Quantum Algorithmic Techniques for Fault-Tolerant Quantum Computers

by

Mária Kieferová

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Physics (Quantum Information)

Waterloo, Ontario, Canada, 2019

Supervisor:	Michele Mosca, Professor, Dept. of Combinatorics & Optimization, University of Waterloo
Supervisor (cotutelle):	Dominic Berry, Assoc. Professor, Dept. of Physics & Astronomy, Macquarie University
Committee Member:	Christine Muschik, Assist. Professor, Dept. of Physics & Astronomy, University of Waterloo
Committee Member:	Daniel Gottesman, Professor, Dept. of Physics & Astronomy, (Perimeter Institute) University of Waterloo
Internal-External Examiner:	Ashwin Nayak, Professor, Dept. of Combinatorics & Optimization, University of Waterloo
External Examiner:	Ashley Montanaro, Reader, School of Mathematics, University of Bristol

Author's Declaration

This thesis is being submitted to Macquarie University and the University of Waterloo in accordance with the Cotutelle agreement dated February 10, 2017.

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis.

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

Mária Kieferová

Statement of Contributions

Chapter 1
MK wrote this chapter as an introduction to the thesis.

Chapter 2
Section 2.3 is based on part 4.1 of my publication [1] but has been extended to cover Hamiltonian simulation techniques in more detail. The rest of the chapter was written by MK solely for this thesis.

- MK wrote the majority of Chapter 4 of [1] with inputs and revisions from other authors, in particular Peter Johnson, Yudong Cao and Ian Kivlichan.

Chapter 3
Chapter 3 is based on [2]. This chapter was edited and Sections 3.1 and 3.5 added for completeness.

The project was proposed by Dominic Berry and Ryan Babbush.

- MK contributed to the antisymmetrization procedure in [2] with ideas and writing.
- MK designed the parallelized quantum comparator with inputs from Artur Scherer, Dominic Berry and Craig Gidney.

Chapter 4
Chapter 4 is based on the publication [3]. This chapter was edited, and Fig. 4.1 added for clarity.

The project was proposed by Dominic Berry.

- MK contributed to all parts of the project.
- MK wrote the majority of the paper together with Artur Scherer and Dominic Berry.

Chapter 5
Chapter 5 is based on [4]. This chapter was extensively edited, updated for new developments, and an introduction to Boltzmann machines was added. We also added multiple figures for clarity.

The project was proposed by Nathan Wiebe.

- MK contributed with ideas for defining a quantum training set.
- MK contributed to the development of POVM and relative entropy training.
- MK performed the simulations in Sections 5.6.3 and 5.6.4 and the comparison between Golden-Thompson and relative entropy training in 5.6.5.
- MK proved Theorem 8.
- MK contributed to the writing of the paper.

Chapter 7
MK wrote this chapter as a conclusion for this thesis.

Appendix A
MK wrote this appendix to provide additional background information for Chapter 5.

Abstract

Quantum computers have the potential to push the limits of computation in areas such as quantum chemistry, cryptography, optimization, and machine learning. Even though many quantum algorithms show asymptotic improvement compared to classical ones, the overhead of running quantum computers limits when quantum computing becomes useful. Thus, by optimizing components of quantum algorithms, we can bring the regime of quantum advantage closer. My work focuses on developing efficient subroutines for quantum computation. I focus specifically on algorithms for scalable, fault-tolerant quantum computers. While it is possible that even noisy quantum computers can outperform classical ones for specific tasks, high-depth and therefore fault-tolerance is likely required for most applications. In this thesis, I introduce three sets of techniques that can be used by themselves or as subroutines in other algorithms.

The first components are coherent versions of classical sort and shuffle. We require that a quantum shuffle prepares a uniform superposition over all permutations of a sequence. The quantum sort is used within the shuffle and as well as in the next algorithm in this thesis. The quantum shuffle is an essential part of state preparation for quantum chemistry computation in first quantization.

Second, I review the progress of Hamiltonian simulations and give a new algorithm for simulating time-dependent Hamiltonians. This algorithm scales polylogarithmic in the inverse error, and the query complexity does not depend on the derivatives of the Hamiltonian. A time-dependent Hamiltonian simulation was recently used for interaction picture simulation with applications to quantum chemistry.

Next, I present a fully quantum Boltzmann machine. I show that our algorithm can train on quantum data and learn a classical description of

quantum states. This type of machine learning can be used for tomography, Hamiltonian learning, and approximate quantum cloning.

Acknowledgement

This thesis would not be possible without a number of people supporting me throughout my PhD studies. I would first like to thank my supervisors Michele Mosca, Dominic Berry and Gavin Brennen for all their time, advice and encouragement. I cannot thank Michele enough for his guidance since my first internship at IQC in 2012. I would also like to thank Dominic for everything I learned from him and his many comments on earlier drafts of this thesis.

I am very grateful to my advisory committee consisting of Daniel Gottesman, Ashwin Nayak, and Roger Melko as well as additional examiners Christine Muschik, Ashley Montanaro, David Poulin and Jingbo Wang for taking the time to read my thesis and for their helpful suggestions.

I want to thank Microsoft Research and Zapata Computing for giving me the opportunities to join them as an intern during my studies. Spending time in a corporate environment enriched my academic experience, and I thoroughly enjoyed both of my internships.

I would also like to thank everyone involved in the FKS competition for opening the doors of physics for me, my former supervisor Daniel Nagaj for introducing me to quantum computing and Nathan Wiebe for his ongoing mentorship. I am also thankful to my friends and colleagues for all the conversations that enriched me academically.

I would not be able to carry out the research in this thesis without my collaborators Ryan Babbush, Yudong Cao, Matthias Degroote, Craig Gidney, Peter D Johnson, Alán Aspuru-Guzik, Ian D Kivlichan, Guang Hao Low, Tim Menke, Jonathan Olson, Borja Peropadre, Sadegh Raeisi, Jonathan Romero, Nicolas PD Sawaya, Sukin Sim, Yuval Sanders, Artur Scherer and Libor Veis. I thank Mike and Ophelia Lazaridis Fellowship, University of Waterloo, Michele Mosca and Macquarie University for financial support.

Lastly, I want to thank my parents for raising me, to Danielka for her friendship since the day she was born and to Yuval for his love and caffeine supply.

Table of Contents

List of Figures

List of Abbreviations

ANN	Artificial Neural Network
BM	Boltzmann Machine
CG	Conjugate Gradient
FY	Fisher-Yates (shuffle)
GT	Golden-Thompson
h.c.	Hermitian conjugate
HHL	Harrow, Hassidim, Lloyd (algorithm [5])
KL	Kullback-Leibler (divergence)
LCU	Linear Combination of Unitaries
NISQ	Noisy Intermediate-Size Quantum
OAA	Oblivious Amplitude Amplification
QAOA	Quantum Approximate Optimization Algorithm
QBM	Quantum Boltzmann Machine

QFT	Quantum Fourier Transform
QSP	Quantum Signal Processing
QSVT	Quantum Singular Value Transformation
SVM	Support Vector Machine
VQE	Variational Quantum Eigensolver

"At the end of a miserable day, instead of grieving my virtual nothing, I can always look at my loaded wastepaper basket and tell myself that if I failed, at least I took a few trees down with me."

David Sedaris (Me Talk Pretty One Day)

1

Introduction

"I don't know what I think until I write it down."

Joan Didion

The promise of quantum algorithms is a major force driving the development of quantum technologies. The beginning of quantum algorithms can be traced back to Feynman [6], who suggested that quantum systems could be used for solving difficult problems. The model of computing using quantum systems was first formulated by David Deutsch and is known as the quantum Turing machine [7]. Soon after, Deutsch and Jozsa [8] formulated a problem that can be provably solved faster on a quantum computer than a classical one, Černy showed that quantum systems can be used for solving NP-complete problems [9] (albeit using an exponential amount of energy) and Bernstein and Vazirani [10], and Simon [11] developed their algorithms for oracular problems.

Three of the major results in the early years of quantum algorithms were Shor's algorithm [12] for factoring, Lloyd's algorithm for simulating quantum systems [13] and Grover's search algorithm [14]. Over the years,

quantum algorithms research expanded and developed subfields such as quantum walks [15–27], Hamiltonian simulation [28, 29, 29–37], and adiabatic quantum computation [28, 38–44]. The applications of quantum computing grew as well. Today, there are quantum algorithms for problems ranging from number theory [12, 45, 46] through simulation of quantum field theories [47–49] to neural networks [50–53].

In the subsequent years, quantum algorithms became increasingly more sophisticated. Nowadays, quantum algorithms often employ multiple subroutines such as Hamiltonian simulation, coherent simulation of classical computation or state preparation.

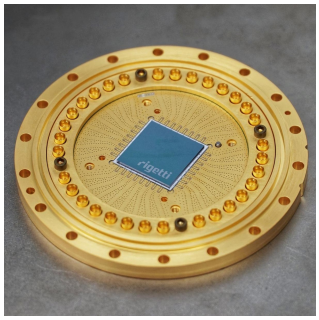
In this thesis, we propose algorithmic techniques that can be used by themselves or as procedures in more complex algorithms. We focus our effort on algorithms for gate-based quantum computers. All of the techniques are also approximate, meaning that they are guaranteed to produce an output within an error ϵ from the actual solution. Whenever possible, we quantify their resource requirements in terms of the size of the input and this error.

At the time of the writing of this thesis, there were no known devices capable of implementing the proposed algorithms. The current devices are quite limited in the size of computation, but the number of available qubits keeps increasing while the noise levels become lower and lower. The holy grail of hardware development is decreasing the noise on a sufficient number of qubits below the threshold where the errors can be efficiently corrected. We call these devices fault-tolerant, and we devised algorithms for them.

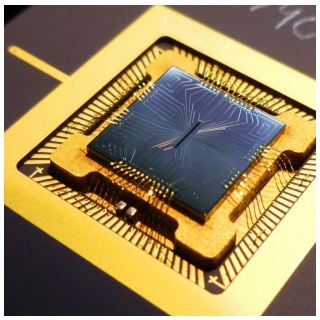
1.1 Quantum Computing Hardware in 2019

Today, we are arguably seeing the dawn of quantum computers. According to John Preskill's speech from December 2017, we are just entering the era of Noisy Intermediate-Scale Quantum (NISQ) devices [54]. This era is characterized by access to chips with 50-100 noisy qubits that run hybrid algorithms such as the variational quantum eigensolver (VQE) [55] or quantum approximate optimization algorithm (QAOA) [56]. The devices in this era are not fault-tolerant and every component of computation

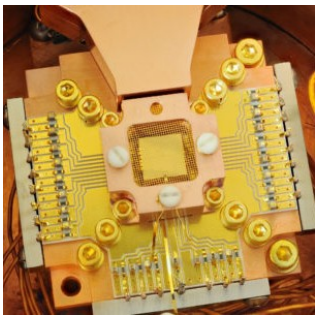
(a) Google [58]



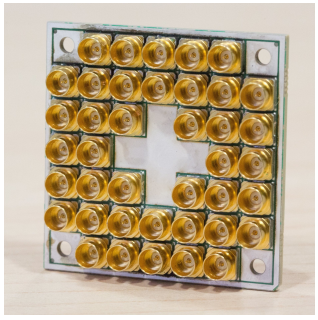
(b) Rigetti [59]



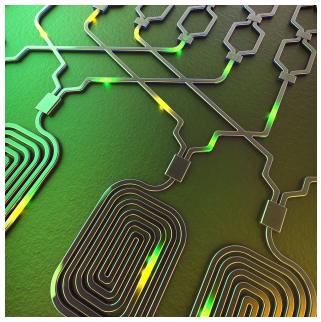
(c) Sandia [60]



(d) IBM [61]



(e) Intel [62]



(f) Bristol/Guangzhou [63]

Figure 1.1: Several NISQ chips in 2019. Many other devices are being developed by academic groups around the world.

introduces errors. These errors come from state initialization, single and multiple qubit gates, measurement as well as the decoherence in the entire system. At the time of the writing, the number of two-qubit gates that can be implemented without a significant loss in accuracy appears to be the main bottleneck for implementation of algorithms.

Currently, there are several proposals for the physical architecture of a quantum computer, each with their respective advantages and drawbacks. For example, the highest number of qubits connected by gates tends to be achieved for superconducting qubits, but ion-trap based computers consistently show higher gate precision. Other measures include the connectivity (when a gate can be applied on a pair of qubits), the ability to initialize qubits and keep them stable, the measurement precision and scalability. All the theoretical requirements for building a scalable universal quantum computer are summarized by DiVincenzo's criteria [57].

On the superconducting front, companies such as Google [58], IBM [64], Intel [62] and Rigetti [65] are developing (or already developed) NISQ chips. We also see the renaissance of ion trap based computers with commercial efforts of ION-Q [66] and Honeywell [67]. Other architectures include photonics, nuclear spins and NMR, analog quantum simulators and annealers such as the D-Wave machines.

It is not clear which hardware architecture is the most suitable for building a quantum computer. It is also possible that future quantum computers will consist of multiple types of quantum systems linked together. At this point, a quantum algorithm theorist such as myself can only speculate. For this reason, we focus on the development of hardware agnostic algorithms.

1.2 Quantum Computing Software in 2019

The capability of NISQ devices is in stark contrast with the requirements of most quantum algorithms. The quantum computing community works on closing this gap from both sides – increasing the power of quantum devices and decreasing the requirements of quantum algorithms.

The decline in the resource requirements of quantum algorithms has been due to the developments in error correction, circuit synthesis, and quantum algorithms. As a consequence, the expected number of (physical) qubits required to break 2048-bit RSA was reduced from 1000 million in 2012 [68] to 20 million while decreasing the overall computational time to under 8 hours [69].

In this thesis, we focus on algorithmic improvements. One of the practical applications we consider is the electronic structure problem in quantum chemistry. The goal is to efficiently estimate the ground state energy of a given fermionic Hamiltonian given a description of the Hamiltonian and a good initial guess of the ground state (from an experiment or a mean-field computation). The problem stated this way has long been known to be polynomially solvable on a quantum computer but it took over a decade of research to decrease the powers of the polynomial [37, 70–74].

Code breaking (for example, factorization for breaking the RSA cryptosystem [75]) and quantum chemistry are two of the areas that can be revolutionized by quantum computing. We also expect improvements in

machine learning and optimization. All of the techniques in this thesis can be used in one or more of these areas. We now proceed to give their overview.

1.3 Overview

During my PhD I co-authored the following publications.

- Sadegh Raeisi, Mária Kieferová, and Michele Mosca. *Novel Technique for Robust Optimal Algorithmic Cooling.* Physical Review Letters, 2019 [76].
- Mária Kieferová, Artur Scherer, and Dominic W Berry. *Simulating the dynamics of time-dependent Hamiltonians with a truncated Dyson series.* Physical Review A, 2019 [3]
- Yudong Cao, Jonathan Romero, Jonathan P Olson, Matthias Degroote, Peter D Johnson, Mária Kieferová, Ian D Kivlichan, Tim Menke, Borja Peropadre, Nicolas PD Sawaya, Sukin Sim, Libor Veis, and Alán Aspuru-Guzik. *Quantum chemistry in the age of quantum computing.* Chemical Reviews, 2018. [1].
- Dominic W Berry, Mária Kieferová, Artur Scherer, Yuval R Sanders, Guang Hao Low, Nathan Wiebe, Craig Gidney, and Ryan Babbush. *Improved techniques for preparing eigenstates of fermionic Hamiltonian.* npj Quantum Information, 2018 [2].
- Mária Kieferová and Nathan Wiebe. *Tomography and generative training with quantum Boltzmann machines.* Physical Review A, 2017 [4].
- Mária Kieferová and Nathan Wiebe. *On the power of coherently controlled quantum adiabatic evolutions.* New Journal of Physics, 2014 [77]

Publications [2–4] constitute the research results presented this thesis. Publication [1] is a review of contemporary quantum computing techniques for quantum chemistry. Publication [77] came from my Master's thesis and I finished it during the first months of my PhD. Publication [76] was an early

project unrelated to my other research and has been recently published in Physical Review Letters.

This thesis is organized as follows. First, we review the existing algorithmic techniques in Chapter 2. Specifically, we review preliminaries in Section 2.1, give a brief overview of commonly used components of quantum algorithms in Section 2.2 and summarize the progress in Hamiltonian simulation in Section 2.3.

The original research from my PhD is the content of Chapters 3 – 5.

Chapter 3 gives coherent versions of sorting and shuffling algorithms, including detailed circuits for their implementation. We also show that an inverse sort can be used as a shuffle. We then use the coherent shuffle for preparation of a completely antisymmetric state in quantum chemistry.

Chapter 4 focuses on the simulation of time-dependent Hamiltonians. Such Hamiltonians arise whenever time-dependent fields or moving nuclei are involved, in simulations in the interaction picture and for digitally simulating adiabatic evolution.

In Chapter 5 we introduce a quantum algorithm that can learn directly from quantum data. We generalize the concept of a training set into a quantum context and show how to represent classical data in this framework. At the same time, our machine learning algorithm gives a new approach to tomography. For readers unfamiliar with machine learning techniques we included a brief introduction in Appendix A.

I conclude this thesis by reviewing the results and presenting a list of open questions.

1.4 Notation

We now briefly review the notation used in this thesis:

- All logarithms are base 2 unless specified otherwise.
- We use the big-$\mathcal{O}$ notation to specify asymptotic upper bounds and Ω stands for asymptotic lower bounds.

- As is customary in quantum computing, the symbol H can stand both for the Hadamard gate, in the context of quantum circuits, and a Hamiltonian, if talking about the evolution of quantum systems. The correct meaning of H should be clear from the context.
- We occasionally refer to complexity classes such as NP or BQP. The definition of these classes can be found in standard textbooks or the Complexity Zoo [78].

2
Background

"I love my computer and its software;
I hug it often though it won't care.
I love each program and every file,
I'd love them more if they worked a while."

Dr. Seuss, The Lost Dr. Seuss Poem

In this chapter, I review the preliminaries and landmark results used in the rest of the thesis. I start by introducing the circuit model, the notion of complexity and fault-tolerance. Next, I review techniques used in my work including quantum simulation of classical computation, phase estimation, and amplitude amplification. The last section focuses on the development of Hamiltonian simulation that is heavily featured in my work. This review serves as a background for algorithms in Chapters 3-5 and is not intended for a quantum computing novice. For a pedagogical introduction to quantum computing and quantum algorithm please see [79–81] or refer to quantum algorithm reviews [82–85].

2.1 Preliminaries

2.1.1 Quantum Gates and the Circuit Model

The first concept we need to introduce is a quantum bit, qubit. A classical bit can be either in state 0 or 1. A qubit can be in any superposition of these states.

Qubit states are typically expressed either as vectors or in Dirac notation. The "classical" states are assigned notation $|0\rangle := \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ for state 0 and $|1\rangle := \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ for state 1 in Dirac and vector notation respectively. These states form the computational basis on a 2-dimensional Hilbert space.

A general qubit state can be expressed as a linear combination $\alpha|0\rangle + \beta|1\rangle$ where α, β are complex numbers and $|\alpha|^2 + |\beta|^2 = 1$. In the vector form, a qubit can be expressed as a normalized $\mathbb{C}^2$ vector

$$|\psi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}. \tag{2.1}$$

With two bits, the potential computational states are 00, 01, 10 and 11. The state on two qubits can be expressed as $\alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle$ for arbitrary α, β, γ and δ that satisfy the normalization condition $|\alpha|^2 + |\beta|^2 + |\gamma|^2 + |\delta|^2 = 1$. A state on n qubits is a superposition over 2^n basis states.

Operations on qubits can be expressed using quantum circuits, described through diagrams as in Fig. 2.1. Quantum circuits show how a potentially complicated operation can be broken down into elementary components, analogously to logic circuits. The components of quantum circuits are simple unitary operations called gates and measurements.

One has a lot of freedom to pick a set of gates used in an algorithm. The choice of gates is often informed by the physical architecture (i.e., superconducting qubits vs. trapped ions) and error-correcting code. A requirement is that the chosen set of gates is universal.

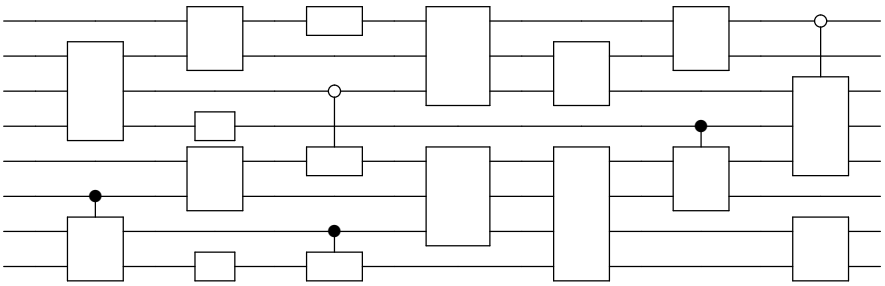
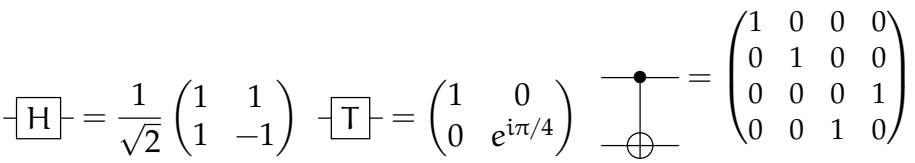


Figure 2.1: A quantum circuit consists of wires (qubits, represented as horizontal lines) and gates (unitary operations, represented as boxes). The circuit represents a sequence of operations with time going from left to right. Some of the gates can be controlled which means that they are executed only if the control bit is in a given state. We use a black full circle to indicate that an operation is applied only if the control qubit is $|1\rangle$ and we use an empty circle for $|0\rangle$. The operation can act in superposition, i.e. if the control qubit is in state $\alpha|0\rangle + \beta|1\rangle$, the (full circle) controlled unitary U will act on the target state $|\psi\rangle$ as $(\alpha|0\rangle + \beta|1\rangle)|\psi\rangle \to \alpha|0\rangle|\psi\rangle + \beta|1\rangle U|\psi\rangle$. The space cost of this circuit is 8, the time cost is 17, and the depth is 7, see definitions below.

A universal set of gates [86, 87] is a finite set of gates such that any finite-dimensional unitary operation can be implemented as a circuit consisting only of gates from this set. One such set is depicted in Fig. 2.2. According to the Solovay-Kitaev theorem [88], any two universal sets of gates can simulate each other with at most polylogarithmic overhead. For convenience, we often add Pauli matrices, phase gate (denoted S and equal to T^2, see Fig. 2.2) and multiply-controlled-NOTs into our available gate set. Synthesizing quantum operations into elementary gates is an active area of research [86, 89] with immense implications for near-term quantum devices.

Quantum computation often requires the use of additional qubits known as ancillae. They play the same role as a working memory in classical computing; many calculations can be more efficient if there is more space available. At the same time, a large number of ancillae is prohibitive for practical purposes.



(a) Hadamard gate

(b) T-gate.

(c) Controlled Pauli X, also known as controlled not or CNOT

Figure 2.2: A universal set of gates consisting of Hadamard, T gate and a C-NOT. This set of gates is arguably the most common in theoretical quantum computing.

Quantum states are converted into classical information through a measurement. In principle, it is always possible to delegate measurements to the end of the circuit. However, it is often advantageous in practice to measure as soon as possible [90, 91].

2.1.2 Complexity

We calculate the cost of computation in terms of the required resources as a function of the length of the input and other relevant parameters (for example, the required precision of computation ϵ). As such, complexity shows how the "difficulty" of the computation scales when the input grows. The space complexity refers to the number of qubits, while the time complexity (often referred to only as the complexity) gives the number of gates. Often, multiple gates can be executed in parallel. The depth of the circuit then refers to the number of rounds or layers when gates were applied (see Fig. 2.1).

The number of required gates can differ based on the gate set one works with. However, a consequence of the Solovay-Kitaev theorem is that this difference is at most $\mathcal{O}\left(\log^c\left(1/\epsilon\right)\right)$ for $c \geqslant 1$. In practice, not all gates are created equal. On near-term noisy devices (with no error correction) [54], two-qubit gates are the most common bottleneck. One may thus wish to minimize the number of two-qubit gates instead of gates in general. For fault-tolerant computation based on the surface code [92], gates such as

T or Toffoli are several hundred times more expensive than Hadamard, Paulis, CNOT and other gates from Clifford group [93, 94].

Apart from gates, it is possible to include oracles (sometimes also called black boxes) in computation. An oracle is an abstract operation that can perform a given computation in a unit of time. We refer to the number of queries to the oracle as the query complexity. There are several reasons why one wants to include oracles in an algorithm. An oracle can represent a separate computational primitive whose implementation can vary. For example of such oracles are common in Hamiltonian simulation; see Subsection 2.3.1. An oracle can be given as a part of the definition of the problem. In this case, one can have oracular access to a function, and our goal is to determine some properties of this function as in Deutsch-Joza [8] or Simon's [11] algorithms. Finally, oracles are often used in the theory of computational complexity to quantify the difficulty of tasks. One can construct a hierarchy of computational difficulty with respect to more and more powerful oracles [95].

2.1.3 Error Correction and the Threshold Theorem

In the real world, neither the qubits nor the operations on them will be implemented perfectly. There are a plethora of complications that come into engineering quantum devices, starting from the difficulty of fabrication through creating a well-isolated environment to the limited life of the qubits. Besides, all operations are imprecise, and the qubit-qubit interactions are the obvious bottleneck for most implementations. These errors accumulate throughout a computation, making a straightforward implementation of an algorithm fail even for modestly sized circuits.

Fortunately for us, if the errors are below a given threshold, it is possible to correct for them faster than they occur. The above statement is made rigorous in the quantum threshold theorem [96].

Theorem 1 (Quantum threshold theorem [96] , Theorem 10 in [97]) *There is a threshold error rate p_T. Suppose we have a local stochastic error model with $p_i \leqslant p < p_T$[1]. Then for any ideal circuit C, and any $\epsilon > 0$, there exists*

[1]Here p_i is the probability of an uncorrelated error at a location C_i, see Definition 11 in [97].

a fault-tolerant circuit C', which, when it undergoes the error model, produces an output which has statistical distance at most ϵ from the output of C. C' has a number of qubits and a number of timesteps which are at most polylog$(|C|/\epsilon)$ times bigger than the number of qubits and timesteps in C, where $|C|$ is the number of locations in C.

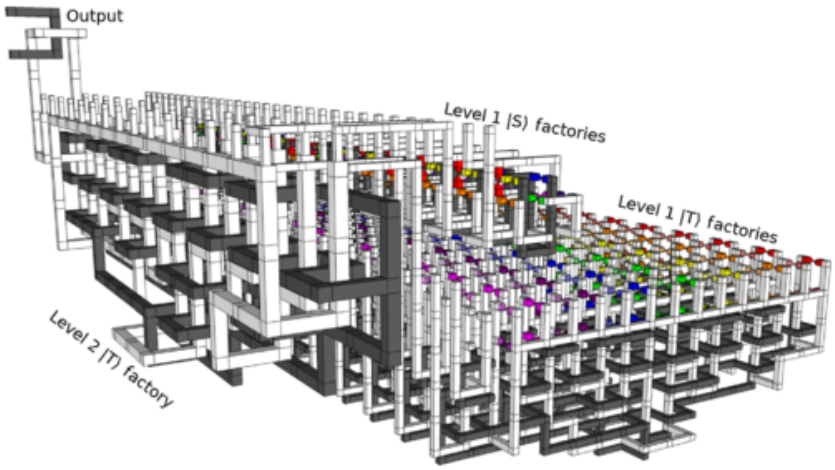


Figure 2.3: Illustration of the difficulty of applying fault-tolerant T gates from [73, Fig. 24]. The figure depicts a "T-factory" in surface code preparing $|T\rangle$ states where $|T\rangle = \cos\beta\,|0\rangle + e^{i\pi/4}\sin\beta\,|1\rangle$ and $\cos(2\beta) = \frac{1}{\sqrt{3}}$. One $|T\rangle$ can be used for fault-tolerant implementation of one T gate [93]. Implementation of T (or different non-Clifford) gates are the bottleneck for fault-tolerant algorithms.

Several things need to be pointed out in the threshold theorem. The first thing is the exact meaning of error rate and how would one estimate it; see [96, 98] for more detail. The second thing is the noise model. Making assumptions about the noise, particularly its locality, is necessary. There is some evidence that error correction for a specific type of non-localized noise might not be possible [99, 100]. In contrast, one can correct errors more effectively given additional promises about the error model. The last and most commonly examined one is how would one go about devising the circuit C'. Quantum error correction and fault-tolerant gadgets address this point.

The main idea is to use redundancy to encode a logical qubit into a

collection of physical qubits. The logical qubit is then encoded into non-local properties of the collection of physical qubits, making it resilient to localized errors. The first example of an error correcting code was Shor's 9-qubit code [101].

Currently, most of practical error correction uses topological codes [102, 103], most prominently surface codes [68, 104, 105] with the stabilizer formalism [94]. Broadly speaking, stabilizers allow us to detect errors without disrupting the computation. Importantly, detecting and correcting errors can be done efficiently.

Surface codes allow us to fault-tolerantly implement logical qubits into physical qubits on a 2-D lattice, which makes them a prime candidate for error correction in superconducting qubits. Even then, the typical ratio between physical and logical qubits can be of the order of thousands[2].

Clifford gates (i.e., Paulis, H, S and CNOT) can be fault-tolerantly implemented in surface codes quite easily. However, it is necessary to include non-Cliffords gates such as the T gate or Toffoli which are implemented using magic states and fault-tolerant gadgets [93]. To illustrate the difficulty of implementing a T gate, we included a picture demonstrating its implementation in a surface code (see Fig. 2.3). Consequently, the cost of computation is predominantly determined by the number of T gates.

We could not possibly give justice to fault-tolerance by such a brief overview. For a more in-depth introduction to error correction see [68, 94] or any of the cited references.

2.2 A Brief Overview of Quantum Algorithms

The speedup in many algorithms stems from a smart application of several quantum transformations. In this chapter, we give an overview of the most used computing primitives and their applications.

[2] A recent result [69] uses 1568 physical qubits for each logical qubit. This estimate uses lattice surgery and code distance $d = 27$.

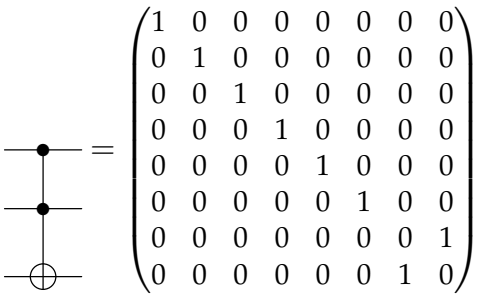


Figure 2.4: The Toffoli gate, also known as doubly-controlled-NOT is universal for classical computation and reversible.

2.2.1 Coherent Classical Computation

Before we move onto quantum algorithm primitives, let us review how a quantum computer can simulate classical computation. Examples of classical computation in quantum algorithms include logic, arithmetic or integral evaluation. In this thesis, we propose coherent versions of classical sorting and shuffling algorithms in Chapter 3.

Even though these algorithms do not provide a speedup compared to their classical counterparts, they are often an unavoidable part of a computation. For example, quantum chemistry algorithms such as the one in [73] require integral evaluation to compute matrix elements of the Hamiltonian. Calculating them classically in advance and accessing them from a lookup table would asymptotically increase the complexity of the algorithm; hence, the integrals are coherently evaluated on the quantum computer.

One of the inspirations for quantum computing was the work by Toffoli [106], Fredkin [107], Landauer [108], Bennett [109,110] and others on classical reversible computation and its connection to thermodynamics and information theory. The reversible Toffoli gate (Fig. 2.4) is universal for classical computation because it can simulate the NAND logic gate; see Fig. 2.5. As such, reversible computation is equivalent to classical computation. Since the Toffoli can be also be implemented as a quantum gate, reversible computation gives us a way to construct classical computing primitives in quantum circuits.

The downside of replacing NAND gates with Toffolis is the number of additional bits. The NAND logic gate takes two bits as inputs and outputs one bit, whereas the Toffoli has three qubits on both the input and output. This means that the number of ancillae would increase linearly with the number of gates in this naive construction. Such a construction quickly becomes wasteful for large circuits or if space is limited. One can save ancillae by reversing the computation to uncompute the information saved in these registers [86, 111] and disentangle them from the output. These techniques are crucial in algorithm design, and we use them in Chapter 4.

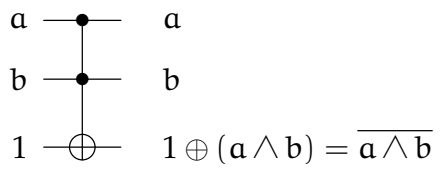


Figure 2.5: A single Toffoli can replicate the NAND operation. The inputs of NAND stay unchanged and the result is encoded into an ancilla.

The space limitation is crucial for the practicability of near-term quantum computers; even modest constant factors can make a difference for implementation on near-term hardware.

Several lines of research aim to make a quantum simulation of classical computation. The use of additional gates, such as NOT or CNOT results in simpler circuits than a naive construction with only Toffolis [112–114]. NOT and CNOT map computational basis states onto computational states; this means that they can be regarded as classical gates. Next, there has been significant effort to efficiently replicate classical computation using quantum components (such as Hadamards or T gates [115]). Lastly, it is often preferred to avoid arithmetic and classical computation altogether [116].

2.2.2 Phase Estimation and Eigenstate Preparation

Another widely used component of algorithms is phase estimation [117, 118]. Given a unitary U and its eigenstate $|\psi_\theta\rangle$ satisfying

$$U|\psi_\theta\rangle = e^{i2\pi\theta}|\psi_\theta\rangle, \tag{2.2}$$

phase estimation can prepare with high probability the state $|\psi_\theta\rangle\,|\widetilde{\theta}\rangle$, where $\widetilde{\theta}$ is an m-digit approximation of θ. It can be shown that if we allow for a failure rate at most ϵ, the algorithm requires $n = m + \left\lceil \log\left(2 + \frac{1}{2\epsilon}\right)\right\rceil$ additional qubits (not including the qubits needed to encode $|\psi\rangle$), 2^n queries to controlled U, and $\mathcal{O}\left(n^2\right)$ elementary gates. We assume that we are given an oracle to implement controlled U. The circuit implementing phase estimation is pictured in Fig. 2.6.

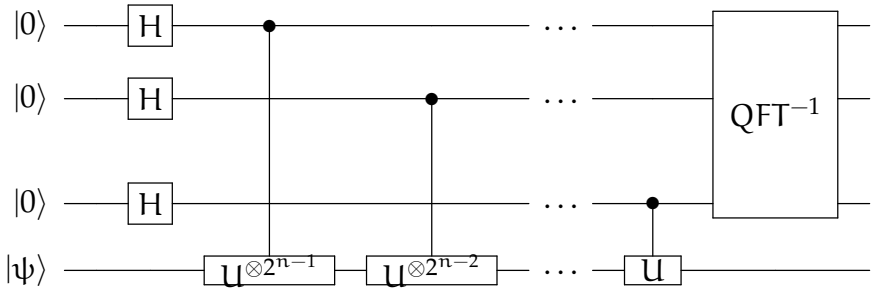


Figure 2.6: Eigenvalue estimation circuit. The last block represents the inverse quantum Fourier transform depicted in Fig. 2.2.2.

Phase estimation starts by creating a uniform superposition on the ancillary register by applying a Hadamard gate on each ancilla qubit.

$$|00\ldots0\rangle\,|\psi\rangle \rightarrow \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} |k\rangle\,|\psi\rangle\,. \tag{2.3}$$

In the next step, controlled U^{2^j} operations are applied.

$$\frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} |k\rangle\,|\psi\rangle \rightarrow \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} |k\rangle\,U^k\,|\psi\rangle \tag{2.4}$$

$$= \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} |k\rangle\,e^{i2\pi k\theta}\,|\psi\rangle\,. \tag{2.5}$$

The last part of phase estimation is the application of the inverse Fourier transform (QFT^{-1}, defined below). If θ can be expressed exactly in n bits, $\widetilde{\theta}$

would be equal to θ. Otherwise, $\widetilde{\theta}$ is an m-digit approximation distributed around θ.

The inverse quantum Fourier transform in the group $\mathbb{Z}_{2^n}$ (all we need in this case) is defined as

$$\mathrm{QFT}_{2^n}^{-1} : |x\rangle \to \frac{1}{\sqrt{2^n}} \sum_{y=0}^{2^n-1} e^{-2\pi i \frac{x}{2^n} y} |y\rangle , \tag{2.6}$$

where $x = x_1 x_2 \ldots x_n$ is an integer and x_i its digits in binary notation. Implementation of $\mathrm{QFT}_{2^n}^{-1}$ on n qubits requires $\mathcal{O}\left(n^2\right)$ gates [81] (or $n \log n$ for an approximate algorithm [119]) and consists of n stages. In the j-th stage, operations controlled by qubits $j+1$ to n are applied on the qubit j. The controlled rotations R_2 to R_{n-j+1} are defined as

$$R_k = \begin{pmatrix} 1 & 0 \\ 0 & e^{-2i\pi/2^k} \end{pmatrix} . \tag{2.7}$$

Rotation R_k applied to qubit j is always controlled by the $(k-j+1)$th qubit. The circuit for $\mathrm{QFT}_{2^n}^{-1}$ on 4 qubits in pictured in Fig. 2.2.2.

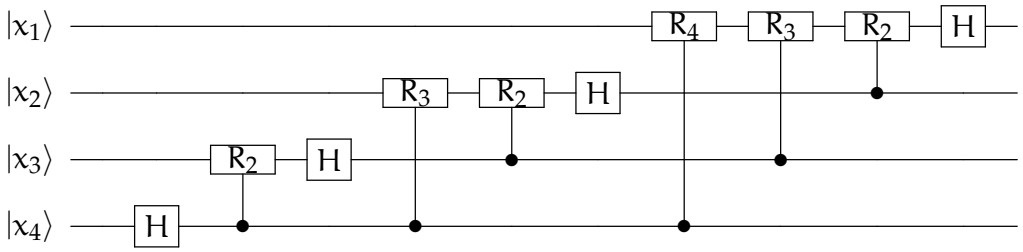


Figure 2.7: Inverse Quantum Fourier Transform on 4 qubits.

The phase estimation algorithm can be used to prepare eigenstates. Let us apply phase estimation on a state $|\phi\rangle$ that is not an eigenvector of U. $|\phi\rangle$ can be then decomposed into eigenvectors as $|\psi\rangle = \sum_i a_i |\psi_i\rangle$. Phase estimation on $|\phi\rangle$ will produce

$$\sum_i a_i |\widetilde{\theta}_i\rangle |\psi_i\rangle , \tag{2.8}$$

where $\widetilde{\theta}_i$ is an approximation of the eigenphase associated with $|\psi_i\rangle$. Measuring the first register will produce one of the eigenstates $|\psi_i\rangle$ with probability $|a_i|^2$.

This method is often used to estimate ground state energies with $U = e^{-iHt}$ for $0 < t < \frac{2\pi}{\|H\|}$ implemented through Hamiltonian simulation techniques (see Section 2.3). Assume that we have a method to prepare a state $|\phi\rangle$ that has a significant overlap with the ground state, i.e., $\langle g|\phi\rangle = a_0$. Furthermore, assume that we are given a promise that any state with energy at most E_{promise} is the ground state. The eigenstate preparation algorithm applied on $|\phi\rangle$ will then produce the ground state with probability $|a_0|^2$. We know we succeeded when we measured $\theta < tE_{\text{promise}}$ in the second register. Since t is known, we can use θ to estimate ground state energy. Several techniques build on phase estimation to estimate energies more efficiently, see [2, 120]. A particularly interesting one is the semi-classical phase estimation [90], later perfected in [121]. This approach replaces all two-qubit gates in QFT^{-1} by measurements and adaptive rotations.

2.2.3 Grover Search and Amplitude Amplification

A large number of quantum algorithms [50, 122, 123] achieve their speedup thanks to amplitude amplification [124]. This idea originated from Grover's algorithm for unstructured search [14].

Grover's algorithm aims to find a marked item in a collection of $N = 2^n$ items. The search problem can be defined formally as finding a preimage of a function in the following setting. Let $f : \{0,1\}^n \to \{0,1\}$ such that x is marked if and only if $f(x) = 1$. The goal of the algorithm is to find such x. If the function f has no structure it is generally not possible for a classical computer to succeed in fewer than $\mathcal{O}(N)$ steps.

To recognize the marked item, we are given an oracle O_{mark}:

$$O_{mark}|x\rangle = \begin{cases} -|x\rangle & \text{if } x \text{ is marked,} \\ |x\rangle & \text{otherwise.} \end{cases} \tag{2.9}$$

The first step of the algorithm is to create a uniform superposition using

Hadamard transform

$$H^{\otimes \log N} |0 \dots 0\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle \,. \tag{2.10}$$

Assume that M out of N items are marked. The uniform superposition can be then decomposed into marked and unmarked states

$$\frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle = \frac{1}{\sqrt{N}} \sum_{x \text{ is marked}} |x\rangle + \frac{1}{\sqrt{N}} \sum_{x \text{ is not marked}} |x\rangle \,, \tag{2.11}$$

$$= \sqrt{\frac{M}{N}} \, |\text{marked}\rangle + \sqrt{\frac{N-M}{N}} \, |\text{unmarked}\rangle \,, \tag{2.12}$$

where we defined the marked and unmarked superposition states

$$|\text{marked}\rangle := \sqrt{\frac{1}{M}} \sum_{x \text{ is marked}} |x\rangle \,, \tag{2.13}$$

$$|\text{unmarked}\rangle := \sqrt{\frac{1}{N-M}} \sum_{x \text{ is not marked}} |x\rangle \,. \tag{2.14}$$

It is straightforward to see that $\langle \text{marked} | \text{unmarked} \rangle = 0$. For convenience, let us introduce the angle θ such that $\sin\theta = \sqrt{\frac{M}{N}}$. In this notation,

$$\frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle = \sin\theta \, |\text{marked}\rangle + \cos\theta \, |\text{unmarked}\rangle \,. \tag{2.15}$$

Let us define the reflection around the uniform superposition

$$R = -\left(\mathbb{I} - 2H^{\otimes \log N} |0\rangle\langle 0| \, H^{\otimes \log N} \right) \tag{2.16}$$

$$\begin{aligned} &= -\cos(2\theta) \, |\text{marked}\rangle\langle\text{marked}| + \cos(2\theta) \, |\text{unmarked}\rangle\langle\text{unmarked}| \\ &\quad + \sin(2\theta) \left(|\text{marked}\rangle\langle\text{unmarked}| + |\text{unmarked}\rangle\langle\text{marked}| \right) \end{aligned} \tag{2.17}$$

and observe that O_{mark} acts as a reflection around the unmarked state.

We can show show that the operation $G = RO_{mark}$ acts as a rotation by an angle 2θ on the subspace span$\{|\text{marked}\rangle, |\text{unmarked}\rangle\}$. Let us apply G on an arbitrary state $|\psi\rangle = \sin\alpha\,|\text{marked}\rangle + \cos\alpha\,|\text{unmarked}\rangle$:

$$\begin{aligned} G\,|\psi\rangle &= RO_{mark}\Big(\sin\alpha\,|\text{marked}\rangle + \cos\alpha\,|\text{unmarked}\rangle\Big) && (2.18)\\ &= R\big(-\sin\alpha\,|\text{marked}\rangle + \cos\alpha\,|\text{unmarked}\rangle\big) && (2.19)\\ &= \sin\alpha(\cos(2\theta)\,|\text{marked}\rangle - \sin(2\theta)\,|\text{unmarked}\rangle) \\ &\quad - \cos\alpha(-\cos(2\theta)\,|\text{unmarked}\rangle - \sin(2\theta)\,|\text{marked}\rangle) && (2.20)\\ &= \sin(\alpha+2\theta)\,|\text{marked}\rangle + \cos(\alpha+2\theta)\,|\text{marked}\rangle && (2.21) \end{aligned}$$

It can be shown that span$\{|\text{marked}\rangle, |\text{unmarked}\rangle\}$ is closed under the application of G. Thus, G rotates the initial state towards $|\text{marked}\rangle$. Assuming that $\sqrt{\frac{N}{M}} \ll 1$, this requires approximately $\frac{\pi}{4}\sqrt{\frac{M}{N}}$ applications of G. In the case where $M \ll N$, Grover's algorithm achieves quadratic speedup over any classical algorithm.

