix–vector Product Verification). Input: A, v, w. Output: Is $Av = w$?

Similarly, we define the following oracular problems where A is part of the problem specification:

- $\mathsf{MM}^A_{\mathcal{S}}$ (Matrix Multiplication). Input: B. Output AB.
- $\mathsf{MPV}^A_{\mathcal{S}}$ (Matrix Product Verification). Input: B, C. Output: Is $AB = C$?
- $\mathsf{MvM}^A_{\mathcal{S}}$ (Matrix–vector Multiplication). Input: v. Output: Av.

- $\mathsf{MvPV}^A_{\mathcal{S}}$ (Matrix–vector Product Verification). Input: v, w. Output: Is $Av = w$?

We also define the following vector problem:

- $\mathsf{vSV}_{\mathcal{S}}$ (vector Sum Verification). Input: v, s. Output: Is $\sum_{i=1}^{n} v_i = s$?

When $\mathcal{S}$ is the Boolean semiring, we prefix B to the problem name instead, e.g., BMM, BvSV, and BMPV^A.

One may wonder why we choose to study so many variants of the problem. Studying the matrix multiplication and product verification problems naturally leads to studying matrix–vector multiplication and product verification. The version of these problems where A is fixed arises in several reductions, making it worthwhile to study independently. Fixing A also allows us to show strong relationships between problems. For example, we show in the next section that over any semiring we have $Q(\mathsf{MM}^A) = \Theta(nQ(\mathsf{MvM}^A))$. On the other hand, no such tight relationship can exist between MvM and MM over arbitrary semirings. Lastly, BMvPV^A is related to other interesting problems in quantum query complexity, such as graph collision and the query complexity of depth-2 AC^0 circuits.

Known results

According to Buhrman and Špalek [BŠ06], the quantum query complexity of matrix product verification was first studied (in an unpublished paper) by Ambainis, Buhrman, Høyer, Karpinski, and Kurur. They presented an algorithm for MPV with query complexity $O(n^{7/4})$. The first published work on the topic is due to Buhrman and Špalek [BŠ06], who noted that $Q(\mathsf{MPV}) = O(n^{5/3})$ over any ring using a generalization of Ambainis' element distinctness algorithm [Amb07]. This algorithm also works over semirings and more general structures. The algorithm can easily be cast in the quantum walk search framework of Magniez, Nayak, Roland and Santha [MNRS11] as explained in the survey by Santha [San08]. More interestingly, they showed how to make this algorithm time efficient over fields and integral domains. Their algorithm uses the same technique used by Freivalds [Fre79] to speed up matrix product verification and since Freivalds' technique also works over rings (see [Wil11]), their algorithm is also time efficient over arbitrary rings.

Buhrman and Špalek were also the first to study the quantum query complexity of Boolean matrix multiplication. They showed that $Q(\mathsf{BMPV}) = O(n^{3/2})$ and gave an output-sensitive algorithm for BMM. An output-sensitive algorithm is one whose complexity depends on the output. They showed that if the number of nonzero entries in the output is ℓ, then $Q(\mathsf{BMM}) = O(n^{3/2}\sqrt{\ell})$ [BŠ06, Section 6.2]. Their algorithm is quite simple: it repeatedly searches for nonzero entries in C. Since checking if a given entry in C is 1 requires $O(\sqrt{n})$ queries, we can search for a nonzero entry among all n^2 entries of C using $O(\sqrt{n^2} \times \sqrt{n})$ queries by composing Grover's

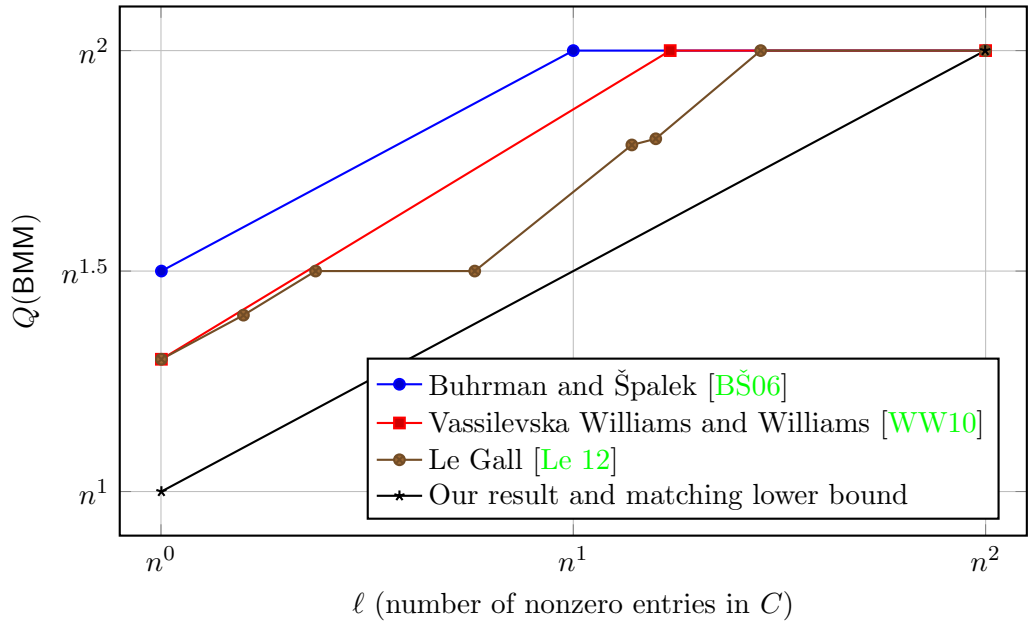

Figure 4.1: Output-sensitive quantum query complexity of Boolean matrix multiplication.

algorithm over a space of size n^2 with the checking subroutine. If there are ℓ nonzero entries, then we can find all of them with $O(\sqrt{n^2\ell} \times \sqrt{n}) = O(n^{3/2}\sqrt{\ell})$ queries, using the version of Grover's algorithm that finds all marked items.

This algorithm was later improved by Vassilevska Williams and Williams [WW10], who used a classical reduction relating Boolean matrix multiplication and triangle finding, and the triangle finding algorithm of Magniez, Santha and Szegedy [MSS07] to get a quantum algorithm for Boolean matrix multiplication with query complexity $\tilde{O}(\min\{n^{1.3}\ell^{17/30}, n^2+n^{13/15}\ell^{47/60}\})$. Their algorithm was then improved by Le Gall [Le 12]. These algorithms are depicted in Figure 4.1.

Our results

Our main result is an optimal output-sensitive algorithm for Boolean matrix multiplication that makes $O(n\sqrt{\ell})$ queries, where ℓ is the number of nonzero entries in the output. We also show a matching lower bound of $\Omega(n\sqrt{\ell})$ when $\ell \leq \epsilon n^2$ for any constant $\epsilon < 1$. The lower bound is based on the observation that if A were the identity matrix, matrix multiplication would be equivalent to learning all the entries of B. These results are presented in Section 4.4.

Our upper bound is based on two ideas: a connection between BMM and the graph collision problem and a new algorithm for graph collision that works better in our application. More precisely, we show how to reduce the Boolean matrix multiplication problem to several instances

of the graph collision problem. The instances of graph collision that arise in our reduction are very dense; the graphs have at most ℓ nonedges. We present a new graph collision algorithm that makes $O(\sqrt{n}+\sqrt{\ell})$ queries and solves graph collision on a graph with at most ℓ nonedges. These two ideas together yield the claimed upper bound. Unlike the previous best algorithms for BMM [WW10, Le 12], our algorithm does not use quantum walks. Our graph collision algorithm is based on Grover's algorithm and its variants, unlike graph collision algorithms used as a subroutine in triangle finding algorithms, which are based on Ambainis' quantum walk for element distinctness [Amb07].

Most of the content of Section 4.2 and Section 4.3, except results attributed to others, does not appear in the literature to the best of my knowledge. However, it is either folklore or relatively easy to show and presented here for completeness.

4.2 Matrix multiplication over rings and semirings

We have defined eight variants of the matrix multiplication problem in Section 4.1. We can visualize these problems arising as a result of three different ways of making the problem easier. First, we could merely verify the product instead of computing it. Second, we could compute a matrix–vector product instead of a matrix–matrix product. Third, we could make the first matrix free, i.e., make it part of the problem specification so that it requires no queries to learn. These three ways of modifying the problem give rise to eight potential problems. We first study these problems and how they relate to each other over semirings.

4.2.1 Matrix multiplication over semirings

Figure 4.2 depicts these problems and their relationships over arbitrary semirings.

The relationships depicted with dotted arrows in Figure 4.2 are obvious and follow from three simple observations: First, any multiplication problem is no easier than the corresponding product verification problem, since computing a value is no easier than verifying it. Second, problems about matrix–matrix products include matrix–vector products as a special case, and cannot be easier. Third, problems in which A is fixed and not part of the input are only easier than the corresponding problem where A must be queried.

Other relationships are not all as straightforward as these and often rely on nontrivial properties of quantum query complexity such as the direct sum (Theorem 1.1) and composition (Theorem 1.2) theorems. We now show that the claimed relationships are correct.

Theorem 4.1 (Matrix multiplication over semirings)**.** *The relationships between problems indicated in Figure 4.2 hold over all semirings. More precisely, for any pair of problems* X *and* Y*, if in*

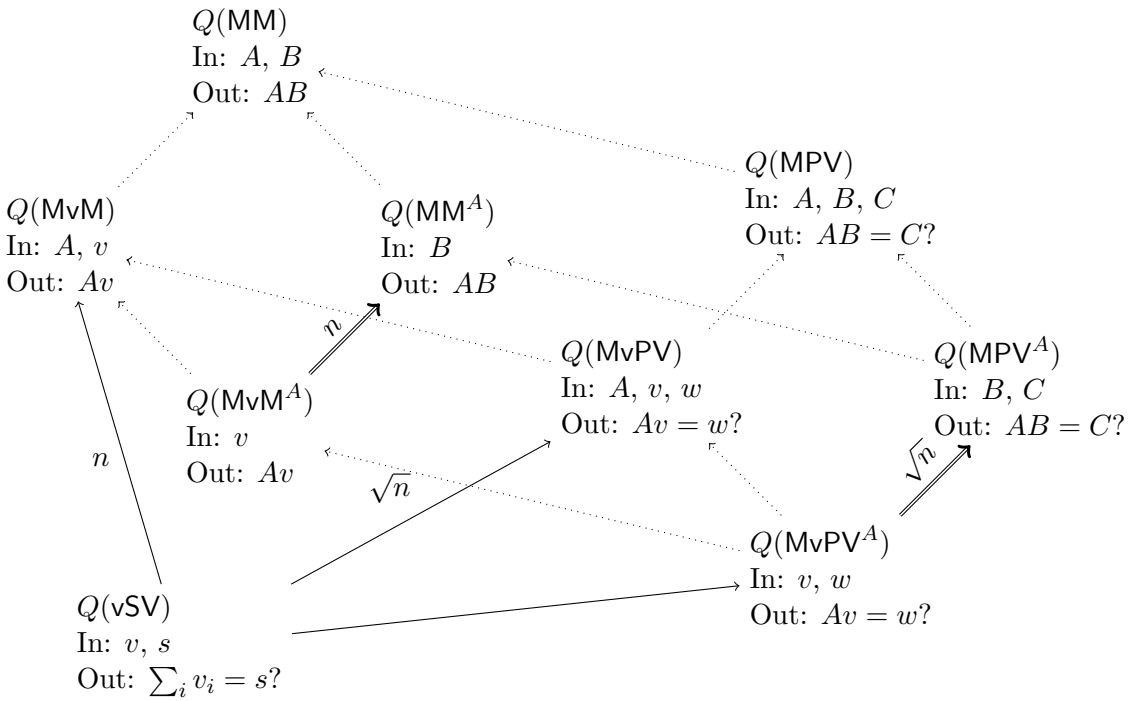

Figure 4.2: Quantum query complexity of matrix multiplication problems over a semiring. The subscript $\mathcal{S}$, for example in $\mathsf{MM}_{\mathcal{S}}$, has been suppressed for readability. Arrows between problems indicate relationships between their query complexities. $Q(\mathsf{X}) \longrightarrow Q(\mathsf{Y})$ and $Q(\mathsf{X}) \dashrightarrow Q(\mathsf{Y})$ mean $Q(\mathsf{Y}) = \Omega(Q(\mathsf{X}))$. $Q(\mathsf{X}) \xrightarrow{f(n)} Q(\mathsf{Y})$ means $Q(\mathsf{Y}) = \Omega(f(n)Q(\mathsf{X}))$. $Q(\mathsf{X}) \xRightarrow{f(n)} Q(\mathsf{Y})$ means $Q(\mathsf{Y}) = \Theta(f(n)Q(\mathsf{X}))$. Dotted arrows indicate obvious relationships.

Figure 4.2 we have $Q(\mathsf{X}) \longrightarrow Q(\mathsf{Y})$ *or* $Q(\mathsf{X}) \dashrightarrow Q(\mathsf{Y})$ *then* $Q(\mathsf{Y}) = \Omega(Q(\mathsf{X}))$*, if* $Q(\mathsf{X}) \xrightarrow{f(n)} Q(\mathsf{Y})$ *then* $Q(\mathsf{Y}) = \Omega(f(n)Q(\mathsf{X}))$*, and if* $Q(\mathsf{X}) \xRightarrow{f(n)} Q(\mathsf{Y})$ *then* $Q(\mathsf{Y}) = \Theta(f(n)Q(\mathsf{X}))$.

Proof. As explained, the relationships indicated using dotted arrows are obvious. Figure 4.2 has 5 other arrows between problems, and thus we have to establish 5 claims. Let us begin with the arrows originating from vSV.

We can embed an instance of vSV into an instance of MvPV^A by taking the first row of A to be the all-ones vector and the remaining rows be all-zeros vectors. Now the first entry of the matrix–vector product of A and v is the sum of all entries in v. Thus $Q(\mathsf{MvPV}^A) = \Omega(Q(\mathsf{vSV}))$. Note that the matrix A is fixed in this reduction. We can reduce more instances of vSV to MvPV when the matrix A is part of the input. By taking the vector in MvPV to be the all-ones vector, we can

verify the vector sum of each row of A. This is equivalent to verifying that n independent instances of vSV are all correct. This is the composition of the AND_n function with vSV, which requires $\Omega(\sqrt{n}Q(\mathsf{vSV}))$ queries (from Theorem 1.2), giving us $Q(\mathsf{MvPV}) = \Omega(\sqrt{n}Q(\mathsf{vSV}))$. Similarly, we can embed n instances of vSV into MvM. In this case, we have to compute the output of n instances of vSV, instead of merely verifying that they are correct. We can use Theorem 1.1 to show that the $Q(\mathsf{MvM}) = \Omega(nQ(\mathsf{vSV}))$. This establishes the correctness of all arrows originating from vSV in Figure 4.2.

We can now establish the correctness of the two double arrows ($\Rightarrow$) in Figure 4.2. First, note that the problem MPV^A is merely n independent instances of MvPV^A, since the i^{th} columns of matrix B and C define an input to MvPV^A for the same matrix A. Since MPV^A is the task of verifying if n instances of MvPV^A are correct, we get $Q(\mathsf{MPV}^A) = \Theta(\sqrt{n}Q(\mathsf{MvPV}^A))$ from Theorem 1.2. Similarly, MM^A is equivalent to computing the output of n independent instances of MvM^A. In this case we use Theorem 1.1 to conclude that $Q(\mathsf{MM}^A) = \Theta(nQ(\mathsf{MvM}^A))$. □

This shows that the relationships indicated in Figure 4.2 hold in general. We use these relationships to show lower bounds for rings below and for the Boolean semiring in Section 4.3.

4.2.2 Matrix multiplication over rings

First we recap known results about the query complexity of matrix multiplication over rings. As stated in Section 4.1, Buhrman and Špalek showed the following result [BŠ06].

Theorem 4.2 (Buhrman and Špalek)**.** *Matrix product verification of $n \times n$ matrices can be performed with $O(n^{5/3})$ queries over any ring or semiring.*

This upper bound is not known to be tight. Buhrman and Špalek showed a lower bound of $\Omega(n^{3/2})$ for MPV over $\mathbb{F}_2$, which we generalize to all rings. But this still leaves a gap between the known upper and lower bounds. Other than this problem, the query complexity of the other problems is easily characterized as shown in Figure 4.3.

To establish these query complexities, we start by showing that $Q(\mathsf{vSV}_{\mathcal{S}}) = \Theta(n)$ over any ring $\mathcal{S}$. Then from Theorem 4.1 we immediately have optimal lower bounds for all other problems. We then provide matching algorithms for all problems other than MPV, which is covered by Theorem 4.2.

Lemma 4.1. *$Q(\mathsf{vSV}_{\mathcal{S}}) = \Theta(n)$ over any ring $\mathcal{S} = (S, +, \cdot)$.*

Proof. Since the input size is $O(n)$, the upper bound follows. For the lower bound, we prove that given oracle access to n elements $v_1, \ldots, v_n$ from a ring S, determining if

$$\sum_{i=1}^{n} v_i = \sum_{i=1}^{\lceil n/2 \rceil} 1_{\mathcal{S}} \quad \text{or} \quad \sum_{i=1}^{n} v_i = \sum_{i=1}^{\lceil n/2 \rceil + 1} 1_{\mathcal{S}},$$

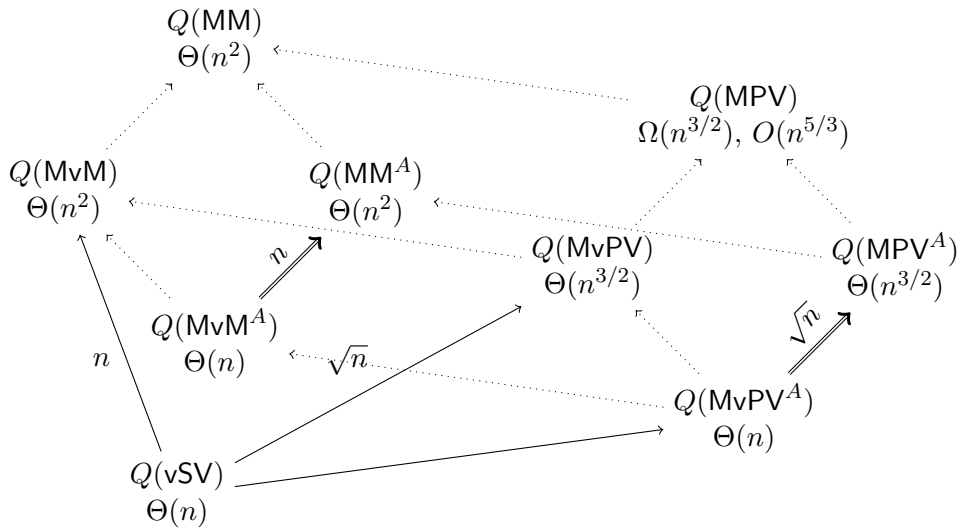

Figure 4.3: Quantum query complexity of matrix multiplication problems over a ring. The subscript $\mathcal{S}$ (e.g., $\mathsf{MM}_{\mathcal{S}}$) has been suppressed. Arrows between problems indicate relationships between their query complexities. $Q(\mathsf{X}) \longrightarrow Q(\mathsf{Y})$ and $Q(\mathsf{X}) \dashrightarrow Q(\mathsf{Y})$ mean $Q(\mathsf{Y}) = \Omega(Q(\mathsf{X}))$. $Q(\mathsf{X}) \xrightarrow{f(n)} Q(\mathsf{Y})$ means $Q(\mathsf{Y}) = \Omega(f(n)Q(\mathsf{X}))$. $Q(\mathsf{X}) \overset{f(n)}{\Longrightarrow} Q(\mathsf{Y})$ means $Q(\mathsf{Y}) = \Theta(f(n)Q(\mathsf{X}))$. Dotted arrows indicate obvious relationships.

promised that one of these is the case, requires $\Omega(n)$ queries. This problem is a special case of vSV, since testing if the sum of the vector equals $\sum_{i=1}^{\lceil n/2 \rceil} 1_{\mathcal{S}}$ solves this problem. The problem is well defined because we are working in a ring and therefore $\sum_{j=1}^{\lceil n/2 \rceil} 1_{\mathcal{S}}$ is not equal to $\sum_{j=1}^{\lceil n/2 \rceil + 1} 1_{\mathcal{S}}$. (This would not be true over the Boolean semiring, for example.)

Consider the set of strings $X \subseteq S^n$, such that $v \in X$ if exactly $\lceil n/2 \rceil$ entries of v are equal to $1_{\mathcal{S}}$ and the remaining entries are $0_{\mathcal{S}}$. Let Y be the set with $\lceil n/2 \rceil + 1$ entries equal to $1_{\mathcal{S}}$ and the rest equal to $0_{\mathcal{S}}$. Note that X contains yes instances of the problem, since $\sum_{i=1}^{n} v_i = \sum_{i=1}^{\lceil n/2 \rceil} 1_{\mathcal{S}}$, and Y contains no instances of the problem. This is exactly the set of hard instances for the $\lceil n/2 \rceil$-threshold problem or the $\mathsf{Majority}$ function. An $\Omega(n)$ lower bound for the majority function can be proved using the polynomial method [BBC+01] or the adversary method [Amb02, Theorem 5.1]. □

We can now show that all the bounds in Figure 4.3 are correct.

Theorem 4.3 (Matrix multiplication over rings)**.** *The query complexities stated in Figure 4.3 are correct.*

Proof. First we establish the lower bounds. From Lemma 4.1 we know that $Q(\mathsf{vSV}) = \Omega(n)$. All other lower bounds follow from this using the general relationships proved in Theorem 4.1.

Moving on to upper bounds, the simple observation that the query complexity of a problem is at most its input size establishes the claimed upper bound for most problems in Figure 4.3. The only remaining problems are MPV, MvPV, and MPV^A. Theorem 4.2 shows that $Q(\mathsf{MPV}) = O(n^{5/3})$. The complexity of MPV^A is characterized by the complexity of MvPV^A, since we know from Theorem 4.1 that $Q(\mathsf{MPV}^A) = \Theta(\sqrt{n}Q(\mathsf{MvPV}^A))$. Finally, $Q(\mathsf{MvPV}) = O(n^{3/2})$ since verifying any given inner product between a row of A and v requires at most $O(n)$ queries and there are n inner products to be verified. Using Theorem 1.2, $Q(\mathsf{MvPV})$ is upper bounded by the product of $Q(\mathsf{AND}_n)$ and $O(n)$, which gives $Q(\mathsf{MvPV}) = O(n^{3/2})$. □

The main open problem of this section is to determine $Q(\mathsf{MPV}_{\mathcal{S}})$ over rings. No better upper or lower bounds are known even for specific rings. Note that the lower bound cannot be improved using the original (positive-weights) adversary method due to the certificate complexity barrier [Sze03, Zha05, LM08, ŠS06].

