On Algorithms, Separability and Cellular Automata in Quantum Computing

by

Donny Cheung

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Combinatorics and Optimization

Waterloo, Ontario, Canada, 2007

Abstract

In Part I of this thesis, we present a new model of quantum cellular automata (QCA) based on local unitary operations. We will describe a set of desirable properties for any QCA model, and show that all of these properties are satisfied by the new model, while previous models of QCA do not. We will also show that the computation model based on Local Unitary QCA is equivalent to the Quantum Circuit model of computation, and give a number of applications of this new model of QCA. We also present a physical model of classical CA, on which the Local Unitary QCA model is based, and Coloured QCA, which is an alternative to the Local Unitary QCA model that can be used as the basis for implementing QCA in actual physical systems.

In Part II, we explore the quantum separability problem, where we are given a density matrix for a state over two quantum systems, and we are to determine whether the state is separable with respect to these systems. We also look at the converse problem of finding an entanglement witness, which is an observable operator which can give a verification that a particular quantum state is indeed entangled. Although the combined problem is known to be NP-hard in general, it reduces to a convex optimization problem, and by exploiting specific properties of the set of separable states, we introduce a classical algorithm for solving this problem based on an Interior Point Algorithm introduced by Atkinson and Vaidya in 1995.

In Part III, we explore the use of a low-depth AQFT (approximate quantum Fourier transform) in quantum phase estimation. It has been shown previously that the logarithmic-depth AQFT is as effective as the full QFT for the purposes of phase estimation. However, with sub-logarithmic depth, the phase estimation algorithm no longer works directly. In this case, results of the phase estimation algorithm need classical post-processing in order to retrieve the desired phase information. A generic technique such as the method of maximum likelihood can be used in order to recover the original phase.

Unfortunately, working with the likelihood function analytically is intractable for the phase estimation algorithm. We develop some computational techniques to handle likelihood functions that occur in phase estimation algorithms. These computational techniques may potentially aid in the analysis of certain likelihood functions.

Contents

List of Tables

List of Figures

Preface

This thesis is presented in three separate parts, representing three research projects which I undertook over the course of my doctoral studies. A summary of the main research results of each project is given at the beginning of each of the three separate parts of this thesis, in Sections 1.1, 7.1, and 10.1.

Part I, entitled *Local Unitary Quantum Cellular Automata*, represents joint work with Carlos Prez-Delgado. In this research project, we develop a new framework for exploring and understanding quantum cellular automata both as a model for quantum computation, and as a tool for modelling homogeneous physical phenomena. We also construct efficient simulations between quantum circuits and quantum cellular automata in order to demonstrate their practical value. Both of these constructions are new.

Part II, entitled *Algorithmic Search for Entanglement Witnesses*, represents joint work with Lawrence Ioannou and Ben Travaglione at the University of Cambridge. Some results from this work has been published in [ITCE04], while some further details are described in [ITC06], available on the arXiv.org e-print server. In this research, we explore the problem of detecting separability and entanglement in terms of a convex optimization problem. We present a new algorithm for finding an entanglement witness given the description of an entangled state, which in turn also solves the problem of distinguishing separable and entangled states.

Part III, entitled *Classical Post-processing for Approximate QFT*, includes joint work with Jamie Batuwantudawe. This research is an extension of research contained in my MMath thesis [Che03, Che04]. We present a new strategy for approaching phase estimation problems which attempts to use the maximum likelihood method to extract as much information as possible from measurement results by using classical post-processing. This strategy becomes useful when considering using low-depth circuits to solve phase estimation problems.

Part I

Local Unitary Quantum Cellular Automata

Chapter 1

Introduction

Cellular automata are computational machines based on the idea of applying operations in parallel to an infinite lattice of cells, which are each in one of finitely many states. The model of computation based on cellular automata has been a topic of interest to computer scientists and physicists alike for a number of decades, due not only to its simplicity, but also the ease with which cellular automata can be used to model a wide variety of physical phenomena, including simple particle motion and fluid dynamics.

In Part I of this thesis, we will explore various notions of quantum cellular automata (QCA), with the eventual goal of introducing a new model of quantum cellular automata based on applying local unitary operations on a lattice of cells containing quantum information. Such a model has a strong relationship to both physical systems which could be modelled using QCA and also possible physical implementations of QCA, since it represents what local and homogeneous physics can achieve in a discrete self-contained system. In Chapter 6, we will consider expanding this model to open quantum systems with dynamics which are not necessarily unitary. However, we will begin by motivating and developing the Local Unitary QCA model.

1.1 Summary of Results

We present a new model of quantum cellular automata (QCA), based on applying local unitary operators to a lattice of cells. We show that this new model has a full range of natural and desirable properties, which are not all properties of previously developed models. We detail how QCA expressed in this model can be efficiently simulated with quantum circuits, and how universal quantum computation can be performed within the model. We also discuss how physical phenomena can be modelled using the Local Unitary QCA model. Finally, we present previously developed models of quantum cellular automata, and describe their relationship to the proposed Local Unitary QCA model.

We also present a model of QCA based on the Local Unitary model called the Coloured QCA model, which has the added advantage of closely modelling some specific potential physical implementations of QCA.

In developing this model of QCA, we also present a classical model of Physical Reversible CA, and we present a theory of reversibility within this model which is quantized to give us the Local Unitary QCA model.

Lastly, we present two methods of constructing valid QCA: The Active Configuration Method described in Section 5.2, and the Coloured QCA Method. To our knowledge, these are the first generic constructive techniques for building QCA.

Chapter 2 will provide an introduction to classical notions of cellular automata and some of the basic concepts and notations used in the study of classical CA. Chapter 3 will introduce the notion of quantum cellular automata, and survey a number of different models for QCA which have been developed.

Chapter 2

Classical Cellular Automata

In this chapter, we will give definitions for the classical Cellular Automaton model, as well as a number of concepts related to classical CA which will be of use as we develop quantum versions of CA.

2.1 Cellular Automata

In the classical model of cellular automata, we begin with a finite set of states, Σ and an infinite lattice of *cells*, each of which is in one of the states in Σ. We have discrete time steps, and at each time step t, the state of the lattice evolves according to a given rule. The rule gives the state of each cell at time $t+1$ as a function of the states of the cells of its *neighbourhood*, which is simply a finite set of cells corresponding to a particular cell.

Definition 2.1.1. *A* Cellular Automaton *(CA) is a 4-tuple* $(L, \Sigma, \mathcal{N}, f)$ *consisting of a d-dimensional lattice of cells indexed by integers* $L = \mathbb{Z}^d$ *, a finite set* Σ *of cell states, a finite neighbourhood scheme* $\mathcal{N} \subseteq \mathbb{Z}^d$*, and a local transition function* $f : L \times \Sigma^{\mathcal{N}} \times \mathbb{Z} \rightarrow \Sigma$*.*

The transition function f simply takes a lattice cell position, $x \in L$, the states of the neighbours of x, which are the cells indexed by the set $x + \mathcal{N}$, and the current time step, $t \in \mathbb{Z}$ to determine the state of cell x at time $t+1$. Usually, we would like to restrict our attention to transition functions with certain properties. In particular, we define a CA to be *time-homogeneous* if the transition function f does not depend on the time step t, and we define a CA to be *space-homogeneous* if f does not depend on the cell location $x \in L$. When a CA is both space- and time-homogeneous, we will simply call it a *homogeneous* CA. A diagram indicating the action of a local transition function is given in Figure 2.1.

When a CA is homogeneous, the state of any cell at any time step depends only on the states of its neighbour cells in the previous time step. The traditional

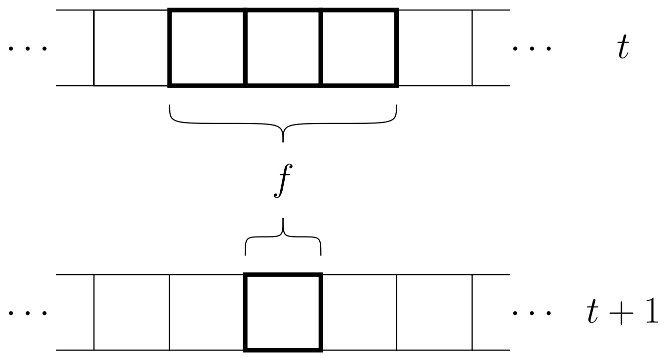


Figure 2.1: Local transition function of a cellular automaton during one time step

definition of CA restricts the model to homogeneous CA, so unless otherwise noted, we will assume that a given CA is homogeneous. In this case, we can give the transition function as $f : \Sigma^{\mathcal{N}} \to \Sigma$.

We may also view the transition function as one which acts on the entire lattice, rather than on individual cells. In this view, we denote the state of the entire CA as a *configuration* $C \in Q^L$ which gives the state of each individual cell. This gives us a *global* transition function which is simply a function which maps $F : Q^L \to Q^L$, or $F : Q^L \times \mathbb{Z} \to Q^L$ if the CA is not time-homogeneous.

2.2 Reversible Cellular Automata

Since our goal is to eventually quantize a model of cellular automata in which the transition functions are unitary operations, it would be helpful to build this model from a reversible classical CA model. In general, however, cellular automata are not reversible. A simple counterexample would be a CA which simply overwrites the entire lattice with one particular symbol.

A homogeneous CA is reversible if for any configuration $C \in \Sigma^L$, there exists a unique predecessor configuration C' such that $C = F(C')$, where F is the global transition operator associated with the CA. It is known that any Turing machine can be efficiently simulated using a reversible CA [Tof77], so the computational power of the CA model is not reduced by this restriction.

Note that the notion of reversibility defined above does not address the local transition function. In general, the reverse of a reversible CA can be expressed as a CA itself. However, the local transition function of the inverse may need a larger neighbourhood as its domain.

While not all CA are reversible, it is possible to encode an non-reversible CA over a d-dimensional lattice within a reversible CA over a $(d + 1)$-dimensional lattice. In this construction, the extra dimension is simply used to store the entire configuration history of the non-reversible CA.

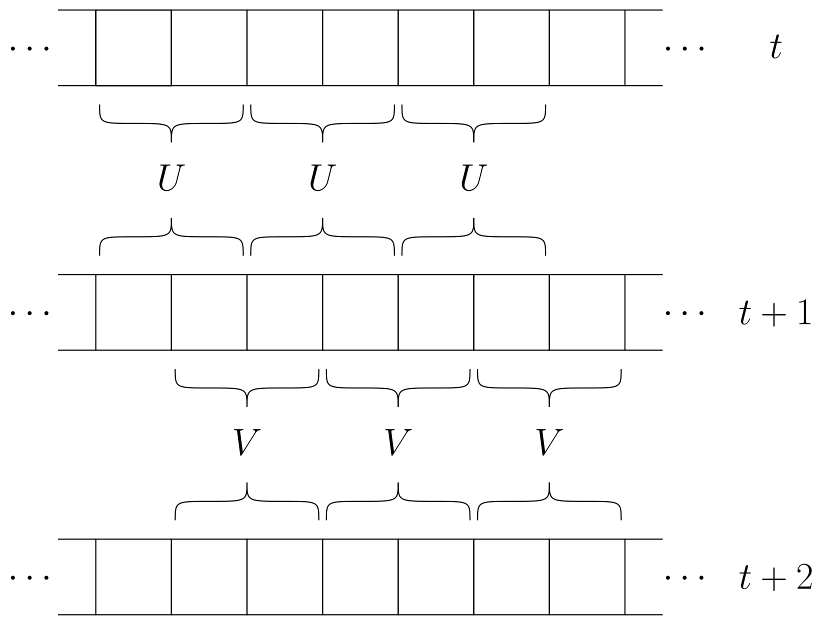


Figure 2.2: A Partitioned Cellular Automaton

2.3 Partitioned Cellular Automata

One method that is used to construct reversible cellular automata is that of *partitioning.* In a partitioned CA, the transition function is composed of local, reversible operations on individual units of a partitioning of the lattice.

While partitioned CA are neither time-homogeneous nor space-homogeneous in general, they are periodic in both space and time, and thus we set both a time period, $T \in \mathbb{Z}$, with $T \geq 1$ and a space period, given as a d-dimensional sublattice, S of $L = \mathbb{Z}^d$. The sublattice S can be defined using a set $\{v_k : k = 1, \ldots, d\}$ of d linearly independent vectors from $L = \mathbb{Z}^d$ as:

$$S = \left\{ \sum_{k=1}^{d} a_k v_k : a_k \in \mathbb{Z} \right\}.$$

Definition 2.3.1. *For a given fixed sublattice $S \subseteq \mathbb{Z}^d$, we define a* block $B \subseteq \mathbb{Z}^d$ *as a finite subset of $\mathbb{Z}^d$ such that $(B + s_1) \cap (B + s_2) = \emptyset$ for any $s_1, s_2 \in S$ with $s_1 \neq s_2$, and such that*

$$\bigcup_{s \in S} (B + s) = \mathbb{Z}^d.$$

The main idea of the partitioned CA is that at different time steps, we act on a different block partitioning of the lattice. The interaction between different partitions is required if the behaviour of the partitioned CA is to be non-trivial. We are now ready to formally define the partitioned CA.