Grover search can be generalized to amplitude amplification. Amplitude amplification can be used to boost the success amplitude for a wide variety of quantum computations. Let us take a unitary U that prepares a state that has λ overlap with the desired state $|\text{good}\rangle$ and can be therefore written as

$$U\,|0\ldots0\rangle = \sin\theta\,|\text{good}\rangle + \cos\theta\,|\text{bad}\rangle\,, \tag{2.22}$$

where $\langle\text{good}|\text{bad}\rangle = 0$. In this notation, $\theta = \frac{\pi}{2}$ means that we reached the desired state $|\text{good}\rangle$ with certainty. Given reflections around the zero state $R_0 = \mathbb{I} - 2\,|0\rangle\langle 0|$ and the correct state $R_{\text{good}} = \mathbb{I} - 2\,|\text{good}\rangle\langle\text{good}|$ we can then construct the operator $G = UR_0U^{-1}R_{\text{good}}$. The operator G is analogous to Grover's operator – the same type of calculation can be used to prove the correctness of the algorithm. Reaching a high probability of success requires approximately $\frac{\pi}{4\theta}$ applications of G.

The presented versions assume that the fraction of marked items $\frac{M}{N}$ and the initial overlap is known. The initial amplitude can be estimated using quantum amplitude estimation [79]. Alternatively, a more technical version of amplitude amplification can be used instead [125] to avoid over-rotating. Another extension of amplitude amplification is oblivious amplitude amplification which we discuss in the next section.

2.2.4 Oblivious Amplitude Amplification

Oblivious amplitude amplification (OAA) extends the framework of amplitude amplification by relaxing the requirement on the reflection. While amplitude amplification reflects around the good state, $|\text{good}\rangle$, OAA reflects only around a control system, leaving the target system untouched. OAA then allows us to turn a probabilistic implementation of an operation [77, 126] into a nearly perfect one without an exponential overhead.

Suppose that W probabilistically implements U such as

$$W|0\rangle_{\text{flag}}|\psi\rangle_{\text{data}} = \sin\theta\,|0\rangle_{\text{flag}} U|\psi\rangle_{\text{data}} + \cos\theta\,|\text{bad}\rangle\,, \tag{2.23}$$

where $\left(|0\rangle\langle 0|_{\text{flag}} \otimes \mathbb{I}_{\text{data}}\right)|\text{bad}\rangle = 0$. In other words, if we measure the ancilla in state $|0\rangle$, U was properly applied on $|\psi\rangle_{\text{data}}$.

Berry et al. showed in [34] that $WRW^\dagger R$ acts as a rotation

$$\left(WRW^\dagger R\right)^k W|0\rangle|\psi\rangle = \sin\left((2k+1)\theta\right)|0\rangle U|\psi\rangle + \cos\left((2k+1)\theta\right)|\text{bad}\rangle\,, \tag{2.24}$$

where R is the reflection around all zero in the flag register $R = 2|0\rangle\langle 0| \otimes \mathbb{I} - \mathbb{I}$. Note that oblivious amplitude amplification differs from amplitude amplification in that R is not a reflection around the initial state. Nevertheless, using a different reflection is in this case sufficient.

In addition, one can show that it is sufficient to constrain ourselves to $\text{span}\{|0\rangle U|\psi\rangle\,, |\text{bad}\rangle\}$. Success amplitude $\sin((2k+1)\theta)$ can be then be boosted arbitrarily close to 1.

So far we assumed that U is a unitary operation. It turns out that oblivious amplitude amplification is robust, meaning that it can handle operations that is only ϵ close to a unitary [34]. At the same time, W is still a unitary operator. Amplification will in this case not be performed perfectly, but the error is at most $\mathcal{O}(\epsilon)$.

2.3 Hamiltonian Evolution Simulation

This section focuses on the problem of simulating the dynamics of a quantum system. Given an initial state of a system $|\psi(0)\rangle$ and a Hamiltonian H,

our aim is to simulate the time evolution $|\psi(0)\rangle \to e^{-iHt}|\psi(0)\rangle$. The goal of Hamiltonian simulation is to design a circuit U consisting of gates and oracles that approximates the time evolution up to an error ϵ such that

$$\left\| U - e^{-iHt} \right\|_2 < \epsilon, \tag{2.25}$$

where $\|\cdot\|_2$ is the spectral norm[3]. We predominantly focus on the query complexity of Hamiltonian simulation.

In Chapter 4 we present a simulation algorithm for a time-dependent Hamiltonian which is an extension of the above-stated problem. Here we present techniques used or related to those in Chapter 4.

Quantum systems are fundamentally difficult to simulate; dynamics of quantum systems is a BQP-hard (or BQP complete for Hamiltonians with natural restrictions) [6, 28, 128]. Even though there are certain cases, such as Clifford circuits [129], when a classical simulation is possible, the complexity of simulating the evolution generally grows exponentially with the number of qubits.

As such, simulation of quantum dynamics is a field where we believe quantum computers can quickly outperform classical ones. In fact, the time evolution of quantum systems was the original application for quantum computers suggested by Feynman in his seminal paper [130] and Lloyd's algorithm [13] from 1996 was one of the early quantum algorithms with exponential speedup.

We start this section by defining the formalism and establishing simulation lower bounds. Then, we follow the chronological progress of Hamiltonian simulation research, starting by Lloyd's [13] paper and follow-up approaches based on Lie-Trotter formula, often referred to as Trotterization or product formulae.

Early Hamiltonian simulation algorithms assumed that the simulated Hamiltonians are given in the form $H = \sum_{j=1}^{m} H_j$, where each H_j is local and

[3]Spectral norm is the correct measure even for algorithms that involve measurements on ancillae. In this case, the diamond norm $\left\| U - e^{-iHt} \right\|_\diamond$ would be an appropriate measure but it can be bounded by $4 \left\| U - e^{-iHt} \right\|_2$ [127]. The proof involves writing the simulation algorithm as a channel where the desired operation (to be applied after post-selection on a measurement outcome) is one of the Kraus operators. The diamond norm is then bounded using norm inequalities and strong convexity.

$e^{-iH_j t}$ can be implemented directly for arbitrary t. The complexity of these algorithms was then generally measured by the number of exponentials needed for the simulation [29]. Nowadays, we do not take these assumptions for granted and instead only assume that the Hamiltonian is a sparse Hermitian matrix and its elements can be accessed through oracles, see Sec. 2.3.1. Non-local Hamiltonians appear in quantum chemistry [131,132] and applications outside of physics such as digital simulation of adiabatic computation [133] and systems of linear equations [5,134]. The complexity of a simulation algorithm is then measured predominantly in terms of oracle queries and then in terms of additional gates [31,33,127]. This will be the convention we follow in this thesis. Of course, there are exceptions to this rule, particularly if the authors are interested in a more specialized class of Hamiltonians.

Simultaneously with the development of Trotterization algorithms, Childs suggested a simulation algorithm based on quantum walks [126]. In 2014, a seminal paper by Berry et al. [135] achieved exponential improvement in the error scaling and was subsequently improved upon in [34,136]. In the last few years, approaches based on singular value processing [35] and qubitization [137] achieved the optimal query complexity.

Let us note that Hamiltonian simulation is often used as a subroutine of a more complex algorithm, for example in phase estimation (described in Section 2.2.2) or for solving linear equations [5,134].

This section is based on part 4.1 of *Quantum Chemistry in the Age of Quantum Computing* [1] but it has been significantly extended and altered. It aims to provide an overview of the existing results from the conception of Hamiltonian simulation to the most recent results. To the best of our knowledge, there are no current surveys covering recent techniques such as quantum signal processing and linear combinations of unitaries (LCU) methods. The older reviews of Hamiltonian simulations include Berry et al. [138] and Georgescu et al. [139].

2.3.1 The Hamiltonian Oracles

In this section, we explain how one can grant access to a Hamiltonian. The oracular representation allows us to work with Hamiltonians that are not necessarily local, for example in adiabatic computation [133] or

quantum chemistry [140]. Storing the Hamiltonian as a full matrix would not be practical because storing such a representation would necessarily require an exponential overhead. Instead, oracles often provide access to the Hamiltonian as an abstraction for computing the matrix elements as discussed in Section 2.1.2.

One type of oracular access is particularly common when the Hamiltonian (in a computational basis) is given by a sparse matrix. We say a Hamiltonian is *row-d-sparse* if each row has at most d non-zero entries. If there is an efficient procedure to locate these entries we moreover say that the Hamiltonian is *row-computable*. In this case, one can efficiently construct oracles

$$O_{loc} |r, k\rangle = |r, k \oplus l\rangle \tag{2.26}$$

$$O_{val} |r, l, z\rangle = |r, l, z \oplus H_{r,l}\rangle . \tag{2.27}$$

Oracle O_{loc} locates the position l of the k-th non-zero element in row r. The oracle O_{val} then gives the value of the matrix element $H_{r,l}$. We compute the cost of algorithms in terms of the number of queries to these oracles.

It is possible to construct different oracles. Any Hermitian matrix can be decomposed into a sum of unitaries

$$H = \sum_{l=0}^{L-1} \alpha_l H_l, \tag{2.28}$$

where for each l, $\alpha_l \leqslant 0$ and H_l is a unitary matrix $\|H_l\| = 1$. This decomposition can be efficiently implemented for sparse Hamiltonians [34]. The coefficients α_l and unitaries H_l can be accessed through oracles

$$O_\alpha |l, z\rangle = |l, z \oplus \alpha_l\rangle \tag{2.29}$$

$$O_{H_l} |l, \psi\rangle = H_l |l, \psi\rangle , \tag{2.30}$$

or, in some cases, described classically.

Berry et al. introduced the decomposition (2.28) in [135] and showed that for d-sparse Hamiltonians, L can be chosen as $L = \mathcal{O}\left(d^2 \|H\|_{max} / \gamma\right)$, where γ is the precision with which the Hamiltonian matrix entries are approximated. Furthermore, oracles (2.29) and (2.30) can be simulated by oracle queries to (2.26) and (2.27) with constant overhead.

Lastly, Low and Chuang [137] introduced a representation of a Hamiltonian through a signal oracle U such that for a signal state defined as $|G\rangle$, $H = \langle G| U |G\rangle$. In other words,

$$U |G\rangle |\psi\rangle = |G\rangle H |\psi\rangle + \sqrt{1 - |H |\psi\rangle|^2} |G_\perp\rangle , \tag{2.31}$$

where $\Big(|G\rangle\langle G| \otimes \mathbb{I} \Big) |G_\perp\rangle = 0$. This approach is based on the Hamiltonian representation in the linear combination of unitaries (LCU) approach [135]. This model is used heavily in algorithms based on quantum signal processing [36]. Low and Chuang [137] showed that the oracle (2.31) can be implemented with $\mathcal{O}(1)$ queries to (2.29) and (2.30). This results allows us to use QSP with LCU [137] instead of the relying on an earlier implementation using quantum walks [35].

Other models in quantum simulation include the fractional query model [135], representation of the Hamiltonian via a density matrix [141], or data structures for non-sparse Hamiltonians [135], but do not appear in the algorithms mentioned in this thesis.

2.3.2 Lower Bounds on Hamiltonian Simulation

Before describing any quantum algorithms, let us first establish what is the best complexity we could hope for. All but one of the lower bounds in this section are stated in the query complexity model. The first lower bound on Hamiltonian simulation is known as the no-fast forwarding theorem [29]. It establishes that no quantum computer can simulate Hamiltonian evolution for time t with complexity sublinear in $\|H\| t$.

Berry, Ahokas, Cleve and Sanders (BACS) [29] proved the no-fast-forwarding theorem by reduction to the parity problem [142, 143]. The work on the parity problem established that the parity of B bits cannot be determined with fewer than $B/2$ queries (a query outputs a single bit) with $1/4$ error. The authors [29] explicitly construct a 2-sparse Hamiltonian whose sublinear simulation would imply a parity algorithm that uses fewer than $N/2$ queries. However, there are still possible speedups for dense Hamiltonians in other parameters besides time.

There were two important lower bounds following the landmark [29] result. Childs and Kothari [144] proved the restrictions for simulating dense

Hamiltonians. They not only extended the no-fast-forwarding theorem to dense Hamiltonians, but also showed that a stricter condition applies. In particular, it is not possible to simulate a general dense $H \in \mathbb{C}^{n\times n}$ for time t with poly $(\|Ht\|, n)$ queries to a matrix entry phase oracle[4] even if given additional information.

More recently, Berry et al. [135] determined the lower bound with respect to the error of the simulation ϵ. Using the parity problem again, they showed that the lower bound in the simulation error ϵ scales as $\Omega\left(\frac{\log(1/\epsilon)}{\log\log(1/\epsilon)}\right)$. At the same time, they gave an algorithm that achieves this scaling.

The next question that needed an answer was the relationship between the scaling in time and the error. Berry et al. showed in [127] that the lower bound on the complexity is additive with respect to the time and the error. We now state this result directly.

Theorem 2 (Hamiltonian simulation lower bound, Theorem 2.2 [127]) *For any $\epsilon, t > 0$, integer $d \geqslant 2$, and a fixed value $\|H\|_{max}$, there exists a d-sparse Hamiltonian H such that simulating H for time t with precision ϵ has query complexity*

$$\Omega\left(\tau + \frac{\log(1/\epsilon)}{\log\log(1/\epsilon)}\right), \tag{2.32}$$

where $\tau = d\,\|H\|_{max}\, t$.

We know that this query complexity lower bound is, in fact, tight with quantum signal processing [137] achieving the optimal simulation complexity.

Lower bounds in terms of gate complexity are much more rare. To the best of our knowledge, the only gate complexity lower bound is due to Haah et al. [145]. The authors show that for any integers n and T such that $1 \leqslant n \leqslant T \leqslant 2^n$ there exists a piecewise constant bounded 1D Hamiltonian $H(t)$ on n qubits whose simulating for time T requires $\tilde{\Omega}(nT)$ 2-qubit gates. This result can be extended into higher dimensions [145].

[4]The matrix entry phase oracle gives the value of $H_{i,j}/\left\|H_{i,j}\right\|$ when queried on the input i, j [144].

2.3.3 Simulation Based on Trotterization

The first quantum simulation algorithms relied on the lowest-order Lie-Trotter formula

$$e^{A+B} = \lim_{r\to\infty} \left(e^{A/r} e^{B/r}\right)^r. \tag{2.33}$$

Lloyd [13] first showed how to use the Lie-Trotter formula for Hamiltonian simulation. Assuming that the Hamiltonian of interest can be decomposed into a sum of of simpler Hamiltonians[5] H_j such that $H = \sum_{j=1}^{m} H_j$, one can decompose the evolution with respect to H into the evolution with respect to each H_j as

$$\widetilde{U} = \left(e^{-iH_1 t/r} e^{-iH_2 t/r} \dots e^{-iH_m t/r}\right)^r. \tag{2.34}$$

If each $e^{-iH_j t/r}$ can be efficiently implemented, one can simulate Hamiltonian evolution using $\mathcal{O}(\text{poly}\log N)$ steps as opposed to $\mathcal{O}(\text{poly}N)$ for a classical simulation.

This approach can be extended to sparse matrices given through the oracles defined in (2.26) and (2.27). Aharonov and Ta-Shma [133] first showed that a sparse Hermitian matrix can be decomposed as $H = \sum_j H_j$, where each $e^{iH_j t}$ can be implemented directly. Then, using the Lie-Trotter formula, d-sparse Hamiltonians can be simulated with $\mathcal{O}(\text{poly}(N, d)(\tau^2/\epsilon))$ complexity.

Two types of improvements were made since the first Trotterization-based simulations. A series of works by Berry, Wiebe, and others [29,30] used more sophisticated Lie-Trotter-Suzuki decompositions [146]. These approximations allow for the approximation error to be an inverse of a polynomial of arbitrary order by devising high-order formulae that better approximate an exponential of a sum. For illustration, the symmetric Trotter formula

$$e^{(A+B)\Delta t} \approx e^{A\Delta t/2} e^{B\Delta t} e^{A\Delta t/2} \tag{2.35}$$

suppresses $\mathcal{O}(\Delta t^2)$ errors [30]. These higher-order formulae can be obtained using the recursion relation

$$S_{2k}(\delta) = [S_{2k-2}(p_k\delta)]^2 \, S_{2k-2}((1-4p_k)\delta) \, [S_{2k-2}(p_k\delta)]^2, \tag{2.36}$$

[5] For example, if H is a sum of local terms as in the Ising model.

where

$$S_2(\delta) = \prod_{j=1}^{m} e^{H_j \delta/2} \prod_{j'=m}^{1} e^{H'_j \delta/2}. \tag{2.37}$$

The number N_{exp} of exponentials of the form $e^{-iH_j t}$ for the k-th order Trotterization algorithm is then bounded by

$$N_{exp} \leqslant \frac{2m5^{2k}(m\tau)^{1+1/2k}}{\epsilon^{1/2k}}, \tag{2.38}$$

for $\epsilon < 2mr^{k-1}$ and $\tau = \|H\| t$ [29].

While Trotterization-based algorithms have poor scaling in error compared to newer techniques, they perform exceptionally well on real-world instances. In a resource study conducted by Childs et al. [147], high-order product formulae [30] showed promising T-gate scaling.

Trotterization-based methods can be further improved by sampling the exponentials at random instead of fixing their order ahead of time. The idea of using randomization to decrease the error in Hamiltonian simulation was first proposed by Childs et al. [148] and subsequently improved by Cambell [149]. Cambell's algorithm, known as the randomized compiling or QDRIFT protocol, works particularly well for shorter evolution times and Hamiltonians of the form $H = \sum_{j=1}^{m} h_j H_j$ where each $\left\|H_j\right\| = 1$, exponentials of the form $e^{-iH_j t}$ can be implemented directly, and h_j vary significantly for different values of j. QDRIFT draws the exponential terms with probabilities proportional to the interaction strengths h_j. This leads to an algorithm dependent on $\sum_{j=1}^{m} \left|h_j\right|$ instead of $m\left(\max_j \left|h_j\right|\right)$.

2.3.4 Sparse Matrix Decomposition

We require that the Hamiltonian can be decomposed into terms that can be efficiently simulated. In addition, this decomposition must be efficiently computable and it is vital that the number of terms in the decomposition be as small as possible. Aharonov and Ta-Shma [133] devised such a decomposition for sparse Hamiltonians.

These decompositions rely on graph coloring techniques. Any Hamiltonian can be assigned a connectivity graph with N vertices and the following

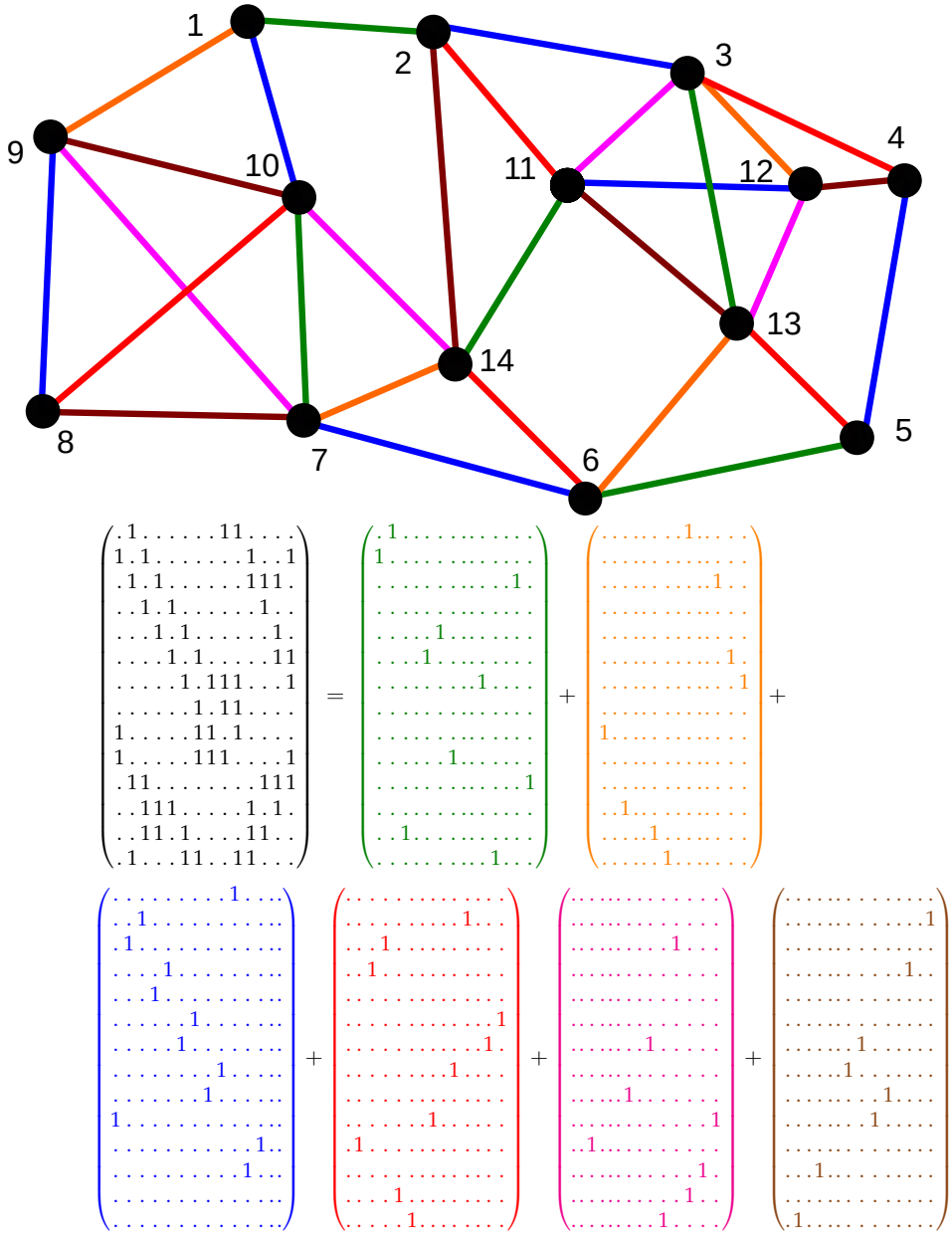


Figure 2.8: An example of Hamiltonian decomposition based on edge coloring. The entire graph corresponds the full, black, matrix. The colored matrices then represent edges of a given color. Note that each colored matrix is only 1-sparse.

property: vertices x and y are connected by an edge if and only if the entry $H_{x,y}$ is nonzero. Since H is Hermitian, this graph is undirected and the sparsity d represents the maximum degree of this graph. The Hamiltonian decomposition is determined from an edge coloring of this graph. This means that each edge is assigned a color (label) such that incident edges (on the same vertex) always have different colors. Edges of a single color in the graph then correspond to a single term in the Hamiltonian decomposition and the number of colors used is the number of terms in the Hamiltonian decomposition as in Fig. 2.8. Each 1-sparse Hamiltonian is then simulated directly using its equivalence to a block diagonal matrix [29, 133].

The decomposition was later improved in a number of papers [29, 135, 150, 151]. The state-of-the-art decomposition is summarized in the theorem below:

Theorem 3 (d-sparse Hamiltonian decomposition [135])
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 $\mathcal{O}(1)$ queries to H.

The new ingredient in the construction [135] is to modify the Hamiltonian to make the graph bipartite[6]. These decompositions are used in nearly all simulations of sparse Hamiltonians including our algorithm in Chapter 4.

2.3.5 The Quantum Walk Approach

A surprising Hamiltonian simulation technique comes from the equivalence between discrete and continuous-time quantum walks [27]. In this section, we describe the quantum walk-based algorithm from [27] and the subsequent work [31]. Both approaches give a quantum analog of a stochastic process on a graph. In the classical case, the transition matrix is the adjacency matrix A of a graph and is required to be stochastic[7]. Every random walk can be modified into its lazy version by changing the transition rule $p \to Ap$ into $p \to (1-\epsilon)p + \epsilon Ap$, where p refers to the probability

[6] A graph is said to be bipartite if the vertices can be split into two distinct sets such that no two vertices within the same set are connected by an edge.

[7] A stochastic matrix must be square and matrix with real, non-negative entries. In our notation, the sum of matrix entries for each column must be equal to 1. This condition ensures that the total probability is preserved.

distribution on the graph and $\epsilon \in (0,1)$. Taking the limit $\epsilon \to 0$ and an infinitesimally short length of each step, one can obtain the continuous time limit of a random walk

$$p(t) = e^{At} p_0, \tag{2.39}$$

which can be interpreted as the diffusion equation.

A continuous-time quantum walk [152] is obtained by replacing (2.39) by the Schrödinger equation

$$|\psi(t)\rangle = e^{-iAt} |\psi_0\rangle , \tag{2.40}$$

with the adjacency matrix as the Hamiltonian. The Hamiltonian is Hermitian if the graph is undirected.

Discrete-time quantum walks take a very different approach. The construction by Szegedy [15] translates any Markov chain on a graph to a quantum process on a quadratically larger Hilbert space. Discrete-time quantum walks are typically defined on symmetric graphs but the construction can be generalized to any $N \times N$ Hamiltonian H. Using the notation from [27], define

$$\left|\psi_j\right\rangle = \frac{1}{\sqrt{\|\mathrm{abs}(H)\|}} \sum_{k=1}^{N} \sqrt{H^*_{j,k} \frac{d_k}{d_j}} \, |j,k\rangle \tag{2.41}$$

for every pair of graph vertices j, k where $\mathrm{abs}(H)$ stands for entry-wise absolute value of H and $|d\rangle = \sum_j d_j |j\rangle$ is the principal eigenvector of $\mathrm{abs}(H)$. This construction allows us to make the quantum walk unitary. A state $|k,j\rangle$ can be assigned to an edge between k and j; j being the vertex where the walker starts in a particular step and k the vertex the walker is moving towards. Let S be the swap operation of these registers

$$S\,|j,k\rangle = |k,j\rangle . \tag{2.42}$$

Next, define an isometry T mapping the states of the graph onto the states of enlarged Hilbert space

$$T = \sum_{j=1}^{N} \left|\psi_j\right\rangle \langle j| . \tag{2.43}$$

One step of the walk can be implemented using $iS(2TT^\dagger - \mathbb{I})$; see [15]. Taking the eigenvalues of the Hamiltonian $\frac{H}{\|abs(H)\|}|\lambda\rangle = \lambda|\lambda\rangle$, it can be shown [153] that the eigenvectors of $U = iS(2TT^\dagger - \mathbb{I})$ are

$$|\mu_\pm\rangle = \frac{1 - e^{\pm i \arccos\lambda} S}{\sqrt{2(1-\lambda^2)}} T|\lambda\rangle \tag{2.44}$$

and corresponding eigenvalues $\mu_\pm = \pm e^{\pm i \arcsin\lambda}$ [27].

Discrete-time quantum walks are already formulated in terms of query complexity. The framework of quantum walks [21], can be applied to a variety of classical randomized algorithms and gives a number of provable speedups [15, 21, 154–156].

The relationship between discrete and continuous random walks translates to a correspondence between discrete and continuous time quantum walks. Childs quantified this correspondence devising an algorithm for simulating continuous-time walks using discrete time quantum walks. Since discrete-time quantum walks can be easily translated into the gate model [157, 158], this result de facto gave a new algorithm for simulating Hamiltonian evolution [31].

Let us take $|\Theta\rangle$ to be an eigenstate of a discrete time quantum walk with an eigenvalue $e^{-i\Theta}$. Furthermore, define P to be an operation that performs phase estimation on the walk and records the outcome in an additional register

$$P|\Theta\rangle = \sum_\phi a_{\phi|\Theta}|\Theta,\phi\rangle\,. \tag{2.45}$$

Next, define a unitary F_{tsin} that computes the sine function and applies an appropriate phase

$$F_{tsin}|\Theta,\phi\rangle = e^{-it\sin\phi}|\Theta,\phi\rangle\,. \tag{2.46}$$

Childs showed in [27] that Hamiltonian evolution can be closely approximated using discrete-time quantum walk techniques

$$\langle\psi|e^{iHt}T^\dagger P^\dagger F_{tsin} P T|\psi\rangle \geqslant 1 - \frac{93t^2}{M^2}, \tag{2.47}$$

where M denotes the number of calls to quantum walk unitary U. Equation (2.47) shows that the fidelity between states produced by the quantum walk algorithm and time evolution is high. Taking $M = \mathcal{O}\left(\frac{t}{\sqrt{\delta}}\right)$, Hamiltonian evolution for time t can be simulated using $\mathcal{O}\left(\left\|\text{abs}(H)t/\sqrt{\delta}\right\|\right)$ steps of a discrete-time quantum walk.

Reference [27] just considered the complexity in terms of the steps of this discrete quantum walk, without showing how to implement them. Berry and Childs [31] provided methods to implement the discrete quantum walk in terms of the oracles for matrix entries of the Hamiltonian (2.26), (2.27), thus providing a complexity that could be compared to the prior work based on Trotterization. Quantum-walk simulation techniques have query complexity linear in simulation time compared to superlinear complexity for Trotterization methods [29].

2.3.6 Hamiltonian Simulation with Linear Combination of Unitaries

We now present a Hamiltonian simulation algorithm [34] that achieves a logarithmic scaling in the inverse error. This algorithm requires the Hamiltonian to be either $\sqrt{L}$ sparse and row-computable or a sum of at most L Pauli operators. In these two cases, the Hamiltonian can be decomposed into a linear combination of unitaries H_l, as

$$H = \sum_{l=0}^{L} \alpha_l H_l. \tag{2.48}$$

This decomposition together with oblivious amplitude amplification is responsible for the speedup compared with previous techniques.

The goal of the algorithm is to implement the unitary

$$U(t) = e^{-iHt} = \sum_{k=0}^{\infty} \frac{(-iHt)^k}{k!} \tag{2.49}$$

given the time t and access to the unitary decomposition of the Hamiltonian H (2.48).

First, we divide the evolution into r segments, each of them t/r long. Each segment is then approximated by a Taylor series truncated at the K-th order

$$\sum_{k=0}^{K} \frac{(-iHt/r)^k}{k!}. \tag{2.50}$$

To limit the error to $\mathcal{O}(\epsilon)$ by the end of evolution, each segment must have error at most $\mathcal{O}(\epsilon/r)$, giving the bound on $K = \mathcal{O}\left(\frac{\log(r/\epsilon)}{\log\log(r/\epsilon)}\right)$ provided that $\|H\|\, t/r < 1$. Note that the operator coming from this approximation is no longer a unitary.

Plugging the decomposition (2.48) into (2.50), we obtain

$$\tilde{U} = \sum_{k=0}^{K} \frac{(-it/r)^k}{k!} \sum_{l_1} \sum_{l_2} \cdots \sum_{l_k} \alpha_{l_1} \alpha_{l_2} \ldots \alpha_{l_k} H_{l_1} H_{l_2} \ldots H_{l_k}, \tag{2.51}$$

where $\tilde{U}$ is an approximation of U and no longer a unitary operation. In (2.51), we expressed $\tilde{U}$ as a sum of unitaries. Each unitary here is a product of $V_{l_1} V_{l_2} \ldots V_{l_k}$ and the coefficients depend on the coefficients in Taylor series and unitary decomposition. Without loss of generality, Eq. (2.51) can be written in the form $\tilde{U} = \sum_{j=0}^{m-1} \beta_j V_j$ where all $\beta_j > 0$ and unitaries V_j are products of Hamiltonian terms H_l and $(-i)$ factors.

The algorithm frist prepares the superposition state $\frac{1}{\sqrt{s}} \sum_{j=0}^{m-1} \sqrt{\beta_j}\, |j\rangle$, where $s = \sum_{j=0}^{m-1} \beta_j$, through a circuit denoted B. Implementation of B can be complicated (see [159]) but it does not require any queries to the Hamiltonians.

Next, we need to able to implement each V_j controlled by the value j. Define

$$\text{select}(V)\, |j\rangle\, |\psi\rangle = |j\rangle\, V_j\, |\psi\rangle\,, \tag{2.52}$$

for each j. Since each V_j consists of a product of at most K terms in the Hamiltonian decomposition, the complexity of implementing $\text{select}(V)$ is K times the complexity of applying H_l.

Let us now introduce

$$W = \left(B^\dagger \otimes \mathbb{I}\right) \text{select}(V)\, (B \otimes \mathbb{I})\,. \tag{2.53}$$

Inductively, W applies the $\widetilde{U}$ if the ancilla is in the correct state (in this case it is $|0\rangle$) and prepares some garbage state if that ancilla is not zero. Formally, $(|0\rangle\langle 0| \otimes \mathbb{I})\, W\, |\psi\rangle\, |0\ldots 0\rangle = \frac{1}{s}\, |0\rangle\, \widetilde{U}\, |\psi\rangle$. If s is close to 2, we can use W to implement oblivious amplitude amplification presented in Sec. 2.2.4. Applying $-WRW^{\dagger}RW$ where $R = \mathbb{I} - (|0\rangle\langle 0| \otimes \mathbb{I})$ results in the state $|0\rangle\, \widetilde{U}\, |\psi\rangle$ with high accuracy [34].

All together, the complexity is

$$\mathcal{O}\left(\tau \frac{\log(\tau/\epsilon)}{\log\log(\tau/\epsilon)}\right), \tag{2.54}$$

where $\tau = L\, \|H\|_{\max}\, t$ for $\sqrt{L}$-sparse matrices. We use similar steps in our algorithm for simulating time-dependent Hamiltonians in Chapter 4 and derive rigorous bounds for our work.

2.3.7 Quantum Signal Processing and Qubitization

Hamiltonian simulation through quantum signal processing started by Low et al. [160] asking a seemingly unrelated question: What single qubit unitaries can we implement given Z-rotations with a fixed angle ϕ and arbitrary X rotations? Surprisingly, this class of operations can be fully classified [36, 160]. We give a simplified question these conditions at the end of this section. Specifically, all circuits with up to k gates can be characterized by four polynomials that satisfy the conditions of Theorem 1 in [160]. Even more surprisingly, given a function of ϕ that meets these conditions, one can construct a circuit approximating this function following the construction in [160].

The above procedure can be extended to a multi-qubit case through the phase kickback explained in Fig. 2.9. This algorithm, named quantum signal processing, describes the transformation of a unitary $W = \sum_{\lambda} e^{i\phi_m}\, |u_m\rangle\langle u_m|$ to

$$W \to V_{\text{ideal}} = \sum_{m} e^{ih(\phi_m)}\, |u_m\rangle\langle u_m| \tag{2.55}$$

for any real function h. The first step of quantum signal processing is the transduction of the phase ϕ_m to the control qubit via phase kickback (see

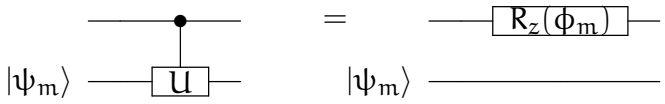


Figure 2.9: Application of a controlled unitary acting on an eigenstate acts as a rotation on the control qubit. Let $|\psi_m\rangle$ be an eigenstate of U associated with eigenvalue $e^{i\phi_m}$. Applying the circuit above on an initial state $(\alpha|0\rangle + \beta|1\rangle)|\psi_m\rangle$ results in $\left(\alpha|0\rangle + e^{i\phi_m}\beta|1\rangle\right)|\psi_m\rangle$. This is equivalent to applying $R_z(\phi_m) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi_m} \end{pmatrix}$ on the first qubit.

Fig. 2.9). Then, single qubit rotations are used to synthesize the desired function as in the above control problem.

Quantum signal processing can be used together with a quantum walk oracle [27, 31]. Recall that the quantum walk operator has eigenvalues $\pm e^{\pm i \arcsin \phi_m}$ where ϕ_m is an eigenvalue of the Hamiltonian. Using the function $h(\theta) = -\tau \sin(\theta)$ gives the proper time evolution. This function can be approximated by the Jacobi-Anger expansion in a manner similar to [31]. Importantly, Jacobi-Anger results in an achievable transformation (satisfying the criteria in [160]) and converges exponentially fast which gives logarithmic scaling in error. In their next work, Low and Chuang [137] presented qubitization. Qubitization allows one to use quantum signal processing beyond the limitations of the signal oracle. Given access to the Hamiltonian as $H = \langle G|U|G\rangle$ (see Subsection 2.3.1), one can construct a "qubiterate" to compute powers of H [137]. The name qubitization refers to decomposition into 2-dimensional invariant subspaces isomorphic to qubits. Qubitization allows one to combine quantum signal processing with LCU methods and other techniques.

Gilyen et al. [36] later simplified and generalized these methods to a technique called quantum singular value transformation (QSVT). We now state their version of quantum signal processing:

Theorem 4 (Quantum Signal Processing [36])
Let $k \in \mathbb{N}$*; there exists* $\Theta = \theta_0, \theta_1, \ldots, \theta_k \in \mathbb{R}^{k+1}$ *such that for all* $x \in [-1, 1]$*:*

$$e^{i\theta_0\sigma_z} \prod_{j=1}^{k} \left(W(x) e^{i\theta_j\sigma_z} \right) = \begin{bmatrix} P(x) & iQ(x)\sqrt{1-x^2} \\ iQ(x)^*\sqrt{1-x^2} & P^*(x) \end{bmatrix}$$

where

$$W(x) = \begin{bmatrix} x & i\sqrt{1-x^2} \\ i\sqrt{1-x^2} & x \end{bmatrix} = e^{i \arccos(x)\sigma_z}$$

if and only if $P, Q \in \mathbb{C}[x]$ *such that*

- $\deg(P) \leqslant k$ *and* $\deg(Q) \leqslant k-1$
- *P has parity (*$k \mod 2$*) and Q has parity (*$k-1 \mod 2$*)*
- $\forall x \in [-1, 1] : |P(x)|^2 + (1-x^2)|Q(x)|^2 = 1$

Simply put, Theorem 4 (Theorem 3 in [36]) states the conditions on the polynomials that can be implemented through quantum signal processing circuits. The proof of this theorem is constructive, giving an algorithm for computing the angles from polynomials. In addition, if P or Q are not known entirely, they can be completed following the proofs of Theorems 3-5 of [36].

2.3.8 Generalization of the Hamiltonian Simulation Problem

In the last subsection of the Background, we discuss several lines of research closely related to Hamiltonian simulation but extend it in various ways.

The first generalization is using a Hamiltonian that depends on time. Techniques for simulating time-dependent Hamiltonians based on the Lie-Trotter-Suzuki decomposition were developed in [161–163], but the complexity scales polynomially with the error. Poulin et al. [163] gave an algorithm whose quantum complexity does not depend on the derivatives of the Hamiltonian because of their use of Monte Carlo integration.

More recent advances providing complexity logarithmic in the error [34,135] mention that their techniques can be generalized to time-dependent scenarios, but do not analyze this case. The most recent algorithms [35,137] are not directly applicable to the time-dependent case. Here we present an explicit algorithm for simulating time-dependent Hamiltonians with complexity logarithmic in the allowable error, matching the complexity of the algorithms for the time-independent case in [34,135], though not those in [35,137]. Low and Wiebe [37] independently achieved similar results.

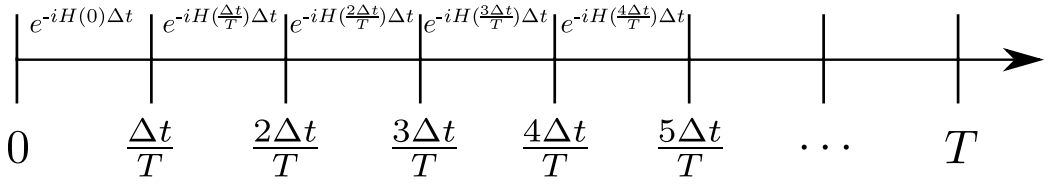


Figure 2.10: The continuous-time evolution is discretized into short timesteps. The state in each step is then evolved with a time-independent Hamiltonian.

Another avenue of research is the simulation of open quantum systems. An open quantum system can always be embedded into a closed one as a consequence of Stinespring dilation [164]. As such, Hamiltonian simulation techniques can also be used for the evolution of an open quantum system. However, extending a quantum system according to Stinespring dilation is prohibitively costly for practical simulations.

If the open system is Markovian, it is possible to simulate the Linbland Master equation directly. Kliesch et al. [165] laid the basis for simulation of Markovian systems by showing that it can be simulated with complexity polynomial in the evolution time and polylogarithmic in the dimension if the Lindbladian is k-local (see [165] for a formal definition) using Trotterization. Childs and Li [166] extended this algorithm to sparse Lindbladians. Recently, Cleve and Wang proposed an algorithm logarithmic in inverse error [167], extending LCU techniques to Markovian dynamics.

3

Quantum Sort and Shuffle

"The order that our mind imagines is like a net, or like a ladder, built to attain something. But afterward you must throw the ladder away, because you discover that, even if it was useful, it was meaningless."

Umberto Eco, The Name of the Rose

Randomness is an essential resource for many classical algorithms used everywhere from casinos to traffic route optimization. An example of a common randomized subroutine is a shuffle. Given objects marked 1 to k, the shuffle produces a sequence of these objects uniformly at random.

In this section, we present a quantum version of the above task: prepare an equal superposition of all permutations of k objects efficiently. Creating a superposition is a natural extension of a shuffle because measurement in the computational basis would reproduce an outcome of a classical shuffle. Note that we are strictly concerned by permutations without repetitions.

We present two different approaches to a quantum shuffle. The first one is a quantization of a classical algorithm, the Fisher-Yates (FY) shuffle. The

second one approaches shuffle as the opposite of a sort. Indeed, if the original list of items is sorted, a sort can be seen as reverting the application of a shuffle. We extend classical sorting algorithms to the quantum framework by providing a reversible version with no asymptotic overhead.

Clearly, neither sort nor shuffle is reversible. We define shuffle as an operation that takes a list in increasing order and outputs a symmetric superposition. A sort is then the inverse operation to the shuffle in the sense that a uniform superposition over possible permutations can be sorted in increasing order.

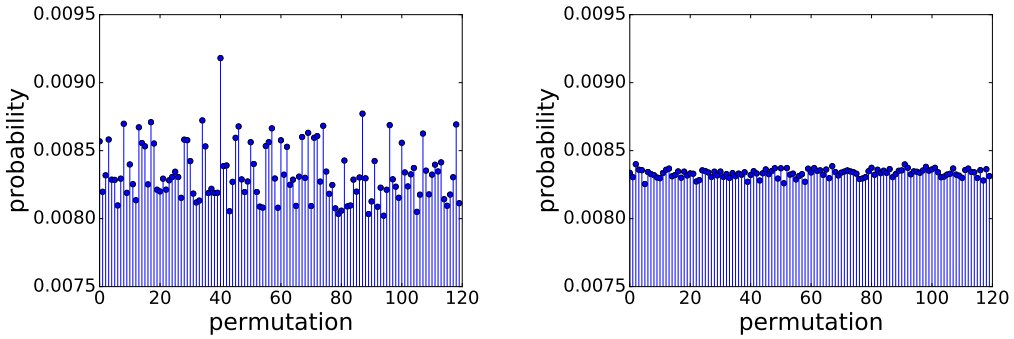
This chapter is based on the paper *Improved techniques for preparing eigenstates of fermionic Hamiltonians* [2] which includes sort and shuffle as subroutines. I participated in the discussion, contributed with ideas, and was involved in the writing process. Since I was primarily engaged in improving antisymmetrization procedure utilizing a quantum sort or a shuffle, I decided to focus this chapter only on these techniques. My main contributions were optimizing the algorithm for logarithmic depth and decomposing the components into elementary gates. Section 3.5 explains how one can use a sort or a shuffle in quantum chemistry as well as additional applications of our work. For instance, the algorithm in Chapter 4 uses the quantum sort as a subroutine.