Open Problem 4.1. What is the quantum query complexity of matrix product verification over rings? It is known that for any ring $\mathcal{S}$, $Q(\mathsf{MPV}_{\mathcal{S}}) = \Omega(n^{3/2})$ and $Q(\mathsf{MPV}_{\mathcal{S}}) = O(n^{5/3})$. Can we improve the upper or lower bound for specific rings?

4.3 Matrix multiplication over the Boolean semiring

We can now study these problems over the Boolean semiring. The Boolean versions of these problems are less understood and are closely related to other problems of interest, such as the graph collision problem (GC), the triangle problem (Triangle) and the query complexity of functions that can be represented by depth-2 linear-sized AC^0 circuits (LC^0_2). We begin by discussing these problems.

Triangle problem

Definition 4.3 (Triangle problem)**.** In the triangle problem or triangle finding problem, denoted Triangle, we are given query access to three $n \times n$ Boolean matrices A, B, and C and we have to decide if there exist $i, j, k \in [n]$ such that $A_{ik} = B_{kj} = C_{ij} = 1$.

In other words, let G be a tripartite graph on $3n$ vertices with tripartition $I \cup J \cup K$, where $|I| = |J| = |K| = n$, specified by biadjacency matrices A, B, and C, such that $A_{ik} = 1$ if $i \in I$ is adjacent to $k \in K$, $B_{kj} = 1$ if $k \in K$ is adjacent to $j \in J$, and $C_{ij} = 1$ if $i \in I$ is adjacent to $j \in J$. Given query access to A, B, and C, we have to determine if G contains K_3 (a triangle) as a subgraph, i.e., if there exists $(i, j, k) \in I \times J \times K$, such that $A_{ik} = B_{kj} = C_{ij} = 1$.

Our definition is slightly different from the usual definition of this problem. In our version, which may also be called the tripartite triangle finding problem, we are promised that the graph under consideration is tripartite. The usual definition of the problem allows arbitrary graphs. However, it is easily seen that the two problems are equivalent up to constant factors. On the one hand, tripartite triangle finding is clearly a special case of triangle finding. On the other hand, given one graph G in which we have to find a triangle, we can make three copies of the vertex set, called I, J, and K, and connect vertex $i \in I$ to vertex $j \in J$ if and only if (i, j) is an edge in G and similarly connect edges between I and K, and J and K. There are no edges within I, J, or K. This graph is tripartite and contains a triangle if and only if G does.

The triangle problem has been intensely studied in quantum query complexity and remains an important open problem. This question was first studied by Buhrman et al. [BDH+05], who gave an $O(n + \sqrt{nm})$ query algorithm for graphs with m edges. When $m = \Theta(n^2)$, this approach uses $O(n^{3/2})$ queries, which matches the performance of the simple algorithm that searches for a triangle over the potential $\binom{n}{3}$ triplets of vertices. This was later improved by Magniez, Santha, and Szegedy [MSS07] to $\tilde{O}(n^{1.3})$, and then by Magniez, Nayak, Roland, and Santha [MNRS11], who removed some log factors. Subsequently, the upper bound was improved by Belovs [Bel12] to $O(n^{35/27})$, and then by Lee, Magniez, and Santha to $O(n^{9/7})$ [LMS13]. More recently, the upper bound has been improved by Le Gall to $\tilde{O}(n^{1.25})$ [Le 14].

However, the best known lower bound for the triangle problem is only $\Omega(n)$ (by a simple reduction from the search problem). This is partly because one of the main lower bound techniques, the quantum adversary method of Ambainis [Amb02], cannot prove a better lower bound due to the certificate complexity barrier [Sze03, Zha05, LM08, ŠS06].

Open Problem 4.2. What is the quantum query complexity of the triangle problem? The best known lower and upper bounds are $Q(\mathsf{Triangle}) = \Omega(n)$ and $Q(\mathsf{Triangle}) = \tilde{O}(n^{1.25})$ [Le 14] respectively.

Graph collision

The next problem we introduce is the graph collision problem. This problem is used as a subroutine in several known triangle finding algorithms [MSS07, Bel12, LMS13, JKM13].

Definition 4.4 (Graph collision)**.** In the graph collision problem with a known $n \times n$ Boolean matrix A, denoted GC^A, we are given query access to two length-n Boolean vectors v and w, and we have to decide if there exist $i, j \in [n]$ such that $A_{ij} = v_j = w_i = 1$.

In other words, let G be a known bipartite graph on $2n$ vertices with adjacency matrix A. We are given query access to two Boolean vectors v and w, which we can interpret as indicating whether a particular vertex is marked or not, i.e., $v_i = 1$ indicates that vertex i is marked. We have to determine if there exists an edge between two marked vertices. An edge between two

marked vertices is called a "collision." Note that in this problem the graph G is known and not part of the input.

Our definition of the graph collision problem is also slightly different from the usual definition. In our version, which may be called the bipartite graph collision problem, we are promised that G is bipartite. The standard definition allows arbitrary graphs. However, the two problems are equivalent up to constant factors. The bipartite version is clearly a special case of the general version and the general version can be reduced to the bipartite version using the same trick as before: we make two copies of the original graph and connect two vertices in different copies if and only if they are connected in the original graph. No vertices are connected within the same copy. This new instance of graph collision is bipartite and has a collision if and only if the original instance does.

The best known lower bound for graph collision is $\Omega(\sqrt{n})$, which follows from a simple reduction from the search problem, while the best known upper bound is $O(n^{2/3})$, as it can be solved by Ambainis' quantum walk for element distinctness [Amb07]. Better upper bounds are known for special graph families, random graphs, and graphs with special properties. See the survey by Ambainis, Balodis, Iraids, Ozols, and Smotrovs [ABI+13] for an overview. The graph collision problem is intimately related to the triangle problem since an improved lower bound for graph collision will lead to an improved lower bound for triangle finding. On the other hand, an improved algorithm for graph collision will improve the known quantum walk algorithms for triangle finding, since they use graph collision as a subroutine.

Open Problem 4.3. What is the quantum query complexity of the graph collision problem? The best known lower and upper bounds are $Q(\mathsf{GC}) = \Omega(\sqrt{n})$ and $Q(\mathsf{GC}) = O(n^{2/3})$ [Amb07] respectively.

Depth-2 LC^0 circuits

Lastly, we introduce the problem of determining the query complexity of functions expressed by depth-2 linear-size circuits using only AND, OR, and NOT gates, where the size of a circuit is the total number of AND and OR gates in it. This problem was first studied by Childs, Kimmel, and Kothari [CKK12], as an example of functions that are not much more complicated than read-once formulas but whose query complexity is unknown. The problem is also closely related to BMvPV^A and yields the best known lower bound for BMvPV and BMPV as we show below.

The class of Boolean functions that can be computed by polynomial-size, constant-depth circuits that use AND, OR, and NOT gates is called AC^0. The class of linear-size AC^0 circuits is called LC^0. The class we are interested in is depth-2 LC^0 circuits.

Definition 4.5 (LC_2^0 problem)**.** Let LC_2^0 denote the class of functions on n bits that can be represented by depth-2 circuits using only AND, OR, and NOT gates with at most n AND and

OR gates. For a function $f \in \mathsf{LC}^0_2$, $Q(f)$ is the quantum query complexity of computing f. Let $Q(\mathsf{LC}^0_2) := \max_{f \in \mathsf{LC}^0_2} Q(f)$ be the query complexity of the hardest function in LC^0_2.

We know that $Q(\mathsf{LC}^0_2) = \Omega(n^{0.555})$ and $Q(\mathsf{LC}^0_2) = O(n^{3/4})$ [CKK12]. I conjecture that the upper bound is closer to the truth. A function based on projective planes that may yield a better lower bound is presented in [CKK12].

Open Problem 4.4. What is $Q(\mathsf{LC}^0_2)$? We know that $Q(\mathsf{LC}^0_2) = \Omega(n^{0.555})$ and $Q(\mathsf{LC}^0_2) = O(n^{3/4})$ [CKK12].

Note that any improvement in the lower bound for this problem will yield an improved lower bound for the Boolean matrix product verification problem (BMPV).

Verifying Boolean matrix product inequalities

It turns out that the three problems introduced above are related to the problem of verifying Boolean matrix product inequalities. In a product verification problem, we have to determine if two Boolean matrices or two Boolean vectors are equal. Testing if two Boolean values a and b are equal can be broken down into two problems, of checking if $a \leqslant b$ and $a \geqslant b$, where we use the obvious ordering on the Booleans, i.e., $0 \leqslant 0$, $0 \leqslant 1$, and $1 \leqslant 1$. Thus we can define inequality versions of any product verification problem in the natural way. We add the subscript $\leqslant$ or $\geqslant$ to denote the version of the problem where we have to test inequality instead of equality. For example, BMPV asks if $AB = C$, while the problems $\mathsf{BMPV}_{\leqslant}$ and $\mathsf{BMPV}_{\geqslant}$ ask if $AB \leqslant C$ and $AB \geqslant C$ respectively.

We can now examine how the inequality versions are related to the standard versions of the problems and to the three problems introduced. Figure 4.4 shows the relationships between these problems.

First note that Figure 4.4 asserts that the equality problems are related to the inequality versions in the following way: $Q(\mathsf{BMvPV}^A) = \Theta(Q(\mathsf{BMvPV}^A_{\leqslant}) + Q(\mathsf{BMvPV}^A_{\geqslant}))$ and $Q(\mathsf{BMPV}) = \Theta(Q(\mathsf{BMPV}_{\leqslant}) + Q(\mathsf{BMPV}_{\geqslant}))$. One direction of this inequality is clear, since we can test equality by testing both inequalities, but it is not obvious that we can test inequality by using only the equality problem. We prove this for BMvPV^A; the result for BMPV can be shown analogously.

Lemma 4.2. $Q(\mathsf{BMvPV}^A) = \Theta(Q(\mathsf{BMvPV}^A_{\leqslant}) + Q(\mathsf{BMvPV}^A_{\geqslant}))$.

Proof. Since we can use algorithms for $\mathsf{BMvPV}^A_{\leqslant}$ and $\mathsf{BMvPV}^A_{\geqslant}$ to solve BMvPV^A, we have $Q(\mathsf{BMvPV}^A) = O(Q(\mathsf{BMvPV}^A_{\leqslant}) + Q(\mathsf{BMvPV}^A_{\geqslant}))$. To establish the other direction, we prove $Q(\mathsf{BMvPV}^A_{\leqslant}) = O(Q(\mathsf{BMvPV}^A))$ and $Q(\mathsf{BMvPV}^A_{\geqslant}) = O(Q(\mathsf{BMvPV}^A))$.

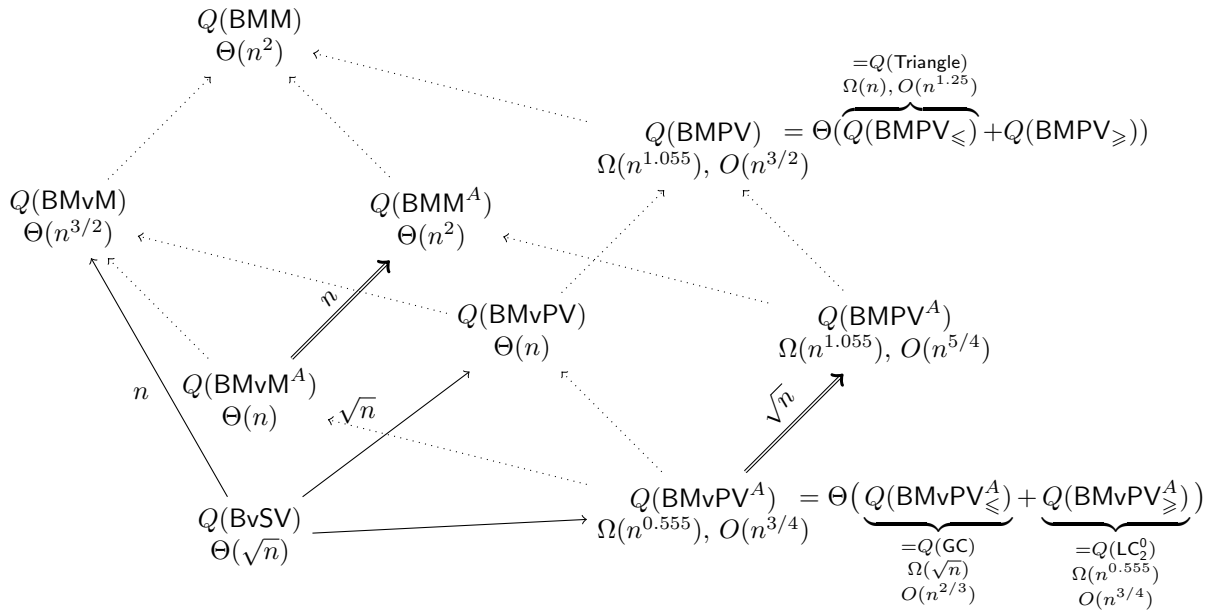

Figure 4.4: Quantum query complexity of Boolean matrix multiplication problems. Arrows between problems indicate relationships between their query complexities. $Q(\mathsf{X}) \longrightarrow Q(\mathsf{Y})$ and $Q(\mathsf{X}) \dashrightarrow Q(\mathsf{Y})$ mean $Q(\mathsf{Y}) = \Omega(Q(\mathsf{X}))$. $Q(\mathsf{X}) \xrightarrow{f(n)} Q(\mathsf{Y})$ means $Q(\mathsf{Y}) = \Omega(f(n)Q(\mathsf{X}))$. $Q(\mathsf{X}) \xdashrightarrow{f(n)} Q(\mathsf{Y})$ means $Q(\mathsf{Y}) = \Theta(f(n)Q(\mathsf{X}))$. Dotted arrows indicate obvious relationships.

We begin with $Q(\mathsf{BMvPV}^A_{\leqslant}) = O(Q(\mathsf{BMvPV}^A))$. To show this, we want to reduce the problem of checking if $Av \leqslant w$ to the problem of checking if $Zx = y$, where Z is a matrix and x and y are vectors. To get some intuition for the reduction, let us consider two Booleans a and b. We want to test if $a \leqslant b$. It is easy to see that $a \leqslant b$ if and only if $a \vee b = b$, since equality fails only when $a = 1$ and $b = 0$. We want to extend this idea to vectors, i.e., we want to use the fact that $Av \leqslant w \Leftrightarrow Av \vee w = w$. This is an equality, but it is not of the form $Zx = y$. It can be brought to this form by defining $Z = \left(\begin{smallmatrix} A & \mathbb{1} \\ \mathbb{0} & \mathbb{0} \end{smallmatrix}\right)$, $x = \left(\begin{smallmatrix} v \\ w \end{smallmatrix}\right)$, and $y = \left(\begin{smallmatrix} w \\ \vec{0} \end{smallmatrix}\right)$. Now $Zx = y$ if and only if $Av \vee w = w$. This completes the proof of $Q(\mathsf{BMvPV}^A_{\leqslant}) = O(Q(\mathsf{BMvPV}^A))$.

Lastly, we have to show that $Q(\mathsf{BMvPV}^A_{\geqslant}) = O(Q(\mathsf{BMvPV}^A))$. As before, we want to reduce the problem of checking if $Av \geqslant w$ to the problem of checking if $Zx = y$. For two Booleans a and b, it is easy to see that $a \geqslant b$ if and only if $a \vee \bar{b} = 1$, where $\bar{b}$ denotes the complement of b. To put this into the form $Zx = y$, we define $Z = \left(\begin{smallmatrix} A & \mathbb{1} \\ \mathbb{0} & \mathbb{0} \end{smallmatrix}\right)$, $x = \left(\begin{smallmatrix} v \\ \bar{w} \end{smallmatrix}\right)$, and $y = \begin{pmatrix} \vec{1} \\ \vec{0} \end{pmatrix}$. Now $Zx = y$ if and only if $Av \vee \bar{w}$ is $\vec{1}$, which is equivalent to $Av \geqslant w$, which completes the proof. □

Figure 4.4 also shows the relationships between the inequality versions of the problems and the three problems we defined earlier in this section. These are straightforward to show. We start with the claim that $Q(\mathsf{BMPV}_{\leqslant}) = Q(\mathsf{Triangle})$. Observe that $AB \leqslant C$ is false only if there exist $i, j \in [n]$, such that $(AB)_{ij} = 1$ and $C_{ij} = 0$, and $(AB)_{ij} = 1$ only if there exists a $k \in [n]$ such that $A_{ik} = B_{kj} = 1$. So $AB \leqslant C$ is false if and only if there exist $i, j, k \in [n]$ such that $A_{ik} = B_{kj} = \bar{C}_{ij} = 1$. This is exactly the definition of the triangle problem, except with $\bar{C}$ instead of C. Similarly, $\mathsf{BMvPV}^A_{\leqslant}$ is false if and only if there exists in $i \in [n]$ such that $(Av)_i = 1$ and $w_i = 0$. $(Av)_i = 1$ if and only if there exists a $j \in [n]$ such that $A_{ij} = 1$ and $v_j = 1$. Thus $\mathsf{BMvPV}^A_{\leqslant}$ is false if and only if there exist $i, j \in [n]$ such that $A_{ij} = v_j = \bar{w}_i = 1$, which is the graph collision problem with $\bar{w}$ instead of w.

Lastly, we show that $Q(\mathsf{BMvPV}^A_{\geqslant}) = \Theta(Q(\mathsf{LC}^0_2))$. We saw in the proof of Lemma 4.2 that $\mathsf{BMvPV}^A_{\geqslant}$ can be alternately expressed as the problem of checking if $Av \vee \bar{w}$ is the all-ones vector. Note that $(Av)_i \vee \bar{w}_i$ is the output of a single OR gate, where the input variables are the bits of v and w, and A is a description of the bits that feed into each OR gate. Checking if $Av \vee \bar{w}$ is the all-ones vector is thus an AND of n OR gates, which is a depth-2 LC^0 circuit. This shows that $Q(\mathsf{BMvPV}^A_{\geqslant}) = O(Q(\mathsf{LC}^0_2))$. For the other direction, we have already noted that LC^0_2 is just the problem of checking if an AND of n OR gates is the all-ones vector. This can be expressed as the problem of checking if Av is the all-ones vector, which is the same as checking if Av is greater than or equal to the all-ones vector.

Thus we have established the relationships stated in Figure 4.4 between the inequality versions of the problems and the rest of the problems.

Query complexities of these problems

Finally we can establish all the other bounds in Figure 4.4. As before, we first look at the vSV problem.

Lemma 4.3. *$Q(\mathsf{vSV}_{\mathcal{S}}) = \Omega(\sqrt{n})$ over any semiring $\mathcal{S}$. In particular, $Q(\mathsf{BvSV}) = \Theta(\sqrt{n})$.*

To show this, we can prove that given black-box access to n elements $x_1, \ldots x_n$ from a semiring R, determining if $\sum_{j=1}^n x_j = 1_{\mathcal{S}}$ requires $\Omega(\sqrt{n})$ queries. We will only need the instances where all entries are equal to $0_{\mathcal{S}}$ and where one entry is $1_{\mathcal{S}}$ and the remaining are zero. In the first case the sum of all entries is $0_{\mathcal{S}}$, and in the second case it is $1_{\mathcal{S}}$. This is the promise version of the OR problem, where we are promised that either there are no marked items or there is only one marked item. This can be shown using the adversary method or the polynomial method.

Finally we can show that all the bounds stated in Figure 4.4 are correct.

Theorem 4.4 (Boolean matrix multiplication)**.** *The query complexities stated in Figure 4.4 are correct.*

Proof. First let us establish the lower bounds. From Lemma 4.1 we know that $Q(\mathsf{BvSV}) = \Omega(n)$. From Lemma 4.2 we know that $Q(\mathsf{BMvPV}^A) = \Omega(\mathsf{LC}_2^0) = \Omega(n^{0.555})$. Finally, $Q(\mathsf{BMvM}^A) = \Omega(n)$, since if A is the identity matrix, this is the problem of learning the entire vector v, which requires $\Omega(n)$ queries. All other lower bounds follow from these using the general relationships proved in Theorem 4.1.

As for the upper bounds, the simple observation that the query complexity of a problem is upper bounded by its input size establishes the claimed upper bound for many problems in Figure 4.3. $Q(\mathsf{BMvM}) = O(n^{3/2})$ since each entry of the product of Av can be computed with $O(\sqrt{n})$ queries using Grover's algorithm and there are n entries to compute, which gives a bound of $O(n^{3/2})$ using the composition theorem (Theorem 1.2). The upper bound on BMPV was shown by Buhrman and Špalek [BŠ06], and follows from a similar idea that checking each entry costs $O(\sqrt{n})$ queries and we can search over all entries to check if all entries are correct. The remaining upper bounds follow from the general relationships shown in Theorem 4.1 and Lemma 4.2. □

Since we have already stated the open problems of determining $Q(\mathsf{GC})$ and $Q(\mathsf{LC}_2^0)$, the main open problem that remains is determining the query complexity of BMPV.

Open Problem 4.5. What is the quantum query complexity of Boolean matrix product verification (BMPV)? We know that $Q(\mathsf{BMPV}) = \Omega(n^{1.055})$ [CKK12] and $Q(\mathsf{BMPV}) = O(n^{3/2})$ [BŠ06].

Since Triangle reduces to BMPV, we cannot improve the upper bound too much without improving the upper bound for Triangle. Since Triangle has withstood attack on the lower bound front for many years, it may be more fruitful to try to improve the lower bound for BMPV, since it is at least as hard as Triangle. Unlike for Triangle, the certificate complexity barrier does not seem to present a problem for BMPV, so there might even be an improved lower bound using the original (positive-weights) adversary method.

4.4 Output-sensitive Boolean matrix multiplication

As discussed in Section 4.1 and shown in Figure 4.1, the output-sensitive version of the Boolean matrix multiplication problem has been studied by several authors. Our main result is an output-sensitive algorithm for Boolean matrix multiplication of $n \times n$ matrices that makes $O(n\sqrt{\ell})$ queries, where ℓ is the number of nonzero entries in the output matrix. More precisely, our algorithm makes $O(n\sqrt{\ell})$ queries when the values of ℓ is known beforehand, and if ℓ is unknown makes $O(n\sqrt{\ell+1})$ queries. The additional $+1$ only affects the asymptotic complexity when $\ell = 0$, since in that case $O(n\sqrt{\ell})$ is 0, whereas the complexity of our algorithm is $O(n)$. Since this is a minor technical point, we only make this distinction precise when we formally prove the complexity of the algorithm. Otherwise when we discuss the algorithm informally we use the expressions $O(n\sqrt{\ell})$ and $O(n\sqrt{\ell+1})$ interchangeably.