Definition 2.3.2. *A* Partitioned CA *is a 6-tuple* $(L, S, T, \Sigma, \mathbf{B}, \mathcal{F})$ *consisting of a d-dimensional lattice of cells indexed by integers* $L = \mathbb{Z}^d$*, a d-dimensional sublattice* $S \subseteq L$*, a time period* $T \geq 1$*, a finite set* Σ *of cell states, a block scheme* $\mathbf{B}$*, which is a sequence* $\{B_0, B_1, \ldots, B_{T-1}\}$ *consisting of* T *blocks relative to the sublattice* S*, and a local transition function scheme* $\mathcal{F}$*, which is a set* $\{f_0, f_1, \ldots, f_{T-1}\}$ *of reversible local transition functions which map* $f_t : \Sigma^{B_t} \to \Sigma^{B_t}$*.*

At time step $t + kT$ for $0 \leq t < T$ and $k \in \mathbb{Z}$, we perform f_t on every block $B_t + s$, where $s \in S$. In order to find the reverse of a partitioned CA, we simply give the reverse block scheme, $\mathbf{B} = \{B_{T-1}, \ldots, B_1, B_0\}$, and the reverse function scheme, $\mathcal{F} = \{f_{T-1}^{-1}, \ldots, f_1^{-1}, f_0^{-1}\}$. A diagram of a partitioned CA is given in Figure 2.2. In particular, the *Margolus Partitioning Scheme*, described in [TM87], is a partitioning scheme using the sublattice $S = 2\mathbb{Z}^d$, period $T = 2$, and the block scheme, $\mathbf{B} = \{B_0, B_1\}$, where

$$B_0 = \{(x_1, x_2, \ldots, x_d) \in L : 0 \leq x_j \leq 1, 1 \leq j \leq d\},$$

which is simply a cube of size 2^d with corners at cells $\mathbf{0} = (0, 0, \ldots, 0)$ and $\mathbf{1} = (1, 1, \ldots, 1)$, and

$$B_1 = B_0 + \mathbf{1},$$

which is simply a translation of the cube B_0.

Although the partitioned CA is neither space- nor time-homogeneous, it can be converted into a reversible homogeneous CA on the lattice S (which is isomorphic to $\mathbb{Z}^d$), with cell states Σ^B, where the new local transition function simulates T time steps of the partitioned CA in one time step. Additionally, the inverse of this reversible CA is also easy to construct using the reverse of the partitioned CA. However, not all reversible CA can be constructed using a partitioned CA.

2.4 Symmetric Cellular Automata

Lastly, we will also describe the notion of a *symmetric* local transition function for a CA. Essentially, a transition function is symmetric if the update rule depends on the contents of the neighbouring cells, but does not distinguish between which of the neighbouring cells actually contains any particular state. In this sense, the transition function depends on the total number of surrounding neighbour cells in each state from Σ.

Definition 2.4.1. *A local transition function* $f : \Sigma^{\mathcal{N}} \to \Sigma$ *is* symmetric *if for any configuration* $C \in \Sigma^{\mathcal{N}}$*, and any* $x, y \in \mathcal{N}$*, we have* $f(C) = f(SWAP_{x,y}(C))$*, where* $SWAP_{x,y}$ *is the operator which swaps cells* x *and* y*.*

A symmetric CA is simply a CA with a symmetric local transition function. Recall that in our definition of a neighbourhood $\mathcal{N}_x$, we generally include the central

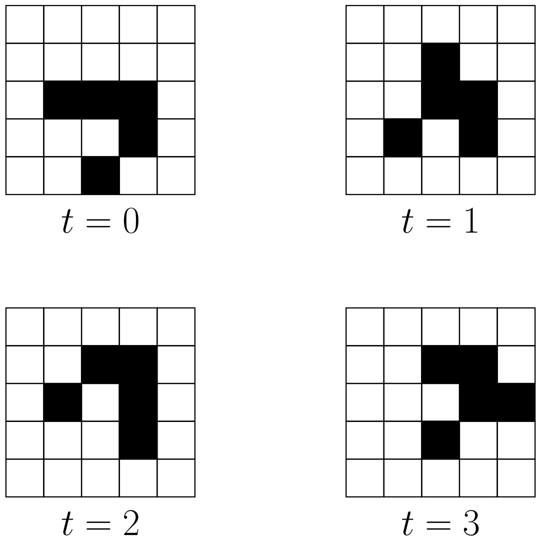


Figure 2.3: Four time steps of Conway's Game of Life cellular automaton

cell $x \in \mathcal{N}_x$. We may also define a variant on the symmetric CA where the central cell can be distinguished from the other cells in its neighbourhood. This is called an *outer-symmetric* CA.

Definition 2.4.2. *A local transition function $f : \Sigma^{\mathcal{N}} \to \Sigma$ is* outer-symmetric *if for any configuration $C \in \Sigma^{\mathcal{N}}$, and any $x, y \in \mathcal{N} \setminus \{0\}$, we have $f(C) = f(SWAP_{x,y}(C))$, where $SWAP_{x,y}$ is the operator which swaps cells x and y.*

The classic example of such a CA is *Conway's Game of Life*, introduced in 1970, which features a two-symbol alphabet over 2-dimensional lattice. The state of each cell at a time step depends only on how many of its 8 surrounding neighbours are in one particular state at the previous time step. Specifically, we have two states labelled "live" and "dead", and at each time step, a "live" cell stays "live" if it has exactly 2 or 3 neighbours, and becomes "dead" otherwise. A "dead" cell becomes "live" if it has exactly 3 neighbours, and stays "dead" otherwise. Note that this local transition rule is not reversible.

Chapter 3

Quantum Cellular Automata

In this chapter, we will explore a number of desirable properties that we may want from a quantum model for cellular automata. We will also present a number of models of quantum cellular automata that have been developed. We will pay particular attention to the models developed by Watrous [Wat95], and Schumacher and Werner [SW04].

3.1 Goals of a Quantum CA Model

In our discussion of models of quantum cellular automata (QCA), we should first define a number of properties that we should reasonably expect from such a model. The first and most natural property a reversible QCA model should have is that it should be a natural extension of the classical model of reversible CA, so that each reversible CA is also a valid QCA. Specifically, the model should include a lattice of cells which contain quantum, rather than classical, information. As the cells of a classical CA contain a state taken from a finite set Σ, the cells of a quantum model should likewise contain quantum states derived from Σ, in the form of the Hilbert space of linear superpositions of these states, $\mathcal{H}^{\Sigma}$.

Because quantum mechanics gives a model of physical reality, it is reasonable that a QCA model also be a model for realistic physical systems. Although it is not possible to directly convert a local transition function from a classical CA into a unitary operation in a quantum system since the domain and range are different, it should still be possible to "implement" the global transition function using local unitary operations. By this, we mean that it should be possible to build the global transition function F as a circuit of depth n, so that $F = f_n f_{n-1} \dots f_2 f_1$, where each operation f_k is a product of unitary operations on disjoint local neighbourhoods of the lattice. Since the lattice contains an infinite number of cells, each of these operations f_k could be a product of infinitely many disjoint local unitary operations. However, the depth of such a circuit must remain finite. Also, note that this model allows us to simulate the action of the global transition function on any finite subset of the lattice using a quantum circuit by restricting our attention to unitary operations which affect the given subset.

Of course, it should be possible to perform universal quantum computation with QCA, just as the classical CA allows for universal computation. It should be possible to efficiently simulate any quantum circuit or quantum Turing machine, using only polynomial overhead for such a simulation. To this end, a model of QCA should be shown to be equivalent to either the quantum circuit or the quantum Turing machine models of computation.

Finally, a model of QCA should be suitable for modelling space- and time-homogeneous phenomena. Ideally, it would be possible to model phenomena with discrete structure exactly. However, it should also be possible to approximate continuous phenomena within this model.

3.2 Watrous QCA

The first attempt to define a quantized version of cellular automata was made by Watrous [Wat95], whose ideas were further explored by van Dam [vD96]. The model considers a one-dimensional lattice of cells and a finite set of basis states Σ for each individual cell, and features a transition function which maps a neighbourhood of cells to a single quantum state instantaneously and simultaneously. Watrous also introduces a model of partitioned QCA in which each cell contains a triplet of quantum states, and a permutation is applied to each cell neighbourhood before the transition function is applied. The partitioned model will be discussed in the Section 3.3.

Definition 3.2.1. *A* Watrous QCA*, acting on a one-dimensional lattice indexed by $\mathbb{Z}$, consists of a 3-tuple $(\Sigma, \mathcal{N}, f)$ consisting of a finite set Σ of cell states, a finite neighbourhood scheme $\mathcal{N}$, and a local transition function $f : \Sigma^{\mathcal{N}} \to \mathcal{H}^{\Sigma}$.*

This model can be viewed as a direct quantization of the classical cellular automata model, where the set of possible configurations of the CA is extended to include all linear superpositions of the classical cell configurations, and the local transition function now maps the cell configurations of a given neighbourhood to a quantum state. In the case that a neighbourhood is in a linear superposition of configurations, f simply acts linearly. Also note that in this model, at each time step, each cell is updated with its new value simultaneously, as in the classical model.

Unfortunately, this definition allows for non-physical behaviour. It is possible to define transition functions which do not represent unitary evolution of the cell tape, either by producing superpositions of configurations which do not have norm 1, or by not being injective, giving configurations which are not reachable by evolution from some other configuration. In order to help resolve this problem, Watrous restricts the set of permissible local transition functions by introducing the notion

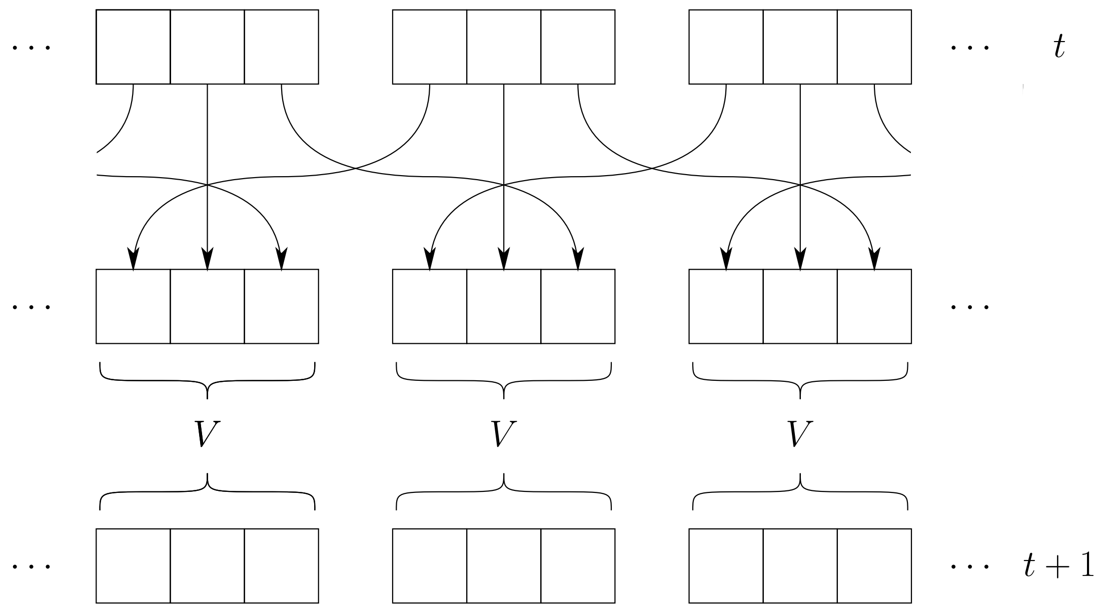


Figure 3.1: Local transition function of a Watrous Partitioned QCA

of *well-formed* QCA. A local transition function is well-formed simply if it maps any configuration to a properly normalized linear superposition of configurations. Because the set of configurations is infinite, this condition is usually expressed in terms of the ℓ_2 norm of the complex amplitudes associated with each configuration.

In order to describe QCA which undergo unitary evolution, Watrous also introduces the idea of a *quiescent* state, which is a distinguished element $\epsilon \in \Sigma$ which has the property that the local transition function f maps $f : \epsilon^{\mathcal{N}} \mapsto \epsilon^{\mathcal{N}}$. We can then define a quiescent QCA as a QCA with a distinguished quiescent state which acts only on $finite$ configurations, which are configurations consisting of finitely many non-quiescent states. It can be shown that a quiescent QCA which is well-formed and injective represents unitary evolution on the lattice. Drr, LThanh and Santha give an efficient algorithm for deciding whether a given local transition function yields a well-formed quiescent QCA [DS96, DLS97].

3.3 Watrous Partitioned QCA

In order to construct examples of valid QCA in this model, Watrous also introduces a model of partitioned QCA in which each cell consists of three quantum states, so that the set of finite states can be subdivided as $\Sigma = \Sigma_l \times \Sigma_c \times \Sigma_r$. Given a configuration in which each cell, indexed by $k \in \mathbb{Z}$, is in the state $(q_k^{(l)}, q_k^{(c)}, q_k^{(r)})$, the transition function of the QCA in one time step first consists of a permutation which brings the state of cell k to $(q_{k+1}^{(l)}, q_k^{(c)}, q_{k-1}^{(r)})$ for each $k \in \mathbb{Z}$, and then performs a local unitary operation V_k on each cell. A diagram showing the action of the local transition function of the Watrous partitioned QCA during one time step is given in Figure 3.1.