3.1 How to Shuffle a Deck of Cards

Consider the following problem. We are given m cards, for simplicity labeled from 1 to m. We want to shuffle them, i.e., output any ordering of the cards with equal probability. What random swaps should we perform to shuffle the cards efficiently?

Intuitively, we could pick random pairs of cards and swap them, but we would need to perform much more than n swaps to guarantee a genuinely random permutation as can be seen in Fig. 3.1. Another strategy would be to pick the first card deterministically and exchange its place with a random card. Unfortunately, a naive approach based on this strategy does not produce uniform probabilities for all permutations.

A successful algorithm that requires m swaps ensures that each card is deterministically chosen only once. Note that a card can be swapped with



(a) 10 random swaps

(b) 1000 random swaps

Figure 3.1: Shuffling by performing random swaps is not very effective. Here we attempt to shuffle 5 cards by picking two cards at random and swapping them. The procedure is repeated 10 times in (a) and 1000 times in (b). Then the statistics is collected from 10^7 repetitions of the shuffle. In an ideal shuffle, all sequences should be equally probable, and the distribution in Figures (a) and (b) should be close to flat (up to a sampling error).

itself, i.e., it does not have to move. This algorithm is called a Fisher-Yates (FY) shuffle [168] or sometimes Knuth's shuffle [169]. There are several variants of FY shuffle; we found the following version to be most suitable for quantization:

Fisher-Yates shuffle

for $k = 1, \ldots, (m-1)$ **do**
  Choose ℓ uniformly at random from $\{0, \ldots, k\}$.
  Swap items in positions k and ℓ.
end for

In other words, in the k-th run, the card on k-th position (indexing from 0) is swapped with any of the previous cards. All the possible scenarios are illustrated as a tree in Figure 3.2 for $m = 4$. Assuming that all swaps are random, the FY shuffle produces a random sequence optimally.

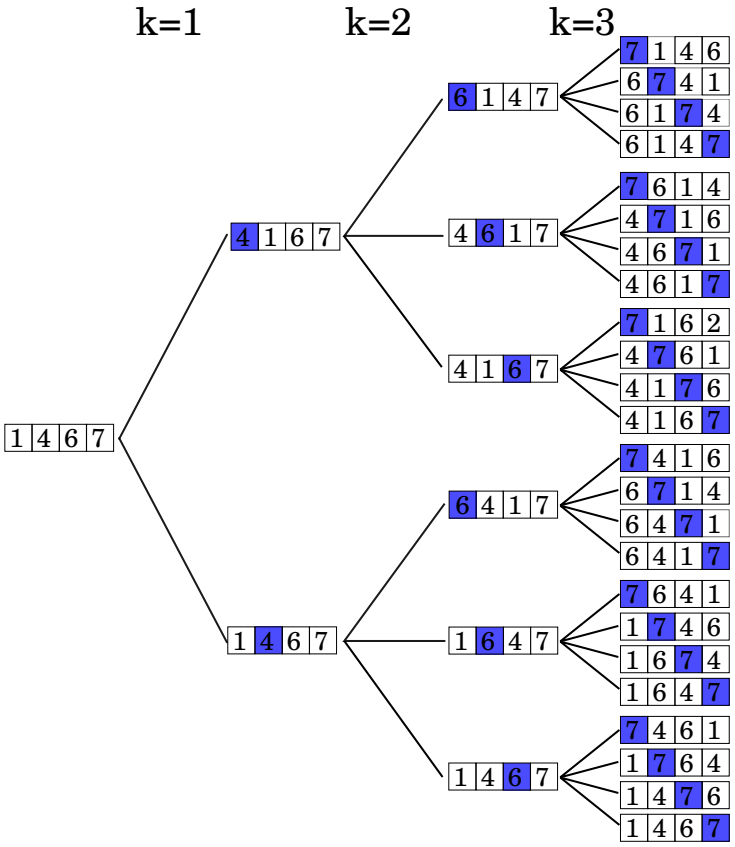


Figure 3.2: A tree diagram for the FY shuffle demonstrates all the moves that the algorithm can make. The blue boxes identify the array entries that have been swapped at each stage of the shuffle. In the first step, the item "2" is moved, in the next one item "3" and item "4" is moved last. Note that the blue boxes also label the largest value in the array truncated to position k.

3.2 Quantum Approach to a Shuffle

The FY algorithm presented above is not a unitary transformation and as such, cannot be directly implemented on a quantum computer. Instead, we propose an algorithm that prepares a uniform superposition of all sequences, analogously to a classical shuffle.

Here we present our algorithm that implements a shuffle in superposi-

tion, i.e. performs the transformation

$$|r_1 \cdots r_m\rangle \mapsto \sum_{\sigma \in S_m} |\sigma(r_1, \cdots, r_m)\rangle, \tag{3.1}$$

where σ iterates over possible permutations. Note that the choice of the state is not unique; adding arbitrary phases $e^{i\phi_\sigma}$[1] to the states $|\sigma(r_1, \cdots, r_m)\rangle$ would still fall within the definition of a quantum shuffle. In particular, the transformation needed in [2] for fermionic state preparation was

$$|r_1 \cdots r_m\rangle \mapsto \sum_{\sigma \in S_m} (-1)^{\pi(\sigma)} |\sigma(r_1, \cdots, r_m)\rangle \tag{3.2}$$

where $\pi(\sigma)$ is the parity of the permutation σ, and we require for the initial state that $r_p < r_{p+1}$ (necessary for this procedure to be unitary). This means that we are solving a restricted version of the problem where the input is always sorted in increasing order.

Although we describe the procedure for a single input $|r_1 \cdots r_m\rangle$, our algorithm may be applied to any superposition of such states. We assume that there are no repetitions among the shuffled items (this assumption holds for fermions in the application of the shuffle in [2]) and the array starts ordered.

We first present a quantum extension of FY shuffle. Unfortunately, its complexity is worse than that of its classical counterpart. Then we show an alternative algorithm utilizing sorting. As a part of the algorithm, we develop a quantum approach to sorting based on sorting networks and synthesize all the components down to elementary gates. This work presents an exponential improvement in depth compared to the prior state-of-art [170].

3.3 Preparing a Uniform Superposition With the Quantum FY Shuffle

In this Section, we present a quantum version of FY shuffle that gives an intuitive approach to antisymmetrization. Assuming that we need to

[1] Of course, some functions ϕ_σ can be difficult to implement

shuffle m numbers between 0 and $N-1$, the size- and depth-complexity are $\mathcal{O}(m^2 \log N)$.

Two key steps turn the FY shuffle into a quantum algorithm. First, our quantum implementation of the shuffle replaces the random selection of swaps with a superposition of all possible exchanges. Unfortunately, this step presents an overhead compared to the classical FY shuffle. To achieve this superposition, the random variable is replaced by an equal-weight superposition $\frac{1}{\sqrt{k+1}} \sum_{\ell=0}^{k} |\ell\rangle$ in an ancillary register (called `choice`).

Second, we must reset the `choice` register at the end of each step. To accomplish this, we introduce an additional `index` register, which initially contains the integers $0, \ldots, m-1$. We shuffle both the length-m `input` register and the `index` register, and the simple form of `index` enables us to easily reset `choice`. The resulting state of the joint `input` $\otimes$ `index` register is still highly entangled; however, provided `input` was initially sorted in ascending order, we can disentangle `index` from `input`.

As we explained in our paper, the quantum FY shuffle consists 6 steps. The high level overview of FY shuffle is illustrated in Fig. 3.3, and Fig. 3.4 provides additional detail. We now briefly review these steps and then explain each in detail.

1. **Initialization.** Prepare the `choice` register in the state $|0\rangle$. Prepare the `index` register in the state $|0, 1, \ldots, m-1\rangle$. Also set a classical variable $k = 1$.

2. **Prepare** `choice`**.** Transform the `choice` register as

$$|0\rangle \to \frac{1}{\sqrt{k+1}} \sum_{\ell=0}^{k} |\ell\rangle \,. \tag{3.3}$$

3. **Execute swap.** Swap element k of `input` with the element specified by `choice`. Also swap element k of `index` with the element specified by `choice`. If preparing an antisymmetric superposition, apply a -1 phase to the `input` register if a non-trivial swap was executed (i.e. if `choice` does not encode k).

4. **Reset** `choice`**.** For each $\ell = 1, \ldots, k$, subtract ℓ from the `choice` register if position ℓ in `index` is equal to k. The resulting state of `choice` is $|0\rangle$.

5. **Repeat.** Increment k by one. If $k < m$, go to Step 2. Otherwise, proceed to the next step.

6. **Disentangle** `index` **from** `input`**.** For each $k \neq \ell = 0, 1, \ldots, m-1$, subtract 1 from position ℓ of `index` if the element at position k in `input` is greater than the element at position ℓ in `input`. The resulting state of `index` is $|0, 0, \ldots, 0\rangle$, which is disentangled from `input`.

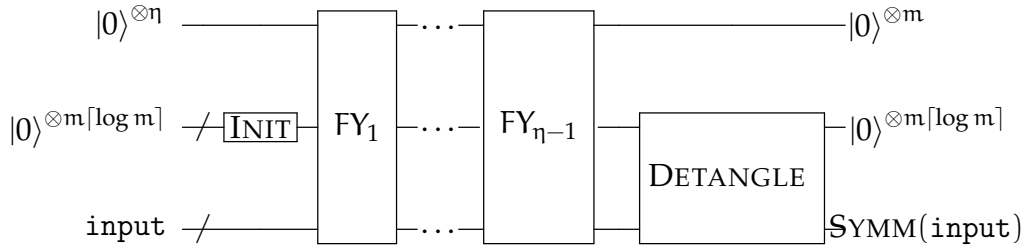


Figure 3.3: High-level view of the FY shuffle as it appeared in [2]. The procedure acts on registers labeled (top to bottom) `choice`, `index` and `input`. The first step block represents **Initialization** and the FY_k blocks consist of steps **Prepare** `choice`, **Execute swap** and **Reset** `choice`, see Fig. 3.4, for k between 1 and $m-1$. The last block aims to **Disentangle** `index` **from** `input`.

After the DETANGLE procedure, the ancillary registers `choice` and `index` are reset to their initial all-zero states and the `input` register has been symmetrized. The structure of FY blocks is depicted in Fig. 3.4. Each block starts with the preparation operator Π_k to `choice`, then we apply selected swaps between `choice` and `index` and between `choice` and `input`. We conclude each block by applying a phase conditioned on `choice` to `input`, and resetting the `choice` register. These procedures are applied a total of $m-1$ times (for each of $k = 1, \ldots, m-1$). Their gate counts and circuits depths are then multiplied by $(m-1)$. Disentangling `index` and `input` is the most expensive component, but it is executed only once, so it contributes only an additive cost to the overall resource requirement.

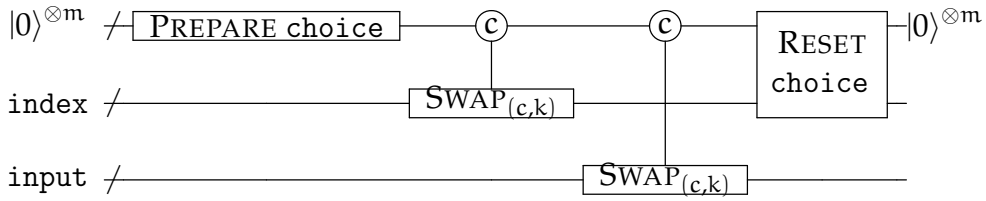


Figure 3.4: Detail for the FY block FY_k from [2]. The first register (labeled `choice`) is used to select the target of the two selected swap steps. Each block FY_k is completed by resetting the `choice` register back to its original state $|0\rangle^{\otimes m}$, i.e. steps steps **Prepare** `choice`, **Execute swap** and **Reset** `choice`.

We now explain each step of the algorithm and justify their corresponding resource contributions. We briefly summarize them here. Step 1 requires $\mathcal{O}(m \log m)$ gates but has a negligible depth $\mathcal{O}(1)$. Step 2 requires $\mathcal{O}(m)$ gates and has depth complexity $\mathcal{O}(m)$. Step 3 requires $\mathcal{O}(m \log N)$ gates and also has depth $\mathcal{O}(m \log N)$. Step 4 requires $\mathcal{O}(m \log m)$ gates but has depth only $\mathcal{O}(\log m)$. As Step 2 to Step 4 are repeated $m-1$ times, the total gate count before Step 6 is $\mathcal{O}(m^2 \log N)$. Finally, Step 6 requires $\mathcal{O}(m^2 \log N)$ gates and has depth $\mathcal{O}\left(m^2 [\log \log N + \log m]\right)$. Thus the total gate count of the quantum FY shuffle is $\mathcal{O}(m^2 \log N)$. Because most of the gates need to be performed sequentially, the overall circuit depth of the algorithm is also $\mathcal{O}(m^2 \log N)$.

3.3.1 Initialization

We assume that the register `choice` is initialized in the state $|0\rangle^{\otimes m}$. The `index` register is set to the state $|0, 1, \ldots, m-1\rangle$ that represents the positions of the entries of `input` in ascending order. Because each of the m entries in `index` must be capable of storing any of the values $0, 1, \ldots, m-1$, the size of `index` is $m\lceil \log m \rceil$ qubits. This step requires $O(m \log m)$ single-qubit gates that can be applied in parallel with circuit depth $\mathcal{O}(1)$ if m is a power of 2. Otherwise, we can create an arbitrary uniform superposition using a method from from [73].

3.3.2 FY Blocks

Each FY block consists three stages: prepare `choice`, executed selected swaps, and reset `choice`. The exact steps depend on the encoding of the `choice` register; in particular, whether it is binary or unary.

We use m qubits (labelled $0, 1, \ldots, m$), define

$$|\texttt{null}\rangle = |0\rangle^{\otimes m} \tag{3.4}$$

and encode

$$|\ell\rangle = X_\ell \, |\texttt{null}\rangle \,, \tag{3.5}$$

where X_ℓ is the Pauli X applied to the qubit labelled ℓ. $|\ell\rangle$ is simply the unary encoding of the value ℓ.

An advantage of our encoding for `choice` is that the selected swaps require only single-qubit controls. A distinct disadvantage is an unnecessary space overhead. While binary encoding would save some space, it would also increase the time cost.

Prepare `choice`

Our preparation procedure has two stages. First, we prepare an alternative unary encoding on k qubits $S_k = \frac{1}{\sqrt{k}} (|10\ldots0\rangle + |110\ldots0\rangle + \cdots + |1111\ldots1\rangle)$. Formally we can define S_k recursively

$$\begin{aligned} S_1 &= |1\rangle \\ S_k &= \frac{\sqrt{k-1}}{\sqrt{k}} S_{k-1} \otimes |0\rangle + \frac{1}{\sqrt{k}} |1\rangle^{\otimes k} \,. \end{aligned} \tag{3.6}$$

We can prepare $|S_k\rangle$ in this encoding with a cascade of controlled rotations of the form

$$R_j := \frac{1}{\sqrt{j+1}} \begin{pmatrix} 1 & -\sqrt{j} \\ \sqrt{j} & 1 \end{pmatrix}, \tag{3.7}$$

see Fig. 3.5. This is a total of $k+1$ gates, all of them applied sequentially.

Second, we need to translate S_k into the desired unary encoding. Each $|11\ldots100\ldots0\rangle$ must be turned into $|00\ldots010\ldots0\rangle$. This can be accomplished with a series of CNOTs 3.5. Thus the total gate count and depth for preparing `choice` are $\mathcal{O}(k) = \mathcal{O}(m)$.

For the circuit for `choice` preparation, please see Figure 3.5.

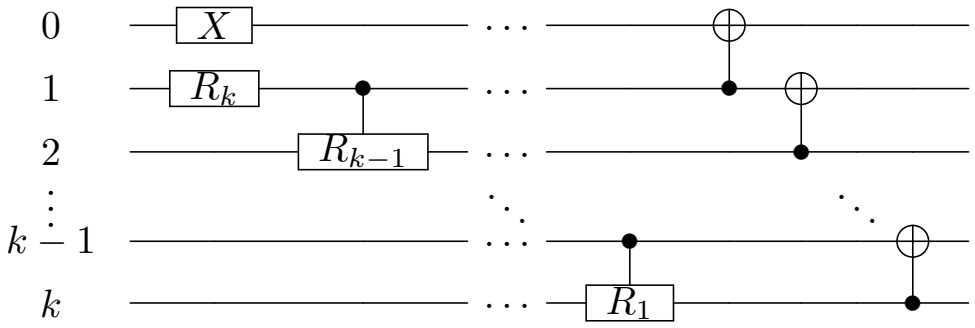


Figure 3.5: Circuit for preparing the `choice` register at the beginning of block FY_k from [2]. See Eq. (3.7) for the definition of R_ℓ. First we prepare S_k state (3.6) using a cascade of controlled rotations. Then we turn it into a unary encoding through a series of CNOTs.

Selected Swap

The next step is to implement the selected swap operation:

$$\textsc{SelSwap}_k := \sum_{c=0}^{k-1} |c\rangle\langle c| \otimes \textsc{Swap}(c, k), \tag{3.8}$$

where the swapped states are either in the `index` or the `input` registers and controlled by the `choice` register. The unary encoding of c the `choice` register allows for a simple implementation of SELSWAP; see Fig. 3.6.

Observe that only the first $k+1$ subregisters are involved of each `choice`, `index` and `input`. For each $i = 0, 1, \ldots, k$, `index`$[i]$ is of size $\lceil \log m \rceil$ whereas `input`$[i]$ is of size $\lceil \log N \rceil$. Hence, the circuit consists of $k\lceil \log m \rceil + k\lceil \log N \rceil$ ordinary 3-qubit controlled-SWAP gates that for the most part must be executed sequentially, see Fig. 3.6. This is the operation responsible for the overhead compared to a classical swap. While the randomized FY algorithm requires only one (random) swap of registers, the coherent $\textsc{SelSwap}_k$ has complexity $O(m \log N)$ for both gate count and depth.

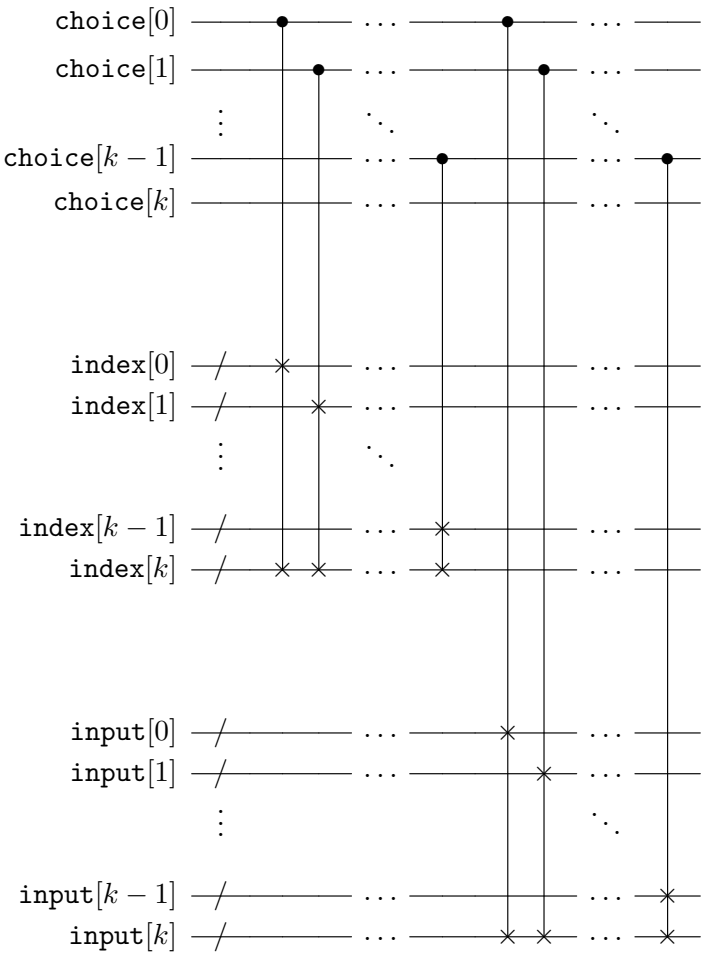


Figure 3.6: Implementation of the two selected swaps SELSWAP$_k$ as part of FY$_k$, with the unary-encoded `choice` as the control register and `index` and `input` as target registers. The figure is from [2]. As each wire of the target registers stands for several qubits, each controlled-SWAP is to be interpreted as many bitwise controlled-SWAPs.

Resetting `choice` register

Next, we need to erase the choice register at the end of each FY block. We can reverse the computation because we previously executed swaps on both `index` and `input`. The erasure is performed by scanning `index` to find out

which value of k was encoded into `choice`. We erase `choice` by applying a NOT operation to `choice`$[\ell]$ if `index`$[\ell] = k$. This can be expressed as a multi-controlled-NOT, as illustrated by an example in Figure 3.7. The control sequence of the multi-controlled-NOT is a binary encoding of the value k.

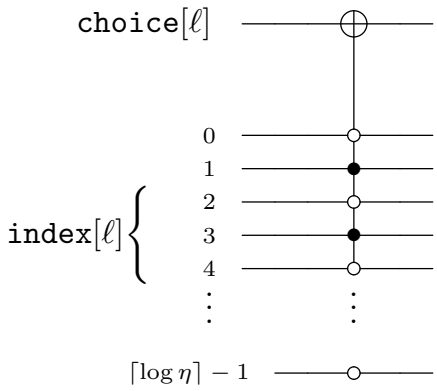


Figure 3.7: Circuit for resetting `choice` register as part of iteration block FY_k from [2]. In this example $k = 10$ and the control sequence is its binary encoding. The NOT erases `choice`$[\ell]$ if `index`$[\ell] = k$.

For compiling multiple controls, see Fig. 4.10 in [81]. Each $\lceil \log m \rceil$-controlled-NOT can be decomposed into a network of $\mathcal{O}(\log m)$ gates (predominantly Toffolis) with depth $\mathcal{O}(\log m)$. Because the $k+1$ multi-controlled-NOTs (for $\ell \leqslant k \leqslant m-1$) can all be executed in parallel, resetting the `choice` register thus requires a circuit with $\mathcal{O}(m \log m)$ gates but only $\mathcal{O}(\log m)$ depth.

3.3.3 Disentangling `index` from `input`

The last task is to clean up and disentangle `index` from `input` by resetting the former to the original state $|0\rangle^{\otimes m\lceil \log m \rceil}$ while leaving the latter in the desired antisymmetrized superposition. The key observation is that the element we moved (blue box in Fig. 3.2) is always the maximum from the array truncated to position k. We can use this observation to reset `index` as follows.

We compare the value carried by each of the m subregisters $\texttt{input}[\ell]$ (labeled by position index $\ell = 0, 1, \ldots, m-1$) with the value of each other subregister $\texttt{input}[\ell']$ ($\ell' \neq \ell$), thus requiring $m(m-1)$ comparisons in total. Note that these subregisters of `input` have all size $\lceil \log N \rceil$. Each time the value held in $\texttt{input}[\ell]$ is larger than the value carried by any other of the remaining $m-1$ subregisters $\texttt{input}[\ell']$, we decrease the value of the corresponding ℓ^{th} subregister $\texttt{index}[\ell]$ of `index` *by* 1. If the value carried by $\texttt{input}[\ell]$ is smaller than $\texttt{input}[\ell']$, we do not decrement the value of $\texttt{index}[\ell]$. This way we identify the original position of the number currently encoded at $\texttt{input}[\ell]$ at thus the value in $\texttt{index}[\ell]$ subregister. After accomplishing all the $m(m-1)$ comparisons within the `input` register and controlled decrements, we have reset the `index` register state to $|0\rangle^{\otimes m\lceil \log m \rceil}$ while leaving the `input` register in the antisymmetrized superposition state.

The comparisons between the values of two subregisters of `input` (each of size $\lceil \log N \rceil$) can be performed with $\mathcal{O}(\log N)$ gates and $\mathcal{O}(\log \log N)$ depth using a quantum comparison from Section 3.4.2. The oracle's output is used to control the '*decrement by 1*' operation, after which the oracle is used again to uncompute the ancilla holding its result.

Decrementing the value of the $\lceil \log m \rceil$-sized index subregister $\texttt{index}[\ell]$ (for any $\ell = 0, 1, \ldots, m-1$) by the value 1 can be achieved by a circuit depicted in Figure 3.8.

Each decrement consists of a total of $\lceil \log m \rceil$ multi-controlled-NOTs. Each multi-controlled-NOT involves $n = \lceil \log m \rceil - 1, \ldots, 0$ controls plus the control on the qubit holding the result of the comparison. These gates can be decomposed into $\mathcal{O}(n)$ Toffoli gates (see [81]), but the majority of the involved Toffoli gates for different values of n will effectively cancel each other out. The resulting cost is then $\mathcal{O}(\log m)$ gates and no ancillae since we can reuse qubits from `choice` register.

3.3.4 Complexity Analysis of the FY shuffle

Putting everything together, the overall circuit size for this step amounts to $\mathcal{O}(m(m-1)[\log N + \log m])$ predominantly Toffoli gates, which can then be further decomposed into CNOTs and single-qubit gates (including T gates) in well-known ways. Because $m \leqslant N$, we thus report $\mathcal{O}(m^2 \log N)$ for

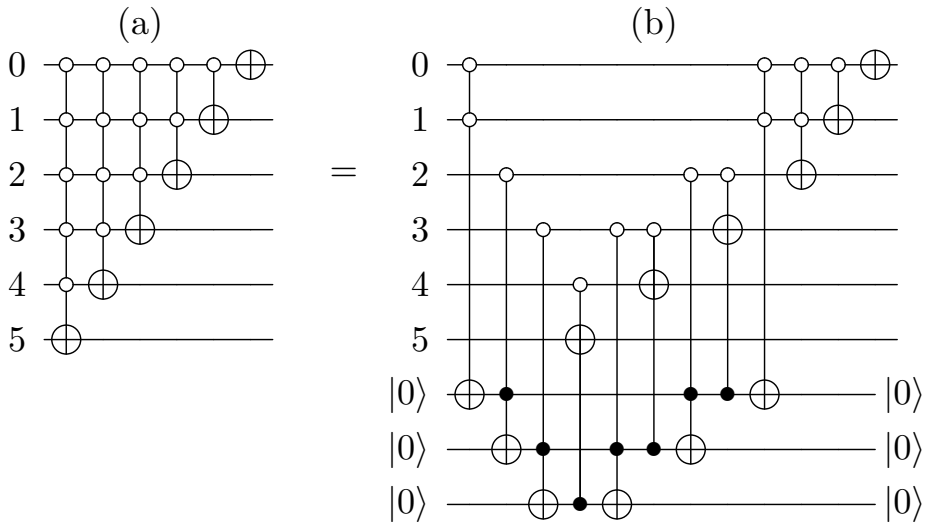


Figure 3.8: Circuit implementing *'decrement by 1'* operation, applied to `index`$[\ell]$ subregisters of size $\lceil \log m \rceil$ from [2]. (a) Example for $m = 64$. (b) Decomposition into a network of $\mathcal{O}(\log m)$ Toffoli gates using $\mathcal{O}(\log m)$ ancillae.

the overall gate count for this step, while its circuit depth is $\mathcal{O}(m^2 \log\log N + m^2 \log m)$.

3.4 Symmetrization Through Quantum Sorting

Our second approach to (anti)symmetrization uses a sorting algorithm. Classically, sorting is not used as a part of a shuffle, because the FY shuffle is more efficient. The situation is different for quantum algorithms, making an approach utilizing a sort more appealing. Sadly, most well-known classical algorithms are not well-suited for quantization. Quicksort's worst case runtime, m^2 applies every time if run in superposition. A standard recursive application of merge sort requires a large space overhead. Other algorithms, for example, heapsort, implement operations whose position depends on values stored in the registers. This overhead is analogical

to selected swaps in FY shuffle. In this Section, we show that sorting networks are well-suited for quantum algorithms and give detailed circuits to implement them.

Quantum sort can speed up the preparation of a symmetrization and antisymmetrization of sequences. Sorting for antisymmetrization was originally proposed in [170], but has complexity $\mathcal{O}(m^2 \log N)$ and is not optimized for depth.

The algorithm that we describe now has the complexity $\mathcal{O}(m \log m \log N)$ and depth $\mathcal{O}(\log m \log\log N)$. We first review the four main steps and then explain them in detail:

1. **Prepare** `seed`**.** Let $\eta > m^2$. In this step, we prepare an ancillary register called `seed` in an even superposition of all possible length-m strings of the numbers $0, 1, \ldots, \eta - 1$ in binary encoding. If η is a power of two, preparing `seed` consist of the Hadamard transform.

2. **Sort** `seed`**.** Sort `seed` using a quantum sorting network. This requires a modification of a sorting network that encoded the performed swaps in a second ancillary register called `record`. Comparing numbers is also performed coherently and can be parallelized. There are several known sorting networks with polylogarithmic runtime, as we discuss below.

3. **Delete collisions from** `seed`. At this point, we need to remove repeated entries from `seed`. Due to the choice of η, repetitions are rare as a consequence of the birthday paradox. As we consider only permutations without repetition, we need to identify and delete the repeats. We further prove that the resulting state of `seed` is disentangled from `record`, meaning `seed` can be discarded after this step.

4. **Apply the reverse of the sort to** `target`**.** Using the comparator values stored in `record`, we apply each step of the sorting network in reverse order to the sorted array `target`. The resulting state of `target` is an evenly weighted superposition of each possible permutation of the original values. If we wish to include a -1 phase for each permutation, we can apply a Z gate after each swap.

Step 4 is the key step for preparing a symmetric superposition. Having prepared (in Step 1-Step 3) a record of the in-place swaps needed to sort a symmetrized, collision-free array, we undo each of these swaps in turn on the sorted `target`. We demonstrate the algorithm on an example in Fig. 3.9.

In the next Section, we explain sorting networks and their implementation on a quantum computer. Then, we show how to compare two numbers coherently, which is an elementary operation necessary in sorting. In the next Subsection 3.4.3, we explain Step 3, **Delete collisions**. We conclude this section by connecting all the components and computing the complexity of our algorithm.

We employ a sorting network, a sorting method used in hardware, because sorting networks have comparisons and swaps at a fixed sequence of locations.

3.4.1 Quantum Sorting Network

Sorting networks are logical circuits that consist of wires carrying values and comparator modules applied to pairs of wires, that compare values and swap them if they are not in the correct order, see Fig. 3.10. They were developed for sorting on a hardware level. Sorting networks are data-oblivious, meaning that performed operations do not depend on the input which makes them ideal for implementation in quantum computers.

Wires represent bit strings (integers stored in binary) in classical sorting networks and qubit strings in their quantum analogs. A classical comparator[2] is a sort on two numbers. In other words, on an ordered pair of inputs A and B, a comparator implements the transformation

$$\text{comparator}\left([A, B]\right) = [\min(A, B), \max(A, B)]\,, \tag{3.9}$$

producing an ordered pair of A, B in increasing order. A quantum comparator is its reversible version where we record whether the items were already sorted (ancilla state $|0\rangle$) or the comparator needed to apply a swap (ancilla state $|1\rangle$); see Figure 3.11.

[2]The name comparator can be somehow misleading because this step involves a conditional step in addition to the comparision. Nevertheless, this is is the term used in the literature.

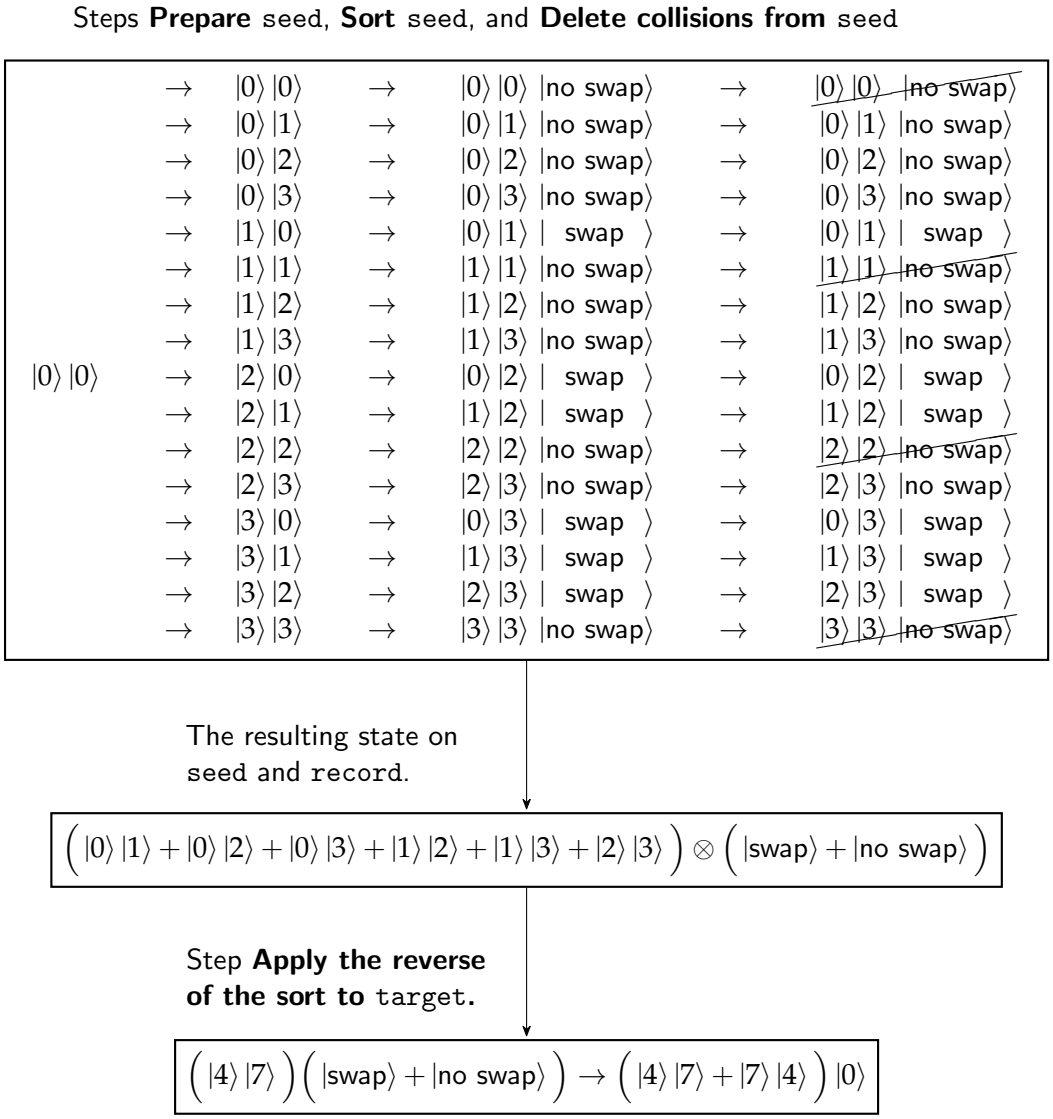


Figure 3.9: Example of the symmetrization by sorting for $N = 4$ and $m = 2$. Steps 1-3 operate only on ancillary registers seed and record and aim to prepare the desired superposition in record. In step 4 (last box), we use record and a reverse sort to symmetrize the data stored in target.

The key feature of sorting networks is that the positions of comparators are predetermined. Therefore, they cannot depend on the inputs, unlike the

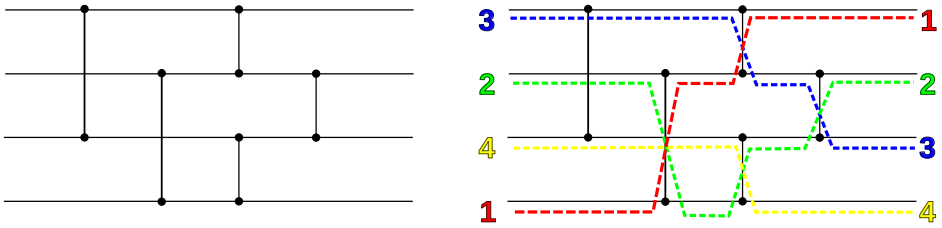


Figure 3.10: A simple sorting network from [171]. The sorting network on 4 inputs on the left hand side that consists of five "comparators", elementary components that compare two numbers and potentially swap them, see Fig. 3.11. The right-hand side figure demonstrates how to sort the input 3-2-4-1 in this network.

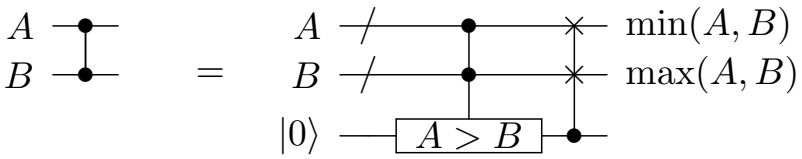


Figure 3.11: The standard notation for a comparator is indicated on the left-hand side from [2]. Its implementation as a quantum circuit is shown on the right. In the first step, we compare two inputs with values A and B and save the outcome (1 if $A > B$ is true and 0 otherwise) in a single-qubit ancilla. In the second step, conditioned on the value of the ancilla qubit, the values A and B in the two wires are swapped.

comparisons in heap sort. This makes sorting networks viable candidates for quantum computing. Many of the sorting networks are also highly parallelizable, thus allowing low-depth, often polylogarithmic, performance. Applications of sorting networks in quantum algorithms has previously been considered in Refs. [142, 172, 173].

Our algorithm allows for any choice of sorting network. Several standard sort algorithms, for example, the insert and bubble sorts, can be represented as sorting networks. However, these algorithms have inferior time complexity and depth. More efficient runtime can be achieved, for example, using the bitonic sort [174, 175], which is illustrated for 8 inputs in Figure 3.12. The bitonic sort uses $\mathcal{O}(m \log^2 m)$ comparators and $\mathcal{O}(\log^2 m)$ depth, thus achieving an exponential improvement in depth compared

to conventional sorting techniques. Similar performance can be obtained using an odd-even mergesort [174].

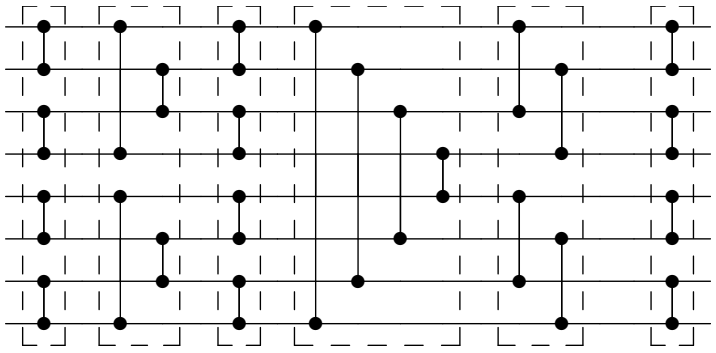


Figure 3.12: Example of a classical *bitonic sort* on 8 inputs from [2]. Comparators in each dashed box can be applied in parallel for depth reduction. A quantum version of of the bitonic sort uses the same order of swaps with comparators defined in Fig. 3.11.

The asymptotically best sorting networks have depth $\mathcal{O}(\log m)$ and complexity $\mathcal{O}(m \log m)$, though there is a large constant which means they are less efficient for realistic m [176, 177]. There is also a sorting network with $\mathcal{O}(m \log m)$ complexity with a better multiplicative constant [178], though its depth is $\mathcal{O}(m \log m)$ (so it is not logarithmic).

For a small number of inputs to be sorted (up to $m = 20$), very tight bounds have been derived for optimized circuit depth as well as the overall number of comparators. Knuth [169] and later Codish *et al.* [179] gave networks for sorting up to 17 numbers that were later shown to be optimal in depth, and up to $m \leqslant 10$ also optimal in the number of comparators. Optimizations for up to 20 inputs have recently been achieved, see Table 1 in [179]. For illustration, the best known sorting networks for 20 numbers require depth 11 and 92 comparators, with lower bounds reported as 10 and 73 respectively. More extensive sorting networks can be built up by an in-place merging of smaller sorting networks, but the construction is sub-optimal.

Assume that a quantum sorting network has m wires, where each wire represents a quantum register with d qubits (i.e., $d = \log N$). We can obtain the resource requirements for a quantum sort by taking a (classical) sorting network and multiplying its resources in terms of comparators by those

needed for a quantum comparator. A quantum comparator consists of a procedure for comparing two numbers explained in Section 3.4.2 and a controlled swap on registers of size d, see Fig. 3.11.

3.4.2 Quantum Comparator

Here we describe how to reversibly implement the comparison of the value held in one register with the value carried by a second equally-sized register, and store the result (larger or not) in a single-qubit ancilla.

We need to use it for implementing the comparator modules of quantum sorting networks as well as in our antisymmetrization approach based on the quantum FY shuffle. We first explain a straightforward method for comparison with depth linear in the length of the involved registers. In the second step, we then convert this prototype into an algorithm with depth logarithmic in the register length using a divide and conquer approach.

Let us compare two values A and B encoded as binaries in two equally sized registers, $\mathtt{A}$ and $\mathtt{B}$. Intuitively, we can compare the registers in a *bit-by-bit* fashion, starting with their most significant bits and going down to their least significant bits. At the very first occurrence of an i such that $\mathtt{A}[i] \neq \mathtt{B}[i]$, i.e., either $\mathtt{A}[i] = 1$ and $\mathtt{B}[i] = 0$ or $\mathtt{A}[i] = 0$ and $\mathtt{B}[i] = 1$, we know that $A > B$ in the first case and $A < B$ in the second case. If $\mathtt{A}[i] = \mathtt{B}[i]$ for all i, then $A = B$.

If we are performing this comparison in superposition, we no longer have the option of stopping once we found the point of difference. Hence, we employ two ancillary registers denoted $\mathtt{A}'$ and $\mathtt{B}'$, that each consist of d qubits initialized $|0\rangle^{\otimes d}$. These registers will encode the results of the comparison. All the bitwise comparisons except for the first one will be conditioned on the values stored in $\mathtt{A}'$ and $\mathtt{B}'$. Once we found the most significant bit where $\mathtt{A}[i] \neq \mathtt{B}[i]$, we flip the bit into 1 in $\mathtt{A}'$ if $A > B$ or flip the bit into 1 in $\mathtt{B}'$ if $B > A$. This means that the results of all remaining comparisons is insignificant and should not be recorded. Instead, we carry the result of the comparison of the most significant bits all the way to the least significant bit and use it for reading the result.

We illustrate the naive comparison protocol on an example in Table 3.1. While this algorithm works, it has the drawback that the bitwise compari-

Register	i=0	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8
A	0	1	0	0	1	0	1	0	1
B	0	1	0	0	0	1	1	1	0
A′	0	0	0	0	1	1	1	1	1
B′	0	0	0	0	0	0	0	0	0

Table 3.1: Example illustrating the idea of reversible bitwise comparison. The index i labels the bits of the registers. At $i = 0$ we compare the most significant bits $A[0]$ and $B[0]$, and write 1 into ancilla $A'[0]$ if $A[0] > B[0]$, or write 1 into ancilla $B'[0]$ if $A[0] < B[0]$. In our example, the ancillas remain as 0. While both $A'[i-1]$ and $B'[i-1]$ are zero, we keep comparing the numbers bit by bit encoding the result into $A'[i]$ and $B'[i]$. The first occurrence of $A[i] \neq B[i]$ is for $i = 4$, at which stage the value of ancilla $A'[4]$ is switched to 1, as $A[4] > B[4]$. For the remaining steps, the remaining bits of A and B will not be compared anymore. Instead, in each step we flip each remaining bit of A' to 1, whereas all bits of B' remain 0. Thus, the last bit of A' will be 1 while $B'[8]$ is 0, implying $A > B$.

son is conducted *sequentially*, which results in circuit-depth scaling $\mathcal{O}(d)$. It also uses more ancilla qubits than necessary.