We also show our algorithm is nearly optimal by providing a matching lower bound when $\ell < \epsilon n^2$ for some $\epsilon < 1$. We start with a high-level overview of our algorithm, followed by our new algorithm for graph collision, followed by our algorithm for BMM. We end with the lower bound and a discussion of its optimality.

4.4.1 High-level overview of the algorithm

We wish to compute the product C of two $n \times n$ Boolean matrices A and B given query access to their entries, while making $O(n\sqrt{\ell})$ queries, where ℓ is the number of nonzero entries in C. As a warmup to solving this problem, let us solve the simpler problem of deciding if $C = \mathbb{0}$. Using our approach, this problem can be solved with $O(n)$ queries, which is optimal. Note that it was already not known how to solve this problem with $O(n)$ queries before our work. The solution to the easier problem will guide us towards the final algorithm for BMM.

A simpler problem: Is $AB = \mathbb{0}$?

Consider the problem of deciding if the product of A and B is $\mathbb{0}$ given query access to the entries of A and B. The straightforward way to solve this problem is by checking if any entry of C is nonzero using the definition. Since any entry of C can be computed using $O(\sqrt{n})$ queries and there are n^2 entries to check, using Grover's algorithm to search for a nonzero entry takes $O(n^{3/2})$ queries, which is suboptimal.

To obtain an optimal algorithm, we take a different perspective on the matrix C. Let e_i be the i^{th} basis vector, i.e., the Boolean vector of length n with 1 at the i^{th} position and zeros everywhere else. In terms of this vector, we can write C as $C = \sum_{ij} C_{ij} e_i e_j^\dagger$. Since $C_{ij} = \sum_k A_{ik} B_{kj}$, we have

$$C = \sum_{ijk} A_{ik} B_{kj} e_i e_j^\dagger = \sum_k \left(\sum_{ij} A_{ik} B_{kj} e_i e_j^\dagger \right) =: \sum_k C_k, \tag{4.1}$$

where we have defined $C_k := \sum_{ij} A_{ik} B_{kj} e_i e_j^\dagger$. In other words, C is the OR of n matrices C_k, where C_k is the outer product of the k^{th} column of A, a_k, with the k^{th} row of B, b_k, as shown below.

$$\text{If } A = \begin{pmatrix} | & | & | \\ a_1 & \cdots & a_n \\ | & | & | \end{pmatrix} \text{ and } B = \begin{pmatrix} — & b_1 & — \\ — & \vdots & — \\ — & b_n & — \end{pmatrix} \text{ then } C_k = \begin{pmatrix} | \\ a_k \\ | \end{pmatrix} \begin{pmatrix} — & b_k & — \end{pmatrix}. \tag{4.2}$$

From this characterization, we see that $C = \mathbb{0}$ if and only if $\forall k \in [n]$, $C_k = \mathbb{0}$. For a fixed k, $C_k = \mathbb{0}$ if the outer product of a_k and b_k is $\mathbb{0}$, i.e., for every i, j, we have $A_{ik} B_{kj} = 0$. This only

happens if either a_k or b_k is $\vec{0}$, which can be checked with $O(\sqrt{n})$ queries. Since we can check if $C_k = \mathbb{0}$ using $O(\sqrt{n})$ queries, we can check if there is any $k \in [n]$ for which $C_k = \mathbb{0}$ using $O(n)$ queries.

The key idea used to solve this simpler problem was the observation that C can be written in an alternate way in terms of outer products of column and row vectors of A and B respectively. Now let us make the problem a little harder by generalizing this simple problem.

An intermediate problem

Consider the problem where we are given query access to matrices A and B, and an explicit description of a matrix C' promised to satisfy $C' \leqslant C = AB$. The problem is to determine if $C' = AB$. While this problem is similar to the Boolean matrix product verification problem, there are two differences: the matrix C' is known and requires no queries to learn, and we are promised that $C' \leqslant C$, which means that all the 1s in C' are also present in C. The simpler problem we just discussed is the special case of this problem when $C' = \mathbb{0}$, which satisfies the promise since $\mathbb{0} \leqslant C$ for all matrices C. We would like to solve this problem with $O(n + \sqrt{n\ell})$ queries, where ℓ is the number of nonzero entries in C'.

As before, we view C as the OR of n matrices C_k. Now the task is to find a nonzero entry in C that is not present in C'. We do this by finding a nonzero entry in C_k that is not present in C', which is equivalent to finding a nonzero entry in $C_k \wedge \overline{C'}$ for some k. For a fixed k, $(C_k \wedge \overline{C'})_{ij}$ is nonzero if and only if $A_{ik} = B_{jk} = 1$ and $\overline{C'}_{ij} = 1$. Since C' is a known matrix, this is exactly the graph collision problem on the graph described by $\overline{C'}$ where the input vectors are the i^{th} column of A and the j^{th} row of B.

The best known algorithm for the graph collision makes $O(n^{2/3})$ queries, so we can find a nonzero entry in $C_k \wedge \overline{C'}$ with $O(n^{2/3})$ queries. Searching over all values of k gives us an algorithm for this problem with query complexity $O(n^{7/6})$ queries. However, this is not an arbitrary instance of the graph collision problem: We know that the bipartite graph described by $\overline{C'}$ has a special property—it has at most ℓ nonedges, since C has ℓ nonzero entries and $C' \leqslant C$. Furthermore, graph collision is easier on graphs with very few nonedges. For example, if the graph has no nonedges, i.e., it is the complete bipartite graph, then graph collision can be solved in $O(\sqrt{n})$ queries as we did before. On the complete graph, graph collision is merely the problem of testing if either of the vectors is $\vec{0}$, which only requires $O(\sqrt{n})$ queries.

We show in the next section that these instances of graph collision can indeed be solved faster. When the graph has at most ℓ nonedges, we can solve GC with $O(\sqrt{n} + \sqrt{\ell})$ queries. We also show that if there are $\lambda > 0$ collisions, we can find all λ collisions with $O(\sqrt{n\lambda} + \sqrt{\ell})$ queries.

Final algorithm

Given these two graph collision algorithms, we now have a natural algorithm for BMM. We can start by setting $C' = \mathbb{0}$. Now we solve our intermediate problem and find a k such that $C_k \wedge \overline{C'}$ has a nonzero entry. Once we find such a k, we find all graph collisions for that k for the graph $\overline{C'}$. Every time we find a collision we learn a new entry of C. After we have found all the nonzero entries of C'_k, we update the matrix C' with the new nonzero entries we have learned and repeat this algorithm until there is no k such that $C_k \wedge \overline{C'}$ has a nonzero entry.

To ensure that this is leading toward our goal, let us perform a back-of-the-envelope estimate for the algorithm's complexity using intuitively reasonable, but unproven, assumptions.

First consider the case when $\ell < n$. It is reasonable to guess that the hardest instances of the problem will have only one witness per nonzero entry, i.e., if $C_{ij} = 1$, then there is only one k for which $(C_k)_{ij} = 1$. It is also reasonable to imagine that in the worst case, each nonzero entry in C will have a different value of k serving as a witness. Thus our algorithm needs to find the ℓ values of k such that each of them provide a witness for one nonzero entry of C. Deciding if a k witnesses a nonzero entry of C is the graph collision problem, which requires only $O(\sqrt{n})$ queries in this case since $\ell \leq n$. Since there are ℓ such k to be found over a search space of size n, we can find them all in query complexity $O(\sqrt{n\ell} \times \sqrt{n}) = O(n\sqrt{\ell})$, which is the claimed upper bound.

In the other case when $\ell > n$, it is still reasonable to imagine that witnesses will be spread uniformly over all values of k so that each k witnesses roughly ℓ/n nonzero entries of C. Since each k witnesses a nonzero entry, we do not need to search for k, we can merely find all nonzero entries in $C_k \wedge \overline{C'}$ for each k. Since each k will have $\lambda = \ell/n$ collisions, graph collision takes $O(\sqrt{n\lambda} + \sqrt{\ell}) = O(\sqrt{\ell})$ queries and there are n different values of k, which gives an algorithm with query complexity $O(n\sqrt{\ell})$. We make this algorithm rigorous in Section 4.4.3.

4.4.2 Graph collision algorithm

As explained in the previous section, we use the graph collision problem as a subroutine for solving BMM. Specifically, we study graph collision on bipartite graphs with at most ℓ nonedges and show that such instances can be solved faster if ℓ is small. As before, let GC^A denote the graph collision problem on a bipartite graph with biadjacency matrix A. Recall that A is known and requires no queries to learn, while the input to the problem is two n-length Boolean vectors v and w, whose entries can be accessed by an oracle. The aim is to decide if there exist $i, j \in [n]$ such that $A_{ij} = v_j = w_i = 1$. For any A, we let ℓ denote the number of zero entries in the matrix, i.e., the number of nonedges in the bipartite graph described by A. Thus the complete bipartite graph, which corresponds to A being the all-ones matrix, has $\ell = 0$. We also study the problem of determining all graph collisions for a given instance, i.e., determining all pairs $i, j \in [n]$ such that $A_{ij} = v_j = w_i = 1$. We denote this problem $\mathsf{GC}^A_{\text{all}}$.

We will also refer to the two sets of vertices, corresponding to the vector v and w, as J and I respectively. We will say a vertex $j \in J$ or $i \in I$ is marked if $v_j = 1$ or $w_i = 1$, respectively. In this terminology, the graph collision problem is to find a marked vertex in J that is adjacent to a marked vertex in I.

Theorem 4.5 (Graph collision on dense graphs)**.** *For an $n \times n$ matrix A with at most ℓ zero entries, $Q(\mathsf{GC}^A) = O(\sqrt{n} + \sqrt{\ell})$ and $Q(\mathsf{GC}^A_{\text{all}}) = O(\sqrt{n\lambda} + \sqrt{\ell})$, where $\lambda > 0$ is the number of graph collisions. When $\lambda = 0$, $Q(\mathsf{GC}^A_{\text{all}}) = O(\sqrt{n} + \sqrt{\ell})$.*

Proof. We start with an algorithm for $\mathsf{GC}^A_{\text{all}}$. As explained at the end of this proof, this algorithm can be easily converted to an algorithm for GC^A by stopping the algorithm once a collision is found. For now let us also assume that $\lambda \neq 0$.

First rearrange the vertices in I in decreasing order of degree and let d_i be the degree of the i^{th} vertex in I. Thus $d_1 \geq d_2 \geq \ldots \geq d_n$. Let $c_i := n - d_i$ be the number of non-neighbors of a vertex $i \in I$, i.e., the number of vertices $j \in J$ that vertex $i \in I$ is not adjacent to. We call c_i the nondegree of i, as it counts the number of nonedges incident on vertex i, much like d_i, the degree, counts the number of edges incident on vertex i.

Algorithm 4.1 Graph collision algorithm (finds all collisions)

Given: An explicit description of a bipartite graph on vertex sets I and J with $|I| = |J| = n$.
Oracle input: Two Boolean vectors v_i and w_j where $i \in I$ and $j \in J$.
Let d_i be the degree of the i^{th} vertex in I, and let $c_i := n - d_i$.
Let the vertices in I be arranged in decreasing order of degree so that $d_1 \geq d_2 \geq \ldots \geq d_n$.

1: Find the highest degree marked vertex in I. Let $r \in I$ be the index of this vertex. $O(\sqrt{n})$
2: **if** $c_r \leq \sqrt{\ell}$ **then**
3: Find all marked neighbors of r in J. Output any graph collisions found. $O(\sqrt{n\lambda})$
4: Read the values of all non-neighbors of r in J. $\leq \sqrt{\ell}$
5: Since v is completely known, find all marked $i \in I$ adjacent to any marked $j \in J$. $O(\sqrt{n\lambda})$
6: **else**
% In this case, using the promise that there are ℓ nonedges, it can be shown that $r \geq n - \sqrt{\ell}$.
7: Read the values of all w_i for $i \geq r$. (We know that $w_i = 0$ for all $i \leq r$.) $\leq \sqrt{\ell}$
8: Since w is completely known, find all marked $j \in J$ adjacent to any marked $i \in I$. $O(\sqrt{n\lambda})$
9: **end if**

Algorithm 4.1 describes our approach for finding all graph collisions assuming $\lambda \neq 0$. Our algorithm starts by finding the highest degree marked vertex in I. This takes $O(\sqrt{n})$ queries [DHHM06]. (We called this as the find-first-one function in Chapter 3.) Let this vertex be the r^{th} vertex in I. We now know that $w_r = 1$ and $w_i = 0$ for all $i \leq r$ since r is the first marked vertex in I.

Now consider two cases. If the number of non-neighbors of r, c_r, is smaller than $\sqrt{\ell}$, this means r is adjacent to $n - \sqrt{\ell}$ vertices in J. Any marked vertex in this set will lead to a collision. Since we know the graph explicitly, we know this set of vertices. We can search for all marked vertices in this set and find them with $O(\sqrt{n\lambda})$ queries, since there are at most λ graph collisions and each marked vertex gives rise to a unique graph collision. Now we know the values v_j of all neighbors of r. Since r has at most $\sqrt{\ell}$ non-neighbors, we can simply read the values of v_j for these neighbors classically. This requires at most $\sqrt{\ell}$ queries, after which we have completely learned the vector v. Now that v is known, we know exactly which vertices in I will lead to graph collisions if marked. We search over this set for all marked vertices. Again, this requires at most $O(\sqrt{n\lambda})$ queries since the search space is of size at most n and there are at most λ marked vertices. This shows that the stated query complexities in the first case are correct and we have found all graph collisions.

In the second case, the number of non-neighbors of r, c_r, is larger than $\sqrt{\ell}$. Since the vertices we arranged in decreasing order of degree, they are arranged in increasing order of nondegree. All vertices $i \in I$ with $i \geq r$ have nondegree greater than c_r. However, there are at most ℓ nonedges, so there cannot be too many vertices with $i \geq r$, since each i contributes c_i nonedges. Indeed, there can be at most ℓ/c_r such vertices, since each vertex has nondegree at least c_r. Since $c_r \geq \sqrt{\ell}$, there can be at most $\sqrt{\ell}$ vertices with $i \geq r$, which means that $r \geq n - \sqrt{\ell}$. Since we know that $w_i = 0$ for all $i \leq r$, the only unknown values of w_i are for $i \geq r$. But there are at most $\sqrt{\ell}$ such entries, which can all be read with at most $\sqrt{\ell}$ queries. Now we have completely learned w, so we know exactly which vertices in v will lead to graph collisions. These can all be found with $O(\sqrt{n\lambda})$ queries as before, since there are at most λ such vertices.

This algorithm has bounded error since all subroutines used are bounded error and are called only a constant number of times. This shows that $Q(\mathsf{GC}^A_{\text{all}}) = O(\sqrt{n\lambda} + \sqrt{\ell})$ when $\lambda \neq 0$. When $\lambda = 0$, we use the same algorithm, but the analysis is slightly different. The subroutines that searched for all marked items and made $O(\sqrt{n\lambda})$ queries now only make $O(\sqrt{n})$ queries before declaring that there is no marked item.

To show that $Q(\mathsf{GC}^A) = O(\sqrt{n} + \sqrt{\ell})$, instead of searching for all graph collisions we stop as soon as we find one graph collision. This replaces the costs $O(\sqrt{n\lambda})$ with $O(\sqrt{n})$ in the analysis above. □

4.4.3 Boolean matrix multiplication algorithm

Given these two graph collision algorithms, we can present and analyze our algorithm for BMM. Before presenting the final algorithm, note that the following simple algorithm works well when $\ell = \Omega(n)$. We just find nonzero entries in $C_k \wedge \overline{C'}$, starting with $C' = \mathbb{0}$, for $k = 1, 2, \ldots, n$. In each step, we update C' to include all the nonzero entries found. Each step requires $O(\sqrt{n\lambda_i} + \sqrt{n} + \sqrt{\ell})$ queries, where the additional $O(\sqrt{n})$ factor accounts for the fact that graph collision still takes

$O(\sqrt{n}+\sqrt{\ell})$ queries when $\lambda_i = 0$. Since each subroutine is bounded error, we boost the success probabilities at the cost of an additional log factor. Thus the total cost of the algorithm is $\sum_k \tilde{O}(\sqrt{n\lambda_k}+\sqrt{n}+\sqrt{\ell}) = \tilde{O}(n^{3/2}+n\sqrt{\ell})$, using the Cauchy–Schwarz inequality and the fact that $\sum_k \lambda_k = \ell$. Note that when $\ell = \Omega(n)$, this is $\tilde{O}(n\sqrt{\ell})$.

Final algorithm

Our final algorithm is similar to this one with one change. Solving graph collision for all values of k is inefficient if ℓ is small and only some of these graph collisions yield nonzero entries. To account for this, we search for the first graph collision instance that has a collision among the n instances defined by different values of k. Specifically, we search for the first k such that $C_k \wedge \overline{C'}$ has a nonzero entry, find all nonzero entries for this values of k, update C', and look for the next k such that $C_k \wedge \overline{C'}$ has a nonzero entry.

We start with $C' = \mathbb{0}$ and want to find the first k such that $C_k \wedge \overline{C'}$ has a nonzero entry. As discussed in Section 3.3.2, we can find the first marked item in a list of size n with $O(\sqrt{p})$ queries in expectation, where p is the position of the first marked item. This follows from the known $O(\sqrt{n})$ algorithm for finding the first 1 in a string of size n [DHHM06], by running that algorithm on the first 2^m bits, for $m = 1, 2, \ldots, \log n$.

If p_1 is the first k with a graph collision, we can compose the algorithm to find the first marked item with the graph collision algorithm that makes $O(\sqrt{n}+\sqrt{\ell})$ queries to get an algorithm that makes $O(\sqrt{p_1}(\sqrt{n}+\sqrt{\ell}))$ queries and finds the first index $k \in [n]$ such that there is a nonzero entry in C_k. Once we find p_1, we find all graph collisions for this instance on the complete bipartite graph, since currently $C' = \mathbb{0}$. This takes $O(\sqrt{n\lambda_1})$ queries, where λ_1 is the number of nonzero entries in C_{p_1}. After we have found all the nonzero entries of C_{p_1}, we update the matrix C' with the new nonzero entries we have learned. At this point, $C' = \sum_{k=1}^{p_1} C_k$, since there are no graph collisions up to C_{p_1} and we have learned all the nonzero entries of C_{p_1}. Now we search for the first $k > p_1$ such that $C_k \wedge \overline{C'}$ has a nonzero entry. Let p_2 be the position of this k relative to p_1, i.e., $k - p_1$. We then find all new nonzero entries of this instance of graph collision and continue this process until we reach the end. As before we boost the success probabilities of all algorithms to make the error probability inverse polynomially small. This gives us Algorithm 4.2.

Note that any point in the algorithm the matrix C' stores a cumulative sum of C_i up to position c, i.e., $C' = \sum_{k=1}^{c} C_k$. In the beginning $C' = \mathbb{0}$, and at each step C' is updated with the new nonzero entries found at the first value of k that has nonzero entries not present in C'. Therefore at the end $C' = \sum_{k=1}^{n} C_k$, which is C.

Let us analyze the complexity of this algorithm. Say the loop runs t times including the last run where no k is found. Let the relative positions of graph collisions be $p_1, p_2, \ldots, p_t$. By relative positions we mean the first k is p_1, the second k is $p_1 + p_2$, and so on. Lastly, in the last run let us say that $k = n + 1$ is the result of the search. This is to account for the fact that the search

Algorithm 4.2 Output-sensitive Boolean matrix multiplication algorithm

$C' \leftarrow \mathbb{0}$
$i \leftarrow 1$
$c \leftarrow 1$
while there exists a $k \geq c$ such that $C_k \wedge \overline{C'}$ has a nonzero entry **do**
    Find the first $k \geq c$ such that $C_k \wedge \overline{C'}$ has a nonzero entry. Let $p_i \leftarrow k - c$. $\tilde{O}(\sqrt{p_i}(\sqrt{n}+\sqrt{\ell}))$
    Find all nonzero entries of $C_k \wedge \overline{C'}$ $\tilde{O}(\sqrt{n\lambda_i}+\sqrt{\ell})$
    $C' \leftarrow C' \vee (C_k \wedge \overline{C'})$
    $c = c + p_i$
    $i \leftarrow i + 1$
end while

in the last run still requires queries to decide that there is no further k to be found such that $C_k \wedge \overline{C'}$ has a nonzero entry. Thus the relative positions p_i sum to $n+1$. Note that $t \leq n$ since there are only n values of k and $t \leq \ell$ since there are only ℓ nonzero entries to be found.

Since the first step of the loop costs $\tilde{O}(\sqrt{p_i}(\sqrt{n}+\sqrt{\ell}))$ queries and the second step needs $\tilde{O}(\sqrt{n\lambda_i}+\sqrt{\ell})$ queries, where λ_i is the number of graph collisions found at step i, the total query complexity of the algorithm is

$$\tilde{O}\left(\sum_{i=1}^{t}\left(\sqrt{p_i n}+\sqrt{p_i \ell}+\sqrt{n\lambda_i}+\sqrt{\ell}\right)\right) = \tilde{O}(n\sqrt{\ell}+n) = \tilde{O}(n\sqrt{\ell+1}), \tag{4.3}$$

where we have used the Cauchy–Schwarz inequality, $\sum_{i=1}^{t}\lambda_i = \ell$, $\sum_{i=1}^{t} p_i = n+1$, $t \leq n$, and $t \leq \ell$. This gives us the following theorem.

Theorem 4.6. *The quantum query complexity of multiplying two $n \times n$ Boolean matrices is $\tilde{O}(n\sqrt{\ell+1})$, where ℓ is the number of nonzero entries in the output matrix.*

Note that this upper bound holds even when the value of ℓ is not known a priori. However, this is still weaker than the claimed result due to log factors present in the upper bound. We need some techniques introduced in the previous chapter to remove these log factors. We show how to apply these techniques in the next section. The removal of log factors is technical and does not conceptually alter the algorithm described above.

4.4.4 Removing log factors

We now show how to improve the upper bound on Boolean matrix multiplication from $\tilde{O}(n\sqrt{\ell+1})$ to $O(n\sqrt{\ell+1})$. These results are based on the composition theorem proved in Section 3.4. As

shown in Section 3.4, the removal of log factors relies on converting quantum algorithms into feasible solutions of the γ_2 SDP while preserving the algorithm's expected query complexity. Let us do this for the two graph collision subroutines used in the algorithm.

The first subroutine solves the graph collision problem on a bipartite graph with $2n$ vertices and at most ℓ nonedges in $O(\sqrt{n}+\sqrt{\ell})$ queries. This query complexity is not input dependent, and thus there is a feasible SDP solution for this problem with $c(x) = O(\sqrt{n}+\sqrt{\ell})$ for all x, using the known characterization of Lee et al. [LMR+11]. Recall that $c(x)$ is a feasible cost function, as defined in Definition 3.2.