Watrous shows that this model of partitioned QCA can be used to simulate a universal quantum Turing machine with polynomial overhead, thereby showing that this model is capable of universal quantum computation.

3.4 Schumacher-Werner QCA

Schumacher and Werner [SW04] take a different approach in the definition of their model of QCA, working in the Heisenberg picture rather than the Schrödinger picture. They introduce a comprehensive model of QCA in which we consider only the evolution of the algebra of observables on the lattice, rather than states of the cell lattice itself. By extending local observables of the cell lattice into a closed observable algebra, the Schumacher-Werner model has a number of useful algebraic properties. In this model, the transition function is simply a homomorphism of the observable algebra which satisfies a locality condition.

In order to avoid problematic issues dealing with observables over an infinite lattices, Schumacher and Werner make use of the *quasi-local* algebra. In order to construct this algebra, we first start with the set of all observables on finite subsets $S \subseteq L$ of the lattice, denoted $\mathcal{A}(S)$, and extend them appropriately into observables of the entire lattice by taking a tensor product with the identity operator over the rest of the lattice. The completion of this set forms the quasi-local algebra.

In this setting, the global transition operator of a QCA is simply defined as a homomorphism $T : \mathcal{A}(L) \to \mathcal{A}(L)$ over the quasi-local algebra which satisfies two specific properties. First, a locality condition must be satisfied: $T(\mathcal{A}(S)) \subseteq \mathcal{A}(S + \mathcal{N})$ for all finite $S \subseteq L$, so that any observable on the region S at a time step $t+1$ will correspond exactly to some observable on the neighbourhood region $S + \mathcal{N}$ in the previous time step t. Secondly, T must commute with lattice translation operators, so that the QCA is space-homogeneous. Now, the QCA can be defined in terms of the lattice L, the neighbourhood scheme $\mathcal{N}$, the single-cell observable algebra, $\mathcal{A}_0$, which takes the place of the alphabet, and the global transition operator T.

The local transition operator of a QCA is simply a homomorphism $T_0 : \mathcal{A}_0 \to \mathcal{A}(\mathcal{N})$ from the observable algebra of a single distinguished cell, such as $\mathbf{0} \in L$, to the observable algebra of the neighbourhood of that cell. Schumacher and Werner show that a local homomorphism T_0 will correspond uniquely to a global transition operator T if and only if for each $x \in L$, the algebras $T_0(\mathcal{A}_0)$ and $\tau_x(T_0(\mathcal{A}_0))$ commute element-wise. Here, τ_x is a lattice translation by x. The global transition operator T given by T_0 is defined by

$$T(\mathcal{A}(S)) = \prod_{x \in S} T_x(\mathcal{A}_x).$$

3.5 Generalized Margolus Partitioned QCA

Schumacher and Werner also introduce a partitioned QCA in [SW04], called the *Generalized Margolus Partitioned QCA*, as a generalization of the Margolus partitioning CA scheme, described in Section 2.3, in which the cell lattice itself is partitioned. Schumacher and Werner present this model as a method of producing valid reversible QCA in their model. In order to describe this model, we will proceed in the same manner as in the definition of the classical partitioned CA.

We start with the d-dimensional lattice $L = \mathbb{Z}^d$, and we fix the sublattice $S = 2\mathbb{Z}^d$ as the set of cells of L with all even co-ordinates, as in the Margolus partitioning scheme. We also fix the time period as $T = 2$. Recall that the block scheme, $\mathbf{B}$ is given as $\{B_0, B_1\}$, where

$$B_0 = \{(x_1, x_2, \ldots, x_d) \in L : 0 \leq x_j \leq 1, 1 \leq j \leq d\},$$

which is simply a cube of size 2^d with corners at cells $\mathbf{0} = (0, 0, \ldots, 0)$ and $\mathbf{1} = (1, 1, \ldots, 1)$, and

$$B_1 = B_0 + \mathbf{1},$$

which is simply a translation of the cube B_0.

Now, as in the regular Schumacher-Werner QCA model, we proceed in the Heisenberg picture. For any block $B_0 + s$, $s \in S$, we have 2^d intersecting blocks from the partition $B_1 + S$. For each block $B_1 + s'$ which intersects with $B_0 + s$, there is a vector $v \in \mathbb{Z}^d$ representing the translation taking $B_0 + s$ to $B_1 + s'$, so that $B_1 + s' = B_0 + s + v$. Indeed, these 2^d intersecting blocks may be indexed by the vectors v, which are simply all vectors of $\mathbb{Z}^d$ whose entries are each ± 1. Hence, we will set $B_v^{(s)} = B_0 + s + v$.

For each block $B_v^{(\mathbf{0})}$, we will fix an observable algebra $\mathcal{B}_v^{(\mathbf{0})}$ as a subalgebra of the observable algebra $\mathcal{A}(B_v^{(\mathbf{0})})$ for the entire block. Then, for each block $B_v^{(s)}$, the observable algebra $\mathcal{B}_v^{(s)}$ is simply the appropriate translation of $\mathcal{B}_v^{(\mathbf{0})}$. Note in particular that $\mathcal{A}(B_{\mathbf{1}}^{(s)})$, which is the observable algebra for the block $B_1 + s = B_{\mathbf{1}}^{(s)}$, contains each of the observable algebras $\mathcal{B}_v^{(s+\mathbf{1}-v)}$. In order for an assignment of subalgebras to be considered valid, these subalgebras $\mathcal{B}_v^{(s+\mathbf{1}-v)}$ must commute and span $\mathcal{A}(B_{\mathbf{1}}^{(s)})$. This occurs if and only if the product of the dimensions of these algebras is $|\Sigma|^{2^d}$.

The transition function then consists first of an isomorphism

$$T_0^{(s)} : \mathcal{A}(B_{\mathbf{0}}^{(s)}) \to \prod_v \mathcal{B}_v^{(s)},$$

followed by the isomorphism

$$T_1^{(s)} : \prod_v \mathcal{B}_v^{(s+\mathbf{1}-v)} \to \mathcal{A}(B_{\mathbf{1}}^{(s)}).$$

Note that since T_0 and T_1 are isomorphisms between observable algebras of equal dimension, with an appropriate choice of basis, they can be represented by unitary operators U_0 and U_1 which map vectors from a complex vector space to another complex vector space of equal dimension. However, they do not represent local unitary evolution, since these complex vector spaces are used to describe two different quantum systems.

3.6 Quantum Lattice Gases

Meyer [Mey96] explored the idea of using QCA as a model for simulating quantum lattice gases. As classical CA are used to model classical physical systems, it is natural to develop QCA models which are capable of modelling quantum physical systems. In order to simulate lattice gases, Meyer uses a model of QCA in which each lattice cell is represented by a computational basis state in a Hilbert space, and the set of states which a given cell can take is replaced with a complex number representing the amplitude of the basis state corresponding to that cell.

This particular model is quite restrictive, as the cells of the lattice do not independently carry quantum information, but are components of one quantum system marking the location of one particle instead. However, this model is sufficient for the task of modelling the evolution of a single particle in a lattice gas, and can be generalized to model certain discrete-time quantum walks.

3.7 Pulse-Driven Spin Chains

Lloyd [Llo93] proposed a model of physical computation based on a chain consisting of a repeating sequence of a fixed number of distinguishable states. In this model, pulses are programmed which are capable of distinguishing the states and performing nearest-neighbour unitary operations. Lloyd also showed that this model is sufficient for implementing universal quantum computation. Although this model does not provide a theoretical descriptive model of all local homogeneous processes that we may wish to classify as QCA, the physical scheme potentially provides a natural platform for implementing QCA.

Chapter 4

Local Unitary QCA

Unfortunately, none of the previous proposed models of QCA satisfy all of the desired properties given in Section 3.1. In this chapter, we will explore why this is the case, and then explore a new model for QCA which does satisfy these properties.

4.1 The Shift-Right QCA

In this section, we will explore one particular QCA which highlights the importance of the use of local unitary operators in defining a model for QCA in general. The *Shift-Right* QCA is defined over a 1-dimensional lattice of qudits. For each lattice cell $x \in L$, we associate with it a quantum state $|\psi_x\rangle \in \mathcal{H}^{\Sigma}$. Although in general, the configuration of a QCA may not be separable with respect to each cell, the configuration can still be described in terms of a linear superposition of these separable configurations. Thus, it suffices to consider such configurations to describe the global transition function of this QCA.

At each time step we wish to have all of the quantum information from each cell in the lattice transferred one cell to the right. In other words, after the first update, each cell x should now store the state $|\psi_{x-1}\rangle$. After k steps each cell x should contain the state $|\psi_{x-k}\rangle$. This QCA is clearly reversible, with the inverse QCA being the analogous *Shift-Left* QCA. We can also see that this QCA is an extension of an analogous classically reversible Shift-Right CA. We will show that, in fact, such a global transition function cannot be implemented by *any* local unitary process, including ones which are not space-homogeneous.

Theorem 4.1.1. *The Shift-Right QCA cannot be implemented by any local unitary process.*

Proof. We decompose the global transition function for the Shift-Right QCA in the local unitary model as described in Section 3.1. Let F be this global transition function, which is decomposed as $F = f_n f_{n-1} \dots f_2 f_1$, such that each operator f_k

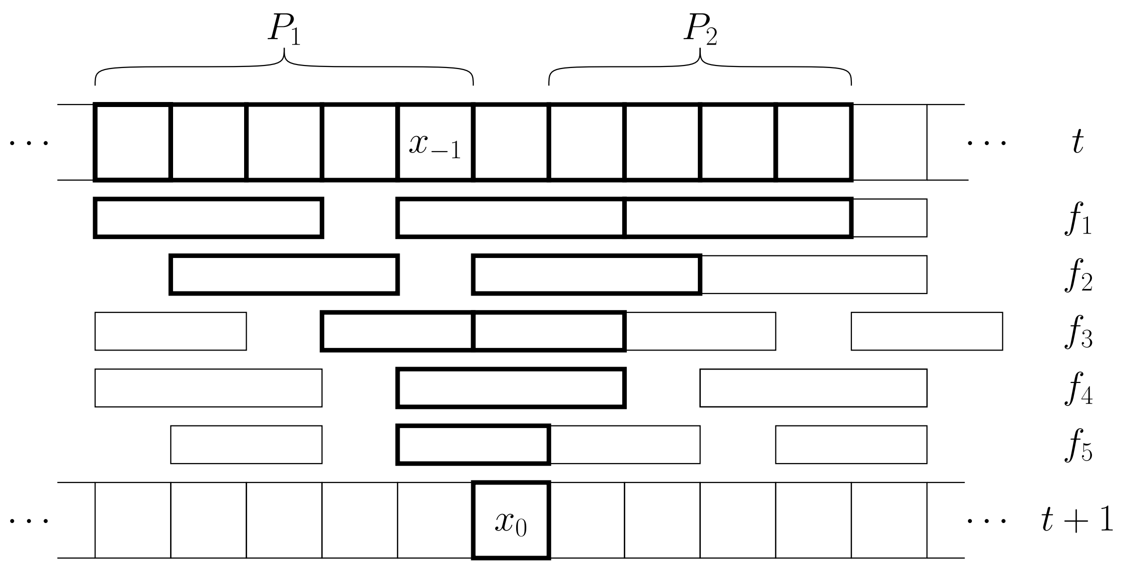


Figure 4.1: An example local unitary implementation of the Shift-Right QCA

is the product of potentially infinitely many local unitary operators on disjoint local neighbourhoods of the 1-dimensional lattice. This gives us the most general description possible of a depth-n quantum circuit implementation of this linear QCA using only local unitary operators. Now, consider an individual cell x_0. By considering the dependencies of the individual local unitary operators which make up the transition function f, it is possible to find a range of cells, $P = \{x : a \leq x \leq b\}$ for some $a, b \in \mathbb{Z}$ such that $x_0 \in P$, and the value of the quantum state at cell x_0 after the application of the transition function depends only on the cells of P. Clearly, this value must be the value of the quantum state in the cell to the immediate left of x_0 before the transition function is applied.

We may also find a minimal set of local unitary operators from F such that the new value of the quantum state at cell x_0 is computed without violating any of the dependency relationships between the local operators. We now divide the transition function F into two functions, $F = hg$, where g is the product of these local operators in the correct order. Then, h simply applies the remainder of the local unitary operators, as appropriate. Note that since g necessarily contains all local unitary operators from f which operate on the cell x_0, the operation h cannot does not include any such operators. Now, let P_1 be the set of cells to the left of x_0, $P_1 = \{x : x < x_0\}$, and let P_2 be the cells to the right of x_0, $P_2 = \{x : x > x_0\}$. Figure 4.1 gives an example diagram of such a construction. Suppose that in an implementation of the global transition function F, that g is performed first, followed by h. We consider the state of the cell lattice immediately after g is performed.