We can reduce the number of ancillae and the depth of the algorithm with a divide-and-conquer approach. Let us split the register A into two parts: A_1 consisting of the first approximately $d/2$ bits and A_2 consisting of the remaining approximately $d/2$ bits. Split register B in the very same way into subregisters B_1 and B_2. We can then determine which number is larger (or whether both are equal) for each pair (A_1, B_1) and (A_2, B_2) separately in parallel (using an unspecified method) and record the results of the two comparisons in ancilla registers (A_1', B_1'), (A_2', B_2'). The least significant bits of these four ancilla registers can be then used to deduce whether $A > B$ or $A < B$ or $A = B$ with just a single bitwise comparison. Thus, we effectively halved the depth by dividing the problem into smaller problems and merging them afterward. This approach can be also be used for comparing subregisters, giving a recursive algorithm. We now explain its bottom-up implementation.

First, slice A and B into pairs of bits – the first slice contains $A[0]$ and $A[1]$, the second slice consists of $A[2]$ and $A[3]$, etc., and in the very same

way for B. The critical step takes the corresponding slices of A and B and overwrites the second bit of each slice with the outcome of the comparison. We ignore the first bit of each slice and store the comparison results stored in the second bits. Therefore, the second bits become the next layer on which bitwise comparisons are performed again. We denote the i'th bit forming the registers of the j^{th} layer by $A^j[i]$ and $B^j[i]$. The original registers A and B correspond to $j = 0$: $A^0 \equiv A$ and $B^0 \equiv B$. The part of the circuit that implements a single bitwise comparison is depicted in Figure 3.13. We denote the corresponding transformation 'COMPARE2', such that it takes two bits from A^j and two bits of B^j registers and sortes the outcome of the comparison in the bits $A^{j+1}[i]$, $B^{j+1}[i]$ storing the comparison result. Note that the comparison can be performed in place with the use of a single ancilla.

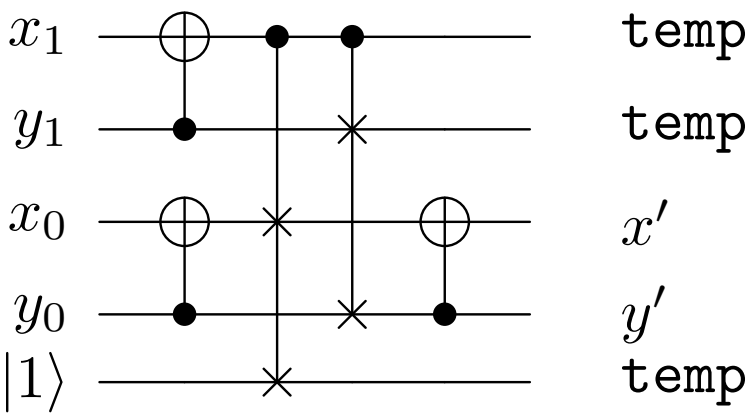


Figure 3.13: The input pair is $(x, y) = (x_0 + 2x_1, y_0 + 2y_1)$. The output pair is (x', y') and will satisfy $\text{sign}(x' - y') = \text{sign}(x - y)$. Output qubits marked "temp" store values that are not needed, and are kept until a later uncompute step where the inputs are restored. Each Fredkin gate within the circuit can be computed using 4 T gates and (by storing an ancilla not shown) later uncomputed using 0 T gates [91, 180]. This circuit was previously published in [2].

At each step, comparisons of the pairs of the original arrays can be performed in parallel, and produce two new arrays with approximately half the size of the original ones to record the results. Thus, at each step,

we approximately halve the size of the problem, while using a constant depth for computing the results. The basic idea is illustrated in Figure 3.14. This procedure is repeated for $\lceil \log d \rceil$ steps until registers $\mathtt{A}^{\text{fin}} := \mathtt{A}^{\lceil \log d \rceil}$ and $\mathtt{B}^{\text{fin}} := \mathtt{B}^{\lceil \log d \rceil}$ both of size 1 have been prepared.

The parallelized algorithm is perfectly suited for comparing arrays whose length d is a power of 2. If d is not a power of 2, we can either pad $\mathtt{A}$ and $\mathtt{B}$ with 0s prior to their most significant bits without altering the result, or introduce comparison of single bits (using only the first two gates from the circuit in Figure 3.13 with targets on $\mathtt{A}^{j+1}$ and $\mathtt{B}^{j+1}$ registers respectively).

Formally, we can express our comparison algorithm as follows, here assuming d to be a power of 2:

Low-depth comparison

for $j = 0, \ldots, \log d - 1$ **do**
  for $i = 0, \ldots, \text{size}(A^j)/2 - 1$ **do**
    $\left(A^{j+1}[i], B^{j+1}[i]\right) = \textsc{Compare2}(A^j[2i], B^j[2i], A^j[2i+1], B^j[2i+1])$
  end for
end for
return $(A^{\log d - 1}[0], B^{\log d - 1}[0])$

The key feature of this algorithm is the parallelization of all the operations of the inner loop. Since one application of COMPARE2 requires only constant depth and a constant number of operations, our comparison algorithm requires only depth $\mathcal{O}(\log d)$.

The above-constructed comparison algorithm can be used to output a result that distinguishes between $A > B$, $A < B$ and $A = B$. Its reversible execution results in the ancillary single-qubit registers $\mathtt{A}^{\text{fin}}$ and $\mathtt{B}^{\text{fin}}$ generated in the very last step of the algorithm holding information about which number is larger or whether they are equal. Indeed, $\mathtt{A}^{\text{fin}}[0] = \mathtt{B}^{\text{fin}}[0]$ implies $A = B$, $\mathtt{A}^{\text{fin}}[0] < \mathtt{B}^{\text{fin}}[0]$ implies $A < B$, and $\mathtt{A}^{\text{fin}}[0] > \mathtt{B}^{\text{fin}}[0]$ implies $A > B$. The three cases are separated into three control qubits by using the circuit shown in Figure 3.15. These individual control qubits can be used to control further conditional operations that depend on the result of the comparison.

After the output bit has been produced, we reverse the comparison

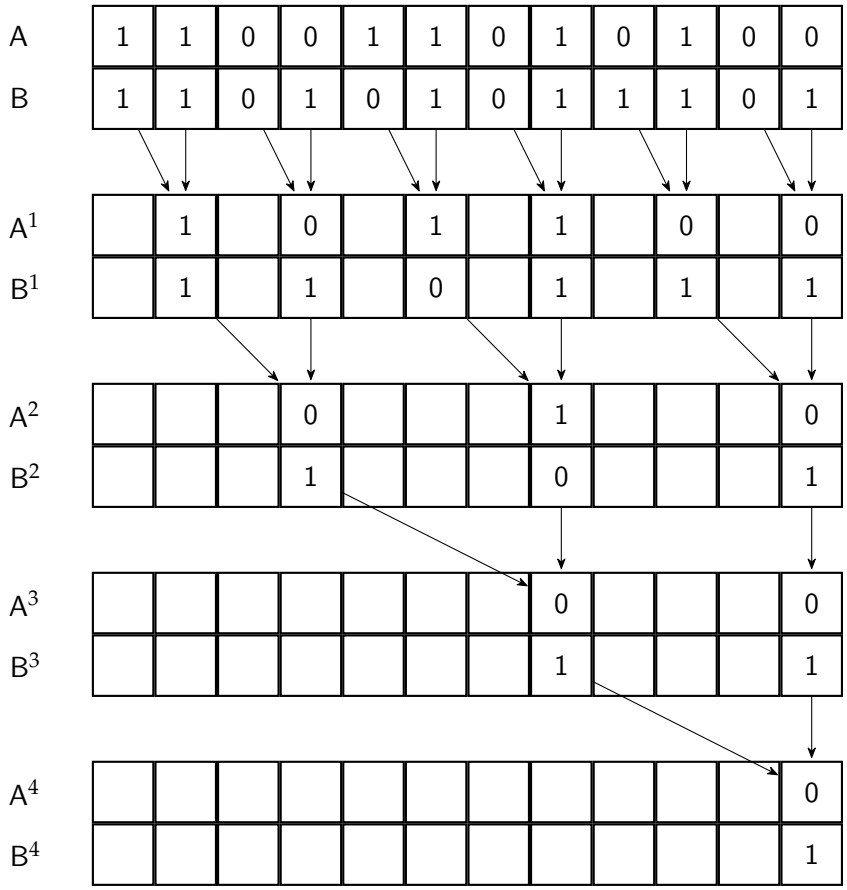


Figure 3.14: Parallelized bitwise comparison. Notice that each step reduces the size of the problem by approximately one half, while using a constant depth for computing the results.

algorithm to clean all the ancillary registers and restore the input registers A and B.

The actual comparison of two numbers thus takes as inputs two size-d registers A and B (holding values A and B) and a single-qubit ancilla q initialized to $|0\rangle$. It reversibly computes whether $A > B$ is true or false by executing the parallelized comparison process presented above. It copies the result (which is stored in A^{fin}) to ancilla q. It then executes the inverse of the comparison process. It outputs A and B *unaltered* and the ancilla q holding the result of the oracle: $q = 1$ if $A > B$ and $q = 0$ if $A \leqslant B$.

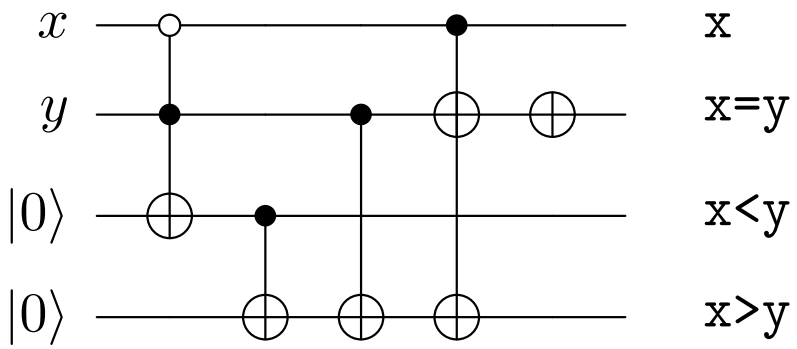


Figure 3.15: A circuit that determines if two bits are equal, ascending, or descending from [2]. When the comparison is no longer needed, the results are uncomputed by applying the circuit in reverse order.

As shown, this oracle has circuit size $\mathcal{O}(d)$ but depth only $\mathcal{O}(\log d)$ and a T-count of $8d + \mathcal{O}(1)$.

The second part of the comparator consists of conditional swaps on two registers of size d. Since all the single-bit swaps are conditioned on the same ancilla qubit, it may appear that these swaps should be applied sequentially, which would imply depth scaling $\mathcal{O}(d)$. However, the conditional swaps can also be parallelized by first copying the bit of the ancilla holding the result of the comparison to additional ancillae initialized in $|0\rangle$ states. Such an expansion of the result to d copies can be attained with a parallelized arrangement of $\mathcal{O}(d)$ CNOTs but with circuit depth only $\mathcal{O}(\log d)$. After copying, all the d controlled elementary swaps can then be executed in parallel (by using the additional ancillae) with circuit depth only $\mathcal{O}(1)$. Next, the $d-1$ additional ancillae used for holding the copied result of comparison are uncomputed again, by reversing the copying process. While this procedure requires $\mathcal{O}(d)$ ancillary space overhead, it optimizes the depth. The overall space overhead of the quantum comparator is also $\mathcal{O}(d)$.

Taking $d = \lceil \log N \rceil$ (the largest registers used in Step 4 of our sort-based antisymmetrization algorithm), conducting the quantum bitonic sort, for instance, thus requires $\mathcal{O}(m \log^2(m) \log N)$ elementary gates but only $\mathcal{O}(\log^2(m) \log \log N)$ circuit depth.

Similar scaling can be achieved using an algorithm from Draper et al. [181]. As presented, our work uses slightly fewer Toffoli gates and ancillae compared to [181]. However, the number of CNOTs and overall depth are larger.

3.4.3 Analysis of 'Delete Collisions' Step

In this Section, we explain how repeated items (collisions) can be deleted from seed. This step is non-deterministic and involves a measurement. We have to show that our algorithm is highly likely to succeed because collisions are relatively rare for a given set of parameters. If this part fails, we can restart the algorithm and repeat steps 1-3. We require that the chance of failure is at most $\frac{1}{2}$; therefore, the expected number of runs of steps 1-3 is at most 2. Additionally, because we want to uncompute `record` during the final step of our algorithm, we have to show that the resulting state of `seed` is disentangled from `record`.

Let us first look at the state of `seed` after Step 1

$$\frac{1}{\eta^{m/2}} \sum_{\ell_0,\ldots,\ell_{m-1}=0}^{\eta-1} |\ell_0,\ldots,\ell_{m-1}\rangle\,. \tag{3.10}$$

The state (3.10) can be decomposed into a part with no repetitions and one when repetitions occur. Clearly, these two subspaces are mutually orthogonal.

To show that the probability of success is high, we need to compute the support of (3.10) on the subspace with no repetitions. This task is exactly opposite to the birthday problem [182]; we want the likelihood of $\ell_i = \ell_j$ for any $i \neq j$ to be small.

Comparing the number of permutations without and with repetitions (just as for the generalized birthday problem), we can show that the norm of the projection onto the repetition-free subspace is

$$\frac{m!}{\eta^m}\binom{\eta}{m}. \tag{3.11}$$

Using Proposition A.1 in [182], the probability of success is lower bounded as

$$\Pr(\text{success}) = \frac{m!}{\eta^m}\binom{\eta}{m} \geqslant 1 - \frac{m(m-1)}{2\eta}, \tag{3.12}$$

which is more than $1/2$ for $\eta \geqslant m^2$.

To detect the repetitions in our algorithm, we need to compare the values of ℓ_i in `seed`. It is advantageous to perform the comparisons in Step 3 after sorting the registers because then it is only necessary to check adjacent entries. Sorting does not affect whether there are repetitions. The repetition-free outcome can be achieved after fewer than two attempts on average. One can improve the success probability by using a larger η or by using amplitude amplification.

We are left to show that `seed` is not entangled with `record` after Step 3. After Step 1, the state of $\texttt{seed} \otimes \texttt{record}$ projected to the repetition-free subspace is proportional to

$$\sum_{0 \leqslant \ell_0 < \ldots < \ell_{m-1} < \eta} \sum_{\sigma \in S_m} |\sigma(\ell_0, \ldots, \ell_{m-1})\rangle_{\texttt{seed}} |\iota\rangle_{\texttt{record}}. \tag{3.13}$$

The register `record` will encode the performed permutation, in terms of performed swaps, applied to `seed`. At this stage, `record` is initialized to the identity permutation denoted as ι.

In Step 2, we sorted `seed` by applying a series of swaps $\sigma_1, \ldots, \sigma_T$ which we recorded in `record`. Since the swaps were intentionally chosen to order the elements in `seed`, we effectively uncompute the initial permutation σ. Formally,

$$\sigma_T \circ \cdots \circ \sigma_1 \circ = \sigma^{-1}. \tag{3.14}$$

Note that the repetitions-free subspace is closed under sorting. Therefore, we do not need to consider repeated elements in this analysis.

All together, Steps 1-2 transform $\texttt{seed} \otimes \texttt{record}$ projected on the repetition-free subspace as

$$\sum_{0 \leqslant \ell_0 < \ldots < \ell_{m-1} < \texttt{eta}} \sum_{\sigma \in S_m} |\sigma(\ell_0, \ldots, \ell_{m-1})\rangle_{\texttt{seed}} |\iota\rangle_{\texttt{record}} \tag{3.15}$$

$$\to \sum_{0 \leqslant \ell_0 < \ldots < \ell_{m-1} < \eta} |\ell_0, \ldots, \ell_{m-1}\rangle_{\texttt{seed}} \sum_{\sigma \in S_m} |\sigma_1, \ldots, \sigma_T\rangle_{\texttt{record}} \tag{3.16}$$

In Step 3, we can flag and eliminate repetitions which leaves us with the state in Eq. (3.16). Notice that the states in registers `seed` and `record` are not entangled. Since we only need the state in `record`, see Fig. 3.9, we can safely discard `seed`.

3.4.4 Complexity Analysis of the Shuffle via Sorting

We now give the complexity of the shuffle via sorting by collecting the complexities of its components. Assuming we use an asymptotically optimal sorting network, the circuit depth for our algorithm is $\mathcal{O}(\log m \log\log N)$ and the gate complexity is $\mathcal{O}(m \log m \log N)$. The dominant cost of the algorithm comes from Step 2 and Step 4, each of which has $\mathcal{O}(m \log m)$ comparators that can be parallelized to ensure the sorting network executes only $\mathcal{O}(\log m)$ comparator rounds. Each comparator for Step 4 has a complexity of $\mathcal{O}(\log N)$ and a depth of $\mathcal{O}(\log\log N)$, as we show in Sec. 3.4.2. The comparators for Step 2 have complexity $\mathcal{O}(\log m)$ and depth $\mathcal{O}(\log\log m)$, which is less because $m < N$. Thus Step 2 and Step 4 each have gate complexity $\mathcal{O}(m \log m \log N)$ and runtime $\mathcal{O}(\log m \log\log N)$.

The other two steps in our algorithm have smaller cost. Step 1 has constant depth and $\mathcal{O}(m \log m)$ complexity. Step 3 requires $\mathcal{O}(m)$ comparisons because be only need to compare adjacent registers on `seed` after sorting. These comparisons can be parallelized over two rounds, with complexity $\mathcal{O}(m \log m)$ and circuit depth $\mathcal{O}(\log\log m)$. Then the result for any of the registers being equal is computed in a single qubit, which has complexity $\mathcal{O}(m)$ and depth $\mathcal{O}(\log m)$. Thus the complexity of Step 3 is $\mathcal{O}(m \log m)$ and the total circuit depth is $\mathcal{O}(\log m)$. Thus, our algorithm has an exponential improvement in depth over the proposal in Refs. [170, 183]. We also have a quadratic improvement in gate complexity, which is $\widetilde{\mathcal{O}}(m)$ for our algorithm but $\widetilde{\mathcal{O}}(m^2)$ for Refs. [170, 183].

The depth of our algorithm is likely optimal for symmetrization, at least in terms of the m scaling. Symmetrization takes a single computational basis state and generates a superposition of $m!$ computational basis states. Each single-qubit operation can increase the number of states in the superposition by at most a factor of two, and two-qubit operations can increase the number of states in the superposition by at most a factor of four. Thus, the number of one- and two-qubit operations is at least

$\log_2(m!) = \mathcal{O}(m \log m)$. In our algorithm, we need this number of operations between the registers. If that is true in general, then m operations can be parallelized, resulting in minimum depth $\mathcal{O}(\log m)$.

Our quoted asymptotic runtime and gate complexity scalings assume the use of sorting networks that are asymptotically optimal. However, these algorithms have a large constant overhead making it more practical to use an odd-even mergesort, leading to depth $\mathcal{O}(\log^2 m \log\log N)$. Note that is possible to obtain complexity $\mathcal{O}(m \log m \log N)$ with a better scaling constant using the sorting network of Ref. [178] but worse depth. All together, we can summarize our results in the theorem below:

Theorem 5 (Quantum shuffle)
Define a quantum shuffle as the transformation

$$|r_1 \cdots r_m\rangle \to \sum_{\sigma \in S_m} |\sigma(r_1, \cdots, r_m)\rangle\,, \tag{3.17}$$

where $0 < r_1 < r_2 \cdots < r_m < N$ *are encoded in binary and* σ *iterates over the permutations. The quantum shuffle can be performed using* $\mathcal{O}(m \log^2 m \log N)$ *gates and depth* $\mathcal{O}(\log m \log\log N)$.

3.5 Applications

The need for a quantum shuffle arises in the state preparation problem in quantum chemistry with first quantization as outlined in Fig. 3.16.

State preparation is the first step for eigenstate (and particularly ground state) preparation algorithms. The goal of this step is to prepare a state that has a significant overlap with the ground state. The state is specified by listing the orbital for each fermion. Note that we described the algorithm as working on a state in the computational basis but a superposition of states is also possible. We can index the orbitals as 0 to N_1. Since the number of orbitals is typically much larger than the number of fermions m, we can perform the Delete collisions step correctly. Since it represents fermions, the state must be completely antisymmetric (exchanging two fermions leads to a (-1) phase). As a result, no two fermions can occupy the same state.

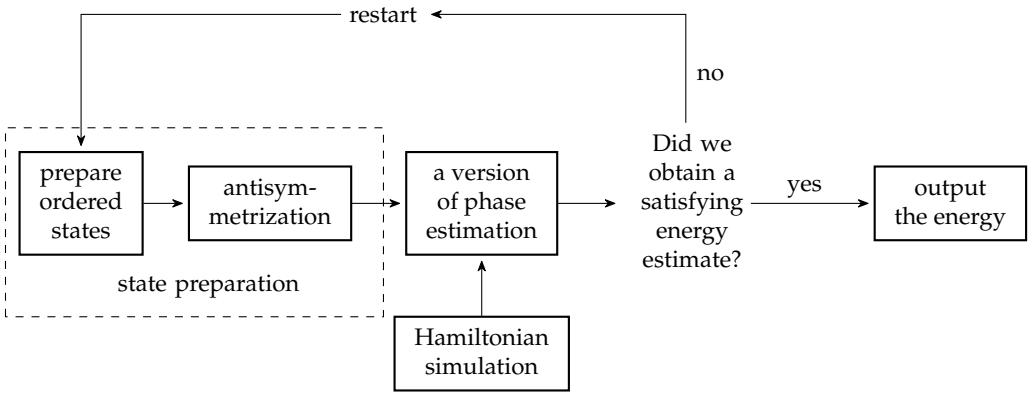


Figure 3.16: An overview of an algorithm for quantum chemistry in the first quantization based on the proposal [183]. The goal of this algorithm is to compute the ground state energy of a given Hamiltonian. The diagram presents the steps of the algorithm left to right. The algorithm works by first preparing a state that has a nontrivial overlap with the ground state and then using a version of phase estimation from Sec. 2.2.2 to extract the ground state energy. The state preparation phase consists of preparing a superposition of ordered states $|r_1 \cdots r_m\rangle$ and their subsequent antisymmetrization. This fermionic state is then used in phase estimation with Hamiltonian simulation used for the unitary U. If phase estimation fails to produce the desired state, the whole procedure is restarted using the same parameters.

Abrams and Lloyd [183] proposed the first algorithm for preparing a completely antisymmetric state. First, we prepare an unsymmetrized state $|r_1, \ldots, r_m\rangle$ where $0 \leqslant r_1 \leqslant r_2 \leqslant \cdots \leqslant r_m < N$, or a superposition of such states. Then, they use sorting and un-sorting for antisymmetrization. However, they suggest using Heapsort, whose quantum complexity is $m^2 \log m$ comparisons and do not go into implementation details.

Modification of the shuffle to prepare a completely antisymmetric state one comes in Step 4 and can be accomplished by applying Z gates with each swap. We select a target qubit in the `input` register – it does not matter which. Then, for each $\ell = 0, 1, \ldots, k-1$, we apply a phase gate controlled on position ℓ of `choice` to the target qubit. The result is that `input` has picked up a phase of (-1) if `choice` specified a value strictly less than k. The total number of gates is $k = \mathcal{O}(m)$, while the depth can be made $\mathcal{O}(1)$.

Quantum sort and comparison has applications beyond symmetrization and antisymmetrization. We utilized a quantum search in Chapter 4 to produce a time ordering. A quantum comparison procedure introduced here is essential for a black-box state preparation algorithm [116]. We used a similar technique in Chapter 4.

3.6 Conclusion

To conclude this Section, we have presented an algorithm for symmetrizing (or antisymmetrizing) a sorted, repetition-free quantum register. We propose two algorithms to accomplish this case. The first one is a quantum version of FY shuffle. While FY shuffle is classically optimal, its quantum version requires linear overhead to replace random swaps with preparing superposition over all possible swaps.

The second approach implements a shuffle by reversing a sort. The choice of a sort presents the dominant cost of this algorithm. We suggest using sorting networks for sorting in quantum hardware and devise circuits for highly efficient sorting and comparison. The resulting asymptotical gate count complexity and depth are, respectively, $O(m \log m \log N)$ and $O(\log m \log\log N)$. This constitutes a polynomial improvement in the first case and exponential in the second case over previous work in Refs. [170, 183]. As in Ref. [183], our algorithm can be used for state preparation of a fermionic wavefunction in first quantization.

Our algorithm requires that the input is always sorted in increasing order. This assumption is easy to satisfy for the purposes for state preparation but one may ask whether it is necessary. It is easy to see that our algorithm fails if the input is not ordered – we will not be able to clean `record`.

We can modify our sort based shuffling algorithm to allow for different types of ordering provided that the input is still in strict order and we can construct an alternative comparator for this order. Then we can apply the reverse sort in Step 4 with respect to this new order. We can easily (anti)symmetrize input in decreasing order with the same complexity but the complexity with respect to N might be different for different encodings.

4

Simulation of Time-Dependent Hamiltonians

"Every moment before this one depends on this one."

Jonathan Safran Foer (Extremely Loud and Incredibly Close)

This chapter is based on the paper [3] and significantly overlaps with its text. I contributed to all parts of the projects including the majority of the writing.

4.1 Unitary Evolution Under a Time-dependent Hamiltonian

The Schrödinger equation

$$i\frac{d}{dt}|\psi(t)\rangle = \hat{H}|\psi(t)\rangle \tag{4.1}$$

is one of the most famous equations in all of physics. It describes the time evolution of a closed quantum system. We described in detail how to solve it on a quantum computer in Section 2.3. The standard process of analytically solving (4.1) (albeit exactly possible only for a few special cases) is to find the eigenvalues E_j and eigenvectors $|\phi_j\rangle$ of the Hamiltonian and express the evolution for a time period T as

$$|\psi(T)\rangle = \sum_j \langle\phi_j|\psi(0)\rangle e^{-iE_j t}|\phi_j\rangle \tag{4.2}$$

While solving the Schrödinger equation classically is almost always very difficult, using a time-dependent Hamiltonian is even more complicated. It is possible to find the *instantaneous* eigenvalues and the eigenvectors; these are however different for each time and Eq. (4.2) no longer applies.

The formal solution of the evolution under a time-dependent Hamiltonian can be written as

$$U(t_0, t_0+T) = \mathcal{T}\exp\left(-i\int_{t_0}^{t_0+T} d\tau\, H(\tau)\right), \tag{4.3}$$

where $\mathcal{T}$ is the time ordering operator. Equation (4.3) can be interpreted as the limit

$$U(t_0, t_0+T) = \lim_{M\to\infty} \mathcal{T}\prod_{n=0}^{M-1} \exp\left\{\frac{-iT}{M} H\left(t_0 + \frac{nT}{M}\right)\right\}. \tag{4.4}$$

Solving the time-dependent Schrödinger equation analytically is in practice almost always impossible. Dynamics of even a small system, say a $\frac{1}{2}$-spin, can exhibit rich behavior that depends on the change of

the Hamiltonian. Many theoretical approaches have been developed to approximate the evolution under a time-dependent Hamiltonian. The adiabatic approximation [184, 185] can be used when the change of the Hamiltonian is slowly varying.

Other approximations include the diabatic approximation and rotating wave approximation. Nevertheless, the evolution under a general time-dependent Hamiltonian is a computationally difficult problem that can be only solved through numerical simulation (for a small system), or simulated on a quantum computer. Let us first define the problem of simulating time dependent Hamiltonians:

Problem 1 (The time-dependent Hamiltonian simulation problem) *Let* H *be a time-dependent Hamiltonian that can be expressed as a time-dependent linear combination of unitaries as either* $H(t) = \sum_{l=0}^{L-1} \alpha_l(t) H_l$ *or* $H(t) = \sum_{l=0}^{L-1} \alpha_l H_l(t)$*, where* L *is polylogarithmic in the dimension of the Hilbert space of* H*. Given an initial state* $|\psi\rangle$ *at time* t_0*, we aim to prepare*

$$\mathcal{T} \exp\left(-i \int_{t_0}^{t_0+T} d\tau\, H(\tau)\right) |\psi\rangle \tag{4.5}$$

for a finite time T*, such that the maximum error in the final state, as quantified by the trace distance, does not exceed* ϵ*.*

4.2 Framework

4.2.1 Oracles

We consider two different representations of $H(t)$: Hamiltonians that are given by a d-sparse time-dependent Hermitian matrix (scenario 1) and Hamiltonians that are time-dependent linear combinations of some time-independent unitaries (scenario 2). While scenario 2 is already in the form required by Problem 1, we show that scenario 1 can be expressed as a linear combination of unitaries as well. The second scenario is also more general than the first one but it can help us to benefit from an additional structure of the Hamiltonians. For example, Hamiltonians in quantum

chemistry, which is one of the main applications of quantum simulations, fall into the second scenario because they are commonly expressed as linear combinations of tensor products of Pauli matrices.

Our algorithms for both scenarios rely on the ability to express the time-dependent Hamiltonian as a linear combination of efficiently implementable unitaries, similarly to [34]. Unlike [34], we need to include the time-dependence in the decomposition. We can choose to add it either in the coefficients or the unitaries. We show that either choice is possible and the preferred approach depends on the structure of the Hamiltonian.

For d-sparse matrices we need compute the decomposition of the Hamiltonian into a sum of unitaries. As such, it is more practical to take the unitaries to be time-dependent while making the coefficients in the decomposition constant. In contrast, for cases where the Hamiltonian is already given in the form of a sum of unitaries, we take the unitaries to be constant, and the time dependence is solely in the coefficients. This form is natural for applications such as quantum chemistry or adiabatic algorithms.

Let us now describe the oracles for both cases in more detail. In the first scenario, access to a d-sparse Hamiltonian is provided through the oracles

$$O_{\mathtt{loc}} |j, s\rangle = |j, \nu(j, s)\rangle \ , \tag{4.6}$$

$$O_{\mathtt{val}} |t, i, j, z\rangle = \left|t, i, j, z \oplus H_{ij}(t)\right\rangle \ , \tag{4.7}$$

where $\oplus$ represents a bitwise XOR. The above equations are a time-dependent analog of oracles (2.26) and (2.27) defined in Section 2.3. Here, $\nu(j, s)$ gives the position of the s'th element in row j of $H(t)$ that may be nonzero at any time. Note that the oracle O_{loc} does not depend on time but $O_{\mathtt{val}}$ does. Oracle O_{val} yields the values of non-zero elements at each instant of time.

Furthermore, we say that a time-dependent Hamiltonian $H(t)$ is d-*sparse on the interval* if the number of entries in any row or column that may be nonzero at any time throughout the interval is at most d. This definition is distinct from the maximum sparseness of the instantaneous Hamiltonians $H(t)$, because some entries may go to zero while others become nonzero. (This definition of sparseness upper bounds the sparseness of instantaneous Hamiltonians.)

4.2.2 Enabling Oblivious Amplitude Amplification

Our algorithm for d-sparse matrices builds on the unitary decomposition of a Hermitian matrix into equal-sized 1-sparse self-inverse parts introduced in Lemma 4.3 in Ref. [135] that we explained in Sec. 2.3.4. The Hamiltonian $H(t)$ can be decomposed using the technique of Ref. [135] for any individual time, giving

$$H(t) = \gamma \sum_{\ell=0}^{L-1} H_\ell(t), \tag{4.8}$$

where the $H_\ell(t)$ are 1-sparse, unitary and Hermitian. The matrix entries in $H_\ell(t)$ can be determined using $\mathcal{O}(1)$ queries according to Lemma 4.4 of Ref. [135]. The single coefficient γ is time-independent, and so is the sum of coefficients $\lambda := L\gamma$ in the decomposition. The value of γ should be chosen as (see Eq. (24) in [135])

$$\gamma \in \Theta(\epsilon/(d^3 T)) \tag{4.9}$$

to give a contribution to the error that is $\mathcal{O}(\epsilon)$. The unitary decomposition (4.8) can be computed with a number of Hamiltonians scaling as

$$L \in \mathcal{O}\left(d^2 H_{\max}/\gamma\right). \tag{4.10}$$

Here we defined

$$H_{\max} = \max_{t\in[t_0,t_0+T]} \|H(t)\|_{\max}. \tag{4.11}$$

We also define a norm for the derivative of the Hamiltonian

$$\dot{H}_{\max} = \max_{t\in[t_0,t_0+T]} \|dH(t)/dt\|, \tag{4.12}$$

where $\|\cdot\|$ indicates the spectral norm and $\|\cdot\|_{\max}$ the max-norm.

In the second scenario, we assume that the Hamiltonian already has the form of a unitary decomposition with time-dependent coefficients:

$$H(t) = \sum_{\ell=0}^{L-1} \alpha_\ell(t) H_\ell. \tag{4.13}$$

In this scenario, the H_ℓ are all unitary and time-independent, while each $\alpha_\ell(t)$ is assumed to be a real-valued differentiable function with modulus

upper bounded by a known constant $\alpha_{\max} \geqslant |\alpha_\ell(t)|$ for $t \in [0, T]$ and all $\ell \in \{0, \ldots, L-1\}$.

The condition $\alpha_\ell(t) \in \mathbb{R}$ can always be attained by decomposing any complex-valued coefficient into its real and imaginary parts and including the complex phase factor $\pm i$ in the associated unitary H_ℓ.

As in the first scenario, access to the Hamiltonian is provided through oracles. We assume there is an efficient procedure to implement an oracle O_{unit} that applies a single unitary from the decomposition (4.13) to the system state according to

$$O_{\text{unit}} |\ell\rangle_a |\Psi\rangle_s = |\ell\rangle_a H_\ell |\Psi\rangle_s . \tag{4.14}$$

The particular unitary H_ℓ is selected by the index value ℓ held in an ancilla register state $|\ell\rangle_a$. In one important example, the H_ℓ are composed of tensor products of Pauli matrices, in which case each such oracle query can be implemented with $\mathcal{O}(L(n + \log L))$ elementary gates; see [34]. However, we do not bind ourselves to a specific implementation and quantify the complexity only in terms of the number of oracle calls and the number of additional gates. Furthermore, we assume access to the time-dependent coefficients through a *coefficient oracle* defined as

$$O_{\texttt{coeff}} |\ell, t, z\rangle = |\ell, t, z \oplus \alpha_\ell(t)\rangle . \tag{4.15}$$

More precisely, the oracle returns a truncated version of the target coefficients as $z \oplus \alpha_\ell^{(\nu)}(t)$, where z is a ν-bit integer encoded into a ν-qubit register, and $\alpha_\ell^{(\nu)}(t) := \lfloor 2^\nu \alpha_\ell(t) \rfloor$ is a ν-bit fixed-point approximation to the real value of $\alpha_\ell(t)$.

The sum of the coefficients, $\lambda(t) := \sum_{\ell=0}^{L-1} \alpha_\ell(t)$, in the decomposition (4.13) generally depends on time, which appears to be a problem for implementing oblivious amplitude amplification [34]. Successful implementation of oblivious amplitude amplification for a given interval requires that the integral of $\lambda(t)$ over that interval should be equal to $\ln 2$. If we were to perform oblivious amplitude amplification on this decomposition directly, we would need to integrate $\lambda(t)$ to find an appropriate length of the time interval, which would increase the computational complexity. How could we avoid this complication? In our construction, we alter the unitary decomposition (4.13) such that the sum of the new coefficients is time-independent

but corresponds to the same Hamiltonian. One option to create such a decomposition is

$$\begin{aligned} H(t) &= \sum_{\ell=0}^{L-1} \frac{\alpha_{max} + \alpha_\ell(t)}{2} H_\ell + \sum_{\ell=0}^{L-1} \frac{\alpha_{max} - \alpha_\ell(t)}{2} (-H_\ell) \\ &= \sum_{\ell=0}^{2L-1} \widetilde{\alpha}_\ell(t) \widetilde{H}_\ell, \end{aligned} \tag{4.16}$$

where

$$\widetilde{\alpha}_\ell(t) := \begin{cases} \frac{\alpha_{max} + \alpha_\ell(t)}{2} & \text{for} \quad \ell = 0, \ldots, L-1 \\ \\ \frac{\alpha_{max} - \alpha_{\ell-L}(t)}{2} & \text{for} \quad \ell = L, \ldots, 2L-1 \end{cases} \tag{4.17}$$

$$\widetilde{H}_\ell := \begin{cases} H_\ell & \text{for} \quad \ell = 0, \ldots, L-1 \\ \\ -H_{\ell-L} & \text{for} \quad \ell = L, \ldots, 2L-1 \, . \end{cases} \tag{4.18}$$

This new decomposition has 2L terms and the sum of its coefficients is by construction time-independent and equal to $\lambda = L\alpha_{max}$. This allows us to satisfy the condition that is sufficient for achieving the oblivious amplitude amplification procedure globally for the entire time interval $[t_0, t_0 + T]$.

4.3 Algorithm Overview

Our algorithm consists of the following steps summarized in the diagram 4.1.

4.3.1 Evolution Discretization

Our goal is to simulate the action of U in Eq. (4.3) within error ϵ. First, we divide the total simulation time T into r time segments of length T/r. Without loss of generality, we analyze the evolution induced within the first segment and set $t_0 = 0$. The simulation of the evolution within

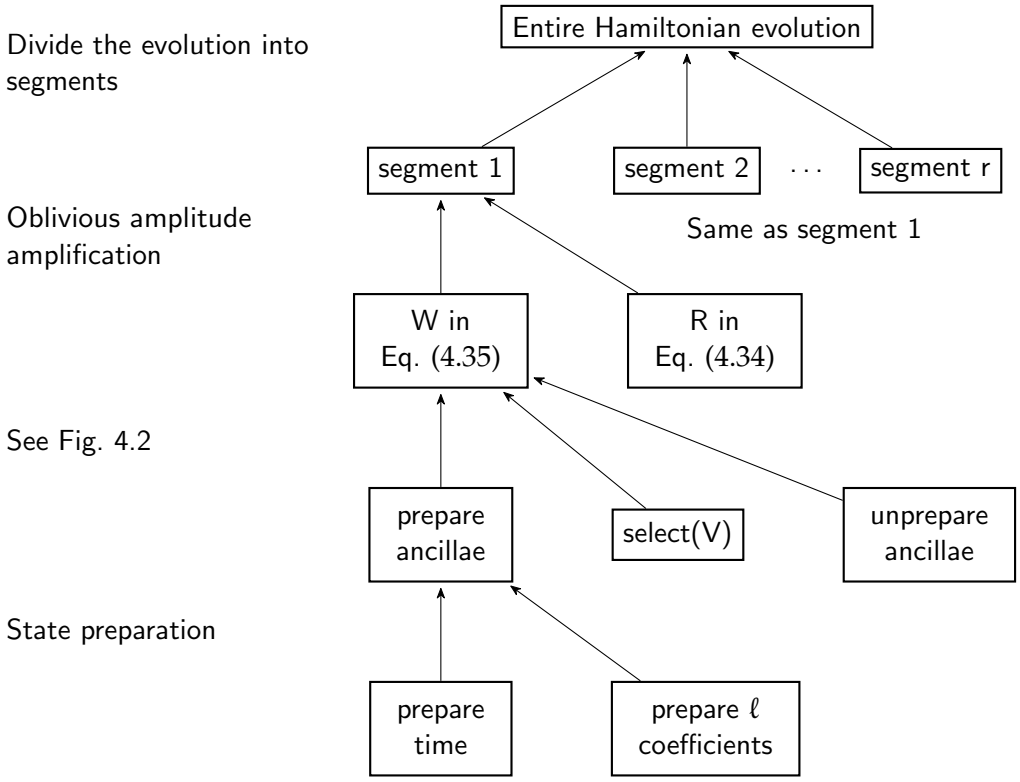


Figure 4.1: An overview of our algorithm. The top level shows the high-level structure and each level below it explains how to implement the corresponding block. We first divide the evolution into segments as outlined in Subsection 4.3.1. In each segment, we approximate the evolution operator by a linear combination of unitaries, see Subsection 4.3.2, that can be implemented using oblivious amplitude amplification. The oblivious amplitude amplification is performed using W, $W^\dagger$ and R defined in (4.35) and (4.34). While R can be trivially implemented, W requires several steps. First, we need to prepare ancillary registers that encode the discretized times (see Subsections 4.4.1 and 4.4.2), and coefficients of the linear combination (see Subsection 4.4.3). Next, we apply the appropriate Hamiltonians specified by the ancillary registers. Lastly, we reverse the preparation of ancillae and repeat the entire process for the next segment.

the following segments is accomplished in the same way. The overall complexity of the algorithm is then given by the number of segments r

times the cost of simulation for each segment. To upper bound the total simulation error by ϵ, we require the error of simulation for each segment to be at most ϵ/r. By using an LCU technique, we can choose these segments longer than for Trotterization algorithms [161].

We need a set of qubits to encode the time over the entire time interval $[0, T]$. It is convenient to take r to be a power of two, so there will be one set of qubits for the time that encodes the segment number, and another set of qubits that gives the time within the segment. The qubits encoding the segment number are only changed by incrementing by 1 between segments, which gives complexity $\mathcal{O}(r \log r)$. If it is not possible to take r as a power of two, one can choose the last segment to be shorter and use an ancilla to perform oblivious amplitude amplification as in [135].

Let us now explain the key steps for decomposing the evolution into a linear combination of unitaries. We approximate the evolution within the first segment by a Dyson series up to K-th order:

$$U(0, T/r) \approx \sum_{k=0}^{K} \frac{(-i)^k}{k!} \mathcal{T} \int_0^{T/r} d\mathbf{t}\, H(t_k) \dots H(t_1)\,, \tag{4.19}$$

where, for each k-th term in the Dyson series, $\mathcal{T} \int_0^{T/r} d\mathbf{t}\,(\cdot)$ represents integration over a k-tuple of time variables $(t_1, \dots, t_k)$ while keeping the times ordered: $t_1 \leqslant t_2 \leqslant \cdots \leqslant t_k$. It is straightforward to show that the error of this approximation is $\mathcal{O}\left(\frac{H_{max}^{K+1}}{(K+1)!}\left(\frac{T}{r}\right)^{K+1}\right)$.

Next, we discretize the integral over each time variable and approximate it by a sum with M terms. For convenience, we may choose to take M to be a power of two. The time-dependent Hamiltonian is thereby approximated by its values at M uniformly separated times $\frac{jT}{rM}$ identified by an integer $j \in \{0, \dots, M-1\}$. Approximating an integral with its Riemann sum yields

$$\begin{aligned} \int_0^{T/r} H(t)dt &= \sum_{j=0}^{M-1} \int_{t_j}^{t_{j+1}} H(t)dt \\ &\approx \sum_{j=0}^{M-1} \left(\frac{T}{rM}\right) H(t_j), \end{aligned} \tag{4.20}$$

where $t_0 = 0$ and $t_M = T/r$. The incurred error can be then bounded by $\sum_{j=0}^{M-1} \int_{t_j}^{t_{j+1}} \left\| H(t) - H(t_j) \right\| dt$.

In each of the time intervals $[t_j, t_{j+1}]$ of length $T/(rM)$ we have

$$\begin{aligned} \|H(t) - H(t_j)\| &\leqslant \frac{T}{rM} \max_z \left\| \frac{dH(z)}{dz} \right\| \\ &\leqslant \frac{T\dot{H}_{max}}{rM}, \end{aligned} \tag{4.21}$$

where $\max_z$ indicates a maximum over that time interval.