The second subroutine finds all graph collisions in an instance with λ collisions using $O(\sqrt{n\lambda}+\sqrt{\ell})$ queries. This upper bound is input dependent, since λ is a function of the input. In this subroutine, the only input-dependent algorithm is the variant of Grover's algorithm that requires $O(\sqrt{nk})$ queries to output all the ones in an n-bit string when there are k ones. There is a feasible cost function for this with $c(x) = O(\sqrt{nk})$, which can be shown in several different ways. One way to show this is to observe that we can obtain an optimal quantum algorithm for this function by finding the first 1 in the string, and then looking for the first 1 after that position and so on. This corresponds to composing the SDP solution for the find-first-one function (Theorem 3.7) with itself repeatedly to find all ones. The cost function of the resultant SDP will satisfy $c(x) = O(\sum_i \sqrt{p_i})$, where p_is are the locations of the ones. By the Cauchy–Schwarz inequality this is $O(\sqrt{nk})$. Another way to show this is to observe that this is the oracle identification problem with $\mathcal{C} = \{0,1\}^N$, for which we have shown that our algorithm gives rise to an SDP solution with cost function $c(x) = \sum_i^r \sqrt{p_i(x)}$. At each stage of the algorithm, the majority string will be the all-ones string, and the algorithm will find all disagreements, which means it will find all the zeros in the string. But by negating the string, this is equivalent to finding all the ones.

Just like in oracle identification, we will break up the BMM algorithm into a sequence of algorithms A_i such that the output of A_i is the input of A_{i+1}, and convert each algorithm into a feasible solution for the corresponding SDP. The BMM algorithm is already of this form: The BMM algorithm breaks up the problem into n instances of graph collision. The algorithm repeatedly searches for the first index i such that the i^{th} graph collision instance has a collision. Then it finds all graph collisions of this instance and repeats. The problem of searching for the first i that has a graph collision is the composition of the find-first-one function (Theorem 3.7) with the graph collision function. This is a composition in the sense that each oracle input bit of the first problem is the output bit of another query problem. It is known that the optimal value of the γ SDP for $f \circ g$ is at most $\gamma(J-F)\gamma(J-G)$ [LMR+11, Lemma 5.1]. Similarly, it can be shown that there is a feasible cost function for $f \circ g$ that is at most the product of the cost functions. This is similar to [LMR+11, Lemma 5.1] or Lemma 3.5, but instead of taking the direct sum of the vectors, we take the tensor product.

Thus if $p_1, \ldots, p_r$ are the relative positions of indices found by the algorithm, the search subroutine requires $O(\sqrt{p_i}(\sqrt{n}+\sqrt{\ell}))$ queries for each i and a feasible cost function with this cost

exists. The algorithm that finds all graph collisions has a feasible cost function $O(\sqrt{n\lambda_i} + \sqrt{\ell})$, where λ_i is the number of graph collisions in the i^{th} graph collision instance. This gives a feasible cost function for BMM with cost $O(\sum_i(\sqrt{p_i}(\sqrt{n} + \sqrt{\ell}) + \sqrt{n\lambda_i} + \sqrt{\ell}))$, which is same calculation we performed in the previous section, but without log factors. This evaluates to $O(n\sqrt{\ell+1})$, which shows $Q(\mathsf{BMM}) = O(n\sqrt{\ell+1})$.

Theorem 4.7 (Output-sensitive Boolean matrix multiplication)**.** *The quantum query complexity of multiplying two $n \times n$ Boolean matrices is $O(n\sqrt{\ell+1})$, where ℓ is the number of nonzero entries in the output matrix. The upper bound holds even when the value of ℓ is unknown.*

4.4.5 Lower bound and discussion

Our algorithm for BMM makes $O(n\sqrt{\ell})$ queries, and we show this is optimal when ℓ is not too close to n^2. More precisely, if $\ell < \epsilon n^2$ for some $\epsilon < 1$, then $Q(\mathsf{BMM}) = \Omega(n\sqrt{\ell})$. Our lower bound also holds if the algorithm is promised that all instances have the same known value of ℓ, except when ℓ is known to be 0 or n^2, because that would uniquely identify the output matrix C. For these extreme cases, we can show that distinguishing between $\ell = 0$ and $\ell = 1$ (or $\ell = n^2$ and $\ell = n^2 - 1$) requires $\Omega(n)$ queries. Formally, we show the following.

Theorem 4.8. *$Q(\mathsf{BMM}) = \Omega(n\sqrt{\min\{\ell, n^2 - \ell\}})$, where ℓ is the number of nonzero entries in the output matrix, even if restricted to instances with a fixed value of ℓ. Additionally, distinguishing between the cases $\ell = 0$ and $\ell = 1$ (or $\ell = n^2$ and $\ell = n^2 - 1$) requires $\Omega(n)$ queries.*

Proof. We fix A to be the identity matrix, so that the output matrix C equals B. Now our problem is the oracle identification problem on a Boolean string of size n^2 with the additional promise that the string has Hamming weight ℓ. Note that the complexity of the problem is the same for Hamming weight ℓ or $n^2 - \ell$, since complementing all the bits in the input string does not change the problem's complexity. Thus we can assume $\ell \leq \frac{1}{2}n^2$ and we only need to show $Q(\mathsf{BMM}) = \Omega(n\sqrt{\ell})$ and that distinguishing $\ell = 0$ from $\ell = 1$ requires $\Omega(n)$ queries.

The latter problem is merely the PromiseOR function, in which we have to compute the OR of n^2 input bits with the promise that the Hamming weight is 0 or 1. The PromiseOR function on n^2 bits requires $\Omega(n)$ queries, which can be shown using any of the lower bound methods. (Indeed, most lower bound proofs for the OR function actually prove a lower bound for this problem.)

We now have to show that the oracle identification problem on n^2 bits with the promise that its Hamming weight ℓ satisfies $0 < \ell \leq \frac{1}{2}n^2$ requires $\Omega(n\sqrt{\ell})$ queries. This is exactly what we showed in Lemma 3.1 in the previous chapter. □

This shows that our algorithm is tight for any $\ell \leq \epsilon n^2$ for any constant $\epsilon < 1$. However, it is not tight for $\ell = n^2 - o(n)$, i.e., when C is very close to being the all-ones matrix J. When ℓ is very

close to n^2, our algorithm makes $O(n^2)$ queries, which is trivial. However, there are nontrivial algorithms that work better in this regime. For example, assume that we know that there is at most one zero entry in C. We can find a zero in C with only $O(n^{3/2})$ queries, since computing one entry of C costs $O(\sqrt{n})$ queries and searching over all n^2 entries adds a multiplicative factor of $O(n)$. This is faster than the trivial $O(n^2)$ algorithm. More generally, if there are m zeros in C, we can find them all in $O(n^{3/2}\sqrt{m})$ queries, which is $o(n\sqrt{\ell}) = o(n^2)$ when $m = o(n)$.

One may expect that the problem behaves symmetrically in terms of ℓ and $n^2 - \ell$, as in the proof of Theorem 4.8. But this is not the case. While our algorithm requires only $O(n)$ queries to decide if $AB = \mathbb{0}$, deciding if $AB = J$ is a strictly harder problem. We showed in the previous section that the LC_2^0 problem is equivalent to checking if $Av = \vec{1}$ for a known matrix A. Thus deciding if $AB = J$, even when A is known, is at least $\sqrt{n}$ times harder than the LC_2^0 problem, using the composition theorem (Theorem 1.2). Thus deciding if $AB = J$ requires $\Omega(\sqrt{n}Q(\mathsf{LC}_2^0)) = \Omega(n^{1.055})$ queries, even when A is known.

The query complexity of BMM when the number of zeros is small remains open.

Open Problem 4.6. What is the output-sensitive query complexity of BMM as parametrized by m, the number of zero entries in the output matrix? We know $Q(\mathsf{BMM}) = O(n^{3/2}\sqrt{m})$ and that the problem of deciding of $AB = J$, where J is the all-ones matrix, requires $\Omega(n^{1.055})$ queries.

Chapter 5

Minor-closed graph properties

Chapter summary: We study the quantum query complexity of minor-closed graph properties, which include such problems as determining whether an n-vertex graph is planar, is a forest, or does not contain a path of a given length. We show that most minor-closed properties—those that cannot be characterized by a finite set of forbidden subgraphs—have quantum query complexity $\Theta(n^{3/2})$. To establish this, we prove an adversary lower bound using a detailed analysis of the structure of minor-closed properties with respect to forbidden topological minors and forbidden subgraphs. On the other hand, we show that minor-closed properties (and more generally, sparse graph properties) that can be characterized by finitely many forbidden subgraphs can be solved strictly faster, in $o(n^{3/2})$ queries. Our algorithms are a novel application of the quantum walk search framework and give improved upper bounds for several subgraph-finding problems.

This chapter is based on the following (conference and journal) papers:

[CK11a] Andrew M. Childs and Robin Kothari. Quantum query complexity of minor-closed graph properties. In *Proceedings of the 28th Symposium on Theoretical Aspects of Computer Science (STACS 2011)*, volume 9 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 661–672, 2011.

[CK12] Andrew M. Childs and Robin Kothari. Quantum query complexity of minor-closed graph properties. *SIAM Journal on Computing*, 41(6):1426–1450, 2012.

5.1 Introduction

In this chapter we study the query complexity of deciding whether a graph has a given property. The query complexity of graph properties has been studied for more than 40 years, yet old and easy-to-state conjectures regarding the deterministic and randomized query complexities of graph properties remain unresolved.

Classical and quantum query complexity of graph properties

For monotone graph properties, a wide class of graph properties including almost all the properties considered in this chapter, a widely believed set of related conjectures, attributed to Stål Aanderaa, Richard Karp, and Arnold Rosenberg, assert that the deterministic and randomized query complexities of recognizing such properties are $\binom{n}{2}$ and $\Theta(n^2)$ respectively, where n is the number of vertices [Ros73, LY02]. For deterministic query complexity this has been proved for prime n [KSS83] and up to a constant factor for all n [RV75]. For randomized query complexity the best known lower bound is $\Omega(n^{4/3}\log^{1/3} n)$ [CK07].

However, the quantum query complexity of graph properties can be harder to pin down than its classical counterparts, since there exist monotone graph properties whose quantum query complexity is $\Theta(n)$, and others with quantum query complexity $\Theta(n^2)$. In fact, one can construct a monotone graph property with quantum query complexity $\Theta(n^{1+\alpha})$ for any fixed $0 \leq \alpha \leq 1$ using known bounds for the threshold function [BBC+01].

Prior work

The quantum query complexity of several specific graph properties has been established in prior work. Dürr, Heiligman, Høyer, and Mhalla [DHHM06] studied the query complexity of several graph problems, and showed in particular that connectivity has quantum query complexity $\Theta(n^{3/2})$. Zhang [Zha05] showed a lower bound of $\Omega(n^{3/2})$ for problems such as bipartiteness and perfect matching. Ambainis et al. [AIN+08] showed that planarity also has quantum query complexity $\Theta(n^{3/2})$. Berzina et al. [BDF+04] showed several quantum lower bounds on graph properties, including Hamiltonicity. Sun, Yao, and Zhang [SYZ04] exhibited a nonmonotone graph property with quantum query complexity $\tilde{O}(\sqrt{n})$ and showed that all nontrivial graph properties have quantum query complexity $\Omega(\sqrt{n})$.

Despite this work, the quantum query complexity of many interesting graph properties remains unresolved. As discussed in Section 4.3, a well-studied graph property whose query complexity is unknown is the property of containing a triangle (i.e., a cycle on 3 vertices) as a subgraph. The best known upper bound is $\tilde{O}(n^{1.25})$ due to Le Gall [Le 14], while the best known lower bound is only $\Omega(n)$. More generally, we can consider the H-subgraph containment problem, in which the task is to determine whether the input graph contains a fixed graph H as a subgraph.

Magniez et al. gave a general algorithm for H-subgraph containment using $\tilde{O}(n^{2-2/d})$ queries, where $d > 3$ is the number of vertices in H [MSS07], which was the best known algorithm for H-subgraph containment at the time of publication of this work, although better algorithms are now known [LMS11, Zhu12, LMS13]. The complexity of the best known upper bound [LMS13] does not have a concise description and can be obtained by solving a linear program. However, the query complexity of these algorithms is still superlinear in n, while the best lower bound known for H-subgraph containment is only $\Omega(n)$.

Our results

In this chapter we study the quantum query complexity of minor-closed graph properties. A property is minor closed if all minors of a graph possessing the property also possess the property. (Graph minors are defined in Section 5.2.) Since minor-closed properties can be characterized by forbidden minors, this can be viewed as a variant of subgraph containment in which we look for a given graph as a minor instead of as a subgraph. The canonical example of a minor-closed property is the property of being planar. Other examples include the property of being a forest, being embeddable on a fixed two-dimensional manifold, having treewidth at most k, and not containing a path of a given length.

While any minor-closed property can be described by a finite set of forbidden minors, some minor-closed properties can also be described by a finite set of forbidden subgraphs, graphs that do not appear as a subgraph of any graph possessing the property. We call a graph property (which need not be minor closed) a *forbidden subgraph property* (FSP) if it can be described by a finite set of forbidden subgraphs. Our main result is that the quantum query complexity of minor-closed properties depends crucially on whether the property is FSP. We show that any nontrivial minor-closed property that is not FSP has query complexity $\Theta(n^{3/2})$, whereas any minor-closed property that is FSP can be decided using $O(n^{\alpha})$ queries for some $\alpha < 3/2$, and in particular the query complexity is $o(n^{3/2})$. In general, the value of α may depend on the property; we give upper bounds on α that depend on the sizes of certain vertex covers.

Figure 5.1 summarizes our understanding of the quantum query complexity of minor-closed graph properties. All subgraph-closed properties, which include minor-closed properties and FSPs, have an easy lower bound of $\Omega(n)$ (Theorem 5.5). Furthermore, all sparse graph properties, which are defined in Section 5.2 and which include all minor-closed properties, have an easy upper bound of $O(n^{3/2})$ (Theorem 5.10). On the lower bound side, our main contribution is to show that minor-closed properties that are not FSP require $\Omega(n^{3/2})$ queries (Theorem 5.8), which tightly characterizes their quantum query complexity. Regarding upper bounds, our main contribution is a quantum algorithm for all sparse graph properties that are FSP using $o(n^{3/2})$ queries (Corollary 5.5).

Our lower bounds (Section 5.3) use the original (positive-weights) quantum adversary method of Ambainis [Amb02]. The basic idea of the main lower bound is similar to the connectivity lower

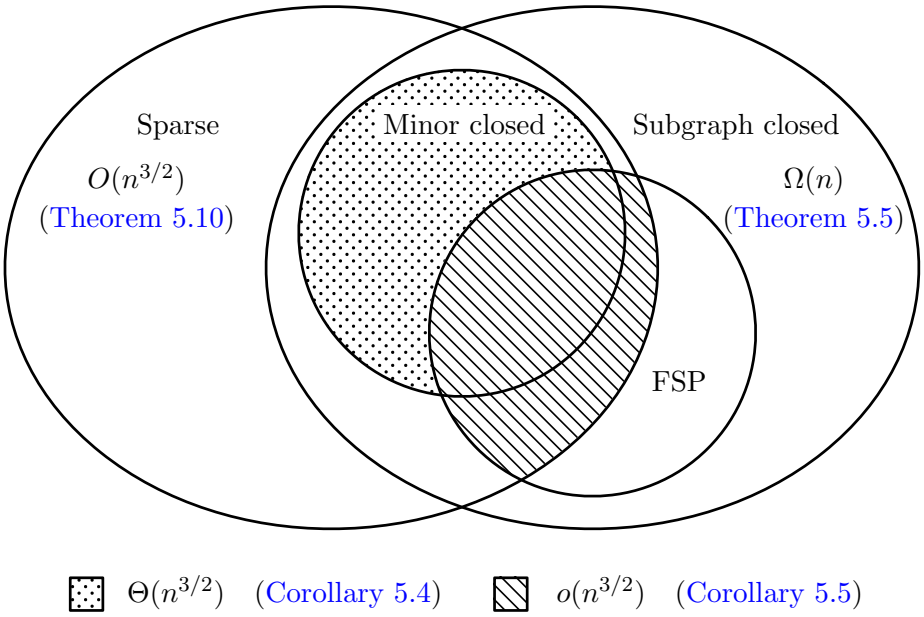

Figure 5.1: Summary of the main results.

bound of Dürr et al. [DHHM06]. However, it is nontrivial to show that this approach works using only the hypothesis that the property is minor closed and not FSP. In fact, we show a slightly stronger result, assuming only that the property is not FSP and can be described by finitely many forbidden topological minors.

Our upper bounds (Section 5.4) use the quantum walk search formalism [MNRS11]. Our approach differs from previous applications of this formalism in several respects. First, the graph underlying our quantum walk is a Hamming graph, rather than the Johnson graph used in previous algorithms. (This difference is not essential, but it simplifies the algorithm, and a similar approach may prove useful in pedagogical treatments of other quantum walk search algorithms.) Second, our algorithms involve several quantum walks occurring simultaneously on different Hamming graphs; although this can be viewed as a single walk on a larger graph, the salient feature is that the walks on different graphs proceed at different speeds, i.e., in each time step a different number of steps are taken on each graph. Third, our quantum walk algorithm makes essential use of the sparsity of the input graph, and to do so the algorithm must make queries even to determine which vertices of the input graph to search over (namely, to find vertices of a given degree).

The fact that our quantum walk can exploit sparsity allows us to improve upon known algorithms for many sparse graph properties, even if they are not necessarily minor closed. In

particular, we give algorithm that outperforms the general algorithm of Magniez et al. [MSS07] whenever H is a bipartite graph.

Finally, as another application, we consider the C_4-subgraph containment problem, where C_4 is the cycle on 4 vertices. This can be viewed as a natural extension of the triangle problem, which is C_3-subgraph containment. We show that C_4 finding can be solved with only $\tilde{O}(n^{1.25})$ queries. This was surprising at the time this work was published because the best known upper bound for triangle finding at that time was $\tilde{O}(n^{1.3})$ [MSS07]. However, the query complexity of triangle finding has recently been improved to $\tilde{O}(n^{1.25})$ [Le 14].

Subsequent developments

After this work was published, some of the algorithmic results of our work were improved by Belovs and Reichardt [BR12]. Using a reduction to s-t connectivity, they show that a path of any fixed length can be detected in $O(n)$ queries. More generally, they show that any minor-closed property that is FSP and can be described by exactly one forbidden subgraph, which must necessarily be a path or a (subdivided) claw, has query complexity $\Theta(n)$. This raises the possibility that all minor-closed properties that are FSP could have linear query complexity, although this remains open for properties characterized by more than one forbidden subgraph.

5.2 Preliminaries

In this chapter, all graphs are simple and undirected. Thus a graph on n vertices is specified by $\binom{n}{2}$ bits. The input graph is accessed by querying a black box to learn any of these $\binom{n}{2}$ bits. Deterministic and randomized algorithms have access to a black box taking two inputs, u and v, and returning a bit indicating whether (u, v) is an edge in the graph. Quantum algorithms have access to a unitary gate that maps $|u, v, b\rangle$ to $|u, v, b \oplus e\rangle$ where $(u, v) \in V \times V$, b is a bit, and e is 1 if and only if $(u, v) \in E$.

Let the deterministic, randomized, and quantum query complexities of determining whether a graph possesses property $\mathcal{P}$ be denoted as $D(\mathcal{P})$, $R(\mathcal{P})$, and $Q(\mathcal{P})$, respectively. Clearly, $Q(\mathcal{P}) \leq R(\mathcal{P}) \leq D(\mathcal{P}) \leq \binom{n}{2}$. Also note that these query complexities are the same for a property $\mathcal{P}$ and its complement $\bar{\mathcal{P}}$, since any algorithm for $\mathcal{P}$ can be turned into an algorithm for $\bar{\mathcal{P}}$ by negating the output, using no additional queries.

A graph property on n vertices is a property of n-vertex graphs that is independent of vertex labeling, i.e., isomorphic graphs are considered equivalent. For a graph G on n vertices and an n-vertex graph property $\mathcal{P}_n$, we write $G \in \mathcal{P}_n$ to mean that graph G has property $\mathcal{P}_n$. A graph property $\mathcal{P} := \{\mathcal{P}_n\}_{n=1}^{\infty}$ is a collection of n-vertex graph properties $\mathcal{P}_n$ for all $n \in N$. For example, the property "the first vertex is isolated" is not a graph property because it depends on

the labeling, and in particular it depends on which vertex we decide to call the first one. However, the property "contains an isolated vertex" is a graph property.

An n-vertex graph property $\mathcal{P}_n$ is nontrivial if there exists a graph that possesses it and one that does not. A graph property $\mathcal{P} = \{\mathcal{P}_n\}_{n=1}^{\infty}$ is nontrivial if there exists an n_0 such that $\mathcal{P}_n$ is nontrivial for all $n > n_0$. Thus a property such as "contains a clique of size 5" is nontrivial, although it is trivial for graphs with fewer than 5 vertices.

In this chapter, K_n and C_n refer to the complete graph and cycle on n vertices, respectively. $K_{s,t}$ is the complete bipartite graph with s vertices in one part and t vertices in the other. The claw graph is $K_{1,3}$ and a star graph is $K_{1,t}$ for any t. For a graph G, $V(G)$ and $E(G)$ denote the vertex and edge sets of the graph; $n := |V(G)|$ and $m := |E(G)|$.

A graph H is said to be a subgraph of G, denoted $H \leq_S G$, if H can be obtained from G by deleting edges and isolated vertices. A graph H is said to be a minor of G, denoted $H \leq_M G$, if H can be obtained from G by deleting edges, deleting isolated vertices, and contracting edges. To contract an edge (u, v), we delete the vertices u and v (and all associated edges) and create a new vertex that is adjacent to all the original neighbors of u and v. The name "edge contraction" comes from viewing this operation as shrinking the edge (u, v) to a point, letting the vertices u and v coalesce to form a single vertex as shown in Figure 5.2. When working with multigraphs, edge contraction may be defined to allow the possibility of multiple edges. However, all graphs are simple in this chapter.

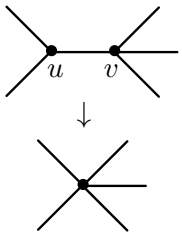

Figure 5.2: Contracting an edge (u, v).