Since h does not operate on the cell x_0, it cannot include any local operation which acts on both cells from P_1 and cells from P_2. Also, since the operation g did not operate on any cells outside of the set P, the cells in P_2 must contain all of the information necessary after g is applied in order to reconstruct all of the quantum data from the cells x_0 and P_2. The notion of $|P_2|$ cells containing the quantum information from $1 + |P_2|$ cells is impossible. We must conclude that constructing the Shift-Right QCA using only unitary operators which act on local neighbourhoods bounded by a fixed size is not possible. □

If we wish to consider QCA on finite one-dimensional structures, such as a ring, in which the cell x_0 does not necessarily divide the structure into two disconnected regions, we may still adapt this theorem to show that it is still impossible to construct circuits with fixed depth and fixed local neighbourhood size which perform the Shift-Right QCA for arbitrarily large rings.

Finally, note that as an extension of this theorem, it is also not possible to implement Shift-Right on a classical reversible CA, using only local reversible operations applied to the lattice. However, to make this notion rigorous, we need to retool the definition of the classical reversible CA to include the considerations of a physical implementation. To do this, we will develop the Physical Reversible CA model.

4.2 Implementing Classical CA

Because we are looking to build a model of QCA based on local unitary operations, we must examine why a QCA as simple and natural as the Shift-Right QCA cannot be implemented in such a model. To resolve this problem, observe that the local unitary model upon which we would like to build our model of QCA is a model of physical implementation. If we look now at the classical model of reversible CA, we note that the model itself does not explicitly address implementation concerns. In particular, a local transition function needs to be applied in parallel to each cell of the lattice in such a way that the information contained within the lattice is updated instantaneously, so that information in one cell is not overwritten before it needs to be read for the local transition function for a neighbouring cell.

In order to be able to implement a CA in a classical setting, for each local transition function that we wish to apply to some local neighbourhood, we must ensure that we copy any cell state information which is needed for any other local neighbourhood to which we have not yet applied the local transition function. Since we wish to be able to implement the global transition function as a finite-depth classical circuit consisting of these local transition functions, we cannot construct an implementation of a CA in general with only a finite amount of additional "memory" contained within a computational machine.

The simplest way to achieve this physical model for classical CA is to use a second lattice L' of the same dimension as the first lattice. The local transition function is applied to the original lattice L, as before, but the output of the function is written to the second lattice L'. After the local transition function has been applied to every local neighbourhood, the original lattice still contains the original configuration, while the second lattice contains the new configuration. At this point, we simply copy the information from L' into L by applying an appropriate single-cell update operation to each cell. While we describe the "workspace" of the physical reversible CA as a second lattice, it is more convenient to represent it as one single lattice, in which each cell contains two symbols, one representing the state of the cell, and the other acting as memory for computing the transition function.

If we aim to build a quantum CA model based on local unitary operations from this physical model for reversible CA, we should ensure that the operations which are being applied are reversible. Specifically, both the local transition function and the cell update function should be reversible. However, reversibility raises the question of the initial state of the memory lattice, since this memory cannot be reset in a reversible model. This issue would be resolved if every configuration $C \in \Sigma^L$ of a given reversible CA could be matched with a dual memory configuration $C' \in \Sigma^{L'}$ such that the reversible local transition function for the implementation of the reversible CA would preserve this dual condition. In order to implement the reversible CA in this model, we would simply need to ensure that the memory is appropriately initialized for the given configuration. We show such a construction below.

We say that a given reversible CA $\mathcal{A} = (L, \Sigma, \mathcal{N}, f)$ with global transition function $F : \Sigma^L \to \Sigma^L$ is *implemented* by a physical reversible CA $\mathcal{A}'$ with global transition function $F' : \Sigma^L \times \Sigma^{L'} \to \Sigma^L \times \Sigma^{L'}$ if there exists a bijection $D : \Sigma^L \to \Sigma^{L'}$ such that $F'(C, D(C)) = (F(C), D(F(C))$.

Now, we present a simple construction for implementing a reversible CA $\mathcal{A}$ in this model. We will let our duality relation simply be the reverse of the global transition function: $D = F^{-1}$. We will also need an addition relation on the alphabet, $+ : \Sigma \to \Sigma$, which can be defined by using some bijection from Σ to $\mathbb{Z}_{|\Sigma|}$. Essentially, the global transition function F' will clear the memory lattice by computing the reverse configuration $F'(C)$ and subtracting it from the memory space, then computing the next configuration $F(C)$. The cell update operation simply swaps the two configurations, and the dual property is preserved. Specifically, we will need the local transition function g from the reversible CA which implements the reverse global transition function of $\mathcal{A}$. The local transition function of $\mathcal{A}'$ then simply applies g to the local neighbourhood of a given cell, subtracting the answer from the corresponding memory cell. Then, it applies f to the local neighbourhood, adding the answer to the memory cell. Finally, after this has been done for each cell in the lattice L, the cell update operation swaps each cell with its corresponding memory cell. Note that the local neighbourhood scheme for the local transition function of $\mathcal{A}'$ must contain the neighbourhood schemes of both $\mathcal{A}$ and its reverse.

Note that one very important property of the local transition function for the physical reversible CA is that while it reads all of the data from a local neighbourhood, it alters only the data of a single memory cell. In this way, the local

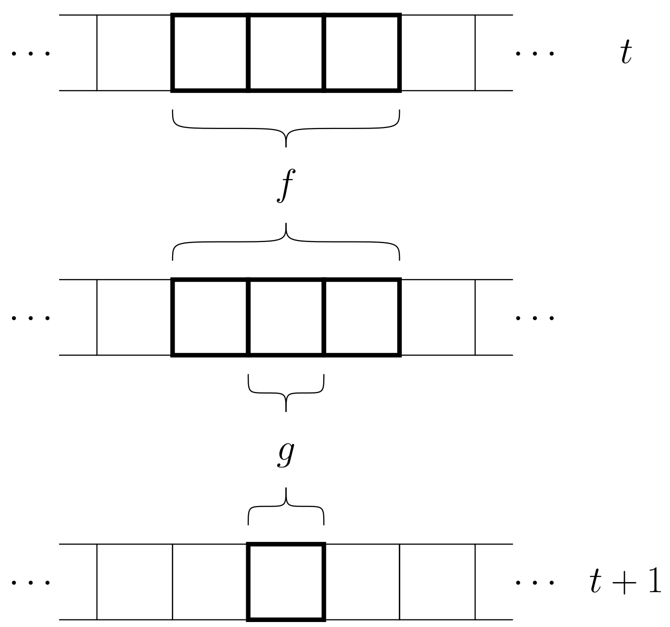


Figure 4.2: Local transition function of a Physical Reversible CA

transition functions may be applied to the lattice in any order. We may generalize this notion slightly, in order to help remove the need to distinguish the data of a cell from its additional memory space, and to allow us to define the local transition function as a function mapping $f : \Sigma^{\mathcal{N}} \to \Sigma^{\mathcal{N}}$ rather than $f : \Sigma^{\mathcal{N}} \to \Sigma$. As long as the local transition functions may be applied in any order, they do not need to conform to any further requirements. To this end, we introduce the property of *translate commutativity.*

Definition 4.2.1. *A lattice function $F : \Sigma^L \to \Sigma^L$ is* translate commutative *if for any lattice translation τ, $[F, \tau F \tau^{-1}] = 0$. A local transition function $f : \Sigma^{\mathcal{N}} \to \Sigma^{\mathcal{N}}$ is translate commutative if the corresponding lattice function is translate commutative.*

We can now consider the data and extra memory of a cell to be part of a combined state space, and give a formal definition for the classical Physical Reversible CA model that we have been developing in this section:

Definition 4.2.2. *A* Physical Reversible Cellular Automaton *(PRCA) is a 5-tuple $(L, \Sigma, \mathcal{N}, f, g)$ consisting of a d-dimensional lattice of cells indexed by integers, $L = \mathbb{Z}^d$, a finite set Σ of cell states, a finite neighbourhood scheme $\mathcal{N} \subseteq \mathbb{Z}^d$, a translate commutative reversible local transition function $f : \Sigma^{\mathcal{N}} \to \Sigma^{\mathcal{N}}$, and a reversible cell update function $g : \Sigma \to \Sigma$.*

A diagram representing the local transition function and the cell update function being applied to a Physical Reversible CA is given in Figure 4.2.

With the Physical Reversible CA, we can resolve the problem of the classical Shift-Right CA. While it is still not possible to shift all of the data within a cell to the right, we can implement a Shift-Right CA by extending the alphabet to include cell memory. Note that the data in the cell memory will not be shifted right. In fact, by necessity, it is shifted *left*.

4.3 Quantizing the CA Model

We now use the Physical Reversible CA to develop a unitary quantum model of cellular automata. In this model, we will again begin with a d-dimensional lattice $L = \mathbb{Z}^d$ of cells, each of which now contains a qudit $|x\rangle \in \mathcal{H}^\Sigma$ from a Hilbert space over a finite set of basis states Σ. We will also use the notation $\mathcal{H}(S)$ to indicate the Hilbert space of quantum states over the cells in a subset of the lattice $S \subseteq L$. In other words, $\mathcal{H}(S) = \left(\mathcal{H}^\Sigma\right)^{\otimes S}$.

Now, we need an analogous definition of translate commutativity for unitary operators:

Definition 4.3.1. *A unitary operator $U : \mathcal{H}(\mathcal{N}) \to \mathcal{H}(\mathcal{N})$ over a local neighbourhood is* translate commutative *if for any lattice translation τ, $[U, TUT^{-1}] = 0$, where T is the quantum operator corresponding to performing the lattice translation τ on a lattice of qudits.*

For each cell $x \in L$, we will use the notation U_x to indicate the unitary operator which performs U over the local neighbourhood $\mathcal{N}_x$ centred at cell x. In this notation, U_0 is translate commutative if $[U_x, U_y] = 0$ for all $x, y \in L$.

Now we can define the local transition operator of a QCA simply as a translate commutative operator $U_0 : \mathcal{H}(\mathcal{N}) \to \mathcal{H}(\mathcal{N})$. The cell update operator is simply a single-qudit operator $V_0 : \mathcal{H}^\Sigma \to \mathcal{H}^\Sigma$. We finally give a formal definition for the Local Unitary QCA:

Definition 4.3.2. *A* Local Unitary Quantum Cellular Automaton *(LUQCA) is a 5-tuple $(L, \Sigma, \mathcal{N}, U_0, V_0)$ consisting of a d-dimensional lattice of cells indexed by integers, $L = \mathbb{Z}^d$, a finite set Σ of orthogonal basis states, a finite neighbourhood scheme $\mathcal{N} \subseteq \mathbb{Z}^d$, a translate commutative local transition function $U_0 : \mathcal{H}(\mathcal{N}) \to \mathcal{H}(\mathcal{N})$, and a local update function $V_0 : \mathcal{H}^\Sigma \to \mathcal{H}^\Sigma$.*

It should be noted that, as with the classical model of cellular automata, the infinite nature of the lattice raises issues of constructibility and computability. For the purposes of finite simulation, there are two simple methods to address these issues. For simulating a closed cyclic system, such as a closed ring of qudits, we may restrict the set of allowable configurations to those which are periodic with respect to a fixed sublattice. To simulate a finite system, we may consider the addition of a *quiescent* state $|q\rangle \in \Sigma$, which has the property that for the local transition function, we have

$$U_0\left(|q\rangle^{\otimes \mathcal{N}}\right) = |q\rangle^{\otimes \mathcal{N}},$$

and for the local update function, we have

$$V_0 \left|q\right\rangle = \left|q\right\rangle .$$

Note that the quiescent state has the property that for any cell $x \in L$ at time step t, if each cell y such that $x \in \mathcal{N}(y)$ is in the quiescent state $\left|q\right\rangle$, then x will be in the quiescent state at time step $t+1$. In other words, if each cell which can influence the state of cell x at time step $t+1$ is quiescent, then x will be quiescent. This is compatible with the classical definition of quiescence.

We now restrict the set of allowable basis configurations to those in which all but a finite number of qudits are in state $\left|q\right\rangle$, and the set of allowable configurations to superpositions of such basis configurations, with the restriction that the ℓ_2-norm of the amplitudes is 1. Note that the availability of a quiescent state assures that this is possible. However, in general, such a construction may be made with any countable set of basis configurations, so long as the set of allowable configurations is closed under the global transition function and the global update function.

Chapter 5

Algorithms for Local Unitary QCA

Using the Local Unitary QCA model presented in Chapter 4, we can now proceed to show that it indeed has all of the desirable properties listed in Section 3.1.

5.1 Simulating QCA with Quantum Circuits

In showing equivalence between the Local Unitary QCA model and other models of quantum computation, we must first define what it means for a QCA to perform a computation. Specifically, if we wish to compute a function, we should be able to encode a finite input state onto a lattice of cells for some QCA, then perform a finite number of time steps after which the desired answer will be in some finite predetermined region of the lattice. Therefore, for any computation that we may wish to perform using a QCA, we need to be able to simulate the action of the QCA on a sufficiently large region to correctly determine the final state of the desired pre-determined region. The scope of this simulation is therefore finite. However, depending on the computation, it may be arbitrarily large. If the final output of a QCA is to be located in a finite region $S \subseteq L$ at time step T, then we must know the state of the QCA in the region $P(S)$ at time step $T-1$, where $P(S)$ is the union of all local neighbourhoods intersecting S. Thus, every cell whose value is used in computing the state of the region S is in $P(S)$. Continuing backwards, we see that it is sufficient to consider the region $P^T(S)$ at time step 0 in order to perform the desired computation.