Hence, the overall error of approximation in the integral over time T/r is $\mathcal{O}\left(\frac{(T/r)^2\dot{H}_{max}}{M}\right)$. That is, the error in the $k = 1$ term for $\widetilde{U}$, and the error for terms with $k > 1$, are higher order. The error for all r segments is then $\mathcal{O}\left(\frac{T^2\dot{H}_{max}}{rM}\right)$. We remark that a slightly tighter bound in terms of a time-average of $\dot{H}(t)$ rather than in terms of $\dot{H}_{max}$ is possible, as was achieved in [37]. Replacing all integrals by sums in expression (4.19) thus yields the following approximation of the time-evolution operator within the first segment:

$$\widetilde{U} := \sum_{k=0}^{K} \frac{(-iT/r)^k}{M^k k!} \sum_{j_1,\ldots,j_k=0}^{M-1} \mathcal{T} H(t_{j_k}) \ldots H(t_{j_1}). \tag{4.22}$$

The overall error of the obtained approximation is thus

$$\|\widetilde{U} - U(0, T/r)\| \in \mathcal{O}\left(\frac{(H_{max}T/r)^{K+1}}{(K+1)!} + \frac{(T/r)^2\dot{H}_{max}}{M} \right). \tag{4.23}$$

Provided that $r \geqslant H_{max}T$, the overall error can be bounded by ϵ/r if we choose

$$K \in \Theta\left(\frac{\log(r/\epsilon)}{\log\log(r/\epsilon)} \right) \quad \text{and} \quad M \in \Theta\left(\frac{T^2\dot{H}_{max}}{\epsilon r} \right). \tag{4.24}$$

This expansion is analogous to truncated Taylor series explained in Section 2.3.6. Unlike in the time-independent algorithm [127] presented in Section 2.3.6, we need to preserve the correct order of Hamiltonians. We

achieve this by introducing an additional multi-qubit control register called `time`. This ancillary register is prepared in a certain superposition state (depending on which approach we take). We use this register to sample the Hamiltonian at different times in superposition while respecting time ordering.

4.3.2 Linear Combination of Unitaries

Substituting the unitary decomposition, as defined in (4.8) or (4.16), into (4.22), the approximation of the time-evolution operator takes the form $\tilde{U} = \sum_{j \in J} \beta_j V_j$. The index j is a multi-index and the coefficients β_j comprise information about both the time discretization and the unitary decomposition weightings as well as the order k within the Taylor series. Explicitly, we define

$$\beta^1_{(k,\ell_1,\ldots,\ell_k,j_1,\ldots,j_k)} := \frac{(\gamma T/r)^k}{M^k k_1! k_2! \ldots k_\sigma!} \theta_k(j_1,\ldots,j_k), \tag{4.25}$$

$$V^1_{(k,\ell_1,\ldots,\ell_k,j_1,\ldots,j_k)} := (-i)^k H_{\ell_k}(t_{j_k}) \ldots H_{\ell_1}(t_{j_1}), \tag{4.26}$$

when dealing with Hamiltonians given by a sparse matrix (scenario 1) giving decomposition (4.8), and

$$\beta^2_{(k,\ell_1,\ldots,\ell_k,j_1,\ldots,j_k)} := \frac{(T/r)^k}{M^k k_1! k_2! \ldots k_\sigma!} \theta_k(j_1,\ldots,j_k)\ \widetilde{\alpha}_{\ell_k}(t_{j_k}) \ldots \widetilde{\alpha}_{\ell_1}(t_{j_1}), \tag{4.27}$$

$$V^2_{(k,\ell_1,\ldots,\ell_k,j_1,\ldots,j_k)} := (-i)^k H_{\ell_k} \ldots H_{\ell_1}, \tag{4.28}$$

when dealing with scenario 2 Hamiltonians given by decomposition (4.13), where $\theta_k(j_1,\ldots,j_k) = 1$ if $j_1 \leqslant j_2 \leqslant \ldots \leqslant j_k$, and zero otherwise. The quantity σ is the number of distinct values of j, and $k_1, k_2, \ldots, k_\sigma$ are the number of repetitions for each distinct value of j. That is, we have the indices j for the times sorted in ascending order, and we have multiplied by a factor of $k!/(k_1! k_2! \ldots k_\sigma!)$ to take account of the number of unordered sets of indices which give the same ordered set of indices. The multi-index set J is defined as

$$\begin{aligned} J^1 := \{(k, \ell_1, \ldots, \ell_k, j_1, \ldots, j_k) &: k \in \{0,\ldots,K\},\ \ell_1,\ldots,\ell_k \in \{0,\ldots,L-1\}, \\ &j_1,\ldots,j_k \in \{0,\ldots,M-1\}\}, \end{aligned} \tag{4.29}$$

or

$$J^2 := \{(k, \ell_1, \ldots, \ell_k, j_1, \ldots, j_k) \,:\, k \in \{0, \ldots, K\},\ \ell_1, \ldots, \ell_k \in \{0, \ldots, 2L-1\},\\ j_1, \ldots, j_k \in \{0, \ldots, M-1\}\}. \tag{4.30}$$

The only difference is the second has 2L rather than L, where we expanded the sum to ensure the λ is independent of time. It will be convenient (though not necessary) to take both L and M to be powers of two, so equal-weight superpositions can be produced with tensor products of Hadamard gates, for example $H^{\otimes \log L}$ when preparing superposition states $\frac{1}{\sqrt{L}} \sum_{\ell=0}^{L-1} |\ell\rangle$.

We use a standard technique to implement linear combinations of unitaries (see Section 2.3.6) involving the use of an ancillary register to encode the coefficients β_j [135]. In the next section, we present two approaches to implement the (multi-qubit) ancilla state preparation

$$B\,|0\rangle_a = \frac{1}{\sqrt{s}} \sum_{j \in J} \sqrt{\beta_j}\,|j\rangle_a \tag{4.31}$$

as part of the LCU approach, where $s := \sum_{j \in J} \beta_j$. Analogously to the time-independent simulation in Section 2.3.6, the operator

$$\textsc{Select}(V) := \sum_j |j\rangle\langle j|_a \otimes V_j \tag{4.32}$$

acts as

$$\textsc{Select}(V)\,|j\rangle_a\,|\Psi\rangle_s = |j\rangle_a\,V_j\,|\Psi\rangle_s \tag{4.33}$$

on the joint ancilla and system states. This operation implements a term from the decomposition of $\widetilde{U}$ *selected* by the ancilla state $|j\rangle_a$ with weight β_j. Following the method in Ref. [34], we also define

$$R := \mathbb{I}_{as} - 2\,(|0\rangle\langle 0|_a \otimes \mathbb{I}_s)\,, \tag{4.34}$$

$$W := \left(B^\dagger \otimes \mathbb{I}_s\right) \textsc{Select}(V)\,(B \otimes \mathbb{I}_s)\,. \tag{4.35}$$

If $\widetilde{U}$ were unitary and $s \leqslant 2$, a single step of oblivious amplitude amplification could be used to implement $\widetilde{U}$. When $\widetilde{U}$ is only approximately unitary, with $\|\widetilde{U} - U(0, T/r)\| \in \mathcal{O}(\epsilon/r)$, a single step of oblivious amplitude amplification yields [34]

$$-WRW^\dagger RW\,|0\rangle_a\,|\Psi\rangle_s = |0\rangle_a\,\tilde{U}\,|\Psi\rangle_s + \mathcal{O}\,(\epsilon/r)\,, \tag{4.36}$$

which is the approximate transformation we aim to implement for each time segment.

The implementation of the unitary transformation W by a quantum circuit is illustrated in Fig. 4.2. It generalizes the LCU technique to time-dependent decompositions. The circuit for scenario 1 is given in Fig. 4.2a and the circuit for scenario 2 is depicted in Fig. 4.2b. In both scenarios, we employ three auxiliary control registers in addition to the system register holding the quantum state on which Hamiltonian evolution is applied. The `k register`, which consists of K qubits, is used to hold the value of k corresponding to the order in the truncated Dyson series encoded in unary. In what follows, we denote the state $\left|1^k 0^{K-k}\right\rangle$ simply as $|k\rangle$. The `time register` consists of K subregisters each of size $\log M$. The `time register` is prepared in a special superposition state $|\text{clock}\rangle$ that also involves the `k register`. These registers are used for sampling the Hamiltonian at different points at a time. Note that the same $|\text{clock}\rangle$ state and the two alternative approaches to its preparation presented in Section 4.4.1 can be used for both considered scenarios. Finally, the `l` register consisting of K subregisters each of size $\log L$ is used to prepare the ancillary register states commonly needed to implement the LCU technique. Its task is to select which term out of the decomposition into unitaries is to be applied. As this controlled selection ought to occur with amplitudes corresponding to the weightings of the involved unitaries in the decomposition, the ancillary `l` register states must also encode those amplitudes. All unitaries in the decomposition (4.8) for scenario 1 have equal weight; thus only equal-weight superpositions need to be prepared in the `l` register in this case. For decompositions (4.13) of scenario 2 Hamiltonians, however, we need to prepare superposition states that encode the amplitudes of the time-dependent weightings $\widetilde{\alpha}_\ell(t)$ according to the unitary transformation

$$\begin{aligned}\text{controlled-PREP}(\alpha)\,|t\rangle_{\texttt{time}_k}|0\rangle_{\texttt{l}_k} &:= \Big(\mathbb{I}\otimes \text{PREP}\big(\alpha(t)\big)\Big)\,|t\rangle_{\texttt{time}_k}\,|0\rangle_{\texttt{l}_k} \\ &:= \frac{1}{\sqrt{L\alpha_{\max}}}\sum_{\ell=0}^{2L-1}\sqrt{\widetilde{\alpha}_\ell(t)}\,|t\rangle_{\texttt{time}_k}\,|\ell\rangle_{\texttt{l}_k}\ .\end{aligned} \tag{4.37}$$

This is a controlled state preparation involving the k-th `l`-subregister as target and the k-th `time` subregister as control, to be executed for each $k = 1,\ldots,K$. By construction, $\text{PREP}(\alpha)$ encodes the coefficients

of the altered (but equivalent) decomposition (4.16). These coefficients have been introduced to achieve a time-independent sum of coefficients, $\lambda = \sum_{\ell=0}^{2L-1} \widetilde{\alpha}_\ell(t) = L\alpha_{\max}$ for all $t \in [0, T]$, in order to fulfill the condition for oblivious amplitude amplification. While the new decomposition (4.16) has twice as many coefficients as the original decomposition (4.13), the cost of implementing the LCU method for the altered decomposition raises only by one additional ancilla qubit, as the selection ranges $\ell = 0, \ldots, L-1$ and $\ell = L, \ldots, 2L-1$ can be differentiated by conditioning on just a single qubit.

The SELECT(V) operation is implemented by a series of K controlled-SELECT(H) transformations, whose action is given by

$$\begin{aligned} &\text{controlled-SELECT}(H) \left|b\right\rangle_{\mathtt{k}_k} \left|t\right\rangle_{\mathtt{time}_k} \left|\ell\right\rangle_{\mathtt{l}_k} \left|\Psi\right\rangle_{\mathtt{s}} \\ &:= \left|b\right\rangle_{\mathtt{k}_k} \left|t\right\rangle_{\mathtt{time}_k} \left|\ell\right\rangle_{\mathtt{l}_k} \left(-iH_\ell(t)\right)^b \left|\Psi\right\rangle_{\mathtt{s}} \end{aligned} \tag{4.38}$$

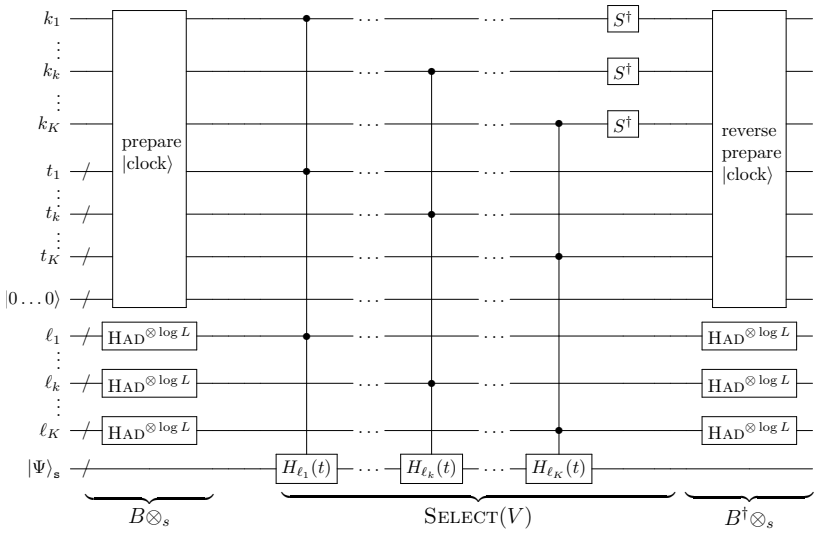
for decompositions (4.8) corresponding to scenario 1, and by

$$\text{controlled-SELECT}(H) \left|b\right\rangle_{\mathtt{k}_k} \left|\ell\right\rangle_{\mathtt{l}_k} \left|\Psi\right\rangle_{\mathtt{s}} := \left|b\right\rangle_{\mathtt{k}_k} \left|\ell\right\rangle_{\mathtt{l}_k} \left(-i\widetilde{H}_\ell\right)^b \left|\Psi\right\rangle_{\mathtt{s}} \tag{4.39}$$

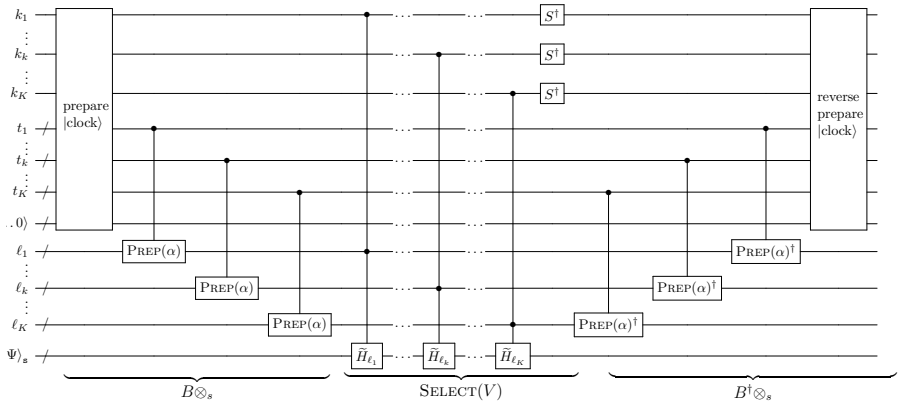
for decompositions (4.16) corresponding to scenario 2. Note that the control on the $\mathtt{time}_k$ register is only necessary in the case of sparse-matrix decompositions, but not for scenario 2 type Hamiltonians, since in the latter case the individual unitaries H_ℓ in the decomposition are all time-independent. The phase factor $(-i)^b$ has to be implemented separately by applying the phase gate $S^\dagger := |0\rangle\langle 0| + (-i)\,|1\rangle\langle 1|$ to each qubit of the $\mathtt{k}$ register. Since the value of k is encoded in unary, applying S on all K wires of the $\mathtt{k}$ register results in the overall factor $(-i)^k$.

4.4 Preparation of Auxiliary Registers

The algorithm relies on efficient implementation of the unitary transformation B, which acts on the composite ancilla registers $\mathtt{k} \otimes \mathtt{time} \otimes \mathtt{l}$ and prepares a joint superposition state given in Eq. (4.31). This state preparation includes encoding the Dyson order k (in register $\mathtt{k}$), the values $j_1 \leqslant \cdots \leqslant j_k$ specifying the ordered time instances (in register $\mathtt{time}$), and the index values $\ell_1, \ldots, \ell_k$ specifying the terms in the Hamiltonian decompositions (in register $\mathtt{l}$), in superposition over all possible combinations of



(a) Hamiltonian is given as a time-dependent sparse matrix (scenario 1)



(b) Hamiltonian is given as a time-dependent linear combination of time-independent unitaries (scenario 2).

Figure 4.2: Quantum circuit implementing the unitary transformation W (see definition in Eq. (4.35)) for the two alternative scenarios from [3]. In both cases, W is composed of a SELECT(V) operation consisting of a series of K controlled-SELECT(H) transformations sandwiched between the operations B and $B^\dagger$ for the preparation and its reverse of the auxiliary register states required for implementing the LCU technique. The boxed subroutine "prepare $|\text{clock}\rangle$" acting on $\mathtt{k} \otimes \mathtt{time}$ is implemented either by the compressed encoding approach outlined in Subsection 4.4.1, or by using a quantum sorting network described in Subsection 4.4.2. Both approaches require additional ancillae indicated by $|0\ldots0\rangle$. The phase gate $S^\dagger = |0\rangle\langle 0| + (-i)\,|1\rangle\langle 1|$ is used to implement the factor $(-i)^k$ as part of the k-th order term of the Dyson series.

the values, and with amplitudes that are the square roots of the weightings $\beta^1_{(k,\ell_1,\ldots,\ell_k,j_1,\ldots,j_k)}$ or $\beta^2_{(k,\ell_1,\ldots,\ell_k,j_1,\ldots,j_k)}$. As noted above, the main difficulty of simulating time-dependent Hamiltonian dynamics is the implementation of time ordering. This is achieved by a weighted superposition named clock prepared as the joint state in the composite auxiliary control register $\mathtt{k} \otimes \mathtt{time}$, which has been introduced in addition to the ancilla register $\mathtt{l}$ used to implement the LCU technique in the time-independent case. Its main purpose and task are to control sampling the Hamiltonian for different instances of time in superposition in a way that respects the correct time order. We here present two alternative approaches to efficient preparation of such clock states in the composite $\mathtt{k} \otimes \mathtt{time}$ register.

The first approach is based upon generating the intervals between successive times according to an exponential distribution, in a similar way as in Ref. [136]. These intervals are then summed to obtain the times. Our second approach starts by creating a superposition over k and then a superposition over all possible times $t_1, \ldots, t_k$ for each k. The times are then reversibly sorted using quantum sorting networks [2, 142, 172] and used to control ℓ as in the previous approach.

4.4.1 Clock Preparation Using Compressed Rotation Encoding

To explain the first approach, it is convenient to consider a conceptually simple but inefficient representation of the time register. An ordered sequence of times $t_1, \ldots, t_k$ is encoded in a binary string $x \in \{0,1\}^M$ with Hamming weight $|x| = k$, where $0 \leqslant k \leqslant K$. That is, if m_j is the j'th value of m such that $x_m = 1$, then $t_j = m_j T/(rM)$. This automatically forces the ordering $t_1 < t_2 < \cdots < t_k$, omitting the terms when two or more Hamiltonians act at the same time. The binary string x then would be encoded into M qubits as $|x\rangle$.

Now consider these M qubits initialized to $|0\rangle^{\otimes M}$, then rotated by a

small angle, to give

$$\begin{aligned}(\alpha|0\rangle+\beta|1\rangle)^{\otimes M} &= \sum_{x} \alpha^{M-|x|}\beta^{|x|}|x\rangle \\ &= \sum_{x,|x|\leqslant K} \alpha^{M-|x|}\beta^{|x|}|x\rangle + \sum_{x,|x|>K} \alpha^{M-|x|}\beta^{|x|}|x\rangle \\ &= \sqrt{1-\mu^2}\,|\Omega\rangle + \mu\left|\Omega^{\perp}\right\rangle, \end{aligned} \tag{4.40}$$

where $\alpha := \sqrt{\frac{rM}{\lambda T+rM}}$ and $\beta := \sqrt{\frac{\lambda T}{\lambda T+rM}}$. The state $|\Omega\rangle$ encodes the sequences of times when the Hamiltonians are applied in Eq. (4.22) in the discretized Dyson series up to the K-th order with the weighting as in Eq. (4.25)

$$|\Omega\rangle = \frac{1}{\sqrt{S}} \sum_{|x|\leqslant K} \left(\frac{\lambda T}{rM}\right)^{|x|/2} |x\rangle, \tag{4.41}$$

with $S := \sum_{|x|\leqslant K} \left(\frac{\lambda T}{rM}\right)^{|x|}$. Recall that the Hamming weight $|x|$ corresponds to the order $0 \leqslant k \leqslant K$ within the truncated Dyson series. The states $|\Omega\rangle$ and $\left|\Omega^{\perp}\right\rangle$ are normalized and orthogonal. The state $\left|\Omega^{\perp}\right\rangle$ is analogous to the higher-order terms omitted in the Dyson series. The amplitude μ satisfies (as shown in [136]):

$$\mu^2 = \mathcal{O}\left(\frac{(\lambda T/r)^{K+1}}{(K+1)!}\right). \tag{4.42}$$

We choose K as in Eq. (4.24), and the choice $r > \lambda T$ implies $\mu^2 = \mathcal{O}(\epsilon/r)$.

Since $|\Omega\rangle$ includes only Hamming weights up to K, it contains highly compressible strings. The lengths of the strings of zeros between successive ones are stored in order to compress the string x. That is, a string $x = 0^{s_1}10^{s_2}10^{s_3}\ldots0^{s_k}10^{\sigma}$ can be represented by the integers $s_1 s_2 \ldots s_k$. There is always a total of $K+1$ entries, regardless of the Hamming weight. In addition, we encode the Hamming weight k in an additional register. Thus, our encoding converts x into

$$\begin{aligned}\Xi|0^{s_1}10^{s_2}10^{s_3}\ldots0^{s_k}10^{\sigma}\rangle = &|s_1+1\rangle\,|s_1+s_2+2\rangle\ldots \\ &|s_1+s_2+\cdots+k+1\rangle\left|\text{junk}_x\right\rangle|k\rangle, \end{aligned} \tag{4.43}$$

where $\left|\text{junk}_x\right\rangle$ is a remnant of the construction as well as a padding for a string with Hamming distance smaller than K. For example, for $K = 4$, the state $|001000001000\rangle$ would be encoded as $|2, 5\rangle \left|\text{junk}_x\right\rangle |2\rangle$. Following Eq. (17) from [136], we define a more complex encoding B_M^K as

$$B_M^K |x\rangle = |s_1, \ldots, s_k\rangle \left(\sum_{j=0}^{q-\sigma-1} \alpha^j \beta |j+\sigma\rangle + \alpha^{q-\sigma} |q\rangle \right) |\phi_q\rangle^{\otimes(K-k)}, \quad (4.44)$$

where $|\phi_q\rangle = \sum_{s=0}^{q-1} \beta \alpha^s |s\rangle + \alpha^q |q\rangle$ and we need to take $q = M$ for the encoding here. According to Theorem 3 in [136],

$$|\phi_q\rangle^{\otimes K+1} = \sum_{x,|x|\leqslant K} \alpha^{M-|x|} \beta^{|x|} B_M^K |x\rangle + \mu \left|\Omega'\right\rangle. \quad (4.45)$$

The idea is that one prepares the state $|\phi_q\rangle^{K+1}$ (see Eq. (13) of [136]) which gives an exponential distribution. Encoding $B_M^K |x\rangle$ includes spacing between consecutive ones, the term in the bracket representing $\left|\text{junk}_x\right\rangle$ and the necessary padding of $|\phi_q\rangle^{\otimes(K-k)}$. However, we require the absolute times, rather than the differences between the times, as well as an additional register encoding k. The state $B_M^K |x\rangle$ can be converted into $\Xi |x\rangle$ following the steps 1 and 2 of Section 4.4 in [136]. First, one computes the absolute position of 1's by computing the prefix sums. Then, it is possible to identify the register $k+1$ by finding the first register larger than m using the term in the bracket as an indicator of overflow following the same steps as in [136]. The Hamming weight k is then recorded in an additional register. Unlike [136], there is no need to clean the $\left|\text{junk}_x\right\rangle$ register or uncompute the Hamming weight.

4.4.2 Clock Preparation Using a Quantum Sort

In this section, we explain an alternative approach to implementing the preparation of the $|\text{clock}\rangle_{k\otimes\texttt{time}}$ state, which establishes time ordering via a reversible sorting algorithm. This approach first creates a superposition over all Dyson series orders $k \leqslant K$ with amplitudes proportional to $\left(\zeta^k/k!\right)^{1/2}$, where $\zeta := \lambda T/(rM)$. It then generates a superposition over all

possible k-tuples of times $t_1, \ldots, t_k$ for each possible value of k. To achieve time-ordered combinations, sorting is applied to each of the tuples in the superposition using a quantum sorting network algorithm.

The clock preparation requires $\log M$ qubits to encode each value t_j (via the integer j). As $k \leqslant K$, the `time` register thus consists of $K \log M$ ancilla qubits. The value of k is encoded in unary into K additional qubits constituting the `k`-register. Additionally, further $\mathcal{O}\left(K\text{poly}(\log K)\right)$ ancillae are required to reversibly perform a quantum sort. The overall space complexity thus amounts to $\mathcal{O}\left(K \log M + K\text{poly}(\log K)\right)$.

Specifically, in the first step we create a superposition over the allowed values of k in unary encoding by applying the following transformation to K qubits initialized to $|0\rangle$:

$$\textsc{Prepare}(k)\,|0\rangle^{\otimes K} := \frac{1}{\mathcal{N}} \sum_{k=0}^{K} \sqrt{\frac{\zeta^k M^k}{k!}} \left|1^k 0^{K-k}\right\rangle, \tag{4.46}$$

where the constant $\mathcal{N} := \left[\sum_{k=0}^{K} \frac{\zeta^k M^k}{k!}\right]^{1/2}$ accounts for normalization. This transformation can be easily implemented (see Section 4 of Ref. [72] for details) using a series of $\mathcal{O}(K)$ controlled rotations, namely, by first rotating the first qubit followed by rotations applied to qubits $k = 2$ to K controlled by qubit $k-1$. We use it to determine which of the times t_j satisfy the condition $j \leqslant k$.

Next, we wish to create an equal superposition over the time indices j. The preparation can be performed via the Hadamard transforms $H^{\otimes \log M}$ on all K subregisters of `time` (each of size $\log M$) to generate the superposition state

$$\frac{1}{\mathcal{N}} \sum_{k=0}^{K} \sqrt{\frac{\zeta^k M^k}{M^K k!}} |k\rangle \sum_{j_1=0}^{M-1} \sum_{j_2=0}^{M-1} \cdots \sum_{j_K=0}^{M-1} |j_1, j_2, \ldots, j_K\rangle \tag{4.47}$$

using $\mathcal{O}(K \log M)$ gates. At this stage, we have created all possible K-tuples of times without accounting for time-ordering. Note that this superposition is over all possible K-tuples of times, not only $k \leqslant K$ of them. We have a factor of M^k/M^K, but for any k in the superposition we ignore the times in subregisters $k+1$ to K. Hence the amplitude for $|j_1, j_2, \ldots, j_k\rangle$ is $\propto \sqrt{\zeta^k/k!}$.

The next step is to sort the values stored in the `time` subregisters. Common classical sorting algorithms are often inappropriate for quantum algorithms because they involve actions on registers that depend on the values found in earlier steps of the algorithm. Sorting techniques that are suitable to be adapted to quantum algorithms are *sorting networks*, because they involve actions on registers in a sequence that is independent of the values stored in these registers. We discussed methods for adapting sorting networks to quantum algorithms in the Chapter 3.

For each $|k\rangle$ in the superposition (4.47), we only want to sort the values in the first k subregisters of `time`. We could perform K sorts for each value of k, or control the action of the comparators such that they perform the SWAP only for subregisters with positions up to value k. A more efficient approach is to sort all the registers regardless of the value of k, and also perform the same controlled-swaps on the registers encoding the value of k in unary. This means that the qubits encoding k still indicate whether the corresponding time register includes a time we wish to use. The controlled $H_\ell(t)$ operations are controlled on the individual qubits encoding k, and will still be performed correctly.

There are many possible sorting networks we can use, for example, the bitonic sort in Fig. 3.12. Since we need to record the positions of the first k registers as well, we perform the same controlled-swap on the k-register too. The bitonic sort requires $\mathcal{O}(K \log^2 K)$ comparisons, but there are more advanced sorting networks that use $\mathcal{O}(K \log K)$ comparisons [178]. That brings the complexity of clock preparation to $\mathcal{O}(K \log K \log M)$ elementary gates. Since each comparison requires a single ancilla, the space overhead is $\mathcal{O}(K \log K)$ ancillae.

4.4.3 Completing the State Preparation

To complete the state preparation, we transform each of the K `l`-subregisters for encoding $\ell_1, \ldots \ell_K$ from $|0\rangle$ into specific superposition states, depending on the representation of the given Hamiltonian.

In the sparse matrix scenario 1, we only need to prepare equal-weight superposition states. If L is a power of two (as we already specified), a Hadamard transform suffices. The complexity is then $\mathcal{O}(K \log L)$ single-qubit operations.

In scenario 2, the target superposition states for the $\mathtt{l}$-subregisters to be encoding $\ell_1, \ldots \ell_K$ are given by (4.37). We propose preparing the required target superposition states using a similar approach as in Ref. [116], namely by transducing the computed values $\widetilde{\alpha}_\ell(t)/\alpha_{\max}$ (using the output of oracle $O_{\mathtt{coeff}}$) into quantum amplitudes $\sqrt{\widetilde{\alpha}_\ell(t)/\alpha_{\max}}$ via 'inequality testing'. According to that approach, the state preparation requires the use of several registers. For each $k = 1, \ldots, K$, in addition to the $\mathtt{l}_k$ subregister for encoding the index value ℓ_k to identify the coefficients in the decomposition (4.16) and the $\mathtt{time}_k$ subregister for encoding a particular time t_k as part of the $|\text{clock}\rangle$ state, we introduce two further registers named '$\mathtt{coeff}_k$' and '$\mathtt{ref}_k$', each consisting of ν qubits. Here, ν is the number of bits used for the ν-bit fixed-point approximations $\widetilde{\alpha}_\ell^{[\nu]}(t) := \lfloor 2^\nu \widetilde{\alpha}_\ell(t)/\alpha_{\max} \rfloor$. Finally, an additional single-qubit ancilla called '$\mathtt{flag}$' is required to distinguish between the two index ranges $\ell = 0, \ldots, L-1$ and $\ell = L, \ldots, 2L-1$, and thus indicate which of the corresponding two alternative amplitudes $\left(\frac{\alpha_{\max}+\alpha_\ell(t)}{2}\right)^{1/2}$ and $\left(\frac{\alpha_{\max}-\alpha_\ell(t)}{2}\right)^{1/2}$ has been generated.

The state preparation by inequality testing can be outlined as follows. For more details, see Ref. [116]. We first create uniform superpositions in both the $\mathtt{l}_k$ and the $\mathtt{ref}_k$ register. We then compute a ν-bit fixed-point approximation $\widetilde{\alpha}_\ell^{[\nu]}(t)$ of the modified coefficient $\widetilde{\alpha}_\ell(t)/\alpha_{\max}$ at time t specified by the $\mathtt{time}_k$ subregister. We obtain the value $\widetilde{\alpha}_\ell^{[\nu]}(t)$ by querying the oracle $O_{\mathtt{coeff}}$ defined in Eq. (4.15) and save it in register $\mathtt{coeff}_k$. These first two steps can be formalized by the following state transformations:

$$
\begin{aligned}
&|0\rangle_{\mathtt{l}_k} |t\rangle_{\mathtt{time}_k} |0\rangle_{\mathtt{coeff}_k} |0\rangle_{\mathtt{ref}_k} |0\rangle_{\mathtt{flag}_k} \\
&\xrightarrow{H^{\otimes(\nu+\log L)}|\cdot\rangle_{\mathtt{l}_k}|\cdot\rangle_{\mathtt{ref}_k}} \frac{1}{\sqrt{L}\sqrt{2^\nu}} \sum_{\ell=0}^{L-1} \sum_{x=0}^{2^\nu-1} |\ell\rangle_{\mathtt{l}_k} |t\rangle_{\mathtt{time}_k} |0\rangle_{\mathtt{coeff}_k} |x\rangle_{\mathtt{ref}_k} |0\rangle_{\mathtt{flag}_k} \\
&\xrightarrow{O_{\mathtt{coeff}}|\cdot\rangle_{\mathtt{l}_k}|\cdot\rangle_{\mathtt{time}_k}|\cdot\rangle_{\mathtt{coeff}_k}} \frac{1}{\sqrt{L}\sqrt{2^\nu}} \sum_{\ell=0}^{L-1} \sum_{x=0}^{2^\nu-1} |\ell\rangle_{\mathtt{l}_k} |t\rangle_{\mathtt{time}_k} \left|\widetilde{\alpha}_\ell^{[\nu]}(t)\right\rangle_{\mathtt{coeff}_k} |x\rangle_{\mathtt{ref}_k} |0\rangle_{\mathtt{flag}_k} .
\end{aligned}
\tag{4.48}
$$

Creating the uniform superpositions in the first step requires only $\nu + \log L$ elementary gates. Computing $\widetilde{\alpha}_\ell^{[\nu]}(t)$ requires only one query to oracle

O_{coeff}. In addition, we need $\mathcal{O}(\nu)$ gates to perform the arithmetic operations needed to obtain $\widetilde{\alpha}_\ell(t)/\alpha_{\max}$. To avoid having to perform generic division (which requires $\mathcal{O}(\nu^2)$ gates), we can take the bound $\alpha_{\max}$ to be a power of 2.

The state preparation then proceeds by applying the operation COMPARE defined in Chapter 3 to the computational registers $\texttt{coeff}_k$ (holding a computed coefficient value $\widetilde{\alpha}_\ell^{[\nu]}(t)$) and $\texttt{ref}_k$ (holding a value $x \in \{0, \ldots, 2^\nu - 1\}$) and flagging the result of the comparison operation (inequality test) by the `flag` qubit:

$$\xrightarrow{\text{COMPARE}\,|\cdot\rangle_{\texttt{coeff}_k}|\cdot\rangle_{\texttt{ref}_k}} \frac{1}{\sqrt{L}}\sum_{\ell=0}^{L-1}|\ell\rangle_{\texttt{l}_k}|t\rangle_{\texttt{time}_k}\left|\widetilde{\alpha}_\ell^{[\nu]}(t)\right\rangle_{\texttt{coeff}_k}\otimes$$
$$\left(\frac{1}{\sqrt{2^\nu}}\sum_{x=0}^{\widetilde{\alpha}_\ell^{[\nu]}(t)-1}|x\rangle_{\texttt{ref}_k}|1\rangle_{\texttt{flag}_k}+\frac{1}{\sqrt{2^\nu}}\sum_{x=\widetilde{\alpha}_\ell^{[\nu]}(t)}^{2^\nu-1}|x\rangle_{\texttt{ref}_k}|0\rangle_{\texttt{flag}_k}\right). \tag{4.49}$$

The state within the brackets of (4.49) is normalized with an amplitude approximately equal to $\sqrt{\frac{\alpha_{\max}+\alpha_\ell(t)}{2\alpha_{\max}}}$ on $|1\rangle_{\texttt{flag}_k}$ and amplitude approximately equal to $\sqrt{\frac{\alpha_{\max}-\alpha_\ell(t)}{2\alpha_{\max}}}$ on $|0\rangle_{\texttt{flag}_k}$, up to error $\mathcal{O}(2^{-\nu})$. This error must be bounded by $\epsilon/(Kr)$, implying $\nu = \Theta\left(\log\frac{Kr}{\epsilon}\right)$. Hence, creating the target superposition states for all K subregisters requires $\mathcal{O}(K)$ oracle queries and

$$\mathcal{O}\left(K\left[\log\left(Kr/\epsilon\right)+\log L\right]\right) \tag{4.50}$$

additional elementary gates. We do not need to implement the full algorithm of Ref. [116] because we do not require erasing the ancillary registers immediately or to perform amplitude amplification. Indeed, we can leave the basis states $|\ell\rangle$ entangled with ancillae similar to state preparation for LCU given in [73], although the latter is different from our approach in that it uses a classical database for amplitudes rather than a quantum oracle. ' The state given in Eq. (4.49) is already operationally equivalent (up to error $\epsilon/(Kr)$) to our target state (4.37), if we regard $\texttt{flag}_k$ as part of an extended `l`-register, while we ignore the entangled registers $\texttt{coeff}_k$ and $\texttt{ref}_k$. The entanglement with the latter registers does not affect the subsequent controlled-SELECT(H) operations. Hence, the uncomputation of registers $\texttt{coeff}_k$ and $\texttt{ref}_k$ may indeed be deferred to the stage when

reversing the above state preparation as part of $B^\dagger$ after completing the SELECT(V) operation.

Next, let us consider the states prepared as a result of these procedures. In the first case, where we prepare the superposition of times by the compressed rotations, we first prepare the state $|\phi_q\rangle^{\otimes K+1}$, which contains the separations between successive times, then add these to obtain the times. The state is therefore

$$\sqrt{\frac{1-\mu^2}{S}} \sum_{k=0}^{K} \zeta^{k/2} |k\rangle \sum_{j_1<j_2\ldots<j_k} |j_1,\ldots,j_k\rangle + \mu \left|\nu''\right\rangle . \tag{4.51}$$

The preparation of the registers encoding $\ell_1,\ldots\ell_K$ gives a factor of $1/L^{K/2}$ for both considered Hamiltonian models. For $k < K$, the registers past k are not used, and the effective amplitude factor is thus $1/L^{k/2}$. We obtain the indices j_1 to j_k in sorted order without repetitions. This means that the weightings of the terms in the sum are $(\gamma T/(rM))^{k/2}$ for scenario 1 and $(\alpha_{max} T/(rM))^{k/2}$ for scenario 2. In Eq. (4.25), when there are no repetitions the $k_1,\ldots,k_\sigma$ are all 1. Therefore we have approximately the desired state preparation from Eq. (4.31) with the correct weightings, and with $s = S/(1-\mu^2)$. There is imprecision of $\mathcal{O}(\epsilon/r)$ due to the additional term weighted by μ, as well as imprecision due to the omitted repetitions, which are bounded in Section 4.5 below.

For amplitude amplification to take one step, we require

$$s = S/(1-\mu^2) \leqslant 2. \tag{4.52}$$

Note that it can be less than 2, and oblivious amplitude amplification can still be performed in a single step using an ancilla qubit, as noted in the Segment Lemma of Ref. [135]. To bound the value of S,

$$\begin{aligned} S &= \sum_{|x|\leqslant K} \zeta^{|x|} \\ &= \sum_{k=0}^{K} \binom{M}{k} \zeta^k \\ &< \sum_{k=0}^{\infty} \frac{M^k \zeta^k}{k!} \\ &= e^{M\zeta} = e^{\lambda T/r}. \end{aligned} \tag{4.53}$$

Therefore, by choosing $r \geqslant \lambda T / \ln[2(1-\mu^2)]$ we can ensure that $s \leqslant 2$, and a single step of amplitude amplification is sufficient. The value of μ^2 is $\mathcal{O}(\epsilon/r)$, and therefore $r = \Theta(\lambda T)$.

In the second case, where we obtain the superposition over the times via a sort, the state is as in Eq. (4.47) except sorted, with information about the permutation used for the sort in an ancilla. When there are repeated indices j, the number of initial sets of indices that yield the same sorted set of indices is $k!/(k_1!k_2!\ldots k_\sigma!)$, using the same notation as in Eq. (4.25). That means for each sorted set of j there is a superposition over this many states in the ancilla with equal amplitude, resulting in a multiplying factor of $\sqrt{k!/(k_1!k_2!\ldots k_\sigma!)}$ for each sorted set of j.

As noted above, because the times t_{k+1} to t_K are ignored, the factors of M in the amplitude cancel. Similarly, when we prepare the state for the registers encoding $\ell_1, \ldots \ell_K$, we obtain a factor of $1/L^{k/2}$. Including these factors, and the factor for the superposition of states in the ancilla, we obtain amplitude $(\gamma T/(rM))^{k/2}/\sqrt{k_1!k_2!\ldots k_\sigma!}$ for scenario 1 and amplitude $(\alpha_{\max} T/(rM))^{k/2}/\sqrt{k_1!k_2!\ldots k_\sigma!}$ for scenario 2. Hence we have the desired state preparation from Eq. (4.31) with the correct amplitudes, and the same s.

We require $s \leqslant 2$ for oblivious amplitude amplification to take one step. We can bound s via

$$\begin{aligned} s &= \sum_{k=0}^{K} \frac{M^k \zeta^k}{k!} \\ &< \sum_{k=0}^{\infty} \frac{M^k \zeta^k}{k!} \\ &= e^{M\zeta} = e^{\lambda T/r}. \end{aligned} \tag{4.54}$$

Therefore, by choosing $r \geqslant \lambda T / \ln 2$ we can ensure that $s \leqslant 2$, and a single step of amplitude amplification is sufficient. Note that it is convenient to take r to be a power of two, so the qubits encoding the time may be separated into a set encoding the segment number and another set encoding the time within the segment. Hence, in either case we choose $r = \Theta(\lambda T)$.

4.5 Complexity Requirements

We now summarize the resource requirements of all the components of the algorithm and provide the overall complexity. We start with elements that are necessary for both scenario 1 sparse matrices and quantum type 2 Hamiltonians and then discuss the complexities of their unique parts.

The full quantum circuit consists of r successively executed blocks, one for each of the r time segments. Since we took $r = \Theta(\lambda T)$ for oblivious amplitude amplification, the overall complexity is multiplied by a factor of λT.

Each segment requires one round of oblivious amplitude amplification, which includes two reflections R, two applications of W and one application of the inverse $W^\dagger$, as in Eq. (4.36). The cost of reflections is negligible compared to that of W. Hence, the overall cost of the algorithm amounts to $\mathcal{O}(r)$ times the cost of transformation W, whose quantum circuit is depicted in Fig. 4.2. For scenario 1, implementing W requires the procedure to prepare the $|\text{clock}\rangle$ state and its inverse, $2K$ applications of $H^{\otimes \log L}$ and K controlled applications of unitaries $H_\ell(t)$. For scenario 2, implementing W requires likewise preparing the $|\text{clock}\rangle$ state and the reverse of that procedure, $2K$ applications of the controlled-$\text{PREP}(\alpha)$ subroutine, and K controlled applications of unitaries H_ℓ.

Choosing K as in Eq. (4.24) yields error due to truncation for each segment scaling as $\mathcal{O}(\epsilon/r)$, and therefore total error due to truncation scaling as $\mathcal{O}(\epsilon)$. Taking $r = \Theta(\lambda T)$, K is chosen as

$$K = \Theta\left(\frac{\log(\lambda T/\epsilon)}{\log\log(\lambda T/\epsilon)}\right). \tag{4.55}$$

Note that $\lambda \geqslant H_{\max}$ implies that the condition $r \geqslant H_{\max} T$ is satisfied, which was required for deriving Eq. (4.24). Recall that $\lambda = L\gamma$ in scenario 1 and $\lambda = L\alpha_{\max}$ in scenario 2.