Another way to understand graph minors is to consider reverse operations: $H \leq_M G$ if G can be obtained from H by adding isolated vertices, adding edges, and performing vertex splits. In a vertex split, we delete a vertex u and add two new adjacent vertices v and w, such that each original neighbor of u becomes a neighbor of either v or w, or both. This is the reverse of the operation depicted in Figure 5.2. In general, this operation does not lead to a unique graph, since there may be many different ways to split a vertex.

A related operation, which is a special case of a vertex split, is known as an elementary subdivision. This operation replaces an edge (u, v) with two edges (u, w) and (w, v), where w is a new vertex. A graph H is said to be a topological minor of G, denoted $H \leq_T G$, if G can be obtained from H by adding edges, adding isolated vertices, and performing elementary subdivisions. We call G a subdivision of H if it is obtained from H by performing any number of elementary subdivisions.

For example, observe that if H is a path, then $H \leq_S G \Leftrightarrow H \leq_T G \Leftrightarrow H \leq_M G$. More generally, if every connected component of a graph H is a subdivision of a star (which includes paths), then $H \leq_S G \Leftrightarrow H \leq_T G$ and these are the only graphs with this property (Lemma 5.4). Similarly, if every connected component of H is a path or a subdivision of a claw ($K_{1,3}$), then $H \leq_S G \Leftrightarrow H \leq_M G$ and these are the only graphs with this property.

Some graph properties can be expressed using a forbidden graph characterization. Such a characterization says that graphs have the property if and only if they do not contain any of some set of forbidden graphs according to some notion of graph inclusion, such as subgraphs or minors. For example, a graph is a forest if and only if it contains no cycle as a subgraph, so forests are characterized by the forbidden subgraph set $\{C_k \colon k \geq 3,\, k \in \mathbb{N}\}$. The property of being a forest can also be characterized by the single forbidden minor C_3, since a graph is a forest if and only if it does not contain C_3 as a minor. If a property can be expressed using a finite number of forbidden subgraphs, we call it a forbidden subgraph property (FSP). A property is said to be subgraph closed if every subgraph of a graph possessing the property also possesses the property. (Note that subgraph-closed properties differ from monotone graph properties in that the latter allow only edge deletion, not vertex deletion.) Similarly, a property is said to be minor closed if all minors of a graph possessing the property also possess the property. In a series of 20 papers spanning over 20 years, Robertson and Seymour proved the following theorem [RS04]:

Theorem 5.1 (Graph minor theorem)**.** *Every minor-closed graph property can be described by a finite set of forbidden minors.*

We also require the following consequence of the graph minor theorem, which follows using well-known facts about topological minors [RS90, Theorem 2.1].

Corollary 5.1. *Every minor-closed graph property can be described by a finite set of forbidden topological minors.*

Note that the converse of this statement is not true, i.e., there exist properties that are not minor closed but can be described by finitely many forbidden topological minors. For example, let H be the graph on 8 vertices that is formed by adding an edge between the degree-3 vertices of two disjoint claw graphs (i.e., it looks like $+\!+$). Then the property of not containing H as a topological minor is not minor closed. It is also easy to verify that this property is not FSP.

We call a graph property *sparse* if there exists a constant c such that every graph G with the property has $|E(G)| \leq c\,|V(G)|$. Nontrivial minor-closed properties are sparse, which is an easy corollary of Mader's theorem [Mad67].

Theorem 5.2. *Every nontrivial minor-closed graph property is sparse.*

Proof. Mader's theorem [Mad67] states that there exists a function f such that any graph G that does not contain K_h as a minor must have $|E(G)| \leq f(h)\,|V(G)|$. (The function f has been subsequently improved [Tho01].)

Since the family is nontrivial, there is some complete graph K_h that is not contained in this family. Since the family is minor closed, no graph which contains K_h as a minor is in this family either. Thus this set of graphs is a subset of the family of graphs which do not contain K_h as a minor, and therefore we have $|E(G)| \leq f(h)|V(G)|$ for all graphs G in this nontrivial minor-closed family. □

5.3 Lower bounds

In this section, we prove lower bounds on the quantum query complexity of graph properties. We first show a simple $\Omega(n)$ lower bound for all subgraph-closed properties. With the exception of Theorem 5.10, this covers all the properties considered in this chapter, since every property considered (or its complement) is subgraph closed.

We then describe an $\Omega(n^{3/2})$ lower bound for the problem of determining whether a graph contains a cycle and explain how a similar strategy can be applied to any graph property that is closed under topological minors, provided we can identify a suitable edge of a forbidden topological minor. Next we introduce a tool used in our general lower bounds, namely a graph invariant that is monotone with respect to topological minors. With this tool in hand, we show an $\Omega(n^{3/2})$ lower bound for any H-topological minor containment problem that does not coincide with H-subgraph containment. We conclude by showing the main result of this section, that every nontrivial minor-closed property $\mathcal{P}$ that is not FSP has $Q(\mathcal{P}) = \Omega(n^{3/2})$.

The quantum lower bounds in this chapter use the original (positive-weights) quantum adversary method of Ambainis [Amb02].

Theorem 5.3 (Adversary bound)**.** *Let $f(x_1, \ldots, x_n)$ be a function of n bits and let X, Y be two sets of inputs such that $f(x) \neq f(y)$ if $x \in X$ and $y \in Y$. Let $R \subseteq X \times Y$ be a relation. Let the values m, m', $l_{x,i}$, $l'_{y,i}$ for $x \in X$, $y \in Y$ and $i \in \{1, \ldots, n\}$ be such that the following hold.*

1. *For every $x \in X$, there are at least m different $y \in Y$ such that $(x, y) \in R$.*
2. *For every $y \in Y$, there are at least m' different $x \in X$ such that $(x, y) \in R$.*
3. *For every $x \in X$ and $i \in \{1, \ldots, n\}$, there are at most $l_{x,i}$ different $y \in Y$ such that $(x, y) \in R$ and $x_i \neq y_i$.*
4. *For every $y \in Y$ and $i \in \{1, \ldots, n\}$, there are at most $l'_{y,i}$ different $x \in X$ such that $(x, y) \in R$ and $x_i \neq y_i$.*

Let $l_{\max}$ be the maximum of $l_{x,i}\, l'_{y,i}$ over all $(x, y) \in R$ and $i \in \{1, \ldots, n\}$ such that $x_i \neq y_i$. Then any quantum algorithm computing f with probability at least 2/3 requires $\Omega\left(\sqrt{\frac{mm'}{l_{\max}}}\right)$ queries.

For the classical lower bound in the next section, we use the following theorem of Aaronson [Aar06].

Theorem 5.4 (Classical adversary bound)**.** *Let $f, X, Y, R, m, m', l_{x,i}, l'_{y,i}$ be as in Theorem 5.3. Let v be the maximum of $\min\{l_{x,i}/m, l'_{y,i}/m'\}$ over all $(x, y) \in R$ and $i \in \{1, \ldots, n\}$ such that $x_i \neq y_i$. Then any randomized algorithm computing f with probability at least 2/3 requires $\Omega(1/v)$ queries.*

5.3.1 Subgraph-closed properties

We begin with the $\Omega(n)$ lower bound for all nontrivial subgraph-closed graph properties.

Theorem 5.5 (Subgraph-closed properties)**.** *For any nontrivial subgraph-closed graph property $\mathcal{P}$, $Q(\mathcal{P}) = \Omega(n)$, $R(\mathcal{P}) = \Theta(n^2)$, and $D(\mathcal{P}) = \Theta(n^2)$.*

Proof. Since $\mathcal{P}$ is nontrivial, for all $n \geq n_0$ there exists a graph on n vertices that is in $\mathcal{P}$, and since $\mathcal{P}$ is subgraph closed, this implies that the empty graph on n vertices is in $\mathcal{P}$. Since $\mathcal{P}$ is nontrivial, there exists a graph H not in $\mathcal{P}$. Since $H \notin \mathcal{P}$, all supergraphs of H are also not in $\mathcal{P}$. Now let $d = \max\{|V(H)|, n_0\}$. Then the empty graph on d vertices is in $\mathcal{P}$, while the complete graph on d vertices, K_d, is not in $\mathcal{P}$.

We prove a lower bound for the problem of distinguishing the empty graph on n vertices and the n-vertex graph formed by K_d and $n - d$ isolated vertices, for all $n \geq d$.

Let X contain only one graph, the empty graph on n vertices. Let Y contain all graphs on n vertices that contain exactly one copy of K_d and $n - d$ isolated vertices. Let R be the trivial relation, $R = X \times Y$. Thus, in the notation of Theorem 5.3, m equals $|Y| = \binom{n}{d}$. Similarly, $m' = |X| = 1$. Since X contains only the empty graph, $l_{x,i}$ counts the number of graphs in Y in which the i^{th} edge is present. Since one edge fixes two vertices of K_d, the number of related graphs in Y with a given edge is $\binom{n-2}{d-2}$. Since X contains only 1 graph, $l'_{y,i} \leq 1$.

The quantum adversary method (Theorem 5.3) gives us $Q(\mathcal{P}) = \Omega\left(\sqrt{\binom{n}{d}/\binom{n-2}{d-2}}\right) = \Omega(n)$. Theorem 5.4 gives us $R(\mathcal{P}) = \Omega(n^2)$ and thus $D(\mathcal{P}) = \Omega(n^2)$. Since all query complexities are at most $\binom{n}{2}$, we have $D(\mathcal{P}) = \Theta(n^2)$ and $R(\mathcal{P}) = \Theta(n^2)$. □

Note that this theorem can also be proved by reduction from the unstructured search problem using Turán's theorem. By Turán's theorem, the densest graph on n vertices that does not contain K_d as a subgraph has $\Theta(n^2)$ nonedges. Thus any algorithm that decides the graph property must be able to distinguish the densest graph from the same graph with one extra edge. Since there are $\Theta(n^2)$ nonedges, the problem of searching a space of size $\Theta(n^2)$ can be reduced to this, giving the appropriate lower bounds.

This theorem shows that all the properties considered in this chapter are classically uninteresting from the viewpoint of query complexity, since their classical (deterministic or randomized) query complexity is $\Theta(n^2)$.

5.3.2 Acyclicity

We now show an $\Omega(n^{3/2})$ lower bound for the problem of determining whether a graph is acyclic (i.e., a forest). We then isolate the main idea of this proof, formulating a lemma that is useful

Figure 5.3: An example of two graphs on 10 vertices such that $(x, y) \in R$.

for establishing lower bounds for more general topological minor-closed graph properties.

The lower bound for acyclicity is similar to the connectivity lower bound of Dürr et al. [DHHM06]. The intuition is that a long path and a long cycle look the same locally. Since algorithms only have access to local information, these two graphs should be hard to distinguish. Unfortunately this is not sufficient, since a path can be easily distinguished from a cycle by searching the entire graph for a degree-1 vertex, which can be done with $O(n)$ queries. Instead, we try to distinguish a path from the disjoint union of a cycle and a path. Now both graphs have 2 degree-1 vertices. We require both the cycle and the path to be long, since a short cycle or path could be quickly traversed.

Theorem 5.6. *Deciding if a graph is a forest requires $\Omega(n^{3/2})$ queries.*

Proof. Let X be the set of all graphs on n vertices isomorphic to the path on n vertices. Let Y be the set of all graphs on n vertices that are the disjoint union of a path and a cycle, such that no vertex is isolated and both the cycle and path have more than $n/3$ vertices. Clearly all graphs in X are forests and all graphs in Y are not. Let $(x, y) \in R$ if there exist 4 vertices a, b, c, d, such that the only difference between x and y is that in x, the induced subgraph on these vertices has only the edges (a, b) and (c, d), but in y, the induced subgraph has only edges (a, c) and (b, d). See Figure 5.3 for an example of two related graphs on $n = 10$ vertices.

Now let us compute the relevant quantities from Theorem 5.3. Recall that m is the minimum number of graphs $y \in Y$ that each graph $x \in X$ is related to. Each graph in X can be transformed to a related graph in Y by selecting edges (a, b) and (c, d) such that the distance between the edges is between $n/3$ and $2n/3$. There are $n - 1$ ways to pick the edge (a, b), and for any choice of (a, b) there are $n/3$ possible edges (c, d), which gives $m = \Omega(n^2)$. Similarly, $m' = \Omega(n^2)$, since in a graph $y \in Y$, we can choose any one edge in the cycle to be (a, c) and any one in the path to be (b, d).

Now let us compute $l_{\max}$. The quantity $l_{x,i}$ counts the number of graphs in Y that are related to x and differ from x at the i^{th} edge. The variable i indexes the bits of the adjacency matrix; let x_i indicate whether the i^{th} edge is present or absent in x. Since $l_{\max}$ is the maximum of $l_{x,i}\, l'_{y,i}$ over all related pairs (x, y) such that $x_i \neq y_i$, let us first compute the maximum of $l_{x,i}\, l'_{y,i}$ over all related pairs (x, y) where $x_i = 0$ and $y_i = 1$.

Since $x_i = 0$, the i^{th} edge is not present in x and is present in the related graphs y, so without loss of generality the i^{th} edge is (a, c). To obtain a graph in Y, we need to choose vertices b and d such that (a, b) and (c, d) are edges in x. We can choose either of a's two neighbors to be b and either of c's two neighbors to be d. This gives at most 4 different $y \in Y$ that are related to this x and differ at the i^{th} edge. Thus $l_{x,i} \leq 4$ when $x_i = 0$. (The reason for the inequality is that sometimes this may not yield a valid graph in y, e.g., when the resulting graph is not of the appropriate form, or when the resulting path or cycle is too short.)

Since $y_i = 1$, $l'_{y,i}$ counts the number of graphs in X related to y that do not have the i^{th} edge. Again, we can assume this edge is (a, c), since it is present in y but not in related graphs in X. If the i^{th} edge is an edge on the path in y (as opposed to the cycle), then choosing any edge in the cycle will give two vertices b and d that give rise to valid graphs in x when the edges (a, b) and (c, d) are added and (a, c) and (b, d) are deleted. Thus there are $O(n)$ possible choices for b and d, which gives $O(n)$ related graphs in X. If (a, c) is an edge on the cycle, then we can choose any edge on the path as (b, d). This again leads to at most $O(n)$ possibilities, which gives us $l'_{y,i} = O(n)$. Thus $l_{x,i}\, l'_{y,i} = O(n)$ for all $(x, y) \in R$ when $x_i = 0$ and $y_i = 1$.

Now we need to compute $l_{x,i}\, l'_{y,i}$ when $x_i = 1$ and $y_i = 0$. The values of $l_{x,i}$ and $l'_{y,i}$ are similar for this case: one is $O(1)$ and the other is $O(n)$. This gives $l_{x,i}\, l'_{y,i} = O(n)$ for all $(x, y) \in R$ such that $x_i \neq y_i$. Thus $l_{\max} = O(n)$.

Using Theorem 5.3, we get a lower bound of $\Omega(\sqrt{n^4/n}) = \Omega(n^{3/2})$. □

Cyclicity can be viewed as the property of containing C_3 as a minor, or equivalently, as a topological minor. Note that for any constant r, the same proof also works for the property of containing C_r as a minor (i.e., containing a cycle of length at least r), since the graphs in X did not contain any cycle, and the graphs in Y contained a cycle of length at least $n/3$, which contains any constant-sized cycle as a minor.

Inspecting the proof technique more closely, we see that no particular property of forests was used, other than the facts that all subdivisions of C_3 are not forests, and that if we delete an edge from a subdivision, the resulting graph is a forest. More precisely, we used the existence of a graph G (in this case C_3) and an edge $(u, v) \in E(G)$ (since C_3 is edge transitive it does not matter which edge is chosen) such that if (u, v) is subdivided any number of times, the resulting graph still does not have the property (in this case, of being a forest) and if (u, v) is replaced by two disjoint paths the resulting graph does have the property. The following lemma formalizes this intuition.

Lemma 5.1. *Let $\mathcal{P}$ be a graph property closed under topological minors. If there exists a graph $G \notin \mathcal{P}$ and an edge (u, v) in G, such that replacing the edge (u, v) by two disjoint paths of any length, one connected to vertex u and the other connected to vertex v, always results in a graph $G_1 \in \mathcal{P}$, then $Q(\mathcal{P}) = \Omega(n^{3/2})$.*

Proof. The proof is similar in structure to Theorem 5.6 and subsumes it if we take $G = C_3$. For the general case, let G be a graph on k vertices.

Let G_1 be the graph G with the edge (u, v) deleted, a path of length n_1 attached at vertex u, and a path of length n_2 attached at vertex v, such that $n_1, n_2 \geq n/3$ and $|V(G_1)| = n$. By assumption, $G_1 \in \mathcal{P}$. Let X be the set of all graphs isomorphic to G_1.

Let G_2' be the graph G with the edge (u, v) subdivided n_1 times, where $n_1 \geq n/3$. This is equivalent to replacing the edge by a path of length $n_1 + 1$. Let G_2 be the disjoint union of G_2' and a path of length n_2, such that $n_2 \geq n/3$ and $|V(G_2)| = n$. Clearly $G_2 \notin \mathcal{P}$, since it contains G as a topological minor. Let Y be the set of all graphs isomorphic to G_2.

As before, let $(x, y) \in R$ if there exist 4 vertices a, b, c, d, such that the only difference between x and y is that in x, the induced subgraph on these vertices has only the edges (a, b) and (c, d), but in y, the induced subgraph has only the edges (a, c) and (b, d).

Each graph in X can be transformed to a related graph in Y by first selecting an edge (a, b) in the first path and then picking another edge (c, d) in the second path. Each path contains more than $n/3$ edges, but we have to satisfy the condition that when the edges (a, b) and (c, d) are removed and replaced with (a, c) and (b, d), the resulting graph is in Y. This means that after swapping the edges, both the long disjoint path and the path between vertices u and v have to be longer than $n/3$. Even with this restriction there are $\Omega(n^2)$ graphs in Y related to any graph $x \in X$. For example, we can choose any edge on the shorter path to be (a, b), and then there are at least $n/6$ edges on the longer path which can be chosen to be (c, d), which will give a graph in Y when the edges (a, b) and (c, d) are removed and replaced with (a, c) and (b, d).

Similarly, $m' = \Omega(n^2)$. We have to choose an edge from the disjoint path and one from the path which connects the vertices u and v. Again, we can choose any edge from the smaller of the two paths, and then there are still at least $n/6$ edges in the other path left to choose, such that if those edges are chosen as (a, c) and (b, d), and then we perform the swap (i.e., (a, c) and (b, d) are removed and replaced with (a, b) and (c, d)), this results in a graph in X.

Now let us upper bound the maximum of $l_{x,i}\, l'_{y,i}$ over all related pairs (x, y) where $x_i = 0$ and $y_i = 1$. Since the i^{th} edge is not present in x, we can take this to be the edge (a, c). To obtain a graph in Y, we need to choose vertices b and d such that (a, b) and (c, d) are edges in x. We can choose any one of a's neighbors to be b and any one of c's neighbors to be d. (For some edges, this procedure may not give a graph in Y, but we only need an upper bound.) Since a and c have $O(1)$ neighbors, $l_{x,i} = O(1)$ when $x_i = 0$.

To compute $l'_{y,i}$, we can assume the i^{th} edge is the edge (a, c), since it is present in the graph y but not in related graphs in X. If the i^{th} edge is an edge on the disjoint path in y or the path connecting vertices u and v, there can be at most $O(n)$ choices for the edge (b, d) on the other path that gives rise to a valid graph in X when the edges (a, b) and (c, d) are added and (a, c) and (b, d) are deleted. As before, sometimes there may be no related graphs with the i^{th} edge

absent, but we only require an upper bound. Thus $l_{x,i}\, l'_{y,i} = O(n)$ for all $(x, y) \in R$ when $x_i = 0$ and $y_i = 1$.

Now we need to compute $l_{x,i}\, l'_{y,i}$ when $x_i = 1$ and $y_i = 0$. The values of $l_{x,i}$ and $l'_{y,i}$ are similar for this case: one is $O(1)$, and the other is $O(n)$. This gives $l_{x,i}\, l'_{y,i} = O(n)$ for all $(x, y) \in R$ such that $x_i \neq y_i$. Thus $l_{\max} = O(n)$.

Using Theorem 5.3, we get a lower bound of $\Omega(\sqrt{n^4/n}) = \Omega(n^{3/2})$. □

5.3.3 A graph invariant for topological minor containment

To identify suitable edges for use in Lemma 5.1, we introduce a graph invariant that is monotone with respect to topological minors. As a simple application, we use this invariant to show an $\Omega(n^{3/2})$ lower bound for all H-topological minor containment properties that are not also H-subgraph containment properties.

Call an edge *internal* if it lies on a cycle or on a path joining two vertices of degree 3 or greater. Call all other edges *external*. Note that an edge is external if and only if it belongs to a *dangling path*, a vertex subset $\{v_1, v_2, \ldots, v_k\}$ such that v_1 is adjacent to v_2, v_2 is adjacent to v_3, etc., where v_1 has degree 1 and $v_2, v_3, \ldots, v_{k-1}$ have degree 2. For a graph G, let $\beta(G)$ denote the number of internal edges in G. We claim that β is monotone with respect to topological minors.

Lemma 5.2. *If $H \leq_T G$ then $\beta(H) \leq \beta(G)$.*

Proof. Let H be a topological minor of G, where G is obtained from H in one step, i.e., by performing an elementary subdivision or by adding a single edge or a single isolated vertex. We show that $\beta(H) \leq \beta(G)$. Then the same inequality follows when H is obtained from G by any number of steps.

It is clear that adding an isolated vertex does not change the β value of a graph. Adding an edge to H results in a supergraph of H, which contains all the high degree vertices and cycles that H does (and possibly more). Therefore each internal edge in H remains an internal edge in G, which shows that the β value cannot decrease.

Finally, since subdividing an edge in H replaces the edge with a path of length 2, this does not change the connectivity of the graph. Any paths that used the subdivided edge can now use the path of length 2 instead. All the vertices of H have the same degree in G. Thus cycles cannot be destroyed by subdivision, and neither can a path between any two particular vertices. □

We use a specific topological minor operation that strictly decreases the invariant.

Lemma 5.3. *In a graph G, deleting an internal edge (u, v) and adding two disjoint paths, one starting from vertex u and one from v, decreases the β value of the resulting graph H.*

Proof. It is clear that merely deleting the internal edge (u, v) decreases the β value of the graph. Let us now show that every internal edge in H is also an internal edge in G.

First, observe that none of the edges that were added to H as part of the two disjoint paths are internal. This follows because the added edges lie on a path with one end connected to u or v and the other end free. No edge on this path is part of a cycle, and the path does not contain any vertices of degree 3 or more.