In order to construct a quantum circuit which simulates the region S of a Local Unitary QCA for T time steps, we start by assigning one qudit to each cell of the region $P^T(S)$. Since the local transition function U_0 is translate commutative, they may be applied to the local neighbourhoods of $P^T(S)$ in any order. In particular, by fixing a (potentially incomplete) tiling of the lattice L using disjoint local neighbourhoods, and using translations of this tiling, we can ensure that there exists a

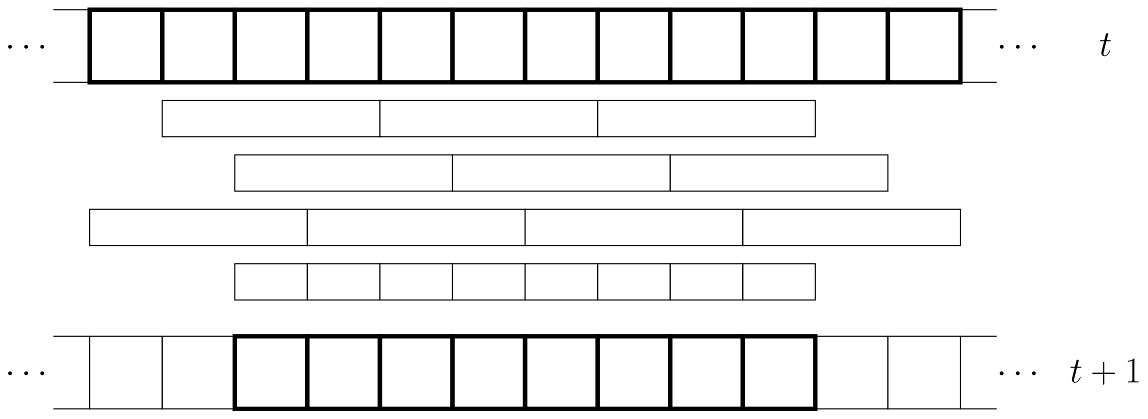


Figure 5.1: Simulation of a finite region of a QCA

fixed circuit depth for the simulation of the transition function $U_L = \prod_{x \in L} U_x$ on any finite region. We finish the simulation of a single time step by performing the single-qudit operation V_x on each qudit. We repeat this procedure for T time steps. A circuit implementation of one time step is illustrated in Figure 5.1. If necessary, we can convert this circuit into a quantum circuit over qubits which applies operators from a fixed universal set of gates. To do this, we encode each qudit as $\lceil \log |\Sigma| \rceil$ qubits, and use appropriate implementations of U_0 and V_0 using operators from the fixed universal gate set. It may be necessary to use approximate implementations of U_0 and V_0. However, given a desired precision for the final state of the region S of the QCA, we may construct such implementations with polynomial overhead.

Although the quantum circuit used to simulate a given finite region $S \subseteq L$ depends on the region in question, this construction still provides us with a family of circuits, indexed by the finite subsets $S \subseteq L$ and the number of time steps being simulated. This gives us a simulation of a finite subregion of a QCA using a binary circuit with $O(\log |\Sigma|)$ overhead in the number of qubits, and poly$(1/\epsilon)$ circuit depth for simulating each time step, for a given error parameter ϵ. Conveniently, we may also consider an "infinite" circuit for simulating the QCA over the entire lattice, where for any finite region S, the corresponding quantum circuit can be constructed simply as the appropriate subset of operations and qubits from this infinite circuit.

5.2 Simulating Quantum Circuits with QCA

We would now like to construct a QCA which can simulate a given quantum circuit encoded as an initial state. Specifically, we will consider quantum circuits operating on qubits, where the universal set of gates consists of a finite number of single-qubit gates and the controlled NOT (CNOT) two-qubit gate operating only on adjacent qubits.

First, we will describe a generic technique for constructing translate commutative operators which can be of use in the construction of many QCA in general. We begin by dividing cell data into *registers*, which are independent pieces of quantum information within the cell. For an n-register cell, we will have separate alphabets $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$ for each register, while the set of all possible states for the complete cell will be $\Sigma = \Sigma_1 \times \Sigma_2 \times \cdots \times \Sigma_n$.

For now, it suffices to consider a three-register cell, $\Sigma = \Sigma_1 \times \Sigma_2 \times \Sigma_3$, where Σ_1 will be a *data* register, Σ_2 will be a *control* register, and Σ_3 will be a *clock* register. For practical purposes, these registers may be subdivided further into more registers, but this three-register cell is sufficient to describe our construction.

The alphabet of the clock register will have two symbols: $\Sigma_3 = \{0, 1\}$. The intention is that all of the control registers in a lattice will be in the same state at any particular moment. If this is not true, we must still ensure that the QCA fails properly, by ensuring that the local transition function remains translate commutative even with bad clock data. Note that to ensure this, it suffices to consider the Σ_3 basis states of the clock registers.

For the control register Σ_2, we will define a QCA $Q^{(2)} = (L, \Sigma_2, \mathcal{N}, U_0^{(2)}, V_0^{(2)})$ which operates only on the control registers. The idea here is to use the clock register to interleave a set of actions on the data registers with one transition of the control register. To achieve this, we define a set of *active configurations* of the control registers Σ_2 to be a set of basis configurations $\mathcal{N}^{\Sigma_2}$ over the control registers with the additional restriction that it is not possible to have intersecting local neighbourhoods $\mathcal{N}_x$ and $\mathcal{N}_y$ be in a basis configuration $(\mathcal{N}_x \cup \mathcal{N}_y)^{\Sigma_2}$ in such a way that the configurations $\mathcal{N}_x^{\Sigma_2}$ and $\mathcal{N}_y^{\Sigma_2}$ are both active. A simple example would be the set $\Sigma_2 = \{0, 1\}$ over a one-dimensional lattice, with the neighbourhood scheme $\mathcal{N} = \{0, 1\}$ consisting of neighbouring pairs of cells. The set of configurations $\{|10\rangle\}$ is active, since it is not possible for two intersecting neighbouring pairs of cells to each be in the state $|10\rangle$. Now, given a set of active configurations, any operation on the registers $\Sigma_1 \times \Sigma_2$ which does not alter the state of the control register Σ_2, and acts non-trivially on Σ_1 only if the state of Σ_2 is active must be a translate commutative operation. We never attempt to apply non-trivial operations to intersecting local neighbourhoods.

We may also consider active configurations for the clock register. Specifically, the set of configurations $\mathcal{N}^{\Sigma_3}$, consisting of the all-0 configuration $|0\rangle^{\otimes \mathcal{N}}$ and the all-1 configuration $|1\rangle^{\otimes \mathcal{N}}$, forms a set of active configurations. Now, we can define a translate commutative local transition function U_0 for the whole QCA.

Consider a neighbourhood $\mathcal{N}_x$. If the clock configuration of the neighbourhood is $|0\rangle^{\otimes \mathcal{N}}$, then we perform a non-trivial operation on the data register only if the control register is in an active configuration. Otherwise, if the clock configuration is $|1\rangle^{\otimes \mathcal{N}}$, we perform the local transition function $U_0^{(2)}$ from the QCA $Q^{(2)}$ corresponding to the control register. Finally, the cell update register V_0 performs the cell update function $V_0^{(2)}$ if the clock is in the state $|1\rangle$, and then performs a *NOT* operation on the clock register.

If the QCA is correctly configured so that the clock registers of the lattice are

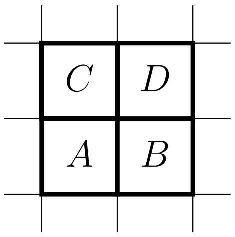


Figure 5.2: Local neighbourhood of a QCA simulating a quantum circuit

either all in the state 0 or all in the state 1, then we see that this QCA alternates between performing operations on the data register, and updating the control register. However, if the QCA is not correctly configured, the local transition function remains translate commutative, even if the computation becomes meaningless.

Now, we will apply this *Active Configuration Method* to the problem of simulating quantum circuits. We will use a 2-dimensional lattice, where one dimension will represent the qubit wires of the circuit, while the second dimension represents individual time steps of the simulation, proceeding from left to right.

The data register $\Sigma_1 = \{0, 1\}$ will simply contain contain qubit data corresponding to a particular wire at a particular time step. The control register Σ_2 will be subdivided into two registers, $\Sigma_2 = \Sigma_A \times \Sigma_B$. The first control register $\Sigma_A = \{0, 1\}$ will indicate whether the current cell is being actively simulated during the current time-step. The second control register state set Σ_B will consist of the finite universal set of single-qubit gates which we are using, including the identity I, along with four other special states, $CNOT_0^{(A)}$, $CNOT_1^{(A)}$, $CNOT_0^{(B)}$, and $CNOT_1^{(B)}$, with the subscripts 0 and 1 indicating that the current qubit is respectively the control or the target of a controlled NOT operation. Finally, the clock register $\Sigma_3 = \{0, 1\}$ is defined as before.

In describing the initial state corresponding to a quantum circuit, we will use the co-ordinates (x, t), where x indicates a specific qubit wire, indexed from 1 to n, and t indicates the current time-step, beginning at 0. We may think of the qubits of the circuit as being passed from cell to cell at each time step, and having an operation performed as indicated by the control register. To accomplish this, we simply encode the initial state of the circuit in the cells $(1, 0)$ to $(n, 0)$. If a single-qubit operation (including the identity operation) is to be performed on cell j at time step t, we encode this in the control register of cell (j, t). If a controlled NOT is to be performed on neighbouring qubits j and $j + 1$, we encode that using the special states $CNOT_0^{(A)}$ and $CNOT_1^{(A)}$ if qubit j is controlling qubit $j + 1$, and $CNOT_1^{(B)}$ and $CNOT_0^{(B)}$ if qubit j is being controlled by qubit $j + 1$.

Now we need to describe the operation on the data register. We will use neighbourhoods of size 2×2, and will label the cells of a given neighbourhood as $A = (x, t)$, $B = (x, t + 1)$, $C = (x + 1, t)$ and $D = (x + 1, t + 1)$ (see Figure 5.2). We operate only if the cells A, B, C, and D have their first control registers in the states 1, 0, 1 and 0, respectively. If the second control register of cell A

indicates a single-qubit operation, we perform the operation on the data register of cell A, and then swap the data registers of cells A and B. If the second control registers of cells A and C indicate a controlled NOT operation by being respectively in states $CNOT_0^{(A)}$ and $CNOT_1^{(A)}$, or in states $CNOT_1^{(B)}$ and $CNOT_0^{(B)}$, then we perform the appropriate controlled NOT operation on the data cells of A and C, and then swap their data registers with that of cells B and D, respectively. Any other configurations of operations will be ignored.

Note that this is not technically an active configuration. However, note that if two overlapping neighbourhoods overlap and both have first control registers in the configuration 1, 0, 1, 0, we only perform an operation on all four cells in the case that we are performing a controlled NOT, and by construction, it is not possible for both overlapping neighbourhoods to be correctly configured for a controlled NOT operation. Without using four separate states to indicate the two different orientations for the controlled NOT gate, ambiguities may arise which violate this overlap rule. It is also not possible for single-qubit operations to overlap with a controlled NOT operation. In other words, the effective set of operations we wish to perform on the entire lattice is still disjoint.

Now, during the update of the control register, we simply shift the first control registers to the right, so that the operations for the next time step will be performed next. Note that in order to shift this information to the right, we require a third control register $\Sigma_C = \{0, 1\}$, containing information which will not be used, to have its information shifted leftwards in order to maintain the local unitary property.

The clock register is initialized all in the state 0, and drives the data and control update procedures as described above.

5.3 Previous QCA Models

In this section, we will compare the Local Unitary QCA model with previous QCA models. In particular, we will look at the models proposed by Watrous [Wat95] and Schumacher and Werner [SW04].

The original QCA model as defined by Watrous, described in Section 3.2, is capable of describing a Shift-Right QCA, and therefore includes QCA which cannot be implemented in the Local Unitary model. Furthermore, because the local transition function maps basis configurations to quantum states over a single cell, it is not possible to construct entangled states from basis configurations. In the Local Unitary QCA model, we can achieve this by simulating a quantum circuit which constructs an entangled state.