In the case where the $|\text{clock}\rangle$ state is prepared using the compressed form of rotations, the operation that is applied is a little different than desired, because all repeated times are omitted. For each k, the proportion of cases with repeated times is an example of the birthday problem and is approximately $k(k-1)/(2M)$. Therefore, denoting by $\widetilde{U}_{\text{unique}}$ the operation

corresponding to $\widetilde{U}$ but with repeated times omitted, we have

$$\begin{aligned}\|\widetilde{U} - \widetilde{U}_{\text{unique}}\| &\lesssim \sum_{k=2}^{K} \frac{(T/r)^k}{k!} \frac{k(k-1)}{2M} H_{\text{maxspec}}^k \\ &= \sum_{k=0}^{K} \frac{(T/r)^{k+2}}{k!} \frac{1}{2M} H_{\text{maxspec}}^{k+2} \\ &< \frac{T^2 H_{\text{maxspec}}^2}{2r^2 M} e^{TH_{\text{maxspec}}/(rM)}, \end{aligned} \tag{4.56}$$

where $H_{\text{maxspec}} := \max_t \|H(t)\|$. The error over r segments due to omitting repeated times is upper bounded by

$$r\|\widetilde{U} - \widetilde{U}_{\text{unique}}\| \lesssim \frac{T^2 H_{\text{maxspec}}^2}{2rM} e^{TH_{\text{maxspec}}/(rM)} . \tag{4.57}$$

Using $r \in \Theta(\lambda T)$ and $\lambda \geqslant H_{\text{maxspec}}$, we should then choose

$$M = \Omega\left(\frac{TH_{\text{maxspec}}^2}{\epsilon\lambda}\right). \tag{4.58}$$

Therefore, when the repeated times are omitted, we would take

$$M = \Theta\left(\frac{T}{\epsilon\lambda}(H_{\text{maxspec}}^2 + \dot{H}_{\max})\right). \tag{4.59}$$

Next we consider the gate complexity for the $|\text{clock}\rangle_{k\otimes \texttt{time}}$ state preparation. In the case of the compressed rotation encoding, the complexity is $\mathcal{O}(K \log M)$, where we have used $r \in \Theta(\lambda T)$. In the case where $|\text{clock}\rangle_{k\otimes \texttt{time}}$ is prepared with a quantum sort, the complexity is $\mathcal{O}(K \log M \log K)$.

4.5.1 Complexity for Scenario 1

Let us now consider the complexity of simulation of a Hamiltonian given as a sparse matrix. The preparation of the registers $\ell_1, \ldots, \ell_k$ requires only creating an equal superposition. As we discussed in Section 4.4.3, the Hadamard transform in this stage requires $\mathcal{O}(K \log L)$ elementary gates.

Next, one needs to implement the controlled unitaries H_ℓ. Each controlled H_ℓ can be implemented with $\mathcal{O}(1)$ queries to the oracles that give the matrix entries of the Hamiltonian [135]. Since the unitaries are controlled by $\mathcal{O}(\log L)$ qubits and act on n qubits, each control-H_ℓ requires $\mathcal{O}(\log L + n)$ gates. Scaling with M does not appear here, because the qubits encoding the times only serve as input to the oracles. As there are K controlled operations in each segment, the complexity of this step for a segment is $\mathcal{O}(K)$ oracle queries and $\mathcal{O}(K(\log L + n))$ additional gates.

As the total cost for the entire simulation is obtained by multiplying the cost per segment by the number r of all segments, the overall oracle query complexity thus amounts to

$$\mathcal{O}(\lambda T K)\,. \tag{4.60}$$

Here $\lambda = L\gamma$ and L is chosen as in Eq. (4.10) as $\Theta(d^2 H_{\max}/\gamma)$, so $\lambda \in \mathcal{O}(d^2 H_{\max})$. In addition, K is given by Eq. (4.55), giving the total query complexity

$$\mathcal{O}\left(d^2 H_{\max} T \frac{\log(d H_{\max} T/\epsilon)}{\log\log(d H_{\max} T/\epsilon)}\right). \tag{4.61}$$

The overall complexity in terms of the additional gates is larger, and depends on the scheme used to prepare the state $|\text{clock}\rangle_{\mathtt{k}\otimes\mathtt{time}}$. In the case where the preparation is performed via the compressed encoding, we obtain the complexity

$$\mathcal{O}\left(\lambda T K[\log L + \log M + n]\right). \tag{4.62}$$

Regardless of the preparation approach, γ is chosen as in Eq. (4.9) as $\Theta(\epsilon/(d^3 T))$. Moreover, $L \in \Theta(d^5 H_{\max} T/\epsilon)$, whereas the first term in the scaling for M in Eq. (4.59) is $O(d^2 H_{\max} T/\epsilon)$. This means that the first term in Eq. (4.59) can be ignored in the overall scaling due to the $\log L$, and we obtain an overall scaling of the gate complexity as

$$\mathcal{O}\left(d^2 H_{\max} T \frac{\log(d H_{\max} T/\epsilon)}{\log\log(d H_{\max} T/\epsilon)}\left[\log\left(\frac{d H_{\max} T}{\epsilon}\right) + \log\left(\frac{\dot{H}_{\max} T}{\epsilon d H_{\max}}\right) + n\right]\right). \tag{4.63}$$

There is also complexity of $\mathcal{O}(r \log r)$ for the increments of the register recording the segment number, but it is easily seen that this complexity is no larger than the other terms above.

In the case where $|\text{clock}\rangle_{\mathtt{k}\otimes\mathtt{time}}$ is prepared using a sorting network, we obtain complexity

$$\mathcal{O}\left(\lambda T K(\log L + \log M \log K + n)\right), \tag{4.64}$$

with M given by Eq. (4.24).

Technically, this complexity is larger than that in Eq. (4.63), because of the multiplication by $\log K$. Nevertheless, preparation by a sorting network may turn out to sometimes be advantageous in practice because the first preparation requires a larger value of M. This is because the compressed encoding approach does not contain repeated times whereas the approach by sorting networks does.

4.5.2 Complexity for Scenario 2

Let us now consider the cost of simulating the time evolution generated by scenario 2 Hamiltonians. For each single time segment, we have $3 \times (2K)$ applications of the controlled-$\textsc{Prep}(\alpha)$ operation as part of B and $B^\dagger$ when implementing W (or $W^\dagger$), which in turn must be applied three times for achieving oblivious amplitude amplification. This requires $\mathcal{O}(K)$ queries to O_{coeff}. Furthermore, we have $3K$ controlled-$\textsc{Select}(H)$ operations as part of the $\textsc{Select}(V)$ transformation that is involved three times in oblivious amplitude amplification, which thus require $\mathcal{O}(K)$ queries to O_{unit}. Note that the mentioned query complexities for $O_{\mathtt{coeff}}$ and O_{unit} are additive. Hence, the overall query complexity for the entire simulation over all r time segments amounts to $\mathcal{O}(rK)$, which, by using $r \in \Theta(\lambda T)$ and $\lambda = L\alpha_{\max}$ as well as the choice for K given in Eq. (4.55), becomes

$$\mathcal{O}\left(L\alpha_{\max} T \frac{\log(L\alpha_{\max} T/\epsilon)}{\log\log(L\alpha_{\max} T/\epsilon)}\right). \tag{4.65}$$

For each time segment, preparation of the auxiliary $\mathtt{1}$-register states requires $\mathcal{O}(K(\log L + \log(rK/\epsilon))$ additional elementary gates, as displayed earlier in Eq. (4.50). Furthermore, we need $\mathcal{O}(K)$ gates to implement the minus signs given in Eq. (4.18) as well as the the $(-i)^k$ factors occurring in the Dyson series. Additional gate complexity comes from the preparation of the

$|\text{clock}\rangle_{k\otimes \texttt{time}}$ state, which is the same as for scenario 1. Preparation via compressed encoding thus requires in total $\mathcal{O}\left(\lambda TK\left[\log L + \log M + \log\left(\frac{\lambda TK}{\epsilon}\right)\right]\right)$ gates, where we have repeatedly used $r \in \Theta(\lambda T)$. Using once more $\lambda = L\alpha_{max}$ and Eqs. (4.55) and (4.59) together with $H_{max} \leqslant L\alpha_{max}$, this finally yields

$$\mathcal{O}\left(\alpha_{max} LT \frac{\log\left(\alpha_{max} LT/\epsilon\right)}{\log\log\left(\alpha_{max} LT/\epsilon\right)}\left[\log\left(\frac{\alpha_{max} LT}{\epsilon}\right) + \log\left(\frac{\dot{H}_{max} T}{\epsilon\,\alpha_{max} L}\right) + \log\left(\frac{\log\left(\alpha_{max} LT/\epsilon\right)}{\log\log\left(\alpha_{max} LT/\epsilon\right)}\right) + \log L\right]\right) \quad (4.66)$$

as the overall gate complexity in this case, where we again ignored the first term in Eq. (4.59) in the overall scaling due to the occurrence of the larger $\log\left(\alpha_{max} LT/\epsilon\right)$ term. Observe that the last two additive terms $\log K$ and $\log L$ in the brackets of Eq. (4.66) are also much smaller compared to the first term; they have also been ignored in the formulation of Theorem 2. Similarly, we obtain the gate complexity $\mathcal{O}\left(\lambda TK[\log L + \log M \log K + \log\left(\frac{\lambda TK}{\epsilon}\right)]\right)$ for implementations based on quantum sorting networks, which is slightly larger due to the additional $\log K$ factor in the second term.

Scaling with n (i.e. with the number of system qubits) does not appear in Eqs. (4.65) and (4.66), because the system qubits are just used as input to the oracle $O_{\texttt{unit}}$ (defined in Eq. (4.14)) as part of controlled-SELECT(H) operations. The scaling with n is therefore hidden in the gate cost of implementing that oracle.

4.6 Results

Let us summarize the results by stating the theorems for both scenarios.

Theorem 6 (Time-dependent Hamiltonian simulation for sparse matrices) *For a Hamiltonian $H(t)$ given by a time-dependent d-sparse matrix, the generated time evolution of a quantum system can be simulated for time T and within error ϵ using a number of oracle queries scaling as*

$$\mathcal{O}\left(d^2 H_{max} T \frac{\log(dH_{max}T/\epsilon)}{\log\log(dH_{max}T/\epsilon)}\right), \quad (4.67)$$

and a number of additional elementary gates scaling as

$$\mathcal{O}\left(d^2 H_{max} T \frac{\log(dH_{max}T/\epsilon)}{\log\log(dH_{max}T/\epsilon)}\left[\log\left(\frac{dH_{max}T}{\epsilon}\right)+\log\left(\frac{\dot{H}_{max}T}{\epsilon dH_{max}}\right)+n\right]\right). \tag{4.68}$$

Theorem 7 (Time-dependent Hamiltonian simulation for LCU)
For a Hamiltonian $H(t)$ *given by a time-dependent linear combination of* L *unitary terms, the generated time evolution of a quantum system can be simulated for time* T *and within error* ϵ *using a number of oracle queries scaling as*

$$\mathcal{O}\left(L\alpha_{max}T\frac{\log(L\alpha_{max}T/\epsilon)}{\log\log(L\alpha_{max}T/\epsilon)}\right) \tag{4.69}$$

and a number of additional elementary gates scaling as

$$\mathcal{O}\left(L\alpha_{max}T\frac{\log\left(L\alpha_{max}T/\epsilon\right)}{\log\log\left(L\alpha_{max}T/\epsilon\right)}\left[\log\left(\frac{L\alpha_{max}T}{\epsilon}\right)+\log\left(\frac{\dot{H}_{max}T}{\epsilon L\alpha_{max}}\right)\right]\right), \tag{4.70}$$

where α_{max} *denotes a known global upper bound on the modulus of each of the* L *time-dependent coefficients in the decomposition, implying* $L\alpha_{max} \geqslant H_{max}$.

4.7 Applications

Our algorithm can be used for a range of problems that require simulating dynamics under a time-dependent Hamiltonian. We now outline several specific applications.

Low and Wiebe recently proposed a new Hamiltonian simulation algorithm in the interaction picture [37]. This algorithm uses simulation of a time-dependent Hamiltonian as a subroutine. Taking a Hamiltonian $H = A + B$ and the Schrödinger equation $i\partial_t |\psi(t)\rangle = (A+B)|\psi(t)\rangle$, we can formulate the simulation problem in the interaction picture as

$$H_I(t) = e^{iAt}Be^{-iAt}, \tag{4.71}$$

$$|\psi_I(t)\rangle = e^{iAt}|\psi(t)\rangle, \tag{4.72}$$

$$i\partial_t|\psi_I(t)\rangle = e^{iAt}Be^{-iAt}|\psi_I(t)\rangle, \tag{4.73}$$

where H_I is the interaction picture Hamiltonian, $|\psi_I(t)\rangle$ a state in the interaction picture and (4.73) the interaction picture Schrödinger equation. Solving (4.73) requires time-dependent Hamiltonian simulation. Low and Wiebe [37] proposed an approach for simulating time-dependent Hamiltonians simultaneously with us. They use a simpler method of preparing the state representing the times by discarding times rather than sorting them. This simplification results in a multiplicative factor in the gate complexity.

Interaction picture simulation is useful when $\|A\| \gg \|B\|$ and e^{-iAt} is straightforward to implement. Using the interaction picture then allows us to reduce the complexity in terms of the norm of the Hamiltonian. This approach is particularly useful for quantum chemistry where it allows for a simulation with complexity sublinear in the number of orbitals [74].

The second application of time-dependent simulations is adiabatic computing and quantum annealing. Adiabatic computation is performed by preparing the ground state $|\psi\rangle$ of an "easy" Hamiltonian H_0 and then evolved under a slowly changing Hamiltonian

$$H(t) = tH_1 + (1-t)H_0, \tag{4.74}$$

where H_1 is the Hamiltonian whose ground state we wish to prepare [186]. If the evolution is slow enough to satisfy the conditions of the adiabatic theorem [187], one has a a good chance of preparing the ground state of H_1. These approaches are favored for heuristics for optimization problems. Implementing them on a fault-tolerant circuit-based quantum computer can allow for a higher accuracy than using an analog quantum annealer. This implementation would simply consist of simulating the evolution under a time-dependent Hamiltonian.

Our work can also be applied to state preparation. Adiabatic state preparation is a popular choice for initializing quantum chemistry algorithms [71]. In this case, one prepares an eigenstate of a simple Hamiltonian and slowly (adiabatically) changes the Hamiltonian to the one of interest.

Lastly, there are many time-dependent phenomena that our algorithm can simulate. One example is evolution of states under a time-dependent electromagnetic field. In quantum chemistry and material science, we can account for nuclear movement by describing the system of electrons by a time-dependent Hamiltonian.

4.8 Conclusion

We have provided a quantum algorithm for simulating the evolution generated by a time-dependent Hamiltonian that is either given by a generic d-sparse matrix or by a time-dependent linear combination of some efficiently implementable unitary terms. For both scenarios, the complexity of the algorithm scales logarithmically in both the dimension of the Hilbert space and the inverse of the error of approximation ϵ. Our work presents an exponential improvement in error scaling compared to techniques based on Lie-Trotter-Suzuki expansion. We utilize the truncated Dyson series, which is based on the truncated Taylor series approach by Berry et al. [34]. It achieves similar complexity, in that it has T times a term logarithmic in $1/\epsilon$. Interestingly, the complexity in terms of queries to the Hamiltonian is independent of the rate of change of the Hamiltonian.

For the complexity in terms of additional gates, the complexity is somewhat larger and depends logarithmically on the rate of change of the Hamiltonian. The complexity also depends on the scheme that is used to prepare the state to represent the times. If one uses the scheme as in Ref. [136], which corresponds to a compressed form of a tensor product of small rotations, then there is an additional error due to the omission of repeated times. Alternatively, one could prepare a superposition of all times and sort them, which eliminates that error, but gives a multiplicative factor in the complexity. This trade-off means that different approaches may be more efficient with different combinations of parameters.

Simultaneously with the publication of our algorithm, Low and Wiebe [37] developed a similar algorithm for time dependent Hamiltonians also based on LCU. Their algorithm is slower than ours by a multiplicative factor. Even more recently, Berry et al. [188] came up with a new algorithm that improved the dependence on the norm of the Hamiltonian. Their work builds onto [149] and can provide more efficient algorithm for simulation Hamiltonians whose norm significantly changes over time, for example in scattering problems.

An interesting open question is whether the complexity could be improved such that the factor of $\log(1/\epsilon)$ is additive rather than multiplicative, as in Ref. [35] for time-independent Hamiltonians. However, those approaches do not appear directly applicable to the time-dependent case.

5

Training and Tomography with Quantum Boltzmann Machines

"What I cannot create, I do not understand."

Richard Feynman

Machine learning (ML) has earned a prominent role in the tech industry for its ability to train computers for complicated tasks without the need of giving them explicit instructions. ML performs exceptionally well on tasks with complex structure such as speech recognition [189–191], computer vision [192–194] or even playing the board game Go [195–197]. Given its importance for computing, it is natural to ask what can arise from the confluence of ML and quantum computing.

In this chapter, we focus on quantum algorithms that learn from quantum data. This chapter is based on our paper [4] and significantly overlaps with it. I altered the text to suit an audience mostly unfamiliar with Boltzmann machines. I included the Theorems 9 and 10 proved by my co-authors but without the full statements of the proofs. My co-author also

carried out the numerical analysis for tomography using relative entropy in Section 5.6.5.

It should be noted that the success of a particular ML algorithm is seldom determined by theoretical analysis alone. Instead, ML algorithms are compared by running against each other on benchmark sets or by their usefulness for practical tasks. Since we do not have quantum hardware that can run QML algorithms, our situation is similar to the one of artificial intelligence in the 60s. At the same time, the foundations of AI laid back before ML became "practical" were crucial for guiding our thinking about algorithm development. Similarly, there is value in foundations research in QML.

5.1 Quantum Machine Learning

ML and quantum physics can overlap in several ways. There are many results that apply (classical) ML techniques on quantum problems in condensed-matter physics [198], quantum control [199] or discovering physical concepts [200]. In these applications, ML is used to uncover the properties of physical systems or devise models for complex phenomena.

Another avenue of research explores the implementation of ML algorithms on quantum hardware. A number of results showed that quantum computing can provide significant speedups for problems such as support vector machines (SVM) [201], nearest neighbor classification [202, 203], boosting [204, 205] and many others [206–210]. However, loading and accessing large data sets in a quantum computer is problematic [211–213]. Nevertheless, there are still applications with polynomial speedup and unexplored ways of combining ML with quantum computing techniques.

The last possible way of combining quantum computing and ML is inventing quantum machine learning (QML) algorithms that run on a quantum computer and are designed to process quantum data. In this setup, quantum states are fed into a quantum computer for analysis directly from an experiment or another quantum device. This area has a lot of potential for speedups. To analyze the data classically, one needs to first perform tomography whose complexity scales exponentially with the size of the system. Quantum machine learning on quantum data might allow us

to extract features from the system without performing full tomography. While we do not expect universal exponential speedups in this area, it is likely that using quantum machine learning will assist in quantum system characterization and Hamiltonian learning [214, 215] for problems that exhibit some structure. We summarize the types of quantum machine learning in the table below.

		The data is	
		classical	quantum
The algorithm runs on a	classical computer	Classical ML	ML on experimental data
	quantum computer	QML for SVM, clustering, fitting ...	This chapter

Figure 5.1: Different areas of QML. Our focus is on quantum algorithms able to process quantum data. However, since classical data are a special case of quantum ones, we show how to learn on them as well.

5.2 Boltzmann Machines

As the name of this chapter implies, we are interested in a particular neural network called a Boltzmann machine (BM). The Boltzmann machine is a physically motivated framework capable of generating new examples "similar" to the training data [216], i.e. generative, unsupervised training introduced in Appendix A1. The inspiration for Boltzmann machines comes from statistical physics, particularly the Boltzmann distribution. For readers unfamiliar with machine learning we recommend reading Appendix A first and then continuing with this section.

Hinton [217] defines a Boltzmann machine as "a network of symmetrically connected, neuronlike units that make stochastic decisions about whether to be on or of." This network has two key features – a graph (or a hypergraph[1]) and an energy function. Just as for general neural networks,

[1] A hypergraph is a generalization of a graph. While an edge in a graph connects two vertices, a hyperedge in a hypergraph can connect any number of vertices.

the information vector $\mathbf{x}$ is encoded in the vertices (called units) x_i that can take on values ± 1. The units of a BM consist of visible units and optional hidden units, similarly to other artificial neural networks.

The edges of the graph have been assigned weights such that an edge connecting nodes i and j has weight W_{ij}. Missing edges automatically have weight 0. We also allow for self-loop like terms, biases b_i. This energy function

$$E(\mathbf{x}) = \mathbf{x}^T W \mathbf{x} + \mathbf{b}^T \mathbf{x} \tag{5.1}$$

can be shown to be equivalent to (A.1) where we added one bit to $\mathbf{x}$ with a fixed value of 1. We demonstrate how to compute an energy for a given configuration in Fig. 5.2.

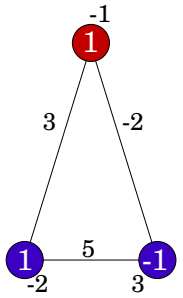
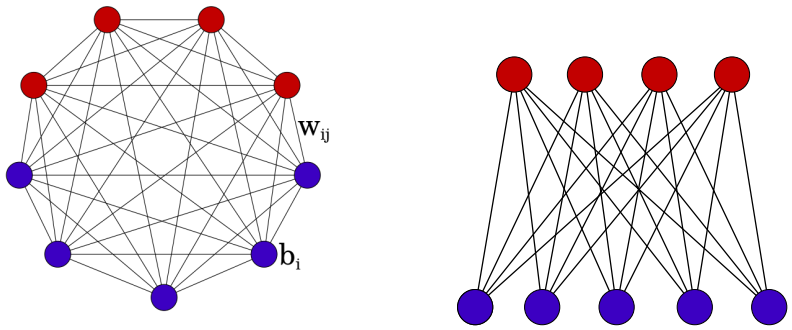


Figure 5.2: Each configuration can be assigned an energy value. In this example, the assigned configuration $E = (-1)\cdot 1 + (-2)\cdot 1 + 3\cdot(-1) + 3\cdot 1\cdot 1 + (-2)\cdot 1\cdot(-1) + 5\cdot 1\cdot(-1) = -6$.

Common choices of the graph include the complete graph and the bipartite graph (see Fig. 5.2) with connections only between visible and hidden units. A BM defined on a bipartite graph is called a Restricted Boltzmann Machine (RMB). RBMs are favored by the ML community because they can be trained using an efficient method called contrastive divergence [218].

A properly trained BM generates new data. Outputs similar to the training set should be generated with high probability while data that is very different than the training set should be very unlikely to appear. To do this, we assume that there is a distribution $P(v)$ that the model learns and the training data are sampled from it. A BM accomplishes this by outputting data from its probability distribution $Q(v)$, defined by weights and biases, that should be close to the distribution $P(v)$. We accomplish



(a) A fully-connected Boltzmann machine. The connections between units form a complete graph.

(b) Restricted Boltzmann Machine. Each hidden unit is connected to each visible unit but not to other hidden units and vice versa. The connections form a bipartite graph.

Figure 5.3: Fully connected vs. restricted Boltzmann machine.

this by teaching the BM to approximately reproduce the training data[2]. In particular, the probability $Q(\mathbf{x})$ for a configuration $\mathbf{x}$ (i.e. on both the visible and the hidden units) is defined as:

$$Q(\mathbf{x}) = \frac{\exp\left(-E(\mathbf{x})\right)}{\sum_{\mathbf{x}} \exp\left(-E(\mathbf{x})\right)}. \tag{5.2}$$

This is the Boltzmann distribution for the energy $E(\mathbf{x})$ defined in (5.1) where we set $kT = 1$. The state $\mathbf{x}$ consists of the state on the hidden units $\mathbf{h}$ and visible units $\mathbf{v}$, so $\mathbf{x} = (\mathbf{v}, \mathbf{h})$. The probability of a state on the visible units can be obtained by summing over all configurations on the hidden units

$$Q(\mathbf{v}) = \frac{\sum_{\mathbf{h}} \exp\left(-E(\mathbf{v}, \mathbf{h})\right)}{\sum_{\mathbf{v},\mathbf{h}} \exp\left(-E(\mathbf{v}, \mathbf{h})\right)}. \tag{5.3}$$

But how do we get the weights and biases required for computing (5.1) and (5.3)? We train the algorithm to make Q close to P using the training data. The "closeness" of two distributions is measured by Kullback-Leibler (KL) divergence. Formally, the KL divergence quantifies the information loss occurring if the distribution generated from the model Q

[2]We want to be able to model the training data but not overfit them.

replaces the underlying distribution of data P:

$$\mathfrak{D}_{KL}(P\|Q) = \sum_{\mathbf{v}} P(\mathbf{v}) \log \frac{P(\mathbf{v})}{Q(\mathbf{v})}. \tag{5.4}$$

Since P is fixed, minimizing $\mathfrak{D}_{KL}(P\|Q)$ is equivalent to maximizing the log-likelihood

$$\mathfrak{L} = \sum_{\mathbf{v}} P(\mathbf{v}) \log Q(\mathbf{v}). \tag{5.5}$$

While computing $\mathfrak{D}_{KL}(P\|Q)$ (and the log-likelihood) is exponentially difficult in the number of units, it is possible to estimate its gradients with respect to b_i and $W_{i,j}$. A straightforward computation yields

$$\partial_{b_i}\mathfrak{L} = \sum_{\mathbf{v}} P(\mathbf{v}) \left[-\frac{\sum_{\mathbf{h}} x_i \exp\left(-E(\mathbf{v},\mathbf{h})\right)}{\sum_{\mathbf{h}} \exp\left(-E(\mathbf{v},\mathbf{h})\right)} + \frac{\sum_{\mathbf{v}',\mathbf{h}} x_i \exp\left(-E(\mathbf{v}',\mathbf{h})\right)}{\sum_{\mathbf{v}',\mathbf{h}} \exp\left(-E(\mathbf{v}',\mathbf{h})\right)} \right] \tag{5.6}$$

for biases and

$$\partial_{W_{i,j}}\mathfrak{L} = \sum_{\mathbf{v}} P(\mathbf{v}) \left[-\frac{\sum_{\mathbf{h}} x_i x_j \exp\left(-E(\mathbf{v},\mathbf{h})\right)}{\sum_{\mathbf{h}} \exp\left(-E(\mathbf{v},\mathbf{h})\right)} + \frac{\sum_{\mathbf{v}',\mathbf{h}} x_i \exp\left(-E(\mathbf{v}',\mathbf{h})\right)}{\sum_{\mathbf{v}',\mathbf{h}} \exp\left(-E(\mathbf{v}',\mathbf{h})\right)} \right] \tag{5.7}$$

for weights. However, we do not have direct access to the probability distribution $P(\mathbf{v})$. Instead, we estimate the gradients as

$$\partial_{W_{i,j}}\mathfrak{L} = \frac{1}{s} \sum_{\mathbf{v}\in\text{training set}} \left[-\frac{\sum_{\mathbf{h}} x_i x_j \exp\left(-E(\mathbf{v},\mathbf{h})\right)}{\sum_{\mathbf{h}} \exp\left(-E(\mathbf{v},\mathbf{h})\right)} + \frac{\sum_{\mathbf{v}',\mathbf{h}} x_i \exp\left(-E(\mathbf{v}',\mathbf{h})\right)}{\sum_{\mathbf{v}',\mathbf{h}} \exp\left(-E(\mathbf{v}',\mathbf{h})\right)} \right] \tag{5.8}$$

where s is the number of training vectors. Here we approximated the underlying distribution $P(v)$ using the training data which we treat as samples from $P(v)$. We estimate the gradients of the biases in the same way. The expression (5.8) consists of expectations values of $x_i x_j$ in two different types of states. The first term in (5.8) is the expectation value when the visible units are "clamped" to a particular training vector v (i.e. fixed to a given configuration) while the hidden units model a thermal distribution. We first estimate these expectation values through sampling for each training vector, and then we average them over the data set. The

second term in (5.8) does not depend on the training data, thus we need to compute it only once for each update of weights. The term $\frac{\sum_{\mathbf{v},\mathbf{h}} x_i \exp(-E(\mathbf{v},\mathbf{h}))}{\sum_{\mathbf{v},\mathbf{h}} \exp(-E(\mathbf{v},\mathbf{h}))}$ is sampled from a thermal distribution on all units. The steps of the training are summarized at the end of this section.

Hinton and Sejnowski [219] proposed the first machine learning algorithm that evaluates the gradient in Eq. (5.8). However, this learning was not efficient because it required many samples from a thermal distribution. There have been many approaches to improve the speed of learning [220, 221, 221–223]. Most notably, the first term in (5.8) can be computed exactly for RBMs and the second term can be roughly approximated using a method called contrastive divergence. Surprisingly, this very rough approximation works exceptionally well [217, 218, 224].

We train a BM using stochastic gradient ascent. We start by initializing the weights and biases to random values and keep updating them to increase $\mathfrak{L}$ (and therefore decrease $\mathfrak{D}_{KL}(P\|Q)$). Many techniques can be used to speed up the convergence of gradient descent, including changing the gradient step and including momentum[3]. This process is repeated for a fixed number of epochs. If the number of epochs is too small, P and Q might not be close. On the other hand, if we trained for too long, it is possible that the BM overfits the data. Since $\mathfrak{L}$ is impossible to estimate for real-world data sets, the quality of the model is in practice estimated using cross-validation data.

We will conclude this section with an overview of the training and the generating algorithms. Once we settle on a graph (i.e., the number of visible and the hidden units and the connections between them) for the BM, the algorithm proceeds as follows:

[3]Momentum often helps accelerate gradient descent. The name is inspired by mechanics and suggests that instead of following the direction of the gradient (acceleration), we should follow the direction of velocity. In each step, the new momentum p_{i+1} is calculated as a linear combination of momentum in the previous step p_i and freshly computed gradient a_{i+1}. The new momentum is $p_{i+1} = (1-\alpha)p_i + \alpha a_{i+1}$ where $\alpha \in [0, 1]$.

Boltzmann machine training

generate starting weights and biases
construct the energy function
for the number of epochs **do**
    create an approximation of the thermal distribution on all units
    evaluate the expectation values of $x_i x_j$ and x_i in this configuration
    for all training vectors **do**
        clamp visible units to a training vector
        create an approximation of the thermal state on hidden units
        evaluate the expectation values $x_i x_j$ and x_i in this configuration
    end for
    compute gradients
    update weights and biases
    update the energy function
end for

Once the algorithm is trained, we can generate new data:

Sampling from a Boltzmann machine

create an approximation of the thermal state on all units
sample from the visible units

5.3 Training Quantum Boltzmann Machines

There are two main reasons why it is worth introducing Boltzmann machines to quantum mechanics. First, classical BMs may require many hidden units to model complicated datasets. We know that quantum mechanics can produce distributions that are difficult to sample from classically. It is possible that quantum Boltzmann machines will be more space efficient than classical ones. Second, we want to perform generative learning on quantum data. We will now introduce several approaches for quantizing BMs and compare them between each other and to their classical counterparts.

A quantum Boltzmann machine (QBM) was first introduced by Amin et al. [225]. The QBM was defined as a generalization of a classical Boltz-

mann machine to the quantum regime. Classical binary units are replaced by qubits and the energy function by a Hamiltonian. Amin et al. constrained their Hamiltonians to the transverse Ising model.

This choice was made to satisfy the hardware abilities; this algorithm was designed to run on a D-Wave machine. While such models are trainable and can outperform classical BMs, the training procedure proposed therein suffers two drawbacks. First, the transverse field cannot be learned by gradient descent from classical data. These terms must be found through brute force techniques which makes finding the full Hamiltonian much more difficult. Second, the transverse Ising models considered are widely believed to be efficiently simulatable using quantum Monte-Carlo methods and therefore do not provide a clear quantum advantage. The state of the QBM can be then described by the density matrix corresponding to the thermal state of the Hamiltonian

$$\sigma = \frac{e^{-H}}{\mathrm{Tr}\left[e^{-H}\right]} \tag{5.9}$$

and the state on the visible units

$$\sigma_v = \mathrm{Tr}_h\left[\sigma\right] = \frac{\mathrm{Tr}_h\left[e^{-H}\right]}{\mathrm{Tr}\left[e^{-H}\right]}, \tag{5.10}$$

where we traced over the hidden units. Amin et al. further defined quantum log-likelihood for training QBMs on classical states. In this chapter, we generalize their version of the QBM in more than one way.

We explicitly design QBMs to process quantum data and work with models that exhibit the full power of quantum computing. The generalization of Boltzmann Machine training into the quantum setting is not unique. We propose three methods that we refer to as POVM-based Golden-Thompson training[4], commutator training, and state-based Relative Entropy training. These approaches present different quantum analogs of the training set and the objective function. We now review the possible realizations of a quantum training set. Next, we follow up with our proposal for a new objective function.

[4]POVM stands for a positive-operator valued measure. This is the most general measurement.

5.3.1 Quantum Training Set

In the classical setting, the training set is a set of vectors representing the data. As a part of the training of classical Boltzmann Machines (BMs), we have to ability to fix (clamp) the visible units to a particular training vector. Additionally, we assume that the training set is sampled from an underlying distribution $P(\mathbf{v})$ over the visible units. The training set was first translated to the quantum world as a set of projectors [225] on computational states. This choice makes it indeed possible to clamp visible units on any desired configuration. The goal of the training is then to learn the probability of each projector.

We propose two ways of generalizing the training set to include quantum as well as classical data. Our first suggestion is that there is no need to constrain ourselves to projectors on computational states. We realized that POVMs provide a natural way to express the training set. Formally, we define the training set to be the following.

Let $\mathcal{H} := \mathcal{H}_V \otimes \mathcal{H}_H$ be a finite-dimensional Hilbert space and let $\mathcal{H}_V$ and $\mathcal{H}_L$ be subsystems corresponding to the visible and hidden units of the QBM. The probability distribution p_ν and POVM $\Lambda = \{\Lambda_\nu\}$, comprise a training set for QBM training if

1. there exists a bijection between the domain of p_ν and Λ and
2. the domain of each Λ_ν is $\mathcal{H}$, and it acts non–trivially only on subsystem $\mathcal{H}_V$.

Note that unlike $P(\nu)$, p_ν corresponds only to the relative frequencies of each outcome in the training set and not necessarily to the expectation values of each Λ_ν in the underlying state (although we assume that they should be close).

This means that if we are training on quantum data, we can take a set of measurements and the frequencies of their corresponding outcomes as our training examples. We can also use POVM training on classical data by pretending that they were sampled from a quantum state.

As a method for generalizing classical data, the choice of POVMs gives a lot of freedom. Let us clarify the freedom of choosing the POVM on

an example. Suppose that we wish to train a model that generates even numbers between 1 and 16 (this was an arbitrary choice) given a set of samples that are all even numbers[5]. Then a sensible training set would be

$$\begin{aligned} \Lambda_n &= |2n\rangle\langle 2n| \text{ for } 1 \leqslant n \leqslant 8 \\ \Lambda_0 &= \mathbb{I} - \sum_{n=1}^{8} \Lambda_n, \qquad p_v = (1-\delta_{v,0})/8. \end{aligned} \tag{5.11}$$

The following equivalent training set can also be used

$$\begin{aligned} \Lambda_1 &= \frac{1}{8}\big(|2\rangle + \cdots + |16\rangle\big)\big(\langle 2| + \cdots + \langle 16|\big), \\ \Lambda_0 &= \mathbb{I} - \Lambda_1, \qquad p_v = \delta_{v,1}. \end{aligned} \tag{5.12}$$

This ambiguity about the form of the training set reveals that the POVM for quantum Boltzmann training can be non-trivial even when a single training vector is used. This allows us to circumvent problems with the Golden-Thompson method that arose in [225] by using inherently non-diagonal POVM elements. POVM training allows for clamping the visible units in the same way as in the classical case. Just as with classical training data, we assume to have a full knowledge about the data set. We know which POVM we measured as well as how often we obtained each outcome[6]. What we do not know is the underlying quantum state that the outcomes were sampled from.

The second option is to train directly from quantum states. This is equivalent to a quantum algorithm sampling from a density matrix. Note that a single density matrix is sufficient since a linear combination of density matrices is a density matrix as well. Since learning a description of a quantum state is the goal of tomography, QBM training can be considered a tomographic tool. The training state is then described by a Hamiltonian such that the thermal state of this Hamiltonian is close to the training state.

These generalizations give very different notions of what a quantum training set really is. While our POVM approach focuses on the generalization of obtaining (measuring) the training examples, the density matrix

[5]This is a very simple example where the training set provides all the necessary information about the concept we are trying to learn without any noise.

[6]It might be possible to relax this assumption and only assume to have oracular access to POVM elements.

approach focuses on the states. We expect that these notions will not be interchangeable in practice. Since we are working with very different objects, we needed to develop different training techniques as well.

5.3.2 Golden-Thompson Training

Our first approach to training is a generalization of the training method introduced by Amin et al. [225]. The goal is to find an analog of the negative log-likelihood as defined in (5.5). The data in their setting is classical and thus can be assigned to a computational basis state. The probability $Q(v)$ of observing a training state $|v\rangle$ can be translated to the quantum setting as

$$Q(v) = \mathrm{Tr}\left[e^{-H}\Lambda_v\right] / \mathrm{Tr}\left[e^{-H}\right], \tag{5.13}$$

where H is the Hamiltonian defining the thermal state. The projector $\Lambda_v = |\mathbf{v}\rangle\langle\mathbf{v}| \otimes \mathbb{1}$ corresponds to visible units clamped to a training binary state $\mathbf{v}$ and the training set is a set of such projectors. This allows us to introduce the quantum log-likelihood.

$$\mathfrak{O}_\Lambda(H) = \sum_{\mathbf{v}} P_{\mathbf{v}} \log\left(\frac{\mathrm{Tr}\left[\Lambda_v e^{-H}\right]}{\mathrm{Tr}\left[e^{-H}\right]}\right). \tag{5.14}$$

Assuming that the Hamiltonian can be written as $H = \sum_{j=1}^{M} \theta_j H_j$, we can attempt to compute the gradient of $\mathfrak{O}_\Lambda(H)$. The derivative of the exponential in (5.14) can be expressed using Duhamel's formula

$$\mathrm{Tr}\left[\Lambda_v \partial_\theta e^{-H}\right] = \mathrm{Tr}\left[\int_0^1 \Lambda_v e^{sH}\left[\partial_\theta H\right] e^{(1-s)H} ds\right]. \tag{5.15}$$

If Λ_v commutes with H, then we recover the expression for the classical gradient. However, in the general, non-commuting case, a closed formula is not available.

Amin et al. [225] overcame this difficulty by lower bounding the objective function instead of estimating it. First define a clamped Hamiltonian $H_v = H - \ln \Lambda_v$. This Hamiltonian introduces a penalty whenever the

visible units are not settled on $\mathbf{v}$. Then, they use the Golden-Thompson inequality

$$\mathrm{Tr}\left[e^{A}e^{B}\right] \geqslant \mathrm{Tr}\left[e^{A+B}\right] \tag{5.16}$$

on expression (5.14), leading to an objective function lower bound

$$\mathfrak{O}_{\Lambda}(H) \geqslant \mathfrak{O}_{\Lambda}^{\mathrm{bound}}(H) = \sum_{\mathbf{v}} P_{\mathbf{v}} \log\left(\frac{\mathrm{Tr}\left[e^{-H_{\nu}}\right]}{\mathrm{Tr}\left[e^{-H}\right]}\right). \tag{5.17}$$

The components of the gradient of $\mathfrak{O}_{\Lambda}^{\mathrm{bound}}(H)$ are

$$\frac{\partial \mathfrak{O}_{\Lambda}^{\mathrm{bound}}(H)}{\partial \theta_j} = \sum_{\mathbf{v}} P_{\mathbf{v}}\Big(-\frac{\mathrm{Tr}[e^{-H_{\nu}}\partial_{\theta_j}H]}{\mathrm{Tr}[e^{-H_{\nu}}]} + \frac{\mathrm{Tr}[e^{-H}\partial_{\theta_j}H]}{\mathrm{Tr}[e^{-H}]}\Big). \tag{5.18}$$

A downside of this approximation is that when $\mathrm{Tr}[e^{-H_{\nu}}\partial_{\theta_j}H] = 0$, the model does not learn because the gradient of the Golden-Thompson approximation is zero. However, this does not mean that the gradient of the log-likelihood is zero as well. Instead, it shows that the upper bound given by the Golden-Thompson inequality is not always tight. As a consequence, Amin et al. in [225] were not able to use this form of training to learn non-diagonal terms of the Hamiltonian.

Our work presents two improvements over [225]. First, our formalism avoids the problem of $\mathrm{Tr}[e^{-H_{\nu}}\partial_{\theta_j}H] = 0$ by explicitly designing non-diagonal POVM elements. We can always see a classical distribution as being sampled from pure states. This allows us to always pick non-diagonal Λ_{ν}.

This freedom allows us to choose a POVM that always produces a non-zero gradient because Λ_{ν} does not commute with H. This avoids the problems in [225] with the transverse Ising model and a training set similar to (5.11).

Second, our algorithm is not constrained by the transverse Ising model. A general Hamiltonian that is a smooth function of its parameters is indeed possible. In our numerical analysis, we used an electronic Hamiltonian which is 4-local. A choice of a 2-local Hamiltonian is preferred because such a Hamiltonian can be represented on a graph instead of a hypergraph.

5.3.3 Commutator Training

Our second approach to POVM training avoids the use of the Golden–Thompson inequality. Instead, we approximate the derivative in the expression (5.14) as a commutator series. If H and Λ_v commute, we recover the classical gradient exactly. In general, if the Hamiltonian is a sum of bounded Hamiltonian terms the expectation value can be written as a commutator series; we have $\mathrm{Tr}[Ce^{-H}]$ for

$$C := \Lambda_v \left(\partial_{\theta_j} H + \frac{[H, \partial_{\theta_j} H]}{2!} + \frac{[H, [H, \partial_{\theta_j} H]]}{3!} + \cdots \right). \tag{5.19}$$

Thus, the gradient of the average log-likelihood becomes

$$\frac{\mathfrak{O}_\Lambda(H)}{\partial \theta_j} = \sum_{\mathbf{v}} P_{\mathbf{v}} \Big(- \frac{\mathrm{Tr}[e^{-H} C]}{\mathrm{Tr}[e^{-H}]} + \frac{\mathrm{Tr}[e^{-H} \partial_{\theta_j} H]}{\mathrm{Tr}[e^{-H}]} \Big). \tag{5.20}$$

Next, we truncate the series at a low order. This makes the commutator series tractable and we will not incur substantial error if $\|[H, \partial_{\theta_j} H]\| \ll 1$. Commutator training is therefore expected to outperform Golden-Thompson training in the presence of L_2 regularization on the quantum terms. However, computing higher-order commutators is computationally more expensive than estimating the gradient from formula (5.18). Lastly, our numerical examples showed that commutator training can be numerically unstable. While it might work for certain data sets or be used for fine-tuning a model trained using the Golden-Thompson approach, we did not find the commutator training to be broadly applicable.

5.3.4 Relative Entropy Training

The relative entropy is the quantum equivalent of KL divergence and as such, represents a natural extension for the learning of quantum states. Relative entropy is defined as

$$S(\rho \| \sigma) = \mathrm{Tr}(\rho \log \rho - \rho \log \sigma), \tag{5.21}$$

where ρ is the data distribution and $\sigma = e^{-\beta H}/Z$ is the thermal state generated by QBM. For simplicity, we take $\beta = 1$. Similarly to KL divergence,

the first part of (5.21) is independent of the parameters of the QBM. Hence, our goal is to maximize the new objective function

$$\mathfrak{O}_\rho(H) = \mathrm{Tr}\left[\rho \log\left(\frac{e^{-H}}{\mathrm{Tr}[e^{-H}]}\right)\right]. \tag{5.22}$$

Minimization of $\mathfrak{O}_\rho(H)$ allows us to train on quantum data, i.e. copies of ρ. We can express the derivatives of $\mathfrak{O}_\rho(H)$ with respect to the parameters of the Hamiltonian θ as

$$\frac{\partial \mathfrak{O}_\rho(H)}{\partial \theta_j} = -\mathrm{Tr}[\rho \partial_{\theta_j} H] + \mathrm{Tr}[e^{-H}\partial_{\theta_j} H]/\mathrm{Tr}[e^{-H}]. \tag{5.23}$$

We can use the expression (5.23) for gradient ascent. We can compute an approximation of the gradient by estimating both terms in (5.23). The first term can be obtained by measuring expectation values on the training state and the corresponding expectation values on the approximation of the thermal state created by a simulation. Furthermore, maximizing $\mathfrak{O}_\rho(H)$ guarantees that the generated data is close to the training data because $S(\rho\|e^{-H}/Z) \geqslant \|\rho - e^{-H}/Z\|^2/2\ln(2)$ if ρ is positive definite. Thus if $S(\rho\|e^{-H}/Z) \to 0$ then $e^{-H}/Z \to \rho$.