It remains to consider edges of H that are also present in G. If an edge is internal in H because it lies on a cycle, then it is also internal in G since all cycles in H are present in G. If an edge is internal in H because it lies on a path between two vertices of degree 3 or more, none of the vertices on that path can be on the added disjoint paths, since all the added vertices have degree 1 or 2, and all vertices that are present in both H and G have exactly the same degree in both graphs. Thus such an edge is internal in G as well. Since G has all the internal edges of H, and at least one more (the edge (u, v)), $\beta(H) < \beta(G)$. □

Finally, the graphs with $\beta(H) = 0$ have a nice characterization.

Lemma 5.4. *For any graph H, H-topological minor containment is equivalent to H-subgraph containment if and only if $\beta(H) = 0$.*

Proof. If H is a subgraph of another graph, then it is also a topological minor of that graph. Thus to show the equivalence of topological minor containment and subgraph containment, we only have to show that if H is a topological minor of G then H is also a subgraph of G.

If $\beta(H) = 0$, then H contains no internal edges, which means each connected component of H is a subdivision of a star graph. Then it is easy to see that every subdivision of H contains H as a subgraph.

To show the converse, note that if H is a graph in which some connected component is not a subdivision of a star, then H must have 2 vertices of degree 3 in a connected component or have a cycle. For these graphs we exhibit a subdivision of H which does not contain H as a subgraph.

Let H be a graph on k vertices that contains a cycle. Let G be the graph obtained by performing k subdivisions on every edge of H. Now the smallest cycle in G is at least k times longer than the smallest cycle in H. Thus G contains no cycles of length less than or equal to k. But H is a graph on k vertices, and contains a cycle, which must be of length less than or equal to k. Therefore H cannot be a subgraph of G.

Finally, let H be a k-vertex acyclic graph with 2 or more vertices of degree 3 in the same connected component. Again let G be the graph obtained by performing k subdivisions on every edge of H. Now the shortest path joining any pair of degree-3 vertices in G has length greater than k. However, any path joining 2 degree-3 vertices in H has length less than or equal to k.

Thus H cannot be a subgraph of G, since deleting edges and isolated vertices cannot decrease the shortest path between two vertices. □

Using this invariant together with Lemma 5.1, we can easily show an $\Omega(n^{3/2})$ lower bound for H-topological minor containment assuming that this property does not coincide with H-subgraph containment. Since Lemma 5.4 characterizes such graphs, we know that H must be cyclic or contain 2 vertices of degree at least 3 in the same connected component.

Theorem 5.7. *For all graphs H, if H-topological minor containment is not equivalent to H-subgraph containment, then the quantum query complexity of H-topological minor containment is $\Omega(n^{3/2})$.*

Proof. To apply Lemma 5.1, $\mathcal{P}$ must be closed under topological minors. Thus, we consider the property of *not* containing H as a topological minor.

We require a graph $G \notin \mathcal{P}$ with the properties stated in Lemma 5.1. Let G be the graph H itself, and let (u, v) be any internal edge, which must exist by Lemma 5.4. By Lemma 5.3, replacing (u, v) with two disjoint paths results in a graph G' with $\beta(G') < \beta(H)$. Thus G' cannot contain H as a minor, which gives $G' \in \mathcal{P}$, and Lemma 5.1 gives us the $\Omega(n^{3/2})$ lower bound. □

5.3.4 Minor-closed properties

We are now ready to prove our main lower bound result, an $\Omega(n^{3/2})$ lower bound for any minor-closed graph property that is not FSP. By Corollary 5.1, any minor-closed graph property can be described in terms of forbidden topological minors. However, so far, we have only considered the case of a single forbidden topological minor. With multiple forbidden topological minors, some internal edges of some forbidden minors may not suffice for use in Lemma 5.1, since subdividing an internal edge of one minor may result in a graph that contains one of the other minors. Hence our main challenge is to identify a suitable edge for use in Lemma 5.1.

We now introduce some terminology that will only be used in this subsection. We call a set of graphs B *equivalent under topological minor containment and subgraph containment* if whenever a graph in B is a topological minor of a graph G, there is some graph in B that is also a subgraph of G.

Lemma 5.5. *For any graph property $\mathcal{P}$ that is not FSP and that is described by a finite set of forbidden topological minors, there exists a graph $G \notin \mathcal{P}$ and an edge $(u, v) \in E(G)$ satisfying the conditions of Lemma 5.1.*

Proof. We define $\mathcal{P}$ using two finite sets of forbidden topological minors, S and B, where the set B is equivalent under topological minor containment and subgraph containment. Clearly, such a

description exists, because we can take B to be the empty set and let S be the set of forbidden topological minors.

Among all possible descriptions of the property $\mathcal{P}$ in terms of finite forbidden sets S and B, we choose a description that minimizes $|S|$. Since $\mathcal{P}$ is not FSP, it cannot be described by a pair S and B where $S = \emptyset$ since B would then provide a forbidden subgraph characterization of $\mathcal{P}$. Let $l := |S| \neq 0$. Order the graphs in S by their β values, so that $S = \{H_1, H_2, \ldots H_l\}$ where $\beta(H_1) \leq \beta(H_2) \leq \ldots \leq \beta(H_l)$.

We claim that H_1 can serve as the required graph G for Lemma 5.1. Clearly, $H_1 \notin \mathcal{P}$. H_1 must contain an internal edge, since otherwise Lemma 5.4 implies that H_1 is equivalent under subgraph and topological minor containment, in which case H_1 could be removed from S and added to B, violating the minimality of S. It remains to show that one of the internal edges of H_1 satisfies the conditions of Lemma 5.1.

Toward a contradiction, assume that none of the internal edges of H_1 could serve as the edge (u, v). This means that for each internal edge there is a pair of disjoint paths such that the graph resulting from replacing the edge with this pair of paths, G', does not possess property $\mathcal{P}$. Since $G' \notin \mathcal{P}$, G' must contain some graph in S or B as a topological minor. Since an internal edge was deleted and replaced with two disjoint paths, the β value of the resulting graph has decreased (by Lemma 5.3). Since $\beta(G') < \beta(H_1)$ and $\beta(H_1) \leq \beta(H_i)$ for all $1 \leq i \leq l$, none of the graphs in S can be a topological minor of G', and thus only a graph in B can be a topological minor of G'.

Hence, for each edge (u, v), there exists a pair of disjoint paths such that when (u, v) is replaced by these paths, the resulting graph G' contains one of the graphs in B as a topological minor, and therefore as a subgraph. Let G'' be a supergraph of G' obtained by adding an extra edge that connects the loose ends of the two disjoint paths. Since G'' is obtained by replacing the edge (u, v) by a long path, it is a subdivision of H_1. Since G' contains a graph in B as a subgraph, so does G''.

It follows that for every internal edge (u, v) of H_1, there is a constant a such that if the edge (u, v) is subdivided a or more times, the resulting graph contains some graph from B as a subgraph. Let the maximum constant a over all internal edges be c. If any internal edge of H_1 is subdivided more than c times, it contains some graph from B as a topological minor. We use this fact to get a forbidden subgraph characterization for $\{H_1\} \cup B$.

Let the number of internal edges in H_1 be k. Let J be any graph that contains some graph from $\{H_1\} \cup B$ as a topological minor. If it contains some graph from B as a topological minor, it also contains some graph from B as a subgraph, so B already has a forbidden subgraph characterization. The only graph whose containment as a topological minor we have to express by a forbidden subgraph characterization is H_1. So let J contain H_1 as a topological minor. Since some subdivision of H_1 is a subgraph of J, let J' be a minimal subgraph of J that is a subdivision of H_1, i.e., no subgraph of J' is a subdivision of H_1.

We claim that if J' has more than $ck + |V(H_1)|$ vertices, then it already contains some graph from B as a subgraph. Since J' is a subdivision of H_1 and has ck more vertices than H_1, it must be obtained after ck subdivisions. Moreover, no subgraph of J' can be a subdivision of H_1. Thus J' must be obtained by subdividing only internal edges of H_1, since subdividing an external edge leads to a supergraph of the original graph (because an external edge must be on a dangling path). Since J' is obtained from H_1 by ck subdivisions of internal edges, at least one internal edge was divided c times. Let the graph with this edge divided c times be called H'. Since the order of subdivisions does not matter, $H' \leq_T J'$. However, by assumption there is a graph in B that is a topological minor of H'. By the transitivity of topological minor containment, there is a graph in B that is a topological minor of J'. But since B-subgraph containment and B-topological minor containment are equivalent, there is a graph in B that is a subgraph of J'. Thus we do not need to forbid any additional subgraphs in order to exclude graphs J' with more than $ck + |V(H_1)|$ vertices.

Now suppose that J' has fewer than $ck + |V(H_1)|$ vertices. Let F be the set of all subdivisions of H_1 with at most $ck + |V(H_1)|$ vertices. Clearly $J' \in F$, and F is a finite set of graphs. The set $F \cup B$ is now a forbidden subgraph characterization for the property of not containing any graph from $\{H_1\} \cup B$ as a topological minor.

This gives us a different representation of $\mathcal{P}$, using the new sets $S' = \{H_2, H_3, \ldots, H_l\}$ and $B' = F \cup B$. But $|S'| < l$, which contradicts the minimality of the original characterization. □

Combining this lemma with Corollary 5.1 and Lemma 5.1, we get the main result of this section.

Theorem 5.8. *For any nontrivial minor-closed property $\mathcal{P}$ that is not FSP, $Q(\mathcal{P}) = \Omega(n^{3/2})$.*

This lower bound cannot be improved due to a matching algorithm shown in Section 5.4. It cannot be extended to minor-closed properties that are also FSP because, as we also show in Section 5.4, every property of this type has query complexity $o(n^{3/2})$.

Since the complement of H-minor containment is minor closed, we have the following.

Corollary 5.2. *If H is a graph for which the property of not containing H as a minor is not FSP, then the quantum query complexity of H-minor containment is $\Omega(n^{3/2})$.*

Note that H-minor containment is not FSP for most graphs H. The only exceptions are graphs in which each connected component is a path or a subdivision of a claw ($K_{1,3}$). It is not hard to show that if H is such a graph, then H-minor containment is equivalent to H-subgraph containment. For such graphs H, one can show that H is a minor of G if and only if it is a topological minor of G [Die05, Proposition 1.7.4]. Then, by Lemma 5.4, such an H is a topological minor of G if and only if it is a subgraph, which proves the claim.

5.4 Algorithms

We now turn to quantum algorithms for deciding minor-closed graph properties, as well as related algorithms for subgraph-finding problems.

5.4.1 Sparse graph detection and extraction

We begin by describing some basic tools that allow us to detect whether a graph is sparse and to optimally extract the adjacency matrix of a sparse graph.

To tell whether a graph is sparse, we can apply quantum counting to determine approximately how many edges it contains. In particular, Theorem 15 of [BHMT02] gives the following.

Theorem 5.9 (Approximate quantum counting)**.** *Let $f\colon \{1,\dots,N\} \to \{0,1\}$ be a black-box function with $|f^{-1}(1)| = K$, and let $\epsilon \in (0,1]$. There is a quantum algorithm that produces an estimate $\tilde{K}$ of K satisfying $|K - \tilde{K}| \leq \epsilon K$ with probability at least $2/3$, using $O(\frac{1}{\epsilon}\sqrt{N/K})$ queries of f in expectation, provided $K > 0$. If $K = 0$, the algorithm outputs $\tilde{K} = 0$ with certainty in $O(\sqrt{N})$ queries.*

Note that the failure probability can be reduced to δ by repeating this algorithm $O(\log \frac{1}{\delta})$ times and taking the median of the resulting estimates.

Applying Theorem 5.9 to approximately count the edges of a graph, we have the following.

Corollary 5.3. *For any constant $\epsilon > 0$ and function $f\colon \mathbb{Z}^+ \to \mathbb{Z}^+$ there is a quantum algorithm using $O(\sqrt{n^2/f(n)}\log \frac{1}{\delta})$ queries which accepts graphs with $m \geq (1+\epsilon)f(n)$ and rejects graphs with $m \leq (1-\epsilon)f(n)$ with probability at least $1-\delta$.*

Proof. Approximate quantum counting with accuracy ϵ can distinguish the two cases. However, if $m \ll f(n)$, then quantum counting requires $O(\sqrt{n^2/m})$ queries in expectation, much more than the claimed $O(\sqrt{n^2/f(n)})$ queries. This can be fixed by adding n more vertices and $f(n)$ edges so the total edge count is always greater than $f(n)$. Now we have to distinguish the cases $m \geq (2+\epsilon)f(n)$ and $m \leq (2-\epsilon)f(n)$. This can be done using accuracy $\epsilon/2$ and only $O(\sqrt{n^2/f(n)})$ expected queries. Since we know the query upper bound explicitly, we can halt the algorithm if it exceeds its query upper bound by a factor of 4. By Markov's inequality, this happens with probability at most $1/4$.

This procedure has constant success probability. Repeating this $O(\log \frac{1}{\delta})$ times and taking the majority vote of the outcomes boosts the success probability to at least $1-\delta$. □

We also use a procedure for extracting all marked items in a search problem.

Lemma 5.6. *Let $f\colon \{1,\dots,N\} \to \{0,1\}$ be a black-box function with $|f^{-1}(1)| = K$. The bounded-error quantum query complexity of determining $f^{-1}(1)$ is $O(\sqrt{NK})$ if $K > 0$, and $O(\sqrt{N})$ if $K = 0$.*

This result and its optimality appear to be folklore (see for example [Amb10]); we include a short proof for completeness. It can also be proved using the techniques of [AIN+08] or [DHHM06].

Proof. First check if $K = 0$ by standard Grover search, using $O(\sqrt{N})$ queries; if so, we are done. Otherwise, by Theorem 17 of [BHMT02], we can exactly determine K with bounded error in $O(\sqrt{NK})$ expected queries. To get a worst-case query upper bound, we first estimate K to a constant factor using Theorem 5.9. If we allow the algorithm in Theorem 5.9 to make up to $O(\sqrt{N})$ queries, it produces an estimate of K with high probability. With an estimate of K, we can halt the algorithm if it makes more than (say) 4 times its expected query complexity, thus exactly determining K with bounded error in $O(\sqrt{NK})$ queries in the worst case.

By Theorem 16 of [BHMT02], given K, we can find a marked item with certainty in $O(\sqrt{N/K})$ queries. We repeat this algorithm K times, unmarking each item after we find it, until there are no more marked items. The number of queries used by this procedure is $O(\sum_{i=0}^{K-1} \sqrt{N/(K-i)})$. Observe that

$$\sum_{i=0}^{K-1} \sqrt{\frac{N}{K-i}} \leq \sqrt{N} \int_0^K \frac{\mathrm{d}x}{\sqrt{x}} = 2\sqrt{NK}. \tag{5.1}$$

Thus $O(\sqrt{NK})$ queries suffice to find the K marked items. □

Applying these results, we find that sparse graph properties can be decided in $O(n^{3/2})$ queries.

Theorem 5.10. *If $\mathcal{P}$ is a sparse graph property, then $Q(\mathcal{P}) = O(n^{3/2})$.*

Proof. Since $\mathcal{P}$ is sparse, there is a constant c such that $G \in \mathcal{P}$ implies $m \leq cn$. By Corollary 5.3, we can reject graphs with $m \geq 2cn$ and keep for further consideration those with $m \leq cn$ with bounded error using $O(\sqrt{n})$ queries. (It does not matter whether graphs with $cn < m < 2cn$ are rejected.) Now all unrejected graphs have $m < 2cn$. By applying Lemma 5.6 we can reconstruct all edges of the graph with bounded error using $O\left(\sqrt{\binom{n}{2}m}\right) = O(n^{3/2})$ queries. Given all the edges of the graph, no further queries are needed to decide $\mathcal{P}$. □

Combining this with Theorem 5.2 and Theorem 5.8, an immediate consequence is

Corollary 5.4. *If $\mathcal{P}$ is nontrivial, minor closed, and not FSP, then $Q(\mathcal{P}) = \Theta(n^{3/2})$.*

Note that this provides an alternative proof that the quantum query complexity of planarity is $\Theta(n^{3/2})$ [AIN+08].

For minor-closed graph properties that are also FSP, the lower bounds from Section 5.3 do not rule out the possibility of an improvement over Theorem 5.10. In fact, we show that an improvement is possible for all such properties.

5.4.2 Quantum walk search

Here we introduce our main algorithmic tool, quantum walk search. Building on work of Ambainis [Amb07] and Szegedy [Sze04], Magniez et al. gave the following general quantum walk search algorithm (Theorem 3 of [MNRS11]):

Theorem 5.11 (Quantum walk search)**.** *Let P be a reversible, ergodic Markov chain with spectral gap $\delta > 0$, and let M be a subset of the states of P (the* marked states*) such that in the stationary distribution of P, the probability of choosing a marked state is at least $\epsilon > 0$. Then there is a bounded-error quantum algorithm that determines whether M is empty using $O(S+\frac{1}{\sqrt{\epsilon}}(\frac{1}{\sqrt{\delta}}U+C))$ queries, where S is the number of queries needed to set up a quantum sample from the stationary distribution of P, U is the number of queries needed to update the state after each step of the chain, and C is the number of queries needed to check if a state is marked.*

Despite the generality of this approach, nearly all previous quantum walk search algorithms take P to be a simple random walk on the Johnson graph $J(N,K)$, whose vertices are the $\binom{N}{K}$ subsets of $\{1,\ldots,N\}$ of size K, with an edge between subsets that differ in exactly one item. For our purposes it will be more convenient to consider a random walk on the Hamming graph $H(N,K)$, with vertex set $\{1,\ldots,N\}^K$ and edges between two K-tuples that differ in exactly one coordinate. This choice simplifies the implementation of our setup step. Although the order of the items has no significance, and the possibility of repeated items only slows down the algorithm, the effect is not substantial.

In particular, both Markov chains have spectral gap $\delta = \Omega(1/K)$. It can be shown that the eigenvalues of the simple random walk on the Johnson graph are $1-\frac{i(N+1-i)}{K(N-K)}$ for $i \in \{0,1,\ldots,K\}$, so the spectral gap is $\delta_{J(N,K)} = \frac{N}{K(N-K)} = \Omega(1/K)$. The Hamming graph is even easier to analyze, since it is the Cartesian product of K copies of the complete graph on N vertices. The normalized adjacency matrix of $H(N,1)$ has spectral gap $\delta_{H(N,1)} = 1$, so the normalized adjacency matrix of $H(N,K)$ has spectral gap $\delta_{H(N,K)} = 1/K$.

More generally, we consider a Markov chain on the tensor product of several $H(N,K_i)$ in which we take α_i steps on the i^{th} coordinate. Then the spectral gap is $\delta = 1 - \max_i(1-\frac{1}{K_i})^{\alpha_i} = \Omega(\min_i \alpha_i/K_i)$, where we assume that $1/K_i = o(1)$.

Note that the stationary distribution of a symmetric Markov chain is uniform. Thus the initial state is a uniform superposition, and to calculate ϵ, it suffices to calculate the probability that a uniformly random state is marked.

5.4.3 Detecting subgraphs of sparse graphs

We now describe algorithms for determining whether a sparse graph G contains a given subgraph H. Our basic strategy is to search over subsets of the vertices of G for one containing a *vertex cover* of H, a subset C of the vertices of H such that each edge of H involves at least one vertex from C. By storing the list of neighbors of every vertex in a given subset, we can determine whether they include a vertex cover of H with no further queries. We exploit sparsity by separately considering cases where the vertices of the vertex cover have a given (approximate) degree.

Let $\mathrm{vc}(H)$ denote the smallest number of vertices in any vertex cover of H. A vertex cover of H with $\mathrm{vc}(H)$ vertices is called a *minimal vertex cover*.

Theorem 5.12. *Let $\mathcal{P}$ be the property that a graph either has more than cn edges (for some constant c) or contains a given subgraph H. Then $Q(\mathcal{P}) = \tilde{O}\left(n^{\frac{3}{2}-\frac{1}{\mathrm{vc}(H)+1}}\right)$.*

Proof. First, we use Corollary 5.3 to determine whether the graph is nonsparse. We accept if it has more than cn edges. Otherwise, we continue, knowing it has fewer than, say, $2cn$ edges.

Now let C be a minimal vertex cover of H. We search for a subset of the vertices of G that include the vertices of C (for some copy of H in G). To take advantage of sparsity, we separately consider different ranges for the degrees of these vertices. We say that a vertex of G has degree *near* q if its degree is within a constant factor of q. For concreteness, let us say that the degree of v is near q if it is between $q/2$ and $2q$. We search for a copy of H in G where vertex i of C has degree near q_i. By considering a geometric progression of values for each of the q_is, we cover all the possible vertex degrees with an overhead of only $O(\log^{\mathrm{vc}(H)} n) = \tilde{O}(1)$.

Since Theorem 5.9 only allows us to estimate the degree of a vertex within error ϵ, if the degree estimate is too close to $q/2$ or $2q$ we might incorrectly accept or reject the vertex. To handle this, we use a geometric progression where the intervals overlap enough that every possible degree is sufficiently far from the end of some interval. For concreteness, we choose the progression of values to be $2, 4, 8, \ldots$, so that the relevant intervals are 1 to 4, 2 to 8, 4 to 16, etc.

For each fixed $(q_1, \ldots, q_{\mathrm{vc}(H)})$, we search over k_i-tuples of vertices of G with degree near q_i for each i from 1 to $\mathrm{vc}(H)$. For each such vertex, we store its complete neighbor list. In one step of the Markov chain, we take α_i steps for the i^{th} component. Here the k_is and the α_is are parameters that can be chosen to optimize the performance of the algorithm.

Let t_i be the number of vertices of G with degree near q_i. Note that we can approximate the t_is at negligible cost using Theorem 5.9. Also note that since each vertex is counted at most 3 times

and the number of edges of G is $O(n)$, we have $\sum_i t_i q_i = O(n)$, and in particular, $t_i q_i = O(n)$ for each i. Choose the ordering of the indices so that $t_1 \leq \cdots \leq t_{\mathrm{vc}(H)}$.

The setup cost of the walk has two contributions. Using Grover's algorithm, we can prepare a uniform superposition over all vertices of degree near q_i using $O(n/\sqrt{t_i})$ queries. To prepare a uniform superposition over all k_i-tuples of such vertices, we simply repeat this k_i times, using $O(k_i n/\sqrt{t_i})$ queries. (Note that if our search involved a Johnson graph instead of a Hamming graph, we would need to prepare a uniform superposition over k_i-subsets of vertices instead of k_i-tuples. Although this could be done, it would make the setup step more complicated, and the performance of the algorithm would be essentially unchanged.) Thus we can prepare a uniform superposition over the k_i-tuples for all i using $O(\sum_i k_i n/\sqrt{t_i})$ queries. Next we compute the list of neighbors of each of these vertices using Lemma 5.6, which takes $O(\sum_i k_i \sqrt{n q_i})$ queries. Since $q_i = O(n/t_i)$, the cost of the neighbor computation can be neglected, so the setup cost is $S = O(\sum_i k_i n/\sqrt{t_i})$.