The partitioned QCA model given by Watrous can be expressed in the Local Unitary QCA model, as long as in the partitioning of the cell alphabet $\Sigma = \Sigma_l \times \Sigma_c \times \Sigma_r$, we have $|\Sigma_l| = |\Sigma_r|$. To achieve this, we separate the permutation into an

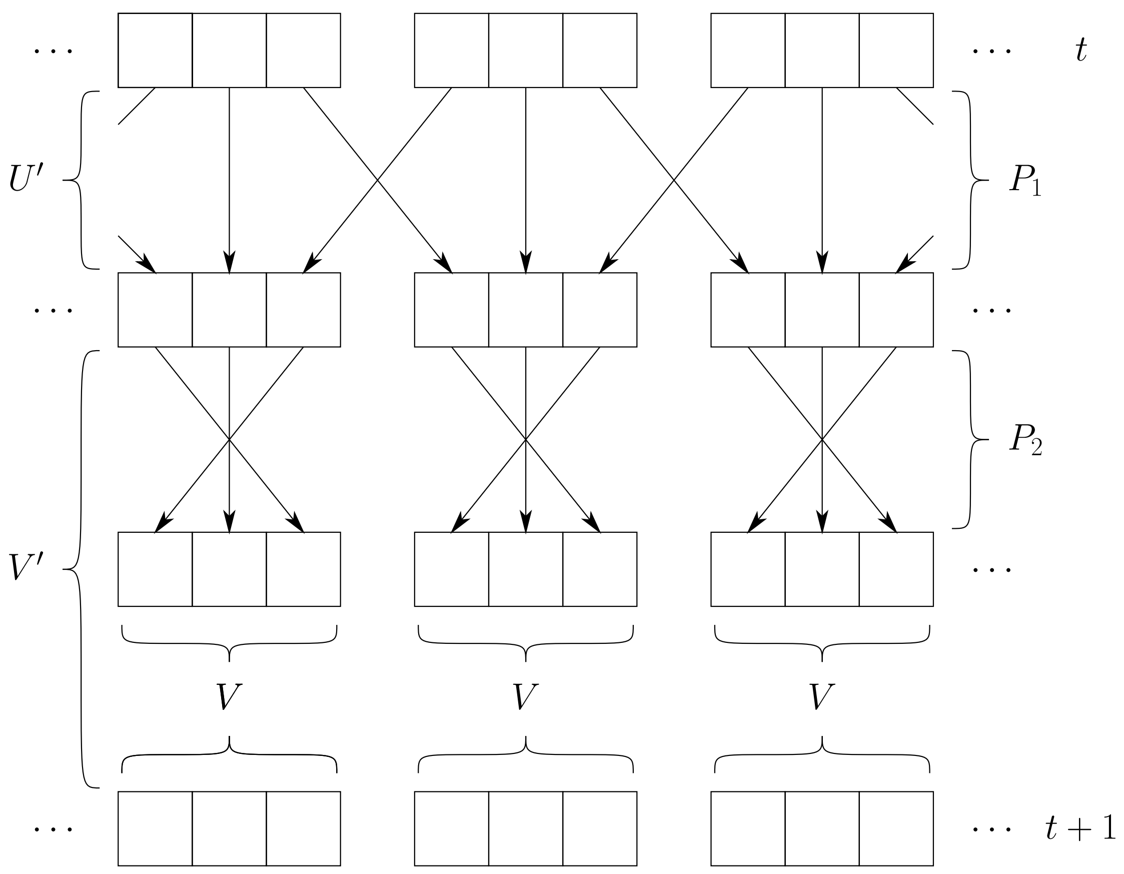


Figure 5.3: Watrous QCA expressed in the Local Unitary QCA model

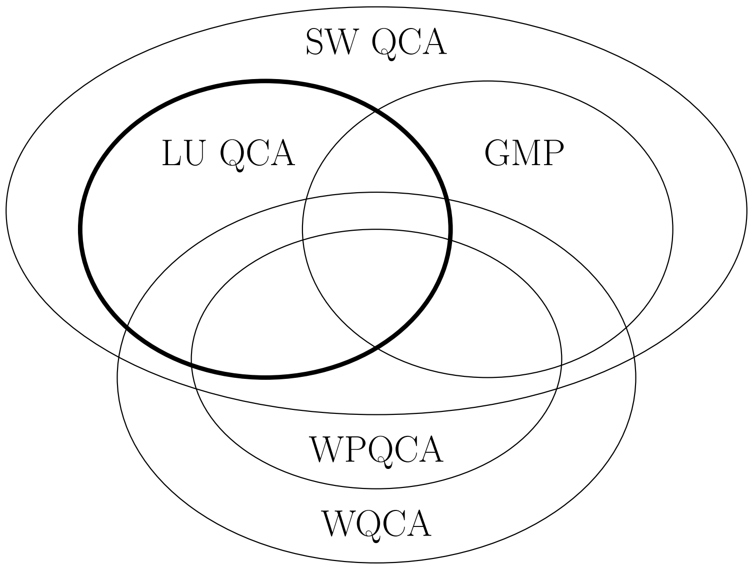


Figure 5.4: Relationships between various models of QCA

operation P_1 which operates on two consecutive cells, mapping

$$\begin{aligned} P_1 : \quad & \Big((q_k^{(l)}, q_k^{(c)}, q_k^{(r)}), (q_{k+1}^{(l)}, q_{k+1}^{(c)}, q_{k+1}^{(r)})\Big) \\ & \mapsto \Big((q_k^{(l)}, q_k^{(c)}, q_{k+1}^{(l)}), (q_k^{(r)}, q_{k+1}^{(c)}, q_{k+1}^{(r)})\Big) \end{aligned}$$

followed by an operation P_2 which operates on a single cell, mapping

$$P_2 : (q_k^{(l)}, q_k^{(c)}, q_k^{(r)}) \mapsto (q_k^{(r)}, q_k^{(c)}, q_k^{(l)}).$$

Note that $P_2 P_1$ performs the desired permutation, and also that P_1 commutes with any lattice translation of P_1. Now, we can express the Watrous partitioned QCA in our QCA model by setting $U' = P_1$ and $V' = V P_2$, as shown in Figure 5.3 (compare with Figure 3.1).

In general when $|\Sigma_l| \neq |\Sigma_r|$, it is possible to construct automata which cannot be implemented using local unitary operators. In particular, when $|\Sigma_l| = |\Sigma_c| = 1$, and $V = I$, we have the Shift-Right QCA.

In the Schumacher-Werner QCA model, every Local Unitary QCA yields a global transition function which is unitary and invariant under lattice translations, so the transition function give us a valid homomorphism over the quasi-local algebra of observables over the lattice. Also, by the very nature of the neighbourhood scheme in their Generalized Margolus Partitioning QCA, we cannot express arbitrary Local Unitary QCA within this model.

However, both Schumacher-Werner models are capable of constructing Shift-Right QCA. Figure 5.4 illustrates the relationships between all of these various models of QCA.

5.4 Quantum Spin Chains

In this section, we will explore the use of QCA in simulating space- and time-homogeneous physical phenomena. In particular, we will be looking at two cases of the Quantum Spin Chain model.

A Quantum Spin Chain consists of a 1-dimension lattice of identical quantum systems, each of which can be in one of some finite number of spin states. For our purposes, we will restrict our attention to the spin-$\frac{1}{2}$ spin chain, where each quantum system is a two-state system: they can have a spin of either $\frac{1}{2}$ or $-\frac{1}{2}$. The coupling between a spin at location n and the neighbouring spin at local $n+1$ can be described as a coupling Hamiltonian $H_{(n,n+1)}$, and the Hamiltonian for the entire chain, of length N is

$$H = \sum_{n=1}^{N-1} H^{(n,n+1)}.$$

This Hamiltonian is for an *open* chain. The chain may also be *closed*, in that the final qubit is connected to the first qubit, giving the Hamiltonian as

$$H = H^{(N,1)} + \sum_{n=1}^{N-1} H^{(n,n+1)}.$$

Either of these models can easily be expressed in terms of a 1-dimensional infinite lattice of cells $L = \mathbb{Z}$. For an open chain, we introduce a third, quiescent state $\epsilon \in \Sigma$ in addition to the spin states of each particle. The quiescent state indicates the absence of a particle. Now, we simply use ϵ as the initial state for any cell which does not correspond to a quantum system of the spin chain. For a closed chain, we simply use an initial state of period N, where each period contains the state of the entire closed spin chain.

Although the evolution of the Hamiltonian over the spin chain occurs in continuous time, we may discretize this evolution by defining a fixed time step Δt. Then, we may give a unitary operator which evolves the system according to the Hamiltonian H for time Δt. This becomes the basis for a discrete-time simulation of the spin chain. Specifically, we have

$$U_L = e^{-iH\Delta t},$$

using the convention of setting the constant $\hbar = 1$ for convenience.

We first explore the Ising spin chain, given by the Ising interaction

$$H^{(n,n+1)} = J\sigma_z^{(n)}\sigma_z^{(n+1)}$$

on neighbouring spins, where J is a coupling strength constant, and the Pauli operators are given as

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

The Hamiltonian for the entire chain will be

$$H = \sum_{n \in L} H^{(n,n+1)} = \sum_{n \in L} J\sigma_z^{(n)}\sigma_z^{(n+1)}.$$

Note that in this system, we have the convenient fact that $[H^{(n,n+1)}, H^{(m,m+1)}] = 0$ for all $n, m \in L$. In other words, the local interaction between spins is translate commutative. This allows us to simplify the modelling greatly. Because these local interaction Hamiltonians commute with each other, for a fixed time step Δt, we have

$$U_L = \exp(-iH\Delta t) = \exp\left(-i\sum_{n \in L} H^{(n,n+1)}\Delta t\right) = \prod_{n \in L} \exp(-iH^{(n,n+1)}\Delta t).$$

Let $U_n = \exp(-iH^{(n,n+1)}\Delta t)$. Since $H^{(n,n+1)}$ is translate commutative, it follows that U_n must also be. This convenient fact allows us to define our QCA immediately. We use the lattice $L = \mathbb{Z}$, $\Sigma = \{+\frac{1}{2}, -\frac{1}{2}\}$, $\mathcal{N} = \{0, 1\}$, $U_0 = \exp(-iH^{(0,1)}\Delta t)$, and $V_0 = I$. This QCA simulates the evolution of an Ising Spin Chain to as fine a resolution as we wish to fix.

In general, the local interaction Hamiltonians $H^{(n,n+1)}$ are not translate commutative. We will now examine the problem of simulating the Heisenberg Spin Chain, which has the local interaction Hamiltonian

$$H^{(n,n+1)} = J(\sigma_x^{(n)}\sigma_x^{(n+1)} + \sigma_y^{(n)}\sigma_y^{(n+1)} + \sigma_z^{(n)}\sigma_z^{(n+1)} - \mathbb{1} \otimes \mathbb{1}).$$

Unfortunately, in this case, the local interaction Hamiltonian is not translate commutative. However, we may still approximate the evolution of this Hamiltonian on a spin chain by making use of the *Suzuki-Trotter Decomposition*, which makes use of the fact that for Hermitian operators A and B and some small constant $\delta > 0$, we have

$$\|e^{\delta(A+B)} - e^{\delta A}e^{\delta B}\| = O(\delta^2)\|[B, A]\|.$$

This formula follows from the Baker-Campbell-Hausdorff (BCH) formula, which gives an expansion of $\log(e^A e^B)$ in terms of a power series for non-commuting operators A and B.

For our application, we will let

$$H_a = \sum_{n \in L} H^{(2n,2n+1)}$$

and

$$H_b = \sum_{n\in L} H^{(2n-1,2n)},$$

and set $A = -iH_a\Delta t$, $B = -iH_b\Delta t$ and $\delta = \frac{1}{k}$ for some integer constant k. We can rewrite the Suzuki-Trotter formula as

$$e^{-i(H_a+H_b)\Delta t} = \left(e^{-iH_a(\Delta t/k)}e^{-iH_b(\Delta t/k)}\right)^k + O\left(\frac{1}{k}\right)\|[H_b, H_a]\|\Delta t^2.$$

This means simulating the evolution of a Heisenberg spin chain for a time period of Δt within a given error tolerance simply requires finding a sufficiently large k. To accomplish this simulation, we need to alternate evolving the spin chain according to the Hamiltonian $e^{-iH_b(\Delta t/k)}$ and the Hamiltonian $e^{-iH_a(\Delta t/k)}$ a total of k times.

Since H_a and H_b are constructed as sums of local interaction Hamiltonians on disjoint neighbourhoods, they are pairwise commuting, and so we have, for some

$$U_a = e^{-iH_a(\Delta t/k)} = \exp\left(-\frac{i\Delta t}{k}\sum_{n\in L} H^{(2n,2n+1)}\right) = \prod_{n\in L} U_{2n}$$

and

$$U_b = e^{-iH_b(\Delta t/k)} = \exp\left(-\frac{i\Delta t}{k}\sum_{n\in L} H^{(2n-1,2n)}\right) = \prod_{n\in L} U_{2n-1},$$

where $U_n = \exp(-iH^{(n,n+1)}(\Delta t/k))$. Unfortunately, U_n is not a translate commutative operator. In order to set this procedure within the Local Unitary QCA framework, we proceed according to the Active Configuration Method, introduced in Section 5.2. We use the local neighbourhood scheme $\mathcal{N} = \{0, 1\}$, and we introduce a control register $\Sigma_2 = \{0, 1\}$ which is initialized to $|0\rangle$ for even-indexed cells $2n$, and $|1\rangle$ for odd-indexed cells $2n-1$. Note that the set $\{|01\rangle\}$is a valid set of active neighbourhood control configurations. This means that the operation U_n, which applies the operation $\exp(-\frac{i\Delta t}{k}H^{(n,n+1)})$ only if the control configuration is $|01\rangle$, is translate commutative. The single-cell update operation V_n simply flips the control bit. Now, the global transition function alternates between applying U_a and U_b, as desired. And as before, if the control registers are not properly initialized, we do not get the desired simulation. However, U_n remains translate commutative, so the evolution is still based on the Local Unitary model.