Training by minimizing the relative entropy has two main advantages. First, unlike Golden-Thompson training, no approximations are needed for computing tomography. Second, relative entropy can be used for directly learning quantum states. Given enough copies of the training state ρ and sufficient computing time, relative entropy training provides a classical description of the training state ρ in the form of the parameters θ_j of a Hamiltonian such that $\rho \approx \frac{e^{-H}}{\mathrm{Tr}[e^{-H}]}$. This provides a novel approach to partial tomography. Unlike tomography, QBM provides a direct description of the state but might be more efficient for high-dimensional states exhibiting clear structure. Additionally, QBM has the ability to create additional approximate copies of ρ and therefore perform approximate cloning. We also propose using QBM as a candidate for a quantum memory (QRAM).

We compare training and generation of Golden-Thompson and relative entropy QBMs in the diagram 5.4.

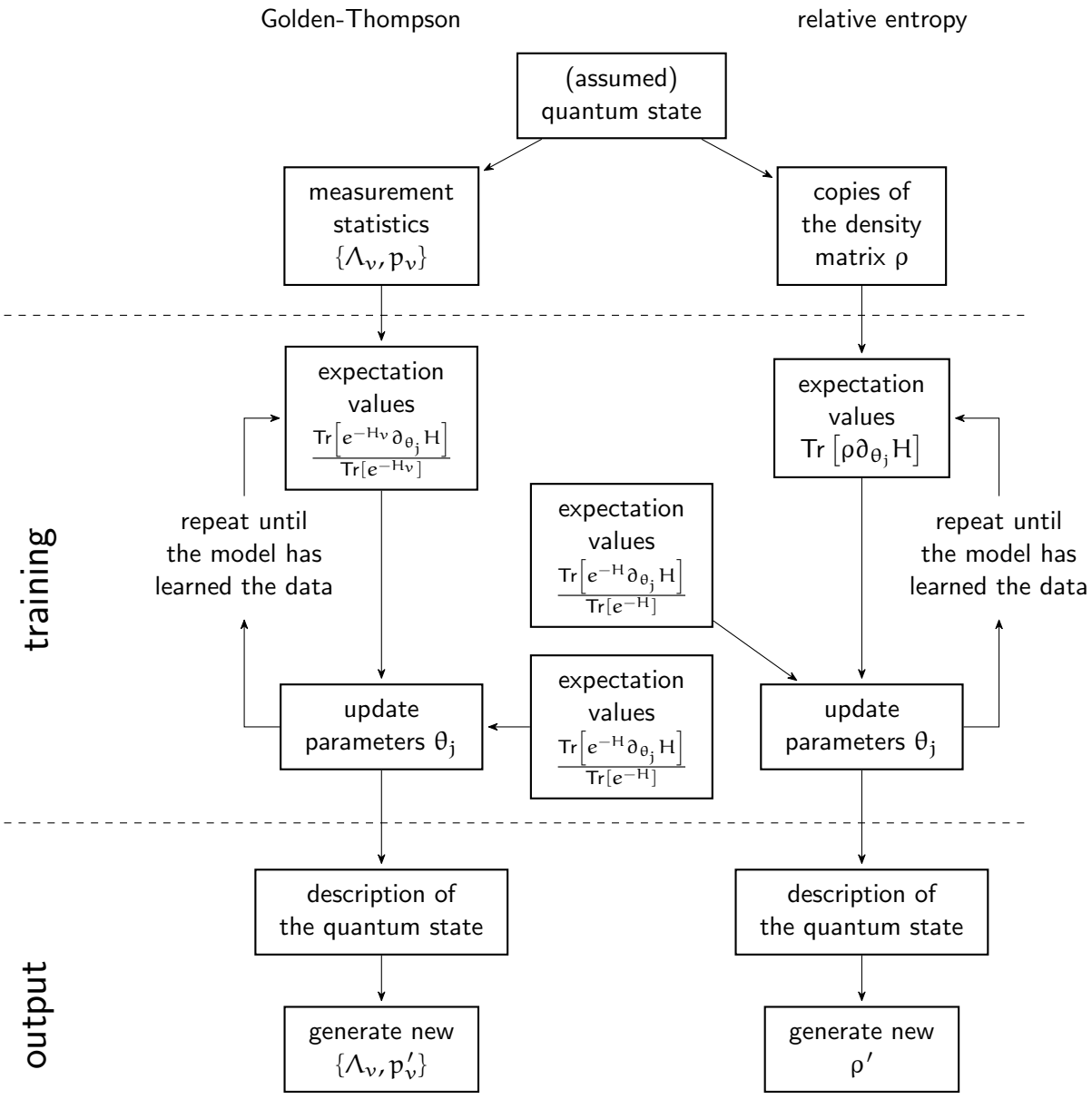


Figure 5.4: Overview of Goldman-Thompson (left) and relative entropy (right) training. In both cases, we are aim to generate samples from an underlying quantum state similarly to the classical generative training in Fig. A.1. We train the models through gradient ascent, where the gradient consists of two terms; one term does depend on the training data and one does not, see Eqs. (5.18) and (5.23). Commutator training is similar to Golden-Thompson training, see Eq. (5.20).

5.4 Complexity Analysis

In this section, we bound the complexity of QBM training. While we are able to obtain several analytic results, the speed of QBM training and the number of copies of the training data must be ultimately determined by benchmarking. In the thesis, we present only the proof of the theorem proved by the author. The theorems proved by our co-author are included for completeness and their proofs can be found in [4].

Let us start by explaining the cost model. We give the complexity of our algorithm as a query complexity with respect to several oracles. We do not explicitly estimate the gate complexity in this work.

The oracle $F_H(\epsilon_H)$ takes the weights and biases of a quantum Boltzmann machine (or equivalently a parameterization of a Hamiltonian) and outputs an approximation σ of the thermal state such that $\|\sigma - e^{-H}/Z\|_{tr} \leqslant \epsilon_H$ for $\epsilon_H \geqslant 0$. Note that we use this oracle for approximating thermal states of H_ν as well as H. In Section 5.5, we review several methods for implementing $F_H(\epsilon_H)$. We do not assume that we have the ability to prepare exact thermal states; this would immediately yield an unrealistic computing model likely more powerful than a quantum computer.

For relative entropy training, we also assume we have access to the training data states ρ through an oracle F_ρ. We take the cost of both oracles to be equal.

Finally, we assume that the POVM elements can be prepared with a constant sized circuit. Therefore, we do not assign a cost to implement them. We do this for two reasons. First, incorporating a cost for POVM elements would affect the implementation of oracle F_H which would force us to specialize to particular state preparation methods. In this sense, the cost for the preparation of POVM elements is incorporated in F_H. Second, the POVM examples are likely to be projectors or similarly simple terms. Their complexity would be therefore comparable to implementing a Hamiltonian term. It might be possible to relax this requirement and only consider oracular implementation of the POVM; we leave this question open for further research.

We first show how the error in approximating the thermal state affects the complexity of POVM training.

Theorem 8 *Let* $H = \sum_{j=1}^{M} \theta_j H_j$ *with* $\|H_j\|_2 = 1 \; \forall \; j$ *be the Hamiltonian for a quantum Boltzmann machine. For notational simplicity, the terms* Λ_ν *are included in the Hamiltonian for POVM-based training. Furthermore, assume that the approximation* σ *of the thermal states are always accessed through the oracle* $F_H(\epsilon_H)$ *such that* $\|e^{-H}/Z - \sigma\|_{tr} \leqslant \epsilon_H$. *Let* G *be an approximation to* $\nabla\mathfrak{O}$ *where* $\mathfrak{O}$ *is the training objective function for either POVM based or relative entropy training. If* $\epsilon > \sqrt{M}\epsilon_H$ *then there exist training algorithms that approximate the gradient using*

$$\mathcal{O}\left(\frac{M}{\epsilon^2 - M\epsilon_H^2}\right)$$

queries to $F_H(\epsilon_H)$ *and the training set times per epoch, where the approximation error* $\mathbb{E}(\|G - G_{true}\|_2^2) \leqslant \epsilon^2$.

We prove this theorem by considering the approximate gradients given by the methods described in this chapter. The algorithms estimate the gradient by sampling the expectation values of local Hamiltonians in the approximate thermal states yielded by $F_H(\epsilon_H)$. Each expectation value is evaluated from n samples. As such, we have a sampling error as well as an error due to using a slightly different state.

The true gradient G_{true} is the vector of expectation values of local Hamiltonians using thermal states $\nabla\mathfrak{O}_\Lambda^{bound}$ or $\nabla\mathfrak{O}_\rho(H)$; see Eqs. (5.8) and (5.23). We evaluate the difference between G_{true} and our estimates G. Since the gradients are vectors, we compute the 2-norm of their difference as

$$\begin{aligned}
\mathbb{E}(\|G - G_{true}\|_2^2) &= \sum_{j=1}^{M} \mathbb{E}\left((G^j - G_{true}^j)^2\right) \\
&= \sum_{j=1}^{M} \left[\mathbb{E}\left((G^j)^2\right) - 2\mathbb{E}\left(G^j G_{true}^j\right) + \mathbb{E}\left((G_{true}^j)^2\right)\right] \\
&= \sum_{j=1}^{M} \left[\mathbb{V}\left(G^j\right) + \mathbb{E}\left(G^j\right)^2 - G_{true}^j 2\mathbb{E}\left(G^j\right) + (G_{true}^j)^2\right] \\
&= \sum_{j=1}^{M} \left[\mathbb{V}\left(G^j\right) + \left(\mathbb{E}\left(G^j\right) - G_{true}^j\right)^2\right]. \qquad (5.24)
\end{aligned}$$

Next, we bound the gradient component $\mathbb{E}\left(G^j\right) = \mathrm{Tr}\left(H_j\sigma\right)$ where σ is an approximation of the thermal state e^{-H}/Z. We are guaranteed that σ is close to the thermal state such that $\left\|\sigma - e^{-H}/Z\right\|_{tr} \leqslant \epsilon_H$. Using standard properties of the trace norm we can bound

$$\left\|H_j e^{-H}/Z - \sigma H_j\right\|_{tr} \leqslant \left\|e^{-H}/Z - \sigma\right\|_{tr} \|H_j\|_2 \leqslant \epsilon_H. \tag{5.25}$$

Thus $|G^j_{true} - \mathbb{E}(G^j)| \leqslant \epsilon_H$ under the assumption that $\|H_j\|_2 \leqslant 1$ for all j.

Next, we estimate the term $\mathbb{V}\left(G^j\right)$ for both relative entropy and POVM training.

For relative entropy training, the components of approximate gradient G_j are estimated using formula (5.23). Assuming $\|H_j\|_2 \leqslant 1$, we can bound the variance of the components

$$\begin{aligned}\mathbb{V}(G^j) &\in \mathcal{O}\left(\frac{1}{n}\max\left\{\mathrm{Tr}\left[\rho H_j\right], \mathrm{Tr}\left[\sigma H_j\right]\right\}\right)\\ &\in \mathcal{O}\left(1/n\right),\end{aligned} \tag{5.26}$$

where ρ is the density matrix corresponding to the ensemble of training vectors. The factor $1/n$ is due to sampling the expectation value n times and $\mathrm{Tr}(\rho H_j)$ and $\mathrm{Tr}(\sigma H_j)$ are both $\mathcal{O}(1)$.

Similarly, we can estimate the variance for Golden-Thompson gradient based on Eq. (5.18). Taking σ_ν to be an approximation of the thermal state $e^{-H_\nu}/\mathrm{Tr}[e^{-H_\nu}]$ within trace distance ϵ_H, we compute the variance

$$\begin{aligned}\mathbb{V}(G^j) &\in \mathcal{O}\left(\frac{1}{n}\max\left\{\mathrm{Tr}\left[H_j\sum_\nu p_\nu\sigma_\nu\right], \mathrm{Tr}\left[H_j\sigma\right]\right\}\right)\\ &\in \mathcal{O}\left(1/n\right).\end{aligned} \tag{5.27}$$

Note that in this context we have implicitly allowed the POVM elements to be considered as Hamiltonian terms in the Boltzmann machine. Thus we can prepare the clamped thermal states e^{-H_ν}/Z_ν within trace distance ϵ_H using one query to $F_H(\epsilon_H)$. Thus in both cases the sample variance of each coordinate of the gradient vector has the same upper bound.

We can plug these results back into (5.24) and bound the error

$$\mathbb{E}(\|G - G_{true}\|_2^2) = \mathcal{O}\left(\frac{M}{n}\right) + M\epsilon_H^2. \tag{5.28}$$

Thus if we wish to take $\mathbb{E}(\|G - G_{\text{true}}\|_2^2) \leqslant \epsilon^2$ it suffices to take $n = \mathcal{O}\left[M/(\epsilon^2 - M\epsilon_H^2)\right]$. This also places a bound on the precision of gradient estimation in terms of precision of the density matrix preparation as $\epsilon > \sqrt{M}\epsilon_H$.

This analysis shows that the gradient can be efficiently estimated by sampling from an imperfect approximation of the gradient. However, we are not able to analytically estimate the number of steps (epochs) needed for the convergence of gradient descent/ascent algorithms. We do not expect that the number of training epochs will scale polynomially with the system size in the worst case scenario. This situation is analogous to classical machine learning, which is often used on NP-complete problems and excel for instances that exhibit additional structure. We will explore convergence on small instances numerically in Section 5.6.

Let us state the additional theorems as they appear in our work. The first result that we show is a lower bound based on tomographic bounds that shows that quantum Boltzmann training cannot be efficient in general if we wish to provide a highly accurate generative model for the training data.

Theorem 9 *The number of queries to F_ρ which yields copies of rank r state operator $\rho \in \mathbb{C}^{D\times D}$ required to train an arbitrary quantum Boltzmann machine using relative entropy such that the quantum state generated by the Boltzmann machine are within trace distance $\epsilon \in (0,1)$ of ρ, and with failure probability $\Theta(1)$, is in $\Omega(Dr/[\epsilon^2 \log(D/r\epsilon)])$.*

The second result states the limitation for training. This result can be proven by Grover's search to Boltzmann training.

Theorem 10 *There does not exist a general purpose POVM-based training algorithm for quantum Boltzmann machines on a training set such that $|\{p_v : p_v > 0\}| = N$ can prepare a thermal state such that $\text{Tr}([\sum_v p_v \Lambda_v] e^{-H}/Z) \geqslant 1/\Delta$ that requires M queries to p_v where $\Delta \in o(\sqrt{N})$ and $M \in o(\sqrt{N})$.*

5.5 Preparing Thermal States

An essential part of Boltzmann machine training is sampling from the thermal distribution. Sadly, preparing the thermal state is NP-hard. Classical

algorithms circumvent this problem by approximating it using contrastive divergence [216]. Additional solutions have been proposed in [50, 226–228]. A high-precision approximation can be obtained using the methods from [91, 229]. We now briefly review a few of these methods.

The method of Chowdhury and Somma is strongly related to the methods in [50, 226, 227]. The main difference between these methods is that their approach uses an integral transformation to allow the exponential to be approximated as a LCU and implemented using Hamiltonian simulation techniques as in Section 2.3.6. The complexity of preparing a thermal state $\rho \in \mathbb{C}^{N\times N}$ within error ϵ, as measured by the 2–norm, is from [229]

$$\mathcal{O}\left(\sqrt{\frac{N}{Z}}\text{polylog}\left(\frac{1}{\epsilon}\sqrt{\frac{N}{Z}}\right)\right), \tag{5.29}$$

for inverse temperature $\beta = 1$, partition function Z and cases where H is explicitly represented as a linear combination of Pauli operators.

Recently, Gilyen et al. [36] improved the above method by using QSVT instead of LCU. This consists of first approximating the transformation by a polynomial series that is probabilistically implemented through QSVT. The success probability is amplified arbitrarily close to 1 with amplitude amplification. If N is the dimension of the Hilbert space, $\mathcal{O}\left(\sqrt{\frac{N}{Z}}\right)$ rounds of amplitude amplification are required.

Yung and Aspuru-Guzik [227] proposed an alternative approach for thermal state preparation based on a quantum walk. In particular, the Metropolis algorithm can be implemented in the quantum setting using a Szegedy walk operator. The complexity of this approach depends on the difference of energies between the ground state and the lowest excited state. These eigenvalues are computed using phase estimation. The number of applications of a controlled walk operator required by phase estimation is $\mathcal{O}\left(\frac{\|H\|^2}{\epsilon\sqrt{\delta}}\log\left(\frac{\|H\|^2}{\epsilon^2}\right)\right)$, where δ is the gap of the transition matrix that defines the quantum walk, and ϵ is the error in the preparation of the thermal state. In addition, one needs to consider the complexity of applying the walk operator. As such, it is not known which out of the described algorithms would be preferable.

Additionally, there are promising empirical methods for preparing thermal states. Recently, Anschuetz and Cao [230] suggested that a QBM can

be trained using the Eigenstate Thermalization Hypothesis [231, 232]. Alternatively, one can achieve an approximation of the thermal state with a quantum annealer [225, 233]. However, this method is applicable only for limited types of Hamiltonians. Lastly, there are thermal state preparation techniques based on QAOA and VQA [234, 235]. Unlike the techniques reviewed in the previous paragraphs, all of these techniques can be implemented on near-term hardware.

5.6 Numerical Results

Now we present our numerical results. We performed two types of simulations. The first type used a fermionic Hamiltonian to learn a classical data set encoded in a quantum state. The second type of simulation performs tomography on a 2-qubit density matrix. We will now discuss the first type of simulation and compare the results to classical training.

5.6.1 The Data Set and the Hamiltonian

Given the limitations of classical computers, we could examine the performance of QBMs only on a small number of units. Even though we were able to simulate only very small QBMs, up to 9 units in total, we performed a thorough sweep through the parameters of the simulation in order to compare the best performance of different models. Since the model was small, we were able to analytically compute the thermal state and the negative log-likelihood. This allowed us to directly assess the performance of the QBM instead of using a validation data set.

We chose a simple data set composed of strings starting with some number of 0s followed by 1s, the step function. We also added noise to the training vectors; each bit had a 5% chance of a flip. We then encoded the data into a single quantum state $|\psi\rangle$ as in Eq. (5.12). For POVM training, we constructed the projectors to be $\Lambda_1 = |\psi\rangle\langle\psi|$ and $\Lambda_0 = \mathbb{I} - |\psi\rangle\langle\psi|$.

To capture the full power of quantum computing (instead of constraining ourselves to stoquastic Hamiltonians), we used a fermionic Hamiltonian

for training from [236]:

$$H = H_1 + \frac{1}{2}H_2 + \frac{1}{2}H_4\,, \tag{5.30}$$

where

$$H_1 = \sum_p h_p \left(a_p + h.c.\right), \tag{5.31}$$

$$H_2 = \sum_{pq} h_{pq} \left(a_p^\dagger a_q + h.c.\right), \tag{5.32}$$

$$H_4 = \sum_{pqrs} h_{pqrs} \left(a_p^\dagger a_q^\dagger a_r a_s + h.c.\right). \tag{5.33}$$

Here a_p and $a_p^\dagger$ are fermionic creation and annihilation operators, which create and destroy fermions at the QBM unit p. They have the properties that $a^\dagger|0\rangle = |1\rangle$, $a^\dagger|1\rangle = 0$ and $a_p^\dagger a_q + a_q a_p^\dagger = \delta_{pq}$. The Hamiltonian here corresponds to the standard Hamiltonian used in quantum chemistry modulo the presence of the non-particle conserving H_p term.

Note that the y-axes in this section depict negative objective values. This is because the objective values (5.17) and (5.22) are both upper bounded by zero.

5.6.2 Parameters of QBM Training

When using any of the training methods outlined in this chapter, one is still left with a freedom to choose a number of additional parameters.

The first one is the number of hidden units. Classically, using hidden units is almost always necessary and a Boltzmann machine with all visible units has limited expressive power. We were changing the number of hidden units for our quantum model between 0 and 2. We found out that our QBMs required much fewer hidden units than their classical counterparts.

The next parameter is the step size within gradient descent/ascent, also called the learning rate. Often, one starts with fairly large steps and keeps decreasing them as the model learns. However, this can give a false sense of convergence which was the case in our earlier simulations. Instead,

we decided to keep the learning rate constant. We searched over different learning rates and used the ones that performed the best. Note that different models preferred different learning rates.

The next parameter is momentum. We noticed that using momentum significantly improved the convergence of learning, especially after many steps of learning when the gradient becomes small. Using momentum helps the training to converge faster if the gradient for consecutive steps points in the same direction.

The last freedom that we explored was regularization. Regularization is a set of methods to prevent overfitting and unnecessarily complicated models. We chose to L2 regularization, that adds a penalty to the objective function proportional to the square of weights h_{pq} and h_{pqrs} corresponding to non-diagonal terms in the Hamiltonian. L2 regularization is a common choice because of the simplicity of its derivatives and good performance for a wide range of models.

We now present our numerical results. We repeated each simulation 100 times and present median values. Since the value of the objective function depends on the dimension as well as the data, we compare the difference between the achieved negative log-likelihood and the maximum one (exact fit) in the plots below.

5.6.3 Golden-Thompson Training Analysis

We focused our efforts on Golden-Thompson training. We explored models with 4 to 8 visible units and a varied number of hidden units. Our QBMs consistently outperformed classical BMs in terms of accuracy, even if we added hidden units to the hidden models. As shown in Fig. 5.5, the quantum models learned to approximate the training data much more closely than their classical counterparts. This suggests that QBMs can be beneficial when classical machine learning techniques underfit data. Surprisingly, adding hidden units to a QBM did not improve the quality of the approximation. The simulations for 6 and more units showed similar results as shown in Fig. 5.5.

We compare the performance of QBMs with a range of numbers of visible units (and zero hidden units) in Fig. 5.6. As expected, larger QBMs

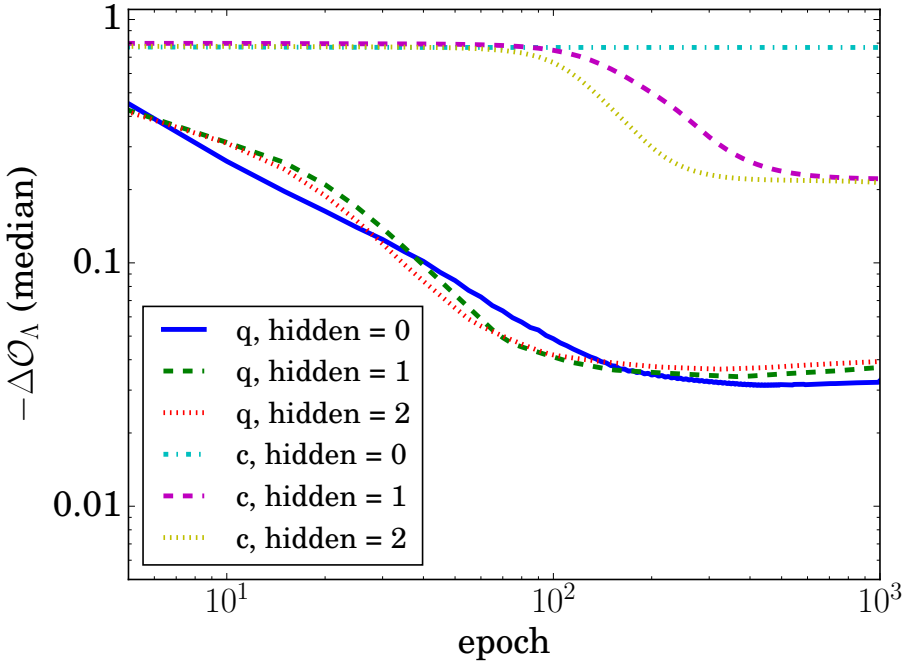


Figure 5.5: Comparison of classical and quantum Boltzmann machines (with Golden-Thompson training) for 5 visible units. This figure first appeared in [4]. The vertical axis shows the difference between asymptotic value of the objective function and the objective function for the learned model in a given epoch.

require longer to learn. We were not able to determine how the complexity of QBM training scales with the number of visible units. Presumably, this can heavily depend on the training data and in some cases grow exponentially.

5.6.4 Commutator Training Analysis

We examine the performance of commutator training for an all-visible Boltzmann machine with 4 visible units and compare it with Golden-Thompson training. We immediately noticed that this type of the training is not numerically stable. In particular, we were not able to use commutator training at all when the learning rate or the gradient were too large – the learning

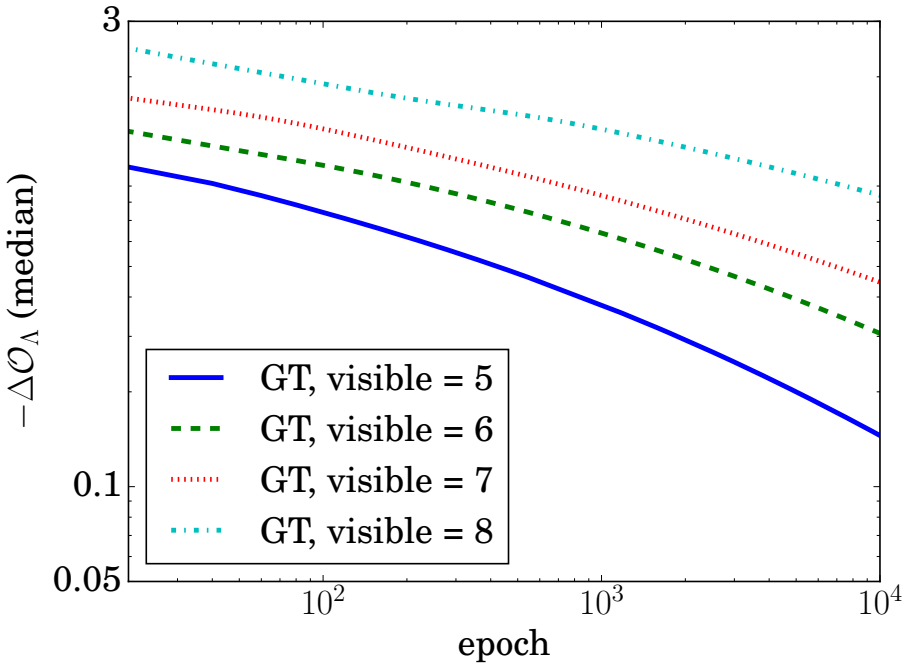


Figure 5.6: Performance of QBMs with a range of numbers of visible units and no hidden units. This figure first appeared in [4].

did not converge.

We were only able to use commutator training for later stages of learning when the gradient estimated by Golden-Thompson was sufficiently small. We then compared commutator and Golden-Thompson training with a fixed set of parameters such as the learning rate and the momentum. Under these conditions and for cases when learning converged, a high-order commutator expansion performed much better than Golden-Thompson; see Fig. 5.7. This, in turn, illustrates the difference between exact gradients and Golden-Thompson gradients.

In particular, we trained for a fixed number of epochs using the Golden-Thompson gradients and then switched to a 5^{th}–order commutator expansion. We saw a dramatic improvement in the objective function as a result. This shows that in some circumstances much better gradients can be found with the commutator method than with Golden-Thompson, albeit at a higher price because more expectation values need to be mea-

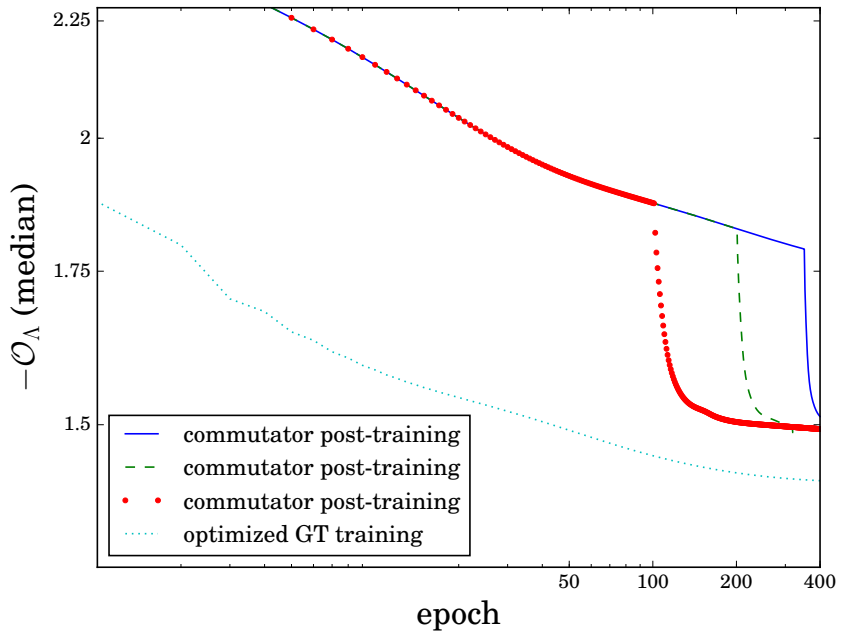


Figure 5.7: Plot showing the accuracy of commutator training for all-visible Boltzmann machines with 4 visible units. The top lines depict training with Golden-Thompson at first and then switching to commutator training where we see a sudden increase in accuracy. We picked the parameters such that the commutator training is stable. The bottom line (dashed) shows the performance of Golden-Thompson training with optimized learning rate and momentum. We always kept the learning rate constant during the training; changing the learning rate might improve the learning outcome but introduces another degree of complexity. This figure first appeared in [4].

sured. In other words the gradients estimated from Golden-Thompson are not always close to the gradients of the negative log-likelihood (without approximation).

We also noticed that the optimal learning rate for this form of training can substantially differ from the optimal learning rate for Golden-Thompson training. We compared our results from commutator training to the results with optimized learning rate for Golden-Thompson training.

We found out that the optimized version of Golden-Thompson training performed much better. This version of Golden-Thompson training used a high learning rate that was incompatible with commutator training because of numerical instabilities. As such, we could not find a case when commutator training proved useful; it only showed an advantage when compared to highly constrained Golden-Thompson training.

This shows that while commutator training can give more accurate gradients, it does not necessarily require fewer gradient steps and its use is very limited. In practice, the method could be used in the last few training epochs after Golden–Thompson training or other forms of approximate training to reach a local optimum, but its use is otherwise limited.

5.6.5 Relative Entropy Analysis

We can train the QBM with relative entropy training using the same classical data set as POVM training. We successfully reproduce results from Golden Thompson training and find out that two approaches converge similarly; see Fig. 5.8. We were not able to train QBMs with hidden units; however, including hidden units from QBMs was in general unnecessary for our simple data.

Next, we train our QBM to reconstruct two-qubit states using relative entropy training. This task demonstrates that QBMs can be used for tomography. We trained QBMs on both pure and mixed states, and in all cases our model was able to reconstruct these states within graphical accuracy (i.e. the learned state is visually indistinguishable from the true state) in 5 epochs as seen in Fig. 5.9. Since we chose our Hamiltonian to consist of every two-qubit Pauli operator, we did not need to use hidden units. A limitation of this approach was the requirement of all visible units. Wossling and Wiebe [237] recently provided a method for training QBMs with hidden units.

5.7 Conclusion

In this chapter, we proposed methods to train quantum Boltzmann machines on quantum data. We demonstrated that the models can successfully

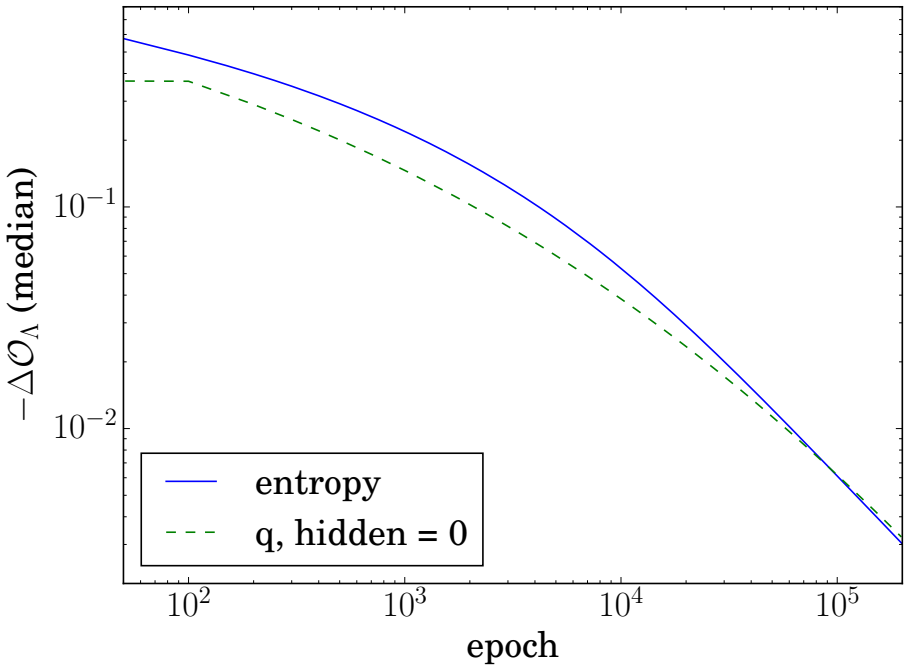


Figure 5.8: Comparison of performance of a QBM using Golden-Thompson POVM training and relative entropy training. In our experiments, the two approaches performed similarly. This figure first appeared in [4].

learn under idealized conditions, but more research is needed to prove the usefulness of QBM training.

The first open question is how well QBMs learn under realistic assumptions. That is, in practice we will not have direct access to thermal states or precise estimates of expectation values. When training classical BMs, the issue of Gibbs sampling is circumvented by using contrastive divergence and explicitly computing certain expectation values. It is not clear how many samples would be required for an accurate estimate of expectation values for gradient computation. Also, it would be interesting to explore whether there is an analog of contrastive divergence for QBM training.

The second question is the inclusion of hidden units for relative entropy training. Hidden units would be necessary for constructing deep quantum Boltzmann machines and for training only with local Hamiltonians. Recently, Wiebe and Wossnig [237] proposed methods for training such

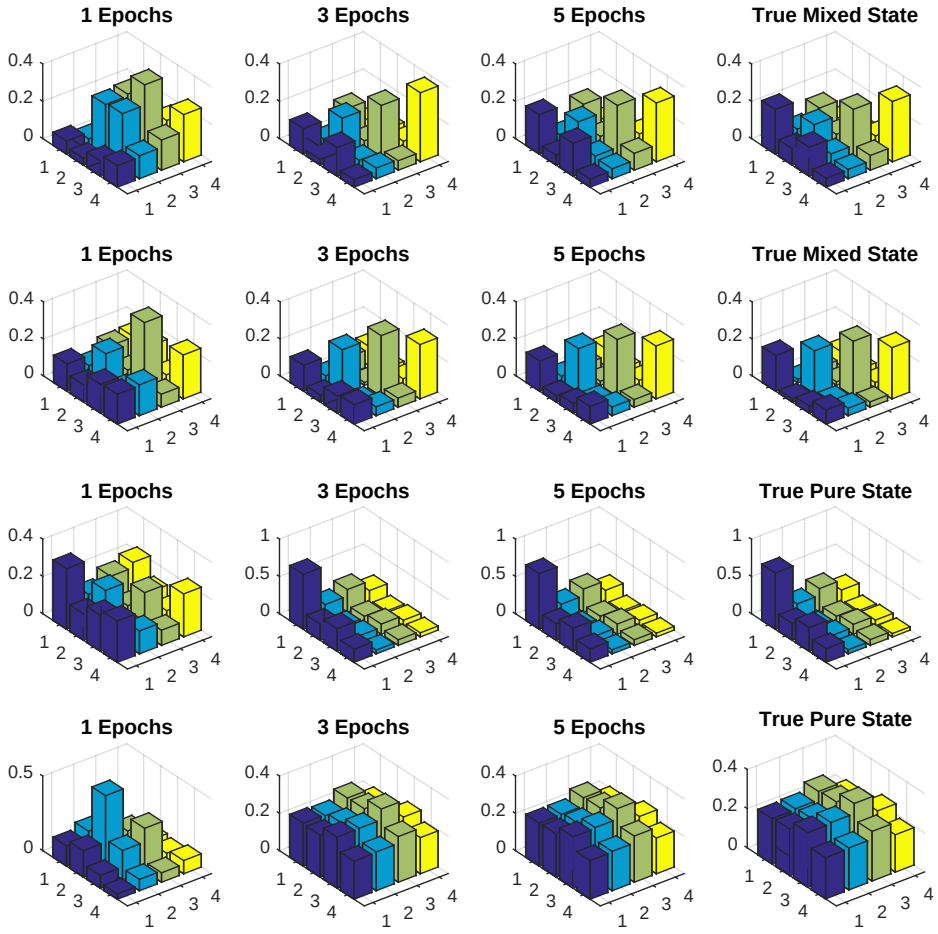


Figure 5.9: Absolute values of tomographic reconstruction of two-qubit pure states and mixed states using relative entropy training with learning rate $\eta = 1$. This figure first appeared in [4].

QBMs.

We mostly avoided the topic of overfitting data. This is because characterizing quantum states is a resource-intensive task and having too much quantum processing power is not likely to be an issue in the near future. Eventually, we will need to introduce methods to prevent us from learning sampling errors. One such method is L2 regularization that we incorporated in our numerical experiments. It would be interesting to examine how classical regularization techniques perform in the quantum setting,

as well as to develop new overfitting techniques. Additionally, we will eventually need to use separate data for validation of QML algorithms.

Lastly, the most important strength of QBMs is their ability to replace tomography for quantum states with structure. Since we require only approximations of thermal states and computation of expectation values, relative entropy training is near ideally suited for near future experiments. We hope that combining ideas from quantum machine learning, quantum Hamiltonian learning, and state estimation will lead to more efficient methods that will be eventually employed in practice.

6

Conclusion and Future Work

"Full stories are as rare as honesty."

Zadie Smith (White Teeth)

In this thesis, we presented several techniques for building efficient quantum algorithms. Our goal was to improve quantum algorithm performance for practical tasks, such as estimating energies in quantum chemistry.

The first set of techniques, quantum comparison, sort, and shuffle, are presented in Chapter 3 and based on our paper [2]. We developed a quantum shuffle to antisymmetrize fermionic states in first quantization quickly. The components of one of our shuffling algorithms, comparison and sort, later found applications in state preparation [116] and simulation of time-independent Hamiltonians [3].

Chapter 4 focused on our algorithm for simulating time-dependent Hamiltonians. This algorithm has applications for simulating quantum chemistry with moving nuclei, a digital simulation of adiabatic computation, and simulations in the interaction picture. The next step will be to estimate the resources of using this method for benchmark molecules in

quantum chemistry. Calculating gate counts will also determine which one of our state preparation methods is more efficient.

Our last methods are for generative learning and tomography with quantum Boltzmann machines based on [4]. We demonstrate how to use quantum Boltzmann machines for training on quantum and classical data. We envision that quantum Boltzmann machines can be used for classical data that conventional algorithms underfit, or for partial tomography on quantum states.

Throughout this thesis, we raised many open questions. The first one is what are the resource requirements of our algorithms. We quantified our algorithms in terms of their asymptotic scaling or through numerical simulations for small examples. In particular, it would be interesting to see how shuffling and truncated Dyson series algorithms perform on quantum chemistry problems. We plan to estimate the explicit gate counts for our Dyson series algorithm in the near future.

In Chapter 5, we showed that we can train quantum Boltzmann machines on quantum data. However, there are many more steps needed to show that such training is efficient. A method for relative entropy training with hidden units [237] is one such step. Another one is creating approximations of training states, aka quantum contrastive divergence. We will also need to carefully examine the performance of QBMs with a growing number of units. Lastly, it would be interesting to investigate whether creating additional layers and forming a deep quantum Boltzmann machine brings any advantages.

Lastly, we would like to apply our techniques on a wider array of problems. Comparison and sort are simple routines with many applications in classical computing and we would like to use our efficient circuits for them in other quantum algorithms. Similarly, we want to know the scope of applications of QBMs for tomography, approximate cloning and possibly state preparation. We hope that the techniques introduced in this thesis will prove to be widely applicable.

References

[1] Yudong Cao, Jonathan Romero, Jonathan P Olson, Matthias Degroote, Peter D Johnson, Mária Kieferová, Ian D Kivlichan, Tim Menke, Borja Peropadre, Nicolas PD Sawaya, et al. Quantum chemistry in the age of quantum computing. *Chemical Reviews*, 2018.

[2] Dominic W Berry, Mária Kieferová, Artur Scherer, Yuval R Sanders, Guang Hao Low, Nathan Wiebe, Craig Gidney, and Ryan Babbush. Improved techniques for preparing eigenstates of fermionic Hamiltonians. *npj Quantum Information*, 4:22, 2018.

[3] Mária Kieferová, Artur Scherer, and Dominic W Berry. Simulating the dynamics of time-dependent Hamiltonians with a truncated Dyson series. *Physical Review A*, 99(4):042314, 2019.

[4] Mária Kieferová and Nathan Wiebe. Tomography and generative training with quantum Boltzmann machines. *Physical Review A*, 96:062327, 2017.

[5] Aram W Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical Review Letters*, 103(15):150502, 2009.

[6] Richard P Feynman. Quantum mechanical computers. *Foundations of Physics*, 16(6):507–531, 1986.

[7] David Deutsch. Quantum theory, the Church–Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 400(1818):97–117, 1985.

[8] David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London Series A*, 439:553–558, 1992.

[9] Vladimír Černý. Quantum computers and intractable (NP-complete) computing problems. *Physical Review A*, 48(1):116, 1993.

[10] Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26(5):1411–1473, 1997.

[11] Daniel R Simon. On the power of quantum computation. *SIAM Journal on Computing*, 26(5):1474–1483, 1997.

[12] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Review*, 41(2):303–332, 1999.

[13] Seth Lloyd. Universal quantum simulators. *Science*, 273(5278):1073–1078, 1996.

[14] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, pages 212–219. ACM, 1996.

[15] Mario Szegedy. Quantum speed-up of markov chain based algorithms. In *45th Annual IEEE Symposium on Foundations of Computer Science*, pages 32–41. IEEE, 2004.

[16] Andris Ambainis. Quantum walk algorithm for element distinctness. *SIAM Journal on Computing*, 37(1):210–239, 2007.

[17] Andrew M Childs, Richard Cleve, Enrico Deotto, Edward Farhi, Sam Gutmann, and Daniel A Spielman. Exponential algorithmic speedup by a quantum walk. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, pages 59–68. ACM, 2003.

[18] Andrew M Childs and Jeffrey Goldstone. Spatial search by quantum walk. *Physical Review A*, 70(2):022314, 2004.

[19] Andrew M Childs. Universal computation by quantum walk. *Physical Review Letters*, 102(18):180501, 2009.

[20] Andris Ambainis, Eric Bach, Ashwin Nayak, Ashvin Vishwanath, and John Watrous. One-dimensional quantum walks. In *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing*, pages 37–49. ACM, 2001.

[21] Frédéric Magniez, Ashwin Nayak, Jérémie Roland, and Miklos Santha. Search via quantum walk. *SIAM Journal on Computing*, 40(1):142–164, 2011.

[22] Frédéric Magniez, Ashwin Nayak, Peter C Richter, and Miklos Santha. On the hitting times of quantum versus random walks. *Algorithmica*, 63(1-2):91–116, 2012.

[23] Stacey Jeffery, Robin Kothari, and Frédéric Magniez. Nested quantum walks with quantum data structures. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete algorithms*, pages 1474–1485. Society for Industrial and Applied Mathematics, 2013.

[24] Neil B Lovett, Sally Cooper, Matthew Everitt, Matthew Trevers, and Viv Kendon. Universal quantum computation using the discrete-time quantum walk. *Physical Review A*, 81(4):042330, 2010.