The cost of performing a step of the walk also has two contributions. To update the vertices, we search for α_i vertices of degree near q_i; this takes $O(\sum_i \alpha_i n/\sqrt{t_i})$ queries. Updating their neighbor lists takes $O(\sum_i \alpha_i \sqrt{n q_i})$ queries, which is again negligible. Therefore, the update cost is $U = O(\sum_i \alpha_i n/\sqrt{t_i})$. Since we perform $\mathrm{poly}(n)$ update steps, we reduce the error probability of each update step to $1/\mathrm{poly}(n)$ so that the final error probability is a constant. This only introduces an overhead of $O(\log n)$.

We mark states of the Markov chain that include a vertex cover of a copy of H in G. Since we also store complete neighbor lists, and every vertex in H is adjacent to some vertex of the vertex cover, no queries are required to determine whether a state is marked. In other words, the checking cost is $C = 0$.

It remains to determine the spectral gap of the chain and the fraction of marked vertices. From Section 5.4.2, the spectral gap is $\delta = \Omega(\min_i \alpha_i/k_i)$. If we choose k_i of the t_i vertices of degree near q_i uniformly at random, the probability of obtaining one particular vertex of degree near q_i is $\Omega(k_i/t_i)$; therefore the fraction of marked vertices is $\epsilon = \Omega(\prod_i k_i/t_i)$.

Applying Theorem 5.11, the number of queries used by this algorithm for any fixed values of $q_1, q_2, \ldots, q_{\mathrm{vc}(H)}$ is

$$O\left(n\left[\sum_i \frac{k_i}{\sqrt{t_i}} + \sqrt{\max_i \frac{k_i}{\alpha_i} \prod_i \frac{t_i}{k_i}} \sum_i \frac{\alpha_i}{\sqrt{t_i}}\right]\right). \tag{5.2}$$

Recall that we have the freedom to choose the α_is and the k_is. If $\mathrm{vc}(H) \geq 2$, we choose them to satisfy

$$\frac{\alpha_i}{\alpha_j} = \frac{k_i}{k_j} = \sqrt{\frac{t_i}{t_j}} \tag{5.3}$$

for all i, j (which still leaves the freedom to choose one of the α_is and one of the k_is). Assume for now that the α_is and the k_is are integers. Then the query complexity is

$$O\left(n\left[\frac{k_1}{\sqrt{t_1}} + \sqrt{\frac{k_1}{\alpha_1}\prod_i \frac{t_i}{k_i}\frac{\alpha_1}{\sqrt{t_1}}}\right]\right) = O\left(n\left[\frac{k_1}{\sqrt{t_1}} + \sqrt{\frac{k_1\alpha_1}{t_1}\prod_i \frac{t_i}{k_i}}\right]\right) \tag{5.4}$$

$$= O\left(n\left[\frac{k_1}{\sqrt{t_1}} + \sqrt{\frac{k_1\alpha_1}{t_1}\left(\frac{\sqrt{t_1}}{k_1}\right)^{\mathrm{vc}(H)}\prod_i \sqrt{t_i}}\right]\right) \tag{5.5}$$

$$= O\left(n\left[\frac{k_1}{\sqrt{t_1}} + \sqrt{\alpha_1\left(\frac{\sqrt{t_1 n}}{k_1}\right)^{\mathrm{vc}(H)-1}}\right]\right) \tag{5.6}$$

where we have used the simple bound $t_i \le n$ in the last line. Now take $\alpha_1 = 1$ and

$$k_1 = \sqrt{t_1} n^{\frac{1}{2} - \frac{1}{\mathrm{vc}(H)+1}}; \tag{5.7}$$

then the total query complexity is

$$O\left(n^{\frac{3}{2} - \frac{1}{\mathrm{vc}(H)+1}}\right) \tag{5.8}$$

as claimed. Furthermore, recall that iterating over the various q_is only introduces logarithmic overhead. Since we repeat this subroutine $\mathrm{poly}(\log n)$ times, we reduce the error probability of the subroutine to $1/\mathrm{poly}(\log n)$, which only introduces an extra $O(\log\log n) = \tilde{O}(1)$ factor to the query complexity.

So far we have assumed that the α_is and the k_is are integers. However, observe that the asymptotic expressions for the query complexity are unchanged if we replace each α_i by any value between α_i and $2\alpha_i$ and each k_i by any value between k_i and $2k_i$. Since $\alpha_1 = 1$ is the smallest α_i and $k_1 = \omega(1)$ is the smallest k_i, and because for any $x \ge 1$ there is always an integer between x and $2x$, the result holds when the α_is and k_is are rounded up to the next largest integers. Similarly, the fact that we only have a multiplicative approximation for the t_is does not affect the asymptotic running time. □

We can apply this algorithm to decide sparse graph properties, and in particular minor-closed properties, that are also FSP: we simply search for each of the forbidden subgraphs, accepting if none of them are present. For minor-closed properties, the nonsparseness condition of Theorem 5.12 can be removed due to Theorem 5.2. Thus, since $\mathrm{vc}(H)$ is a constant for any fixed graph H, we have

Corollary 5.5. *If $\mathcal{P}$ is sparse and FSP, then $Q(\mathcal{P}) = O(n^\alpha)$ for some $\alpha < 3/2$.*

Note that Theorem 5.12 also holds if we ask whether H is contained as an induced subgraph, since when we check whether H is present for a certain subset of edges, we have access to their complete neighbor lists.

For many subgraphs, we can improve Theorem 5.12 further by storing additional information about the vertices in the minimal vertex cover: in addition to storing their neighborhoods, we can also store basic information about their second neighbors. In particular, we have the following.

Theorem 5.13. *Let $\mathcal{P}$ be the property that a graph either has more than cn edges (for some constant c) or contains a given subgraph H. Let H' be the graph obtained by deleting all degree-one vertices of H that are not part of an isolated edge. Then $Q(\mathcal{P}) = \tilde{O}\left(n^{\frac{3}{2} - \frac{1}{\text{vc}(H')+1}}\right)$.*

Proof. As before, we begin by using Corollary 5.3 to determine whether the graph is nonsparse, accepting if this is the case.

Otherwise, we use the technique of color coding [AYZ95] to handle the degree-one vertices of H. Suppose that H has ℓ degree-one vertices, and label them $1, \ldots, \ell$; label the other vertices $\ell + 1$. Assign labels from the set $\{1, \ldots, \ell, \ell + 1\}$ uniformly at random to the vertices of G. If there is a copy of H in G, then with probability at least $(\ell + 1)^{-|V(H)|} = \Omega(1)$, the vertices of this copy of H in G have the correct labels. We assume this is the case, increasing the cost of the algorithm by a factor of $O(1)$.

We augment the algorithm of Theorem 5.12 by storing additional information about each vertex: in addition to the neighborhood, we also store whether each vertex has a second neighbor with each possible label. Computing this information for any one vertex of G of degree near q_i with known neighborhood takes $O(\sqrt{nq_i})$ queries, so the additional setup cost of $O(\sum_i k_i \sqrt{nq_i})$ and update cost of $O(\sum_i \alpha_i \sqrt{nq_i})$ to store this information is negligible. Now we can recognize H by storing only a minimal vertex cover of H', still with zero checking cost. Thus the same analysis applies with $\text{vc}(H)$ replaced by $\text{vc}(H')$, and the result follows. □

In the published version of this work, we use Theorem 5.13 to obtain improved algorithms for properties characterized by a single forbidden minor (and equivalently, a single forbidden subgraph). However, since the query complexity of such properties is now known to be $\Theta(n)$ by the work of [BR12], we do not study algorithms for such properties in this thesis.

5.4.4 Relaxing sparsity

In the previous section, we focused on the case of sparse graphs, since this is the relevant case for minor-closed graph properties. However, our algorithms easily generalize to the case where the number of edges is at most some prescribed upper bound, which need not be linear in n, leading to further applications.

Theorem 5.14. *Let $\mathcal{P}$ be the property that an n-vertex graph either has more than $\bar{m}$ edges, where $\bar{m} = \Omega(n)$, or contains a given subgraph H. Let H' be the graph obtained by deleting all degree-one vertices of H that are not part of an isolated edge. Then $Q(\mathcal{P}) = \tilde{O}(\sqrt{\bar{m}}n^{1-\frac{1}{\mathrm{vc}(H')+1}})$.*

Note that Theorem 5.14 subsumes Theorem 5.13, which in turn subsumes Theorem 5.12.

Proof. We apply the algorithm from the proof of Theorem 5.13 (which in turn depends on the analysis in the proof of Theorem 5.12), replacing the promise that $m = O(n)$ with the promise that $m = O(\bar{m})$, which implies that $q_i = O(\bar{m}/t_i)$. In this algorithm, the setup cost is

$$S = O\left(\sum_i k_i n/\sqrt{t_i} + \sum_i k_i\sqrt{nq_i}\right) \tag{5.9}$$

$$= O\left(\sum_i \frac{k_i}{\sqrt{t_i}}(n + \sqrt{n\bar{m}})\right) \tag{5.10}$$

$$= O\left(\sqrt{n\bar{m}}\sum_i \frac{k_i}{\sqrt{t_i}}\right), \tag{5.11}$$

and by a similar calculation, the update cost is

$$U = O\left(\sqrt{n\bar{m}}\sum_i \frac{\alpha_i}{\sqrt{t_i}}\right); \tag{5.12}$$

the checking cost remains $C = 0$, and the spectral gap and fraction of marked vertices are also unchanged. Again choosing

$$\frac{\alpha_i}{\alpha_j} = \frac{k_i}{k_j} = \sqrt{\frac{t_i}{t_j}}, \tag{5.13}$$

the query complexity from Theorem 5.11 is

$$O\left(\sqrt{n\bar{m}}\left[\frac{k_1}{\sqrt{t_1}} + \sqrt{\frac{k_1}{\alpha_1}\prod_i \frac{t_i}{k_i}\frac{\alpha_1}{\sqrt{t_1}}}\right]\right) = O\left(\sqrt{n\bar{m}}\left[\frac{k_1}{\sqrt{t_1}} + \sqrt{\alpha_1\left(\frac{\sqrt{t_1}n}{k_1}\right)^{\mathrm{vc}(H)-1}}\right]\right). \tag{5.14}$$

(cf. (5.4–5.6)). Taking $\alpha_1 = 1$ and

$$k_1 = \sqrt{t_1}n^{\frac{1}{2}-\frac{1}{\mathrm{vc}(H')+1}} \tag{5.15}$$

gives a query complexity of

$$O(\sqrt{\bar{m}}n^{1-\frac{1}{\mathrm{vc}(H')+1}}) \tag{5.16}$$

and the result follows. □

In conjunction with the Kövári–Sós–Turán theorem [KST54], this algorithm has applications to subgraph-finding problems that are not equivalent to minor-finding problems.

Theorem 5.15 (Kövári–Sós–Turán)**.** *If an n-vertex graph G does not contain $K_{s,t}$ as a subgraph, where $1 \leq s \leq t$, then $|E(G)| \leq c_{s,t}\, n^{2-\frac{1}{s}}$, where $c_{s,t}$ is a constant depending only on s and t.*

In particular, we use this theorem to show the following.

Theorem 5.16. *If H is a d-vertex bipartite graph, then H-subgraph containment has quantum query complexity $\tilde{O}(n^{2-\frac{1}{d}-\frac{2}{d+2}}) = \tilde{O}(n^{2-\frac{3d+2}{d(d+2)}})$.*

Proof. By Theorem 5.15, a graph that does not contain $K_{s,t}$ (where $1 \leq s \leq t$) has at most $c_{s,t}\, n^{2-\frac{1}{s}}$ edges, so a graph with more than $c_{s,t}\, n^{2-\frac{1}{s}}$ edges must contain H. Theorem 5.9 shows that we can determine whether the input graph has more than $2c_{s,t}\, n^{2-\frac{1}{s}}$ edges using $o(n)$ queries. If so, we accept; otherwise, we apply Theorem 5.14 with $\bar{m} = 2c_{s,t}\, n^{2-\frac{1}{s}}$ and $\mathrm{vc}(H') \leq d/2$, giving the desired result. □

Recall that for $d > 3$, Theorem 4.6 of [MSS07] gives an upper bound of $O(n^{2-\frac{2}{d}})$ for finding a d-vertex subgraph. For bipartite subgraphs, Theorem 5.16 is a strict improvement.

Note that a better bound may be possible by taking the structure of H into account. In general, if H is a bipartite graph with the i^{th} connected component having vertex bipartition $V_i \cup U_i$ with $1 \leq |V_i| \leq |U_i|$, then we can replace $d/2$ by $\sum_i |V_i|$, since a graph that does not contain $K_{\sum_i |V_i|, \sum_i |U_i|}$ does not contain H, and $\mathrm{vc}(H') \leq \mathrm{vc}(H) = \sum_i |V_i|$. As a simple example, if $H = K_{1,t}$ is a star on $t+1$ vertices, then H-subgraph containment can be solved with $\tilde{O}(n)$ quantum queries (which is essentially optimal due to Theorem 5.5). This shows that the quantum query complexity of deciding if a graph is t-regular is $\tilde{\Theta}(n)$. (In fact it is not hard to show that the query complexity of this problem is $\Theta(n)$.)

As another example, consider the property of containing a fixed even-length cycle, i.e., C_{2l}-subgraph containment. Since C_{2l} is bipartite, this is a special case of the problem considered above. Theorem 5.16 gives an upper bound of $\tilde{O}(n^{2-\frac{3l+1}{2l(l+1)}})$, which approaches $O(n^2)$ as the cycle gets longer (i.e., as $l \to \infty$). As concrete examples, it gives upper bounds of $\tilde{O}(n^{1.41\overline{6}})$ for C_4 containment and $\tilde{O}(n^{1.58\overline{3}})$ for C_6 containment.

For even cycles of length 6 or greater, this upper bound can be significantly improved by replacing Theorem 5.15 with the following result of Bondy and Simonovits [BS74].

Theorem 5.17 (Bondy–Simonovits)**.** *Let G be a graph on n vertices. For any $l \geq 1$, if $|E(G)| > 100ln^{1+1/l}$ then G contains C_{2l} as a subgraph.*

Using this upper bound instead of Theorem 5.15 in Theorem 5.16 gives us the following upper bound for even cycles.

Theorem 5.18. *The C_{2l}-subgraph containment problem can be solved using $\tilde{O}(n^{\frac{3}{2}-\frac{l-1}{2l(l+1)}})$ queries.*

For C_4 containment, the upper bound given by this theorem matches the one given by Theorem 5.16. However, for all longer even cycles, the bound given by this theorem is strictly better than the one given by Theorem 5.16. For example, we get an upper bound of $\tilde{O}(n^{1.41\overline{6}})$ for C_6 containment, as compared to the upper bound of $\tilde{O}(n^{1.58\overline{3}})$ given by Theorem 5.16. Moreover, as the cycles get longer, the upper bound of Theorem 5.18 approaches $O(n^{3/2})$ instead of $O(n^2)$.

We can sometimes improve over Theorem 5.14 by introducing a nontrivial checking cost. The following is a simple example of such an algorithm for C_4 containment that performs better than the $\tilde{O}(n^{1.41\overline{6}})$ query algorithm given by both Theorem 5.16 and Theorem 5.18.

Theorem 5.19. *C_4-subgraph containment can be solved in $\tilde{O}(n^{1.25})$ quantum queries.*

Proof. By Theorem 5.15, a graph with $\Omega(n^{3/2})$ edges must contain $C_4 = K_{2,2}$, so by applying Theorem 5.9 and accepting graphs with $\Omega(n^{3/2})$ edges, we can assume that $m = O(n^{3/2})$. Then, for various values of q in a geometric progression, we search for a vertex v of the input graph that has degree near q and that belongs to a 4-cycle. It suffices to search for v using Grover's algorithm instead of the more general Theorem 5.11. Let t denote the number of vertices of the input graph with degree near q, and let $|\psi\rangle$ denote a uniform superposition over all such vertices, where for each vertex we store a list of its $O(q)$ neighbors. Grover's algorithm starts from the state $|\psi\rangle$ and alternates between reflecting about $|\psi\rangle$ and about vertices that belong to a 4-cycle, detecting whether such a vertex exists in $O(\sqrt{t})$ iterations. By Grover's algorithm, we can prepare a uniform superposition over the vertices of degree near q using $O(n/\sqrt{t})$ queries; by Lemma 5.6, we can compute the neighborhood of a vertex with degree near q in $O(\sqrt{nq})$ queries. Thus we can prepare or reflect about $|\psi\rangle$ in $O(n/\sqrt{t} + \sqrt{nq})$ queries. Given the neighbors of a vertex with degree near q, we can decide whether that vertex is part of a 4-cycle by searching over all vertices in the graph for a vertex that is adjacent to 2 of its neighbors. This can be done in $O(\sqrt{nq})$ queries. Thus the query complexity of searching for a 4-cycle is

$$O\left(\sqrt{t}\left[\frac{n}{\sqrt{t}} + \sqrt{nq}\right]\right) = O(n + \sqrt{nqt}). \tag{5.17}$$

Since $qt = O(n^{3/2})$, we see that the number of amplitude amplification steps is $O(n^{5/4})$. As in previous algorithms, iterating over values of q and error reduction of subroutines only introduces logarithmic overhead, so the total query complexity is $\tilde{O}(n^{5/4})$ as claimed. □

5.5 Open problems

Since the best known lower bound for forbidden subgraph properties is the simple $\Omega(n)$ bound from Theorem 5.5, an obvious open question is to find improved lower bounds for such properties. While the original (positive-weights) quantum adversary method cannot prove a better lower bound due to the certificate complexity barrier [Sze03, Zha05, LM08, ŠS06], it might be possible to apply the negative-weights adversary method [HLŠ07] or the polynomial method [BBC+01]. Note that sparsity makes forbidden subgraph properties potentially more difficult to lower bound; this is precisely the feature we took advantage of in the algorithms of Section 5.4. Proving a superlinear lower bound for any subgraph-finding problem—even one for which dense graphs might not contain the subgraph, such as in the case of triangles—remains a major challenge.

Open Problem 5.1. Can we prove an $\omega(n)$ lower bound on the quantum query complexity of any H-subgraph containment property?

After our work and the results of Belovs and Reichardt [BR12], the main open problem about minor-closed properties is whether we can show a dichotomy result for their query complexity. We have already shown that minor-closed properties that are not FSP require $\Theta(n^{3/2})$ queries. Is it the case that minor-closed graph properties that are FSP require $\Theta(n)$ queries?

Open Problem 5.2. Is the following conjecture true? The query complexity of a minor-closed property is either $\Theta(n)$ or $\Theta(n^{3/2})$, depending on whether or not the property is FSP.

Note that the conjecture is true for minor-closed properties that are characterized by exactly one forbidden minor [BR12]. Showing the general conjecture would require proving an $O(n)$ query upper bound for all minor-closed properties that are FSP. Note that while our algorithms only take advantage of the sparsity of such properties, minor-closed families of graphs have other special properties, such as bounded degeneracy, which might also be exploited.

Lastly, the quantum analogue of the Aanderaa–Karp–Rosenberg conjecture remains open. Using the best known randomized lower bound of $\Omega(n^{4/3}\log^{1/3} n)$ [CK07], it can be shown that any nontrivial monotone graph property has quantum query complexity $\Omega(n^{2/3}\log^{1/6} n)$ [MSS07].

Open Problem 5.3. Does a quantum analogue of the Aanderaa–Karp–Rosenberg conjecture hold? Is it the case that the quantum query complexity of deciding any nontrivial monotone graph property is $\Omega(n)$? The best known lower bound is $\Omega(n^{2/3}\log^{1/6} n)$ [MSS07].

The best lower bound we could hope for is $\Omega(n)$, since there exist nontrivial monotone graph properties with query complexity $O(n)$.

Chapter 6

Conclusion

In this thesis we studied the quantum query complexity of a variety of problems using several different techniques. We briefly summarize the problems solved and key technical contributions.

In Chapter 2 we introduced the linear combinations of unitaries (LCU) algorithm that provides nearly optimal algorithms for the sparse Hamiltonian simulation problem and the problem of simulating the continuous- and fractional-query models in the standard discrete-query model. The LCU algorithm is based on a new technique introduced called "oblivious amplitude amplification." In Chapter 3 we presented the first optimal quantum algorithm for the oracle identification problem. The solution is based on ideas from classical learning theory and a new composition theorem for quantum query complexity that allows the removal of log factors that arise from composition in certain situations. The composition theorem is also used to remove log factors from the algorithm presented in the next chapter. In Chapter 4 we surveyed known results on matrix multiplication and provided an almost optimal algorithm for output-sensitive Boolean matrix multiplication. The algorithm is based on a reduction to the graph collision problem and a new algorithm for graph collision that performs better than known algorithms when the underlying graph is dense. Lastly in Chapter 5 we characterized the quantum query complexity of all minor-closed graph properties that did not have a forbidden subgraph characterization and showed that the remaining minor-closed properties are strictly easier to solve. The lower bound is based on a detailed analysis of the structure of such properties with respect to forbidden subgraphs and topological minors, while the upper bound is based on a novel application of the quantum walk framework that exploits the fact that the underlying graphs are sparse.

We have already seen many open questions related to the problems studied in this thesis, ranging from well-known and widely studied problems such as algorithms for Hamiltonian simulation (Open Problem 2.2) and the query complexity of monotone graph properties (Open Problem 5.3) to lesser-known problems like the query complexity of LC_2^0 circuits (Open Problem 4.4) and Boolean matrix multiplication with dense output matrices (Open Problem 4.6). More generally,

quantum query complexity is rich in open problems and continues to remain, in my opinion, an exciting area of research.