5.5 Coloured QCA

First, we will define the notion of a *symmetric* transition function for QCA. It is the quantum analog of symmetric CA, introduced in Section 2.4, in which the transition function depends only on the total number of neighbourhood cells in each of the possible cell states. Essentially, a transition update function is symmetric when it affects only the value of the target cell in a manner which depends only on how many of the cells in its neighbourhood are in particular states, rather than on the state that any particular neighbour is in.

Definition 5.5.1. *Given a QCA $Q = (L, \Sigma, \mathcal{N}, U_0, V_0)$, we call the update function $U_0 : \mathcal{N} \to \mathcal{N}$ symmetric if we have $tr_{\{0\}}U_0 = \mathbb{1}_{\mathcal{N}\setminus\{0\}}$, and U_0 commutes with every operator $SWAP_{x,y}$, which simply swaps the contents of cells x and y, where $x, y \in \mathcal{N} \setminus \{0\}$. If Q has a symmetric update function, then we call Q a symmetric QCA.*

Note that this definition of a quantum symmetric operation is actually analogous to the classical definition of an outer-symmetric transition function. In order to ensure that the operation is unitary, we cannot use the information in the target cell to control the operation being applied.

Next, we wish to formalize the notion of a colored QCA. For this model, we will fix the neighbourhood scheme to include only directly adjacent cells. That is, $\mathcal{N} = \{x \in \mathbb{Z}^d : \|x\|_1 \leq 1\}$, so that the a neighbourhood contains the target cell and $2d$ adjacent cells. However, first we will define the set of permissible colourings of a lattice.

Definition 5.5.2. *Given a lattice $L = \mathbb{Z}^d$, and a neighbourhood scheme $\mathcal{N}$ we define a correct k-colouring for a lattice as a periodic mapping $C : L \to 0, 1, \ldots, k-1$ such that no two adjacent cells in L are assigned the same colour.*

With our fixed neighbourhood $\mathcal{N}$, we may define adjacency to mean that one cell is in the neighbourhood of the other cell.

Definition 5.5.3. *Now, given a neighbourhood scheme $\mathcal{N}$ and a correct k-colouring of the lattice, we call an update function $U_0 : \mathcal{N} \to \mathcal{N}$ colour-symmetric if the following properties hold:*

1. *U_0 acts only on the target cell: $tr_{\{0\}}U_0 = \mathbb{1}_{\mathcal{N}\setminus\{0\}}$,*
2. *There exists a specific colour j, $0 \leq j < k$ such that U_x acts non-trivially only if the target cell has colour $C(x) = j$, and*
3. *U_0 commutes with every operator $SWAP_{x,y}$, such that $x, y \in \mathcal{N} \setminus \{0\}$ and $C(x) = C(y)$.*

The final condition means that the transition function U_0 essentially cannot distinguish between cells of the same colour within its neighbourhood. We can now finally give a definition for the colored QCA. Recall that the neighbourhood scheme $\mathcal{N}$ is fixed.

Definition 5.5.4. *A Coloured QCA or CQCA is a 4-tuple $(L, C, \Sigma, \mathcal{U})$ consisting of a lattice $L = \mathbb{Z}^d$, a correct k-colouring C, a finite set Σ of cell states, and a sequence of T colour-symmetric unitary operators $\mathcal{U} = \{U_0^{(0)}, U_0^{(1)}, \ldots, U_0^{(T-1)}\}$, which each map $U_0^{(j)} : (\mathcal{H}^\Sigma)^{\otimes \mathcal{N}} \to (\mathcal{H}^\Sigma)^{\otimes \mathcal{N}}$. The local transition operation consists of applying $U_x^{(j)}$ to each cell x, at time step $t = j + nT$, where $0 \leq j < T$ and $n \in \mathbb{Z}$.*

Note that since C is a correct k-colouring, no two operators $U_x^{(j)}$ act non-trivially on the same cell at the same time t. This means that, assuming a correct k-colouring, the operators $U_x^{(j)}$ are translate commutative. We will show that the Coloured QCA and Local Unitary QCA models can simulate each other.

Theorem 5.5.1. *For every Coloured QCA Q there exists a Local Unitary QCA Q' that simulates the same evolution exactly.*

Proof. We add an extra register to each cell which contains its colour information $C(x)$. Then, we add one extra clock register to each cell, initialized to 0. The update operator U_x simply applies $U_x^{(j)}$ conditional on both $C(x)$ and the clock register of cell x being set to j. In order to ensure that U_x commutes with its translates, the operator U_x also does not act non-trivially unless the colours of all the neighbours of x are consistent with the colouring C. The read operator V_x simply increments the clock register, modulo T. $\square$

Theorem 5.5.2. *For every Local Unitary QCA Q there exists a Coloured QCA Q' that simulates the same evolution exactly.*

Proof. Given the QCA $Q = (L, \Sigma, \mathcal{N}, U_0, V_0)$, we will use the same lattice L and alphabet Σ. The neighbourhood scheme for the CQCA, $\mathcal{N}'$ is fixed by definition. We also need to provide a correct k-colouring of the lattice. To this end, it suffices to provide a colouring C with the property that that no neighbourhood $\mathcal{N}_x$ of Q or $\mathcal{N}'_x$ of Q' contains two cells with the same colour. Now, we need to construct a sequence $\mathcal{U}$ of update operators. Note that conditional on a cell x having a particular colour, single-qudit operations on x, and the controlled-NOT operation targetting x are colour-symmetric operations. Because we insisted on a colouring such that no neighbourhood $\mathcal{N}'_x$ contains two cells of the same colour, such controlled-NOT operations can always be controlled by a single neighbouring qubit. Now, given an implementation of the unitary update operation U_0 of Q using single-qudit and nearest-neighbour controlled-NOT operations, we can give a sequence of colour-symmetric operations which perform U_0 on a neighbourhood $\mathcal{N}_x$ of a cell x of a specific colour. By performing a similar sequence of operations for each colour in our colouring C, we effectively perform U_x for each cell x. Since each update operation U_x commutes with the other update operations, we have effectively simulated the update transition operation of Q. Finally, we can perform the single-qudit operations V_x for each cell colour. $\square$

Besides providing us with another technique with which to construct Local Unitary QCA, the Colour QCA model also accurately models the pulse-driven spin chain, used by Lloyd [Llo93] in his proposed implementation of a quantum computer. Combining these techniques with implementations within the Colour QCA model, it may be possible to build physical implementations of Local Unitary QCA, assuming the availability of the appropriate resources.

5.6 Quantum Walks and Quantum Lattice Gases

It is also possible to simulate discrete-time quantum walks, described in [AAKV01], on a lattice $L = \mathbb{Z}^d$ using a quantum cellular automaton. However, in a quantum walk, the permissible states correspond to the lattice nodes themselves, rather than having each cell carry its own quantum information, as in the QCA. Meyer [Mey96] also uses this technique as the basis for his simulation of a quantum lattice gas.

In order to simulate such a restriction, we consider a binary QCA over the lattice $L = \mathbb{Z}^d$. We represent the state corresponding to a particular cell x using the QCA configuration corresponding to having each cell contain the state $|0\rangle$, with the exception of the cell x, which contains the state $|1\rangle$. This configuration will correspond to the logical state $|x\rangle$, corresponding to the given cell. Now, we need to construct a valid local transition function which gives a global transition function which is closed with respect to linear superpositions of these logical states. In other words, the transition function maps logical states $|x\rangle$ to linear superpositions of other logical states.

In order for the local transition function to be valid, it should act correctly when the configuration is a valid logical state, and act in a manner which remains translate commutative when it is not. In order to do this, we can use a local transition function which operates non-trivially only if the target cell is in the configuration $|1\rangle$, and the remaining cells of its neighbourhood are in the configuration $|0\rangle$. At this point, we should distinguish the neighbourhood of a vertex of the graph used in the quantum walk from the neighbourhood of the corresponding cell in the simulating QCA. Since we need to ensure that we do not have a situation where we need to apply two non-trivial operations on the same cell, the neighbourhood of a cell in the simulating QCA must contain all cells corresponding to vertices in the graph which have a common neighbour with the vertex corresponding to the target cell. Finally, when the local transition function acts non-trivially, the result must be a linear superposition of valid configurations, which have exactly one cell in the configuration $|1\rangle$.

It is also possible to simulate discrete-time quantum walks which use "coin" information to affect the transition function. To do this, we add a coin register for each cell, where the active coin information is stored in the coin register corresponding to the active cell which is in the configuration $|1\rangle$. The local transition function will be responsible for ensuring that this information remains with the active cell. This can be accomplished using controlled-SWAP operations on the coin registers.

It should also be noted that the neighbourhood scheme in the original graph needs to be transitive with respect to translations in order for this construction to work. However, if this is not the case, we may also store information about the neighbourhood of each vertex in the graph within each corresponding cell. In such a simulation, the local transition function does not alter this neighbourhood information.

Chapter 6

Further Directions

In this chapter, we will discuss some natural extensions of the Local Unitary QCA model, and speculate on further possible applications of QCA theory to other areas of quantum information science.

6.1 General Local QCA

It is possible to define a model of QCA based on an open quantum system which include dynamics that are not necessarily unitary, including full or partial measurements of cell information, and other general interactions with the external environment. The simplest way to augment the Local Unitary model of QCA to include such operations is to allow the single-cell operator V to be an arbitrary single-cell completely positive map. We may view this operation as a circuit consisting of unitary operations and partial measurements of the information contained within each single cell. We may also use the idea of dissipation as a primitive operation, where some register within a cell is set to a specific ground state through interaction with the environment.

Using dissipation, we may express any non-reversible CA as a General Local QCA, by using a memory register, as in the Physical Reversible CA model described in Section 4.2. After each time step, the single-cell operation V simply dissipates the memory register of each cell.

General Local QCA may also form the basis for constructing fault-tolerant forms of QCA, since fault tolerance in quantum circuits can be achieved using local unitary operations and dissipation [Sho96]. Fault-tolerant constructions for classical CA were introduced by Harao and Noguchi [HN75], Toom [Too76], and Gcs [Gác83], and are fairly complicated. It is reasonable to guess that corresponding constructions for QCA will be at least as complicated.

6.2 Non-Lattice QCA

In the application of QCA to implementations on specific physical systems, it may be desirable to consider QCA acting on lattice structures which are not cubic. Although it is always possible to consider embeddings of such structures in lattices $L = \mathbb{Z}^d$, we may also wish to consider QCA over a generic class of structures called *Cayley graphs*.

In a Cayley graph, the set of vertices corresponds to the elements of a group G, and a neighbourhood scheme is given as a set of group elements $\mathcal{N} \subseteq G$. For a given cell corresponding to the vertex $g \in G$, the local transition function is a unitary translate commutative operation acting on the cells corresponding to the vertices $g\mathcal{N} = \{gh : h \in \mathcal{N}\}$. This definition allows us to explore both finite and infinite structures G. The lattice $L = \mathbb{Z}^d$ itself can be represented by the free Abelian group generated by d independent elements $g_1, g_2, \ldots, g_d$.

We may also use these structures in the context of Coloured QCA, introduced in Section 5.5, where we may use separate neighbourhood schemes for each different cell colour. In this context, we may generalize the definition of a correct colouring by fixing a particular subgroup $H \leq G$, which would correspond to the sublattice in the previous definition of a correct colouring. For each right-coset of the subgroup H, we assign each cell corresponding that coset with the same colour. However, it is not necessary that different cosets are assigned different colours.

Part II

Algorithmic Search for Entanglement Witnesses

Chapter 7

Introduction

There has been much interest in the study of the entanglement and the separability of quantum states, both from a theoretical and a practical viewpoint. Entanglement is a key phenomenon which differentiates quantum theory from classical theory, and can be viewed as a resource which enables quantum computation to supersede classical computation.

In Part II of this thesis, we will examine the problem of determining whether or not a given quantum state contains entanglement from a theoretical viewpoint. We will show how this problem can be framed as a problem in convex optimization, and explore *entanglement witnesses*, which are operators which can potentially verify the presence of entanglement in a given quantum state.

7.1 Summary of Results

We will discuss the quantum separability problem, and explore its relationships to other known problems. In order to do this, we describe a setting of the quantum separability problem in terms of a convex optimization problem. Specifically, by using some structural properties of the set of separable states, we develop a new classical algorithm that solves the quantum separability problem more efficiently than the natural algorithm which arises from the definition of separability. This algorithm requires a polynomial number of calls to an oracle which solves an optimization problem over the set of separable states.

Chapter 8

Quantum Separability and Entanglement

In this chapter, we will review some of the theoretical background behind the study of quantum separability and entanglement of states.