[25] Ashley Montanaro. Quantum walk speedup of backtracking algorithms. *arXiv:1509.02374*, 2015.

[26] Andrew M Childs, David Gosset, and Zak Webb. Universal computation by multiparticle quantum walk. *Science*, 339(6121):791–794, 2013.

[27] Andrew M Childs. On the relationship between continuous-and discrete-time quantum walk. *Communications in Mathematical Physics*, 294(2):581–603, 2010.

[28] Dorit Aharonov, Wim van Dam, Julia Kempe, Zeph Landau, Seth Lloyd, and Odev Regev. Adiabatic Quantum Computation is Equivalent to Standard Quantum Computation. In *45th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, 2004.

[29] 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, 2006.

[30] Nathan Wiebe, Dominic Berry, Peter Høyer, and Barry C Sanders. Higher order decompositions of ordered operator exponentials. *Journal of Physics A: Mathematical and Theoretical*, 43(6):065203, 2010.

[31] Dominic W Berry and Andrew M Childs. Black-box Hamiltonian simulation and unitary implementation. *Quantum Information and Computation*, 12:0029–0062, 2012.

[32] Dominic W Berry, Andrew M Childs, and Robin Kothari. Hamiltonian simulation with nearly optimal dependence on all parameters. In *Proceedings of the 56th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 792–809, 2015.

[33] Dominic W Berry, Richard Cleve, and Rolando D Somma. Exponential improvement in precision for Hamiltonian-evolution simulation. *arXiv:1308.5424*, 2013.

[34] Dominic W Berry, Andrew M Childs, Richard Cleve, Robin Kothari, and Rolando D Somma. Simulating Hamiltonian dynamics with a truncated Taylor series. *Physical Review Letters*, 114(9):090502, 2015.

[35] Guang Hao Low and Isaac L Chuang. Optimal Hamiltonian simulation by quantum signal processing. *Physical Review Letters*, 118:010501, 2017.

[36] András Gilyén, Yuan Su, Guang Hao Low, and Nathan Wiebe. Quantum singular value transformation and beyond: exponential improvements for quantum matrix arithmetics. *arXiv:1806.01838*, 2018.

[37] Guang Hao Low and Nathan Wiebe. Hamiltonian simulation in the interaction picture. *arXiv:1805.00675*, 2018.

[38] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. Quantum computation by adiabatic evolution. *arXiv:0001106*, 2000.

[39] Wim Van Dam, Michele Mosca, and Umesh Vazirani. How powerful is adiabatic quantum computation? In *Proceedings 2001 IEEE International Conference on Cluster Computing*, pages 279–287. IEEE, 2001.

[40] Andrew M Childs, Edward Farhi, and John Preskill. Robustness of adiabatic quantum computation. *Physical Review A*, 65(1):012322, 2001.

[41] Jérémie Roland and Nicolas J Cerf. Adiabatic quantum search algorithm for structured problems. *Physical Review A*, 68:62312, 2003.

[42] Tameem Albash and Daniel A Lidar. Adiabatic quantum computation. *Reviews of Modern Physics*, 90(1):015002, 2018.

[43] Rolando D Somma, Daniel Nagaj, and Mária Kieferová. Quantum Speedup by Quantum Annealing. *Physical Review Letters*, 109(5):50501–50506, 2012.

[44] Rolando D Somma, Daniel Nagaj, and Mária Kieferová. Quantum speedup by quantum annealing. *Physical Review Letters*, 109(5):050501, 2012.

[45] Greg Kuperberg. A subexponential-time quantum algorithm for the dihedral hidden subgroup problem. *SIAM Journal on Computing*, 35(1):170–188, 2005.

[46] Daniel J Bernstein, Stacey Jeffery, Tanja Lange, and Alexander Meurer. Quantum algorithms for the subset-sum problem. In *International Workshop on Post-Quantum Cryptography*, pages 16–33. Springer, 2013.

[47] Stephen P Jordan, Keith SM Lee, and John Preskill. Quantum algorithms for quantum field theories. *Science*, 336(6085):1130–1133, 2012.

[48] Esteban A Martinez, Christine A Muschik, Philipp Schindler, Daniel Nigg, Alexander Erhard, Markus Heyl, Philipp Hauke, Marcello Dalmonte, Thomas Monz, Peter Zoller, et al. Real-time dynamics of lattice gauge theories with a few-qubit quantum computer. *Nature*, 534(7608):516, 2016.

[49] Gavin K Brennen, Peter Rohde, Barry C Sanders, and Sukhwinder Singh. Multiscale quantum simulation of quantum field theory using wavelets. *Physical Review A*, 92(3):032315, 2015.

[50] Nathan Wiebe, Ashish Kapoor, and Krysta M Svore. Quantum deep learning. *Quantum Information and Computation*, 16:0541–0587, 2016.

[51] MV Altaisky. Quantum neural network. *arXiv:0107012*, 2001.

[52] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. The quest for a quantum neural network. *Quantum Information Processing*, 13(11):2567–2586, 2014.

[53] Edward Farhi and Hartmut Neven. Classification with quantum neural networks on near term processors. *arXiv:1802.06002*, 2018.

[54] John Preskill. Quantum computing in the NISQ era and beyond. *arXiv:1801.00862*, 2018.

[55] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J Love, Alán Aspuru-Guzik, and Jeremy L O'Brien. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5:4213, 2014.

[56] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm. *arXiv:1411.4028*, 2014.

[57] David P DiVincenzo. The physical implementation of quantum computation. *Fortschritte der Physik: Progress of Physics*, 48(9-11):771–783, 2000.

[58] Julian Kelly. A pReview of Bristlecone, Google's new quantum processor. `https://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html`, 2018.

[59] Rigetti's 19Q superconducting quantum processor. `https://www.eurekalert.org/multimedia/pub/162061.php`, 2018.

[60] Quantum information science in the national spotlight. `https://www.aps.org/publications/apsnews/201810/quantum.cfm`, 2018.

[61] IBM's newly revealed 50-qubit quantum processor will be available in next generation of IBM Q systems. `http://www.softcarecs.com/ibms-newly-revealed-50-qubit-quantum-processor-will-be-availale-in-next-generation-of-ibm-q-systems/`, 2017.

[62] Making the world's first integrated quantum system. https://newsroom.intel.com/wp-content/uploads/sites/11/2018/05/49-qubit-processor-tangle-lake-infographic.jpg, 2018.

[63] Xiaogang Qiang, Xiaoqi Zhou, Jianwei Wang, Callum M Wilkes, Thomas Loke, Sean O'Gara, Laurent Kling, Graham D Marshall, Raffaele Santagati, Timothy C Ralph, et al. Large-scale silicon quantum photonics implementing arbitrary two-qubit processing. *Nature Photonics*, 12(9):534, 2018.

[64] Making the world's first integrated quantum system. https://www.research.ibm.com/ibm-q/system-one/, 2019.

[65] Chad Rigetti. The Rigetti 128-qubit chip and what it means for quantum. https://medium.com/rigetti/the-rigetti-128-qubit-chip-and-what-it-means-for-quantum-df757d1b71ea, 2018.

[66] The Rigetti 128-qubit chip and what it means for quantum. https://ionq.co/.

[67] The path to the world's most powerful quantum computer. https://www.honeywell.com/en-us/newsroom/news/2019/01/the-path-to-the-world-s-most-powerful-quantum-computer, 2019.

[68] Austin G Fowler, Matteo Mariantoni, John M Martinis, and Andrew N Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, 86(3):032324, 2012.

[69] Craig Gidney and Martin Ekerå. How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. *arXiv:1905.09749*, 2019.

[70] Ivan Kassal, Stephen P Jordan, Peter J Love, Masoud Mohseni, and Alán Aspuru-Guzik. Polynomial-time quantum algorithm for the simulation of chemical dynamics. *Proceedings of the National Academy of Sciences*, 105(48):18681–18686, 2008.

[71] Alán Aspuru-Guzik, Anthony D Dutoi, Peter J Love, and Martin Head-Gordon. Simulated quantum computation of molecular energies. *Science*, 309(5741):1704, 2005.

[72] Ryan Babbush, Dominic W Berry, Ian D. Kivlichan, Annie Y. Wei, Peter J. Love, and Alán Aspuru-Guzik. Exponentially more precise quantum simulation of fermions in second quantization. *New Journal of Physics*, 18(3):033032, 2016.

[73] Ryan Babbush, Craig Gidney, Dominic W Berry, Nathan Wiebe, Jarrod McClean, Alexandru Paler, Austin Fowler, and Hartmut Neven. Encoding electronic spectra in quantum circuits with linear t complexity. *Physical Review X*, 8(4):041015, 2018.

[74] Ryan Babbush, Dominic W Berry, Jarrod R. McClean, and Hartmut Neven. Quantum simulation of chemistry with sublinear scaling to the continuum. *arXiv:1807.09802*, 2018.

[75] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[76] Sadegh Raeisi, Mária Kieferová, and Michele Mosca. Novel technique for robust optimal algorithmic cooling. *Physical Review Letters*, 122:220501, Jun 2019.

[77] Mária Kieferová and Nathan Wiebe. On the power of coherently controlled quantum adiabatic evolutions. *New Journal of Physics*, 16(12):123034, 2014.

[78] Scott Aaronson, Greg Kuperberg, and Christopher Granade. The Complexity Zoo. `https://complexityzoo.uwaterloo.ca/Complexity_Zoo`, 2005.

[79] Phillip Kaye, Raymond Laflamme, and Michele Mosca. *An introduction to quantum computing*. Oxford University Press, 2007.

[80] Chris Ferrie. *Quantum Computing for Babies*. Jabberwocky, 2017.

[81] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2010.

[82] Stephen Jordan. Quantum Algorithm Zoo. `https://math.nist.gov/quantum/zoo/`.

[83] Michele Mosca. Quantum algorithms. *Encyclopedia of Complexity and Systems Science*, pages 7088–7118, 2009.

[84] Dave Bacon and Wim VAn DAm. Recent progress in quantum algorithms. *Communications of the ACM*, 53(2):84–93, 2010.

[85] Ashley Montanaro. Quantum algorithms: an overview. *npj Quantum Information*, 2:15023, 2016.

[86] Adriano Barenco, Charles H Bennett, Richard Cleve, David P DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *Physical Review A*, 52:3457–3467, 1995.

[87] Michael A Nielsen, Michael J Bremner, Jennifer L Dodd, Andrew M Childs, and Christopher M Dawson. Universal simulation of Hamiltonian dynamics for quantum systems with finite-dimensional state spaces. *Physical Review A*, 66(2):022317, 2002.

[88] Christopher M Dawson and Michael A Nielsen. The Solovay-Kitaev algorithm. *arXiv:0505030*, 2005.

[89] Vadym Kliuchnikov, Dmitri Maslov, and Michele Mosca. Asymptotically optimal approximation of single qubit unitaries by Clifford and T circuits using a constant number of ancillary qubits. *Physical Review Letters*, 110:190502, 2013.

[90] Robert B Griffiths and Chi-Sheng Niu. Semiclassical Fourier transform for quantum computation. *Physical Review Letters*, 76:3228–3231, 1996.

[91] Craig Gidney. Halving the cost of quantum addition. *Quantum*, 2:74, 2018.

[92] Austin G Fowler, Ashley M Stephens, and Peter Groszkowski. High-threshold universal quantum computation on the surface code. *Physical Review A*, 80(5):052312, 2009.

[93] Sergey Bravyi and Alexei Kitaev. Universal quantum computation with ideal Clifford gates and noisy ancillas. *Physical Review A*, 71(2):022316, 2005.

[94] Daniel Gottesman. Stabilizer codes and quantum error correction. *arXiv:9705052*, 1997.

[95] Larry J Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

[96] Dorit Aharonov and Michael Ben-Or. Fault-tolerant quantum computation with constant error rate. *arxiv:9906129*, 1999.

[97] Daniel Gottesman. An introduction to quantum error correction and fault-tolerant quantum computation. In *Quantum information science and its contributions to mathematics, Proceedings of Symposia in Applied Mathematics*, volume 68, pages 13–58, 2010.

[98] Yuval R Sanders, Joel J Wallman, and Barry C Sanders. Bounding quantum gate error rate based on reported average fidelity. *New Journal of Physics*, 18(1):012002, 2015.

[99] John Preskill. Sufficient condition on noise correlations for scalable quantum computing. *arXiv:1207.6131*, 2012.

[100] Gil Kalai. Quantum computers: noise propagation and adversarial noise models. *arXiv:0904.3265*, 2009.

[101] Peter W Shor. Scheme for reducing decoherence in quantum computer memory. *Physical Review A*, 52:R2493–R2496, 1995.

[102] Chenyang Wang, Jim Harrington, and John Preskill. Confinement-Higgs transition in a disordered gauge theory and the accuracy threshold for quantum memory. *Annals of Physics*, 303(1):31–58, 2003.

[103] Michael Freedman, Alexei Y Kitaev, Michael Larsen, and Zhenghan Wang. Topological quantum computation. *Bulletin of the American Mathematical Society*, 40(1):31–38, 2003.

[104] Sergey B Bravyi and Alexei Y Kitaev. Quantum codes on a lattice with boundary. *arXiv:9811052*, 1998.

[105] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. Topological quantum memory. *Journal of Mathematical Physics*, 43(9):4452–4505, 2002.

[106] Tommaso Toffoli. Reversible computing. In *International Colloquium on Automata, Languages, and Programming*, pages 632–644. Springer, 1980.

[107] Edward Fredkin and Tommaso Toffoli. Conservative logic. *International Journal of Theoretical Physics*, 21(3-4):219–253, 1982.

[108] Rolf Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5(3):183–191, 1961.

[109] Charles H Bennett. Notes on the history of reversible computation. *ibm Journal of Research and Development*, 32(1):16–23, 1988.

[110] Charles H Bennett. Notes on Landauer's principle, reversible computation, and Maxwell's Demon. *Studies In History and Philosophy of Science Part B: Studies In History and Philosophy of Modern Physics*, 34(3):501–510, 2003.

[111] Alex Parent, Martin Roetteler, and Michele Mosca. Improved reversible and quantum circuits for Karatsuba-based integer multiplication. *arXiv:1706.03419*, 2017.

[112] Vivek V Shende, Aditya K Prasad, Igor L Markov, and John P Hayes. Synthesis of reversible logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(6):710–722, 2003.

[113] Mehdi Saeedi and Igor L Markov. Synthesis and optimization of reversible circuits–a survey. *ACM Computing Surveys (CSUR)*, 45(2):21, 2013.

[114] D Michael Miller, Dmitri Maslov, and Gerhard W Dueck. A transformation based algorithm for reversible logic synthesis. In *Proceedings 2003. Design Automation Conference (IEEE Cat. No. 03CH37451)*, pages 318–323. IEEE, 2003.

[115] Mathias Soeken, Martin Roetteler, Nathan Wiebe, and Giovanni De Micheli. Logic synthesis for quantum computing. *arXiv:1706.02721*, 2017.

[116] Yuval R Sanders, Guang Hao Low, Artur Scherer, and Dominic W Berry. Black-box quantum state preparation without arithmetic. *Physical Review Letters*, 122:020502, 2019.

[117] Richard Cleve, Artur Ekert, Chiara Macchiavello, and Michele Mosca. Quantum algorithms revisited. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 454, pages 339–354. The Royal Society, 1998.

[118] A Yu Kitaev. Quantum measurements and the Abelian stabilizer problem. *arxiv:9511026*, 1995.

[119] Lisa Hales and Sean Hallgren. An improved quantum Fourier transform algorithm and applications. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 515–525. IEEE, 2000.

[120] David Poulin, Alexei Kitaev, Damian S. Steiger, Matthew B. Hastings, and Matthias Troyer. Quantum algorithm for spectral measurement with a lower gate count. *Physical Review Letters*, 121:010501, 2018.

[121] Brendon L. Higgins, Dominic W Berry, Stephen D. Bartlett, Howard M. Wiseman, and Geoff J. Pryde. Entanglement-free Heisenberg-limited phase estimation. *Nature*, 450:393–396, 2007.

[122] Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald De Wolf. Quantum lower bounds by polynomials. *Journal of the ACM (JACM)*, 48(4):778–797, 2001.

[123] Andris Ambainis, Kaspars Balodis, Jānis Iraids, Martins Kokainis, Krišjānis Prūsis, and Jevgēnijs Vihrovs. Quantum speedups for exponential-time dynamic programming algorithms. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1783–1793. SIAM, 2019.

[124] Gilles Brassard, Peter Hoyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Contemporary Mathematics*, 305:53–74, 2002.

[125] Theodore J Yoder, Guang Hao Low, and Isaac L Chuang. Fixed-point quantum search with an optimal number of queries. *Physical Review Letters*, 113(21):210501, 2014.

[126] Andrew M Childs and Nathan Wiebe. Hamiltonian simulation using linear combinations of unitary operations. *Quantum Information and Computation*, 12(11):901, 2012.

[127] Dominic W Berry, Andrew M Childs, and Robin Kothari. Hamiltonian simulation with nearly optimal dependence on all parameters. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 792–809. IEEE, 2015.

[128] Seth Lloyd. Universal Quantum Simulators. *Science*, 273(5):1073–1078, 1996.

[129] Scott Aaronson and Daniel Gottesman. Improved simulation of stabilizer circuits. *Physical Review A*, 70(5):052328, 2004.

[130] Richard P Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6-7):467–488, 1982.

[131] James D. Whitfield, Jacob Biamonte, and Alán Aspuru-Guzik. Simulation of electronic structure Hamiltonians using quantum computers. *Molecular Physics*, 109(5):735–750, 2011.

[132] James Daniel Whitfield, Peter John Love, and Alan Aspuru-Guzik. Computational complexity in electronic structure. *Physical Chemistry Chemical Physics*, 15(2):397–411, 2013.

[133] Dorit Aharonov and Amnon Ta-Shma. Adiabatic quantum state generation and statistical zero knowledge. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, pages 20–29. ACM, 2003.

[134] Andrew M Childs, Robin Kothari, and Rolando D Somma. Quantum algorithm for systems of linear equations with exponentially improved dependence on precision. *SIAM Journal on Computing*, 46(6):1920–1950, 2017.

[135] 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 Annual ACM Symposium on Theory of Computing*, STOC '14, pages 283–292, New York, NY, USA, 2014. ACM.

[136] Dominic W Berry, Richard Cleve, and Sevag Gharibian. Gate-efficient discrete simulations of continuous-time quantum query algorithms. *Quantum Information & Computation*, 14(1-2):1–30, 2014.

[137] Guang Hao Low and Isaac L Chuang. Hamiltonian simulation by qubitization. *Quantum*, 3:163, 2019.

[138] Dominic W. Berry, Graeme Ahokas, Richard Cleve, and Barry C. Sanders. Quantum Algorithms for Hamiltonian Simulation. In Louis Kauffman and Samuel J. Lomonaco, editors, *Mathematics of Quantum Computation and Quantum Technology*, Chapman & Hall/CRC Applied Mathematics & Nonlinear Science, pages 89–112. CRC Press, 2007.

[139] I M Georgescu, S Ashhab, and Franco Nori. Quantum simulation. *Reviews of Modern Physics*, 86(1):153–185, 2014.

[140] A. Aspuru-Guzik, A. Dutoi, P. J. Love, and M. Head-Gordon. Simulated quantum computation of molecular energies. *Science*, 309(5741):1704–1707, 2005.

[141] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum principal component analysis. *Nature Physics*, 10:631–633, 2014.

[142] Robert Beals, Stephen Brierley, Oliver Gray, Aram W Harrow, Samuel Kutin, Noah Linden, Dan Shepherd, and Mark Stather. Efficient distributed quantum computing. *Proceedings of the Royal Society A*, 469(2153), 2013.

[143] 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, 1998.

[144] Andrew M Childs and Robin Kothari. Limitations on the simulation of non-sparse Hamiltonians. *arXiv:0908.4398*, 2009.

[145] Jeongwan Haah, Matthew Hastings, Robin Kothari, and Guang Hao Low. Quantum algorithm for simulating real time evolution of lattice hamiltonians. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 350–360. IEEE, 2018.

[146] Masuo Suzuki. Fractal decomposition of exponential operators with applications to many-body theories and Monte Carlo simulations. *Physics Letters A*, 146(6):319–323, 1990.

[147] Andrew M Childs, Dmitri Maslov, Yunseong Nam, Neil J Ross, and Yuan Su. Toward the first quantum simulation with quantum speedup. *Proceedings of the National Academy of Sciences*, 115(38):9456–9461, 2018.

[148] Andrew M Childs, Aaron Ostrander, and Yuan Su. Faster quantum simulation by randomization. *Quantum*, 3:182, 2019.

[149] Earl Campbell. Random compiler for fast hamiltonian simulation. *Physical Review Letters*, 123(7):070503, 2019.

[150] Andrew M Childs. *Quantum information processing in continuous time*. PhD thesis, Massachusetts Institute of Technology, 2004.

[151] Andrew M Childs and Robin Kothari. Simulating sparse Hamiltonians with star decompositions. In *Conference on Quantum Computation, Communication, and Cryptography*, pages 94–103. Springer, 2010.

[152] Andrew M Childs, Edward Farhi, and Sam Gutmann. An example of the difference between quantum and classical random walks. *Quantum Information Processing*, 1(1):35–43, 2002.

[153] 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.

[154] Stacey Jeffery. *Frameworks for Quantum Algorithms*. PhD thesis, University of Waterloo, 2014.

[155] Andris Ambainis. Quantum walks and their algorithmic applications. *International Journal of Quantum Information*, 1(04):507–518, 2003.

[156] Harry Buhrman, Christoph Durr, Mark Heiligman, Peter Hoyer, Frédéric Magniez, Miklos Santha, and Ronald De Wolf. Quantum algorithms for element distinctness. In *Computational Complexity, 16th Annual IEEE Conference on, 2001.*, pages 131–137. IEEE, 2001.

[157] Chen-Fu Chiang, Daniel Nagaj, and Pawel Wocjan. Efficient circuits for quantum walks. *arXiv:0903.3465*, 2009.

[158] T Loke and JB Wang. Efficient quantum circuits for szegedy quantum walks. *Annals of Physics*, 382:64–84, 2017.

[159] Leonardo Novo and Dominic W Berry. Improved Hamiltonian simulation via a truncated Taylor series and corrections. *arXiv: 1611.10033*, 2016.

[160] Guang Hao Low, Theodore J Yoder, and Isaac L Chuang. Methodology of resonant equiangular composite quantum gates. *Physical Review X*, 6:041067, 2016.

[161] Nathan Wiebe, Dominic W Berry, Peter Høyer, and Barry C Sanders. Simulating quantum dynamics on a quantum computer. *Journal of Physics A: Mathematical and Theoretical*, 44(44):445308, 2011.

[162] David Poulin, Angie Qarry, Rolando Somma, and Frank Verstraete. Quantum simulation of time-dependent Hamiltonians and the convenient illusion of Hilbert space. *Physical Review Letters*, 106:170501, 2011.

[163] David Poulin, Angie Qarry, Rolando Somma, and Frank Verstraete. Quantum simulation of time-dependent Hamiltonians and the convenient illusion of hilbert space. *Physical Review Letters*, 106(17):170501, 2011.

[164] W Forrest Stinespring. Positive functions on C*-algebras. *Proceedings of the American Mathematical Society*, 6(2):211–216, 1955.

[165] Martin Kliesch, Thomas Barthel, Christian Gogolin, Michael Kastoryano, and Jens Eisert. Dissipative quantum Church-Turing theorem. *Physical Review Letters*, 107(12):120501, 2011.

[166] Andrew M Childs and Tongyang Li. Efficient simulation of sparse markovian quantum dynamics. *arXiv:1611.05543*, 2016.

[167] Richard Cleve and Chunhao Wang. Efficient quantum algorithms for simulating Lindblad evolution. *arXiv:1612.09512*, 2016.

[168] Richard Durstenfeld. Algorithm 235: random permutation. *Communications of the ACM*, 7(7):420, 1964.

[169] Donald Ervin Knuth. *The art of computer programming*, volume 3. Pearson Education, 1997.

[170] Daniel S Abrams and Seth Lloyd. Quantum algorithm providing exponential speed increase for finding eigenvalues and eigenvectors. *Physical Review Letters*, 83(2):5162–5165, 1999.

[171] Sorting network entry on Wikipedia. `https://en.wikipedia.org/wiki/Sorting_network`. Accessed: 2019-04-17.

[172] Sheng-Tzong Cheng and Chun-Yen Wang. Quantum switching and quantum merge sorting. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 53(2):316–325, 2006.

[173] Michael J. Bremner, Ashley Montanaro, and Dan J. Shepherd. Achieving quantum supremacy with sparse and noisy commuting quantum computations. *Quantum*, 1:8, April 2017.

[174] K E Batcher. Sorting networks and their applications. *Communications of the ACM*, 32:307–314, 1968.

[175] K J Liszka and K E Batcher. A generalized bitonic sorting network. *International Conference on Parallel Processing*, 1:105–108, 1993.

[176] M Ajtai, J Komlós, and E Szemerédi. An $O(N \log N)$ sorting network. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, pages 1–9, New York, NY, USA, 1983. ACM.

[177] M S Paterson. Improved sorting networks with $O(\log N)$ depth. *Algorithmica*, 5:75–92, 1990.

[178] Michael T Goodrich. Zig-zag sort: A simple deterministic data-oblivious sorting algorithm running in $O(N \log N)$ time. In *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 684–693, New York, NY, USA, 2014. ACM.

[179] Michael Codish, Lus Cruz-Filipe, Thorsten Ehlers, Mike Müller, and Peter Schneider-Kamp. Sorting networks: To the end and back again. *Journal of Computer and System Sciences*, 2016.

[180] Cody Jones. Low-overhead constructions for the fault-tolerant Toffoli gate. *Physical Review A*, 87(2):022328, 2013.

[181] Thomas G Draper, Samuel A Kutin, Eric M Rains, and Krysta M Svore. A logarithmic-depth quantum carry-lookahead adder. *arXiv:0406142*, 2004.

[182] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of the cipher block chaining message authentication code. *Journal of Computer and System Sciences*, 61(3):362 – 399, 2000.

[183] Daniel S Abrams and Seth Lloyd. Simulation of many-body Fermi systems on a universal quantum computer. *Physical Review Letters*, 79:2586–2589, 1997.

[184] Donny Cheung, Peter Høyer, and Nathan Wiebe. Improved error bounds for the adiabatic approximation. *Journal of Physics A: Mathematical and Theoretical*, 44(41):415302, 2011.

[185] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. Quantum computation by adiabatic evolution. *arXiv:0001106*, 2000.

[186] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. Quantum Computation by Adiabatic Evolution. *MIT-CTP-2936*, 2000.

[187] Donny Cheung, Peter Hoyer, and Nathan Wiebe. Improved error bounds for the adiabatic approximation. *Journal of Physics A: Mathematical and Theoretical*, 44(41):415302, 2011.

[188] Dominic W Berry, Andrew M Childs, Yuan Su, Xin Wang, and Nathan Wiebe. Time-dependent Hamiltonian simulation with L^1-norm scaling. *arXiv:1906.07115*, 2019.

[189] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Brian Kingsbury, et al. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine*, 29, 2012.

[190] George E Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary

speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1):30–42, 2012.

[191] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649. IEEE, 2013.

[192] Simon J D Prince. *Computer vision: models, learning, and inference*. Cambridge University Press, 2012.

[193] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European conference on computer vision*, pages 430–443. Springer, 2006.

[194] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[195] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484, 2016.

[196] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[197] Yutian Chen, Aja Huang, Ziyu Wang, Ioannis Antonoglou, Julian Schrittwieser, David Silver, and Nando de Freitas. Bayesian optimization in alphago. *arXiv:1812.06855*, 2018.

[198] Juan Carrasquilla and Roger G Melko. Machine learning phases of matter. *Nature Physics*, 13(5):431, 2017.

[199] Sandeep Mavadia, Virginia Frey, Jarrah Sastrawan, Stephen Dona, and Michael J Biercuk. Prediction and real-time compensation of

qubit decoherence via machine learning. *Nature Communications*, 8:14106, 2017.

[200] Raban Iten, Tony Metger, Henrik Wilming, Lídia Del Rio, and Renato Renner. Discovering physical concepts with neural networks. *arXiv:1807.10300*, 2018.

[201] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. Quantum support vector machine for big data classification. *Physical Review Letters*, 113(13):130503, 2014.

[202] Esma Aïmeur, Gilles Brassard, and Sébastien Gambs. Machine learning in a quantum world. *Advances in Artificial Intelligence*, pages 431–442, 2006.

[203] Nathan Wiebe, Ashish Kapoor, and Krysta M Svore. Quantum nearest-neighbor algorithms for machine learning. *Quantum Information and Computation*, 15, 2018.

[204] Kristen L Pudenz and Daniel A Lidar. Quantum adiabatic machine learning. *Quantum Information Processing*, 12(5):2027–2070, 2013.

[205] Hartmut Neven, Vasil S Denchev, Geordie Rose, and William G Macready. Training a binary classifier with the quantum adiabatic algorithm. *arXiv:0811.0416*, 2008.

[206] Patrick Rebentrost, Maria Schuld, Francesco Petruccione, and Seth Lloyd. Quantum gradient descent and Newton's method for constrained polynomial optimization. *arXiv:1612.01789*, 2016.

[207] Nathan Wiebe, Ashish Kapoor, and Krysta M Svore. Quantum perceptron models. *arXiv:1602.04799*, 2016.

[208] Iordanis Kerenidis and Anupam Prakash. Quantum recommendation systems. *arXiv:1603.08675*, 2016.

[209] Alex Monràs, Gael Sentís, and Peter Wittek. Inductive supervised quantum learning. *Physical Review Letters*, 118(19):190503, 2017.

[210] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. Prediction by linear regression on a quantum computer. *Physical Review A*, 94(2):022342, 2016.

[211] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum random access memory. *Physical Review Letters*, 100(16):160501, 2008.

[212] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Architectures for a quantum random access memory. *Physical Review A*, 78(5):052310, 2008.

[213] Srinivasan Arunachalam, Vlad Gheorghiu, Tomas Jochym-O'Connor, Michele Mosca, and Priyaa Varshinee Srinivasan. On the robustness of bucket brigade quantum RAM. *New Journal of Physics*, 17(12):123010, 2015.

[214] Christopher E Granade, Christopher Ferrie, Nathan Wiebe, and David G Cory. Robust online Hamiltonian learning. *New Journal of Physics*, 14(10):103013, 2012.

[215] Jianwei Wang, Stefano Paesani, Raffaele Santagati, Sebastian Knauer, Antonio A Gentile, Nathan Wiebe, Maurangelo Petruzzella, Jeremy L O'Brien, John G Rarity, Anthony Laing, et al. Experimental quantum Hamiltonian learning. *Nature Physics*, 13(6):551, 2017.

[216] Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002.

[217] Geoffrey Hinton. Boltzmann machines. *Encyclopedia of Machine Learning and Data Mining*, pages 1–7, 2014.

[218] Miguel Á Carreira-Perpiñán and Geoffrey E Hinton. On contrastive divergence learning. In *AISTATS*, 2005.

[219] Geoffrey E Hinton and Terrence J Sejnowski. Analyzing cooperative computation. In *Proceeding of the 5th Annual Congress of the Cognitive Science Society*, 1983.

[220] Radford M Neal. Connectionist learning of belief networks. *Artificial Intelligence*, 56(1):71–113, 1992.

[221] David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. A learning algorithm for boltzmann machines. *Cognitive Science*, 9(1):147–169, 1985.

[222] Ruslan Salakhutdinov and Geoffrey Hinton. Deep Boltzmann machines. In *Artificial Intelligence and Statistics*, pages 448–455, 2009.

[223] Nitish Srivastava and Ruslan R Salakhutdinov. Multimodal learning with deep boltzmann machines. In *Advances in neural information processing systems*, pages 2222–2230, 2012.

[224] Geoffrey Hinton. A practical guide to training restricted Boltzmann machines. *Momentum*, 9(1):926, 2010.

[225] Mohammad H Amin, Evgeny Andriyash, Jason Rolfe, Bohdan Kulchytskyy, and Roger Melko. Quantum Boltzmann machine. *Physical Review X*, 8(2):021050, 2018.

[226] David Poulin and Pawel Wocjan. Sampling from the thermal quantum Gibbs state and evaluating partition functions with a quantum computer. *Physical Review Letters*, 103(22):220502, 2009.

[227] Man-Hong Yung and Alán Aspuru-Guzik. A quantum–quantum Metropolis algorithm. *Proceedings of the National Academy of Sciences*, 109(3):754–759, 2012.

[228] Nathan Wiebe, Ashish Kapoor, Christopher Granade, and Krysta M Svore. Quantum inspired training for Boltzmann machines. *arXiv:1507.02642*, 2015.

[229] Anirban Narayan Chowdhury and Rolando D Somma. Quantum algorithms for Gibbs sampling and hitting-time estimation. *arXiv:1603.02940*, 2016.

[230] Eric R Anschuetz and Yudong Cao. Realizing quantum Boltzmann machines through eigenstate thermalization. *arXiv:1903.01359*, 2019.

[231] J M Deutsch. Quantum statistical mechanics in a closed system. *Physical Review A*, 43:2046–2049, 1991.

[232] Mark Srednicki. Chaos and quantum thermalization. *Physical Review E*, 50:888–901, 1994.

[233] Misha Denil and Nando De Freitas. Toward the implementation of a quantum rbm. In *NIPS Deep Learning and Unsupervised Feature Learning Workshop*, volume 5, 2011.

[234] Guillaume Verdon, Michael Broughton, and Jacob Biamonte. A quantum algorithm to train neural networks using low-depth circuits. *arXiv:1712.05304*, 2017.

[235] Jingxiang Wu and Timothy H Hsieh. Variational thermal quantum simulation via thermofield double states. *arXiv:1811.11756*, 2018.

[236] James D Whitfield, Jacob Biamonte, and Alán Aspuru-Guzik. Simulation of electronic structure Hamiltonians using quantum computers. *Molecular Physics*, 2011.

[237] Nathan Wiebe and Leonard Wossnig. Generative training of quantum Boltzmann machines with hidden units. *arXiv:1905.09902*, 2019.

[238] Sandhya Samarasinghe. *Neural networks for applied sciences and engineering: from fundamentals to complex pattern recognition*. Auerbach publications, 2016.

[239] Mohamad H Hassoun et al. *Fundamentals of artificial neural networks*. MIT press, 1995.

[240] Geoffrey Hinton. Neural networks and machine learning. `https://www.cs.toronto.edu/~hinton/coursera_lectures.html`.

A

A Brief Introduction To Machine Learning

Let us briefly introduce a few classical ML concepts that are essential for our work. ML algorithms can be divided into supervised, unsupervised, and reinforcement learning. In supervised learning, an algorithm has access to a database of examples with labels. Given enough training, the algorithm is then expected to assign correct labels to new data.

Our algorithm is an unsupervised one, which means that it gets only a set of data with no labels. A canonical example of an unsupervised algorithm is clustering. The goal of the clustering algorithm is to group "similar data" together and divide the data set into two or more clusters. For example, a clustering algorithm can take a set of hand-written digits and divide them into ten groups, one for each digit (without labels). Intuitively, the algorithm learns a model that accurately represents the data. Reinforcement learning is a cross between supervised and unsupervised learning and is not relevant to this thesis.

A1 Generative Modelling

Machine learning algorithms can also be divided into discriminative and generative. In a discriminative model, the algorithm learns how to classify the training samples (represented as binary vectors). After such an algorithm is trained, it is supposed to assign each vector to a group of "similar" vectors. In contrast, a generative algorithm learns a model that generates the training data. In this setting, the training data are seen as samples from an underlying distribution. A properly trained generative algorithm can be used for sampling from this distribution and to generate examples similar to the training data as in Fig. A.1.

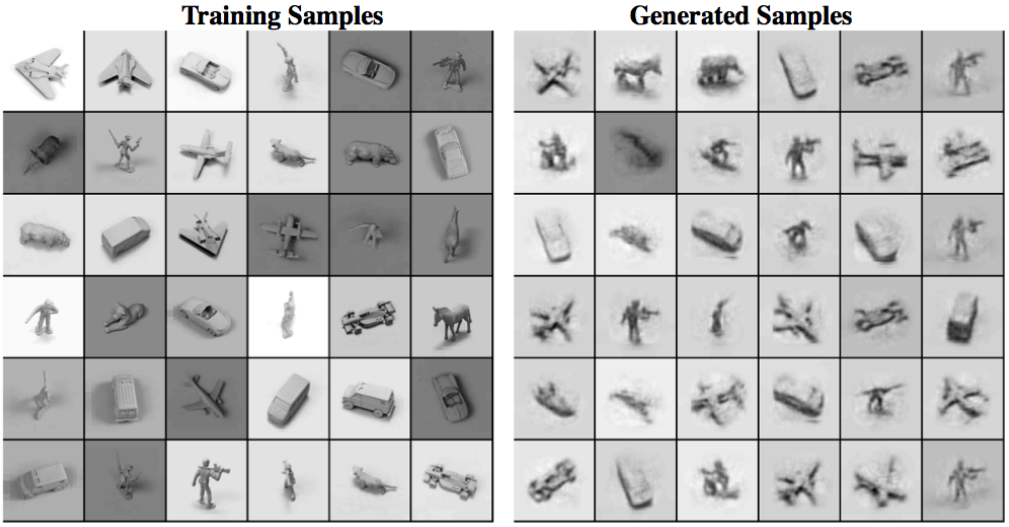


Figure A.1: An example of generative modelling from [222]. The examples on the left were used to train a deep Boltzmann machine (a restricted Boltzmann machine with multiple layers). Then, the algorithm generated the images on the right-hand side.

If the model is not rich enough, it can underfit the data, i.e. not capture important properties of the distribution. Such a model can neither fit the training data nor generate new data.

In the other extreme, the model can also overfit data – capture any sampling errors. Such a model appears to perform very well on training data – indeed, we can fit any data perfectly given enough parameters, but

it will not generalize. We demonstrate the problem of underfitting and overfitting on an example in Fig. A.2.

A2 Artificial Neural Networks

Artificial neural networks (and their variants, deep nets) emerged as one of the main ML tools. We will now briefly review them and introduce the terminology used in this chapter. Artificial neural networks are an ML framework inspired by biology. In biological neural networks, neurons connected by synapses are responsible for the activity in animals' and humans' brains.

An artificial neural network (from now on referred to only as a neural network or ANN) is defined by its graph. The vertices of the graph (also called *units*) represent neurons and store information as $+1$ or -1 states[1]. Some of the units also serve as input or output of the algorithms, and these are called *visible units*. Remaining units are referred to as *hidden units* and can provide additional complexity to the network. The edges of the graph represent connections between units and encode the relationship between the bits of the information. In addition, one can define functions on this graph, such as the activation function or the energy function, that specifies how the information propagates through the network.

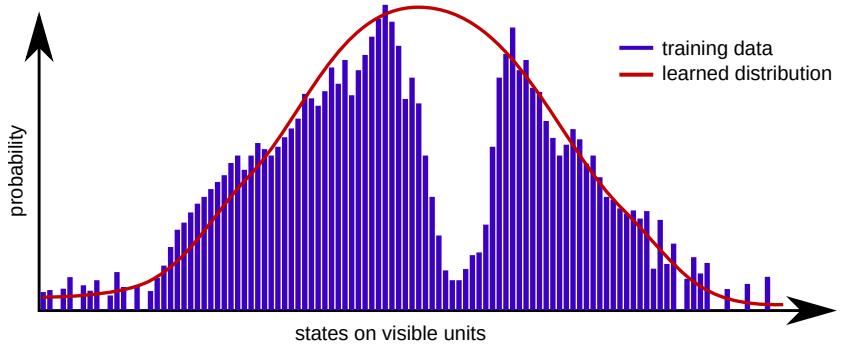
It is possible to distinguish two types of ANNs based on the properties of the underlying graph structure. Oriented graphs correspond to feed-forward networks, where the information flows only in one direction, and the graphs do not include cycles. Their counterparts are recurrent neural networks that can include loops and undirected graphs.

From the physics perspective, neural networks with undirected graphs are particularly interesting because they allow defining "energy" as a bilinear function

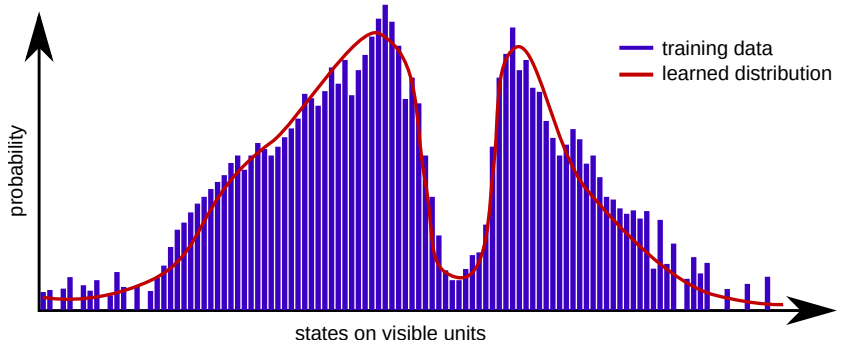
$$\mathrm{E}(\mathbf{x}) = \mathbf{x}^{\mathrm{T}} W \mathbf{x}, \tag{A.1}$$

where W is the adjacency matrix of the graph and the binary vector $\mathbf{x}$ is defined on the units. A Boltzmann machine is an example of a neural

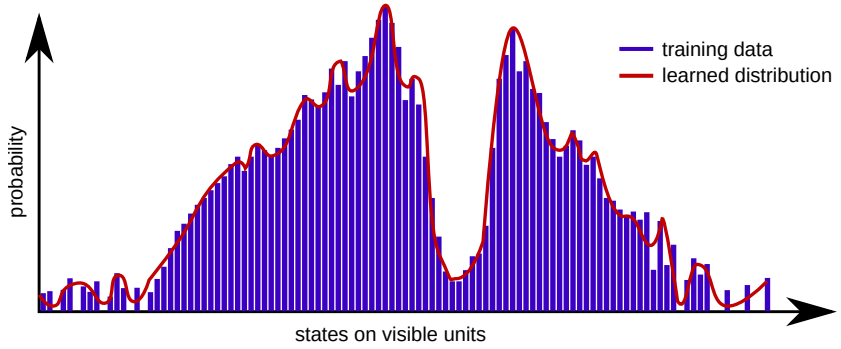
[1] Machine learning literature often prefers to use 0 and 1, but the notations are equivalent.



(a) Underfitting.

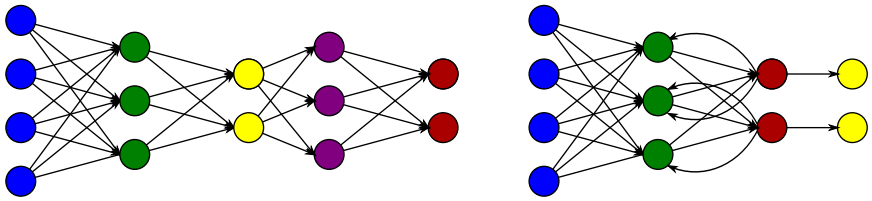


(b) Good fit.



(c) Overfitting.

Figure A.2: Examples of underfitting, good fit and overfitting. The histogram represents the training data and the red fit is the distribution that supposedly generated it. This example is purely illustratory; in practice, it is unlikely to have any training vector represented more than once in the training set. In addition, it is in general not possible to directly determine what constitutes a good fit – the quality of a learned model is determined from its performance on test data.



(a) Feed-forward ANN. (b) Recurrent ANN.

Figure A.3: Examples of ANN architectures.

network that can be trained using the energy function. For an introduction to neural networks see [238, 239] or this excellent lecture series by Geoffrey Hinton [240].