References

[Aar06] Scott Aaronson. Lower bounds for local search by quantum arguments. *SIAM Journal on Computing*, 35(4):804–824, 2006. [p. 96]

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009. [pp. 3, 6]

[ABI+13] Andris Ambainis, Kaspars Balodis, Jānis Iraids, Raitis Ozols, and Juris Smotrovs. Parameterized quantum query complexity of graph collision. In *Proceedings of the Workshop on Quantum and Classical Complexity*, pages 5–16, 2013. [p. 74]

[ABSdW13] Andris Ambainis, Arturs Backurs, Juris Smotrovs, and Ronald de Wolf. Optimal quantum query bounds for almost all Boolean functions. In *Proceedings of the 30th Symposium on Theoretical Aspects of Computer Science (STACS 2013)*, volume 20 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 446–453, 2013. [p. 41]

[ACR+10] Andris Ambainis, Andrew M. Childs, Ben W. Reichardt, Robert Špalek, and Shengyu Zhang. Any AND-OR formula of size N can be evaluated in time $N^{1/2+o(1)}$ on a quantum computer. *SIAM Journal on Computing*, 39(6):2513–2530, 2010. [p. 14]

[AIK+04] Andris Ambainis, Kazuo Iwama, Akinori Kawachi, Hiroyuki Masuda, Raymond H. Putra, and Shigeru Yamashita. Quantum Identification of Boolean Oracles. In *Proceedings of the 21st International Symposium on Theoretical Aspects of Computer Science (STACS 2004)*, volume 2996 of *Lecture Notes in Computer Science*, pages 105–116. Springer, 2004. [pp. 41, 42, 43]

[AIK+07] Andris Ambainis, Kazuo Iwama, Akinori Kawachi, Rudy Raymond, and Shigeru Yamashita. Improved algorithms for quantum identification of Boolean oracles. *Theoretical Computer Science*, 378(1):41 – 53, 2007. [pp. 7, 40, 41, 42, 43]

[AIN+08] Andris Ambainis, Kazuo Iwama, Masaki Nakanishi, Harumichi Nishimura, Rudy Raymond, Seiichiro Tani, and Shigeru Yamashita. Quantum query complexity of Boolean functions with small on-sets. In *Proceedings of the 19th International Symposium on Algorithms and Computation*, pages 907–918, 2008. [pp. 90, 107, 108]

[AIN+09] Andris Ambainis, Kazuo Iwama, Masaki Nakanishi, Harumichi Nishimura, Rudy Raymond, Seiichiro Tani, and Shigeru Yamashita. Average/worst-case gap of quantum query complexities by on-set size. *arXiv preprint* arXiv:0908.2468, 2009. [p. 43]

[Amb02] Andris Ambainis. Quantum lower bounds by quantum arguments. *Journal of Computer and System Sciences*, 64(4):750–767, 2002. [pp. 45, 71, 73, 91, 96]

[Amb07] Andris Ambainis. Quantum walk algorithm for element distinctness. *SIAM Journal on Computing*, 37(1):210–239, 2007. [pp. 66, 68, 74, 108]

[Amb10] Andris Ambainis. A new quantum lower bound method, with an application to strong direct product theorem for quantum search. *Theory of Computing*, 6(1):1–25, 2010. [p. 107]

[AMRR11] Andris Ambainis, Loïck Magnin, Martin Roetteler, and Jérémie Roland. Symmetry-assisted adversaries for quantum state generation. In *Proceedings of the 26th IEEE Conference on Computational Complexity (CCC 2011)*, pages 167–177, 2011. [p. 13]

[Ang88] Dana Angluin. Queries and Concept Learning. *Machine Learning*, 2:319–342, 1988. [p. 44]

[AS05] Alp Atıcı and Rocco Servedio. Improved Bounds on Quantum Learning Algorithms. *Quantum Information Processing*, 4:355–386, 2005. [pp. 41, 44, 59]

[AT03] Dorit Aharonov and Amnon Ta-Shma. Adiabatic quantum state generation and statistical zero knowledge. In *Proceedings of the 35th ACM Symposium on Theory of Computing (STOC 2003)*, pages 20–29, 2003. [pp. 11, 12, 30]

[AYZ95] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995. [pp. 3, 112]

[BACS07] Dominic W. Berry, Graeme Ahokas, Richard Cleve, and Barry C. Sanders. Efficient quantum algorithms for simulating sparse Hamiltonians. *Communications in Mathematical Physics*, 270(2):359–371, 2007. [pp. 3, 11, 12, 30, 31, 33, 39]

[BBBV97] Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing. *SIAM Journal on Computing (special issue on quantum computing)*, 26:1510–1523, 1997. [pp. 4, 41]

[BBC+01] Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald de Wolf. Quantum lower bounds by polynomials. *Journal of the ACM*, 48(4):778–797, 2001. [pp. 16, 33, 34, 45, 71, 90, 116]

[BBHT98] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight Bounds on Quantum Searching. *Fortschritte der Physik*, 46(4-5):493–505, 1998. [p. 47]

[BC12] Dominic W. Berry and Andrew M. Childs. Black-box Hamiltonian simulation and unitary implementation. *Quantum Information and Computation*, 12(1–2):29–62, 2012. [pp. 12, 13, 38, 39]

[BCC+14] Dominic W. Berry, Andrew M. Childs, Richard Cleve, Robin Kothari, and Rolando D. Somma. Exponential improvement in precision for simulating sparse Hamiltonians. In *Proceedings of the 46th ACM Symposium on Theory of Computing (STOC 2014)*, pages 283–292, 2014. [pp. 7, 10, 11, 15, 20, 38]

[BCG14] Dominic W. Berry, Richard Cleve, and Sevag Gharibian. Gate-efficient discrete simulations of continuous-time quantum query algorithms. *Quantum Information and Computation*, 14(1–2):1–30, 2014. [pp. 18, 19]

[BDF+04] Aija Berzina, Andrej Dubrovsky, Rusins Freivalds, Lelde Lace, and Oksana Scegulnaja. Quantum query complexity for some graph problems. In *SOFSEM 2004: Theory and Practice of Computer Science*, volume 2932, pages 1–11, 2004. [p. 90]

[BDH+05] Harry Buhrman, Christoph Dürr, Mark Heiligman, Peter Høyer, Frédéric Magniez, Miklos Santha, and Ronald de Wolf. Quantum algorithms for element distinctness. *SIAM Journal on Computing*, 34(6):1324–1330, 2005. [p. 73]

[BdW02] Harry Buhrman and Ronald de Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288(1):21 – 43, 2002. [pp. 2, 6]

[Bel12] Aleksandrs Belovs. Span programs for functions with constant-sized 1-certificates. In *Proceedings of the 44th ACM Symposium on Theory of Computing (STOC 2012)*, pages 77–84, 2012. [p. 73]

[BHMT02] Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. In *Quantum computation and information*, volume 305 of *Contemporary Mathematics*, pages 53–74. AMS, 2002. [pp. 106, 107]

[BR12] Aleksandrs Belovs and Ben W. Reichardt. Span programs and quantum algorithms for st-connectivity and claw detection. In *Algorithms — ESA 2012*, volume 7501 of *Lecture Notes in Computer Science*, pages 193–204. Springer, 2012. [pp. 93, 112, 116]

[BS74] J. Adrian Bondy and Miklós Simonovits. Cycles of even length in graphs. *Journal of Combinatorial Theory, Series B*, 16(2):97–105, 1974. [p. 114]

[BŠ06] Harry Buhrman and Robert Špalek. Quantum verification of matrix products. In *Proceedings of the 17th ACM-SIAM Symposium On Discrete Algorithms (SODA 2006)*, pages 880–889, 2006. [pp. 66, 67, 70, 78]

[BV97] Ethan Bernstein and Umesh Vazirani. Quantum Complexity Theory. *SIAM Journal on Computing*, 26(5):1411–1473, 1997. [pp. 41, 42, 61]

[BZ13] Dan Boneh and Mark Zhandry. Quantum-Secure Message Authentication Codes. In *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 592–608. Springer, 2013. [p. 42]

[CCD+03] Andrew M. Childs, Richard Cleve, Enrico Deotto, Edward Farhi, Sam Gutmann, and Daniel A. Spielman. Exponential algorithmic speedup by quantum walk. In *Proceedings of the 35th ACM Symposium on Theory of Computing (STOC 2003)*, pages 59–68, 2003. [pp. 11, 12]

[CCJY09] Andrew M. Childs, Richard Cleve, Stephen P. Jordan, and David Yonge-Mallo. Discrete-query quantum algorithm for NAND trees. *Theory of Computing*, 5(5):119–123, 2009. [pp. 11, 14]

[CDEL04] Matthias Christandl, Nilanjana Datta, Artur Ekert, and Andrew J. Landahl. Perfect state transfer in quantum spin networks. *Physical Review Letters*, 92(18):187902, 2004. [p. 33]

[CGM+09] Richard Cleve, Daniel Gottesman, Michele Mosca, Rolando D. Somma, and David Yonge-Mallo. Efficient discrete-time simulations of continuous-time quantum query algorithms. In *Proceedings of the 41st ACM Symposium on Theory of Computing (STOC 2009)*, pages 409–416, 2009. [pp. 15, 18, 19, 34]

[Chi04] Andrew M. Childs. *Quantum information processing in continuous time*. PhD thesis, Massachusetts Institute of Technology, 2004. [pp. 11, 12]

[Chi10] Andrew M. Childs. On the relationship between continuous- and discrete-time quantum walk. *Communications in Mathematical Physics*, 294(2):581–603, 2010. [pp. 12, 14, 38]

[CK07] Amit Chakrabarti and Subhash Khot. Improved lower bounds on the randomized complexity of graph properties. *Random Structures and Algorithms*, 30(3):427–440, 2007. [pp. 90, 116]

[CK11a] Andrew M. Childs and Robin Kothari. Quantum query complexity of minor-closed graph properties. In *Proceedings of the 28th Symposium on Theoretical Aspects of Computer Science (STACS 2011)*, volume 9 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 661–672, 2011. [p. 89]

[CK11b] Andrew M. Childs and Robin Kothari. Simulating sparse Hamiltonians with star decompositions. In *Theory of Quantum Computation, Communication, and Cryptography (TQC 2010)*, volume 6519 of *Lecture Notes in Computer Science*, pages 94–103. Springer, 2011. [pp. 3, 12, 31]

[CK12] Andrew M. Childs and Robin Kothari. Quantum query complexity of minor-closed graph properties. *SIAM Journal on Computing*, 41(6):1426–1450, 2012. [pp. 8, 89]

[CKK12] Andrew M. Childs, Shelby Kimmel, and Robin Kothari. The quantum query complexity of read-many formulas. In *Algorithms — ESA 2012*, volume 7501 of *Lecture Notes in Computer Science*, pages 337–348. Springer, 2012. [pp. 74, 75, 78]

[CKOR13] Andrew M. Childs, Robin Kothari, Maris Ozols, and Martin Roetteler. Easy and Hard Functions for the Boolean Hidden Shift Problem. In *8th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2013)*, volume 22 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 50–79, 2013. [pp. 41, 43]

[CV86] Richard Cole and Uzi Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70(1):32 – 53, 1986. [p. 3]

[DHHM06] Christoph Dürr, Mark Heiligman, Peter Høyer, and Mehdi Mhalla. Quantum query complexity of some graph problems. *SIAM Journal on Computing*, 35(6):1310–1328, 2006. [pp. 47, 82, 84, 90, 92, 98, 107]

[Die05] Reinhard Diestel. *Graph theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, Berlin, third edition, 2005. [pp. 6, 105]

[Fey82] Richard P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6–7):467–488, 1982. [p. 11]

[FG98] Edward Farhi and Sam Gutmann. Analog analogue of a digital quantum computation. *Physical Review A*, 57(4):2403–2406, 1998. [p. 14]

[FGG08] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum algorithm for the Hamiltonian NAND tree. *Theory of Computing*, 4(8):169–190, 2008. [p. 14]

[FGGS98] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. Limit on the speed of quantum computation in determining parity. *Physical Review Letters*, 81(24):5442–5444, 1998. [pp. 16, 33, 34, 45]

[Fre79] Rūsiņš Freivalds. Fast probabilistic algorithms. In *Mathematical Foundations of Computer Science*, volume 74 of *Lecture Notes in Computer Science*, pages 57–69. Springer, 1979. [pp. 63, 66]

[FRPU94] Uriel Feige, Prabhakar Raghavan, David Peleg, and Eli Upfal. Computing with noisy information. *SIAM Journal on Computing*, 23(5):1001–1018, 1994. [p. 4]

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th ACM Symposium on Theory of Computing (STOC 1996)*, pages 212–219, 1996. [pp. 4, 41]

[Heg95] Tibor Hegedűs. Generalized teaching dimensions and the query complexity of learning. In *Proceedings of the 8th Conference on Computational Learning Theory (COLT 1995)*, pages 108–117, 1995. [pp. 44, 48, 49]

[HHL09] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical Review Letters*, 103(15):150502, 2009. [p. 11]

[HLŠ07] Peter Høyer, Troy Lee, and Robert Špalek. Negative weights make adversaries stronger. In *Proceedings of the 39th ACM Symposium on Theory of Computing (STOC 2007)*, pages 526–535, 2007. [p. 116]

[HMP+10] Markus Hunziker, David A. Meyer, Jihun Park, James Pommersheim, and Mitch Rothstein. The geometry of quantum learning. *Quantum Information Processing*, 9(3):321–341, 2010. [pp. 7, 40, 41, 45, 60]

[HR90] Jacky Huyghebaert and Hans De Raedt. Product formula methods for time-dependent Schrödinger problems. *Journal of Physics A*, 23(24):5777, 1990. [p. 35]

[JKM12] Stacey Jeffery, Robin Kothari, and Frédéric Magniez. Improving Quantum Query Complexity of Boolean Matrix Multiplication Using Graph Collision. In *Automata, Languages, and Programming*, volume 7391 of *Lecture Notes in Computer Science*, pages 522–532. Springer, 2012. [pp. 8, 62]

[JKM13] Stacey Jeffery, Robin Kothari, and Frédéric Magniez. Nested quantum walks with quantum data structures. In *Proceedings of the 24th ACM-SIAM Symposium on Discrete Algorithms (SODA 2013)*, pages 1474–1485, 2013. [p. 73]

[JKS10] Rahul Jain, Hartmut Klauck, and Miklos Santha. Optimal direct sum results for deterministic and randomized decision tree complexity. *Information Processing Letters*, 110(20):893 – 897, 2010. [p. 4]

[Jor75] Camille Jordan. Essai sur la géométrie à n dimensions. *Bulletin de la Société Mathématique de France*, 3:103–174, 1875. [p. 22]

[Juk12] Stasys Jukna. *Boolean Function Complexity: Advances and Frontiers*. Algorithms and Combinatorics. Springer, 2012. [pp. 3, 6]

[KLM06] P. Kaye, R. Laflamme, and M. Mosca. *An Introduction to Quantum Computing*. Oxford University Press, 2006. [p. 6]

[Kot14] Robin Kothari. An optimal quantum algorithm for the oracle identification problem. In *Proceedings of the 31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014)*, volume 25 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 482–493, 2014. [pp. 7, 40]

[KSS83] Jeff Kahn, Michael Saks, and Dean Sturtevant. A topological approach to evasiveness. In *Proceedings of the 24th Symposium on Foundations of Computer Science (SFCS 1983)*, pages 31–33, 1983. [p. 90]

[KST54] Thomason Kövari, Vera T. Sós, and Pál Turán. On a problem of K. Zarankiewicz. *Colloquium Mathematicum*, 3:50–57, 1954. [p. 114]

[Le 12] François Le Gall. Improved output-sensitive quantum algorithms for Boolean matrix multiplication. In *Proceedings of the 23rd ACM-SIAM Symposium On Discrete Algorithms (SODA 2012)*, pages 1464–1476, 2012. [pp. 67, 68]

[Le 14] François Le Gall. Improved quantum algorithm for triangle finding via combinatorial arguments. *arXiv preprint* arXiv:1407.0085, 2014. [pp. 73, 90, 93]

[Lit88] Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4):285–318, 1988. [p. 46]

[Llo96] Seth Lloyd. Universal quantum simulators. *Science*, 273(5278):1073–1078, 1996. [pp. 11, 12]

[LM08] Sophie Laplante and Frédéric Magniez. Lower bounds for randomized and quantum query complexity using Kolmogorov arguments. *SIAM Journal on Computing*, 38(1):46–62, 2008. [pp. 72, 73, 116]

[LMR+11] Troy Lee, Rajat Mittal, Ben W. Reichardt, Robert Špalek, and Mario Szegedy. Quantum query complexity of state conversion. In *Proceedings of the 52nd IEEE Symposium on Foundations of Computer Science (FOCS 2011)*, pages 344–353, 2011. [pp. 5, 13, 15, 44, 53, 54, 86]

[LMS11] Troy Lee, Frédéric Magniez, and Miklos Santha. A learning graph based quantum query algorithm for finding constant-size subgraphs. *arXiv preprint* arXiv:1109.5135, 2011. [p. 91]

[LMS13] Troy Lee, Frédéric Magniez, and Miklos Santha. Improved quantum query algorithms for triangle finding and associativity testing. In *Proceedings of the 24th ACM-SIAM Symposium On Discrete Algorithms (SODA 2013)*, pages 1486–1502, 2013. [pp. 73, 91]

[LY02] Laszlo Lovasz and Neal E. Young. Lecture notes on evasiveness of graph properties. *arXiv preprint* arXiv:cs/0205031, 2002. [p. 90]

[Mad67] Wolfgang Mader. Homomorphieeigenschaften und mittlere Kantendichte von Graphen. *Mathematische Annalen*, 174:265–268, 1967. [p. 95]

[Man89] Udi Manber. *Introduction to Algorithms: A Creative Approach*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989. [p. 6]

[MNRS11] Frédéric Magniez, Ashwin Nayak, Jérémie Roland, and Miklos Santha. Search via quantum walk. *SIAM Journal on Computing*, 40(1):142–164, 2011. [pp. 66, 73, 92, 108]

[Moc07] Carlos Mochon. Hamiltonian oracles. *Physical Review A*, 75(4):042313, 2007. [pp. 14, 35]

[Mon13] Ashley Montanaro. A composition theorem for decision tree complexity. *arXiv preprint* arXiv:1302.4207, 2013. [p. 5]

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995. [pp. 6, 37]

[MSS07] Frédéric Magniez, Miklos Santha, and Mario Szegedy. Quantum algorithms for the triangle problem. *SIAM Journal on Computing*, 37(2):413–424, 2007. [pp. 67, 73, 91, 93, 114, 116]

[MW05] Chris Marriott and John Watrous. Quantum Arthur–Merlin games. *Computational Complexity*, 14(2):122–152, 2005. [p. 20]

[NC00] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge Series on Information and the Natural Sciences. Cambridge University Press, 2000. [pp. 6, 9]

[NWZ09] Daniel Nagaj, Pawel Wocjan, and Yong Zhang. Fast amplification of QMA. *Quantum Information and Computation*, 9(11-12):1053–1068, 2009. [p. 20]

[Pól04] George Pólya. *How to Solve It: A New Aspect of Mathematical Method*. Princeton Science Library. Princeton University Press, 2004. [p. 1]

[PS13] Adam Paetznick and Krysta M. Svore. Repeat-until-success: Non-deterministic decomposition of single-qubit unitaries. *arXiv preprint* arXiv:1311.1074, 2013. [pp. 18, 20]

[Ros73] Arnold L. Rosenberg. On the time required to recognize properties of graphs: a problem. *SIGACT News*, 5(4):15–16, 1973. [p. 90]

[RS90] Neil Robertson and Paul D. Seymour. Graph minors. VIII. A Kuratowski theorem for general surfaces. *Journal of Combinatorial Theory, Series B*, 48(2):255–288, 1990. [p. 95]

[RS04] Neil Robertson and Paul D. Seymour. Graph minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004. [p. 95]

[RV75] Ronald L. Rivest and Jean Vuillemin. A generalization and proof of the Aanderaa-Rosenberg conjecture. In *Proceedings of the 7th ACM Symposium on Theory of Computing (STOC 1975)*, pages 6–11, 1975. [p. 90]

[San08] Miklos Santha. Quantum walk based search algorithms. In *Theory and Applications of Models of Computation*, volume 4978 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2008. [p. 66]

[SG04] Rocco A. Servedio and Steven J. Gortler. Equivalences and Separations Between Quantum and Classical Learnability. *SIAM Journal on Computing*, 33(5):1067–1092, 2004. [pp. 41, 59, 60]

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997. [p. 4]

[ŠS06] Robert Špalek and Mario Szegedy. All quantum adversary methods are equivalent. *Theory of Computing*, 2:1–18, 2006. [pp. 72, 73, 116]

[Str69] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969. [p. 63]

[SYZ04] Xiaoming Sun, Andrew C. Yao, and Shengyu Zhang. Graph properties and circular functions: How low can quantum query complexity go? In *Proceedings of the 19th IEEE Conference on Computational Complexity (CCC 2004)*, pages 286–293, 2004. [p. 90]

[Sze03] Mario Szegedy. On the quantum query complexity of detecting triangles in graphs. *arXiv preprint* arXiv:quant-ph/0310107, 2003. [pp. 72, 73, 116]

[Sze04] Mario Szegedy. Quantum speed-up of Markov chain based algorithms. In *Proceedings of the 45th IEEE Symposium on Foundations of Computer Science (FOCS 2004)*, pages 32–41, 2004. [p. 108]

[Tho01] Andrew Thomason. The extremal function for complete minors. *Journal of Combinatorial Theory, Series B*, 81(2):318–338, 2001. [p. 95]

[vD98] Wim van Dam. Quantum Oracle Interrogation: Getting All Information for Almost Half the Price. In *Proceedings of the 39th IEEE Symposium on Foundations of Computer Science (FOCS 1998)*, page 362, 1998. [p. 41]

[vDHI06] Wim van Dam, Sean Hallgren, and Lawrence Ip. Quantum Algorithms for Some Hidden Shift Problems. *SIAM Journal on Computing*, 36(3):763–778, 2006. [p. 41]

[Ver98] Nikolai K. Vereshchagin. Randomized Boolean decision trees: Several remarks. *Theoretical Computer Science*, 207(2):329 – 342, 1998. [p. 3]

[Wat09] John Watrous. Zero-knowledge against quantum attacks. *SIAM Journal on Computing*, 39(1):25–58, 2009. [p. 20]

[WBHS11] Nathan Wiebe, Dominic W. Berry, Peter Høyer, and Barry C. Sanders. Simulating quantum dynamics on a quantum computer. *Journal of Physics A*, 44(44):445308, 2011. [p. 31]

[Wil11] Ryan Williams (http://cstheory.stackexchange.com/users/225/ryan-williams). Answer to "What is the most general structure on which matrix product verification can be done in $O(n^2)$ time?". Theoretical Computer Science Stack Exchange, 2011. http://cstheory.stackexchange.com/a/5942 (version: 2011-04-07). [p. 66]

[WR14] Nathan Wiebe and Martin Roetteler. Quantum arithmetic and numerical analysis using repeat-until-success circuits. *arXiv preprint* arXiv:1406.2040, 2014. [p. 20]

[WW10] Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *Proceedings of the 51st IEEE Symposium on Foundations of Computer Science (FOCS 2010)*, pages 645–654, 2010. [pp. 67, 68]

[Zha05] Shengyu Zhang. On the power of Ambainis' lower bounds. *Theoretical Computer Science*, 339:241–256, 2005. [pp. 72, 73, 90, 116]

[Zhu12] Yechao Zhu. Quantum query complexity of constant-sized subgraph containment. *International Journal of Quantum Information*, 10(03):1250019, 2012. [p. 91]