8.1 Bipartite Quantum State Separability

Given two disjoint quantum systems represented by the finite-dimensional Hilbert spaces $\mathcal{H}_1$ and $\mathcal{H}_2$, we say that a given pure state $|\psi\rangle \in \mathcal{H}_1 \otimes \mathcal{H}_2$ is *separable* if the state can be considered as two separate states on the two Hilbert spaces, $|\psi_1\rangle \in \mathcal{H}_1$ and $|\psi_2\rangle \in \mathcal{H}_2$, with $|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle$. In order to define separability more generally for mixed quantum states, we represent quantum states as density matrices, so that a pure state given in vector notation as $|\psi\rangle$ is given as the matrix $\rho = |\psi\rangle\langle\psi|$. In general, density matrices are Hermitian positive semidefinite matrices with trace 1. We can also express mixed quantum states as convex combinations of corresponding pure states. Specifically, given a basis $|\psi_j\rangle$, $j = 1 \ldots d$, of a Hilbert space $\mathcal{H}$ of dimension d, we can represent a mixed quantum state as

$$\rho = \sum_{j=1}^{d} p_j |\psi_j\rangle\langle\psi_j|,$$

where we have $p_j \geq 0$ for each $j = 1 \ldots n$, and

$$\sum_{j=1}^{d} p_j = 1.$$

This represents a mixed quantum state consisting of a classical mixture of basis states $|\psi_j\rangle$ with corresponding probabilities p_j. A given mixed state $\rho \in \mathcal{H}_1 \otimes \mathcal{H}_2$ is separable if for each index j with $p_j \neq 0$, the corresponding basis state $|\psi_j\rangle\langle\psi_j| \in$

$\mathcal{H}_1 \otimes \mathcal{H}_2$ is a separable pure state. Note that this means that the set of separable quantum states is simply the convex hull of the set of separable pure quantum states. Thus, given two Hilbert spaces $\mathcal{H}_1$ of dimension d_1 and $\mathcal{H}_2$ of dimension d_2, we can describe the set of separable quantum states of the Hilbert space $\mathcal{H} = \mathcal{H}_1 \otimes \mathcal{H}_2$ of dimension $d = d_1 d_2$ as a convex region of the set of $d \times d$ Hermitian matrices.

We may represent the set of $d \times d$ Hermitian matrices over the complex numbers $\mathbb{C}$ as a vector of size d^2 over the real numbers $\mathbb{R}$ by use of the generalized Gell-Mann basis. The generalized Gell-Mann matrices consist of $d^2 - 1$ Hermitian matrices with trace 0, and are a generalization of the Pauli matrices in dimension $d = 2$:

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

and the regular Gell-Mann matrices in dimension $d = 3$:

$$G_{1,2} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad G_{1,3} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \quad G_{2,3} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

$$G_{2,1} = \begin{pmatrix} 0 & -i & 0 \\ i & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad G_{3,1} = \begin{pmatrix} 0 & 0 & -i \\ 0 & 0 & 0 \\ i & 0 & 0 \end{pmatrix} \quad G_{3,2} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -i \\ 0 & i & 0 \end{pmatrix}$$

$$G_{1,1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad G_{2,2} = \frac{1}{\sqrt{3}} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -2 \end{pmatrix}.$$

For higher dimensions, we may construct the generalized Gell-Mann matrices as follows. Let $e_{j,k}$ be the $d \times d$ matrix with 1 as the (j,k)-th entry, and 0 as every other entry. Then, the generalized Gell-Mann matrices consist of

$$G_{j,k} = e_{j,k} + e_{k,j}$$

for $1 \le j < k \le d$,

$$G_{j,k} = i(e_{j,k} - e_{k,j})$$

for $1 \le k < j \le d$, and finally

$$G_{j,j} = \sqrt{\frac{2}{j(j+1)}} \left(\left(\sum_{k=1}^{j} e_{k,k} \right) - j e_{j+1,j+1} \right)$$

for $1 \le j < d$.

The generalized Gell-Mann matrices form an $\mathbb{R}$-basis for the set of $d \times d$ trace-0 Hermitian matrices. In other words, the set of real linear combinations of these matrices span the trace-0 Hermitian matrices. By including a normalized identity

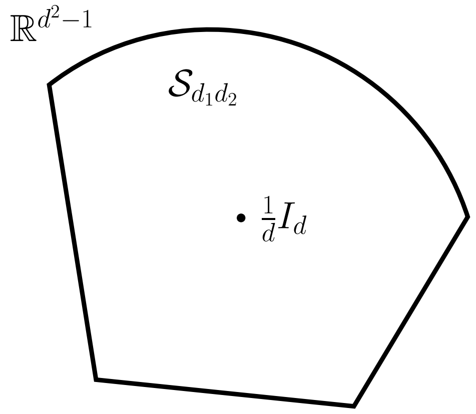


Figure 8.1: The set of separable states $\mathcal{S}_{d_1,d_2}$ as a region of $\mathbb{R}^{d^2-1}$

matrix

$$G_{d,d} = \frac{1}{d} I_d,$$

with trace 1, we form an $\mathbb{R}$-basis which spans all of the $d \times d$ Hermitian matrices. This gives us a representation of all $d \times d$ Hermitian matrices over the complex numbers $\mathbb{C}$ as a d^2-dimensional real vector space. Furthermore, since all density matrices corresponding to quantum states on $\mathcal{H}$ have trace 1, they must all lie on the corresponding hyperplane spanned by only the trace 0 generalized Gell-Mann matrices. Also, since convex combinations of positive semidefinite matrices with trace 1 remain positive semidefinite while preserving trace, the region on this hyperplane corresponding to the density matrices must also remain convex. We may either consider the set of density matrices as a convex region on this hyperplane, or alternatively, as a convex region on the vector space $\mathbb{R}^{d^2-1}$ corresponding to the matrices $\rho - \frac{1}{d}I_d$ represented in the generalized Gell-Mann basis for each density matrix ρ. Note that the vector $\mathbf{0} \in \mathbb{R}^{d^2-1}$ corresponds to the density matrix $\frac{1}{d}I_d$, which gives the maximally mixed state.

Finally, note that since this transformation is linear, the set of separable quantum states as represented in $\mathbb{R}^{d^2-1}$ remains a convex set. Given the quantum system $\mathcal{H} = \mathcal{H}_1 \otimes \mathcal{H}_2$ where $\dim(\mathcal{H}_1) = d_1$ and $\dim(\mathcal{H}_2) = d_2$, we will denote the set of separable states of $\mathcal{H}$ relative to $\mathcal{H}_1$ and $\mathcal{H}_2$ represented in the vector space $\mathbb{R}^{d^2-1}$ as $\mathcal{S}_{d_1,d_2} \subseteq \mathbb{R}^{d^2-1}$. This set is illustrated as a generic convex set in Figure 8.1. However, the actual structure of the set $\mathcal{S}_{d_1,d_2}$ itself is quite complicated, and has been the subject of much study.

8.2 Conditions for Separability

Given a quantum state $\rho \in \mathcal{H} = \mathcal{H}_1 \otimes \mathcal{H}_2$, we say that ρ is *entangled* if it is not separable. Although there is currently no known set of necessary and sufficient conditions for efficiently testing the separability of a given quantum state, there are numerous partial results which give either necessary or sufficient conditions for separability.

One such condition is called the Positive Partial Transpose (PPT) criterion [HHH96, Per96], which gives a necessary condition for separability. The partial transpose is given by the map

$$\rho^{T_2} = (I \otimes T)(\rho),$$

where $T : \rho \to \rho^T$ maps a matrix ρ to its transpose ρ^T. In other words, given a block matrix representation of $\rho \in \mathcal{H} = \mathcal{H}_1 \otimes \mathcal{H}_2$ in terms of matrices $\rho_{ij} \in \mathcal{H}_2$:

$$\rho = \begin{pmatrix} \rho_{11} & \cdots & \rho_{1d_1} \\ \vdots & \ddots & \vdots \\ \rho_{d_1 1} & \cdots & \rho_{d_1 d_1} \end{pmatrix},$$

the partial transpose is given as

$$\rho^{T_2} = \begin{pmatrix} \rho_{11}^T & \cdots & \rho_{1d_1}^T \\ \vdots & \ddots & \vdots \\ \rho_{d_1 1}^T & \cdots & \rho_{d_1 d_1}^T \end{pmatrix}.$$

We can similarly define the partial transpose $\rho^{T_1} = (T \otimes I)(\rho)$. However, ρ^{T_1} is simply the transpose of ρ^{T_2}. The PPT criterion states that if the density matrix of a given quantum state $\rho \in \mathcal{H} = \mathcal{H}_1 \otimes \mathcal{H}_2$ is separable, then its partial transpose ρ^{T_2} must be positive semidefinite. However there do exist entangled states which satisfy the PPT condition in the case that $d = d_1 d_2 > 6$. Otherwise, for $d \leq 6$, the PPT condition characterizes the set of separable states.

It is also possible to give sufficient conditions for separability. We may consider the distance between a separable state ρ and the maximally mixed state $\frac{1}{d} I_d$ using the Hilbert-Schmidt norm resulting from the Hilbert-Schmidt inner product on $\mathcal{H}$, given by $\langle A, B\rangle = \mathrm{tr}(A^\dagger B)$. In particular, we may consider the radius of the largest possible ball centered on the maximally mixed state $\frac{1}{d} I_d$ which lies within the set of separable density matrices. It was shown in [GB02] that all density matrices ρ which satisfy

$$\left\| \rho - \frac{1}{d} I_d \right\|_{HS} = \sqrt{\mathrm{tr}\left(\rho - \frac{1}{d} I_d\right)^2} \leq \frac{1}{\sqrt{d(d-1)}}$$

are separable. Furthermore, this is indeed the size of the largest separable ball with respect to the Hilbert-Schmidt norm. This provides us with a sufficient condition for separability. However, many separable states will not satisfy this condition.

There are many other criteria which have been formulated to determine either entanglement or separability of a given quantum state. In particular, [EHGC04] gives an approach in which the separability problem is formulated as an optimization problem. A efficient hierarchy of either necessary or sufficient conditions can be formed by considering a series of relaxations of this optimization problem as semidefinite programs which approximate the original optimization problem.

8.3 Entanglement Witnesses

Recall that the set of separable density matrices may be expressed in the generalized Gell-Mann basis as the region $\mathcal{S}_{d_1,d_2} \subseteq \mathbb{R}^{d^2-1}$ lying on a hyperplane of $\mathbb{R}^{d^2}$ corresponding to Hermitian matrices with trace 1. We may extend the notion of separability to all Hermitian matrices by defining the Hermitian matrix ρ to be separable if there exists a separable density matrix σ with $\mathrm{tr}(\sigma) = 1$ such that

$$\rho = \alpha\sigma$$

for some real scalar $\alpha \geq 0$. With this extended definition of separability, the set of separable Hermitian matrices expressed in the generalized Gell-Mann basis is simply the cone corresponding to $\mathcal{S}_{d_1,d_2}$ in $\mathbb{R}^{d^2}$. This cone is also a convex set.

Since the set of separable Hermitian matrices forms a convex set, for any Hermitian matrix ρ which is entangled (not separable), it should be possible to find a hyperplane which separates ρ from the set of separable Hermitian matrices. At this point, we should distinguish the different notions of separable matrices, as described in Section 8.1, and the separation of two sets by means of a hyperplane. Specifically, for an entangled Hermitian matrix ρ, there must exist a Hermitian operator W and a real number α such that

$$\langle W, \rho\rangle > \alpha$$

and

$$\langle W, \sigma\rangle \leq \alpha$$

for all separable Hermitian matrices σ.

We may adjust the constant α until the hyperplane $\langle W, x\rangle = \alpha$ is tangent to set of separable Hermitian matrices. Since the matrix $\mathbf{0}$ is separable, we must have $\langle W, \mathbf{0}\rangle = 0 \geq \alpha$. Moreover, if $\langle W, \sigma\rangle < 0$ for some separable Hermitian matrix σ, then $\langle W, x\rangle$ is unbounded from below for separable Hermitian matrices x, since any positive multiple of σ is also separable. Thus, it suffices to search for Hermitian operators W such that

$$\langle W, \rho\rangle > 0$$

and

$$\langle W, \sigma\rangle \leq 0$$

for all separable Hermitian matrices σ. Such an operator is called an *entanglement witness.*

Finally, we need to consider how to represent an entanglement witness in the generalized Gell-Mann basis. Note that for any two distinct matrices G_1, G_2 in the

Gell-Mann basis, we have $\langle G_1, G_2 \rangle = 0$. This means that the generalized Gell-Mann basis is an orthogonal basis with respect to the Hilbert-Schmidt inner product. We may normalize the elements of the Gell-Mann basis to construct an orthonormal basis. Given two $d \times d$ matrices A and B, the Hilbert-Schmidt inner product $\langle A, B \rangle$ can be thought of as a dot product of vectors of length d^2 corresponding to the entries of the matrices A and B. In other words, indexing the entries of A and B as A_{jk} and B_{jk}, respectively,

$$\langle A, B \rangle = \sum_{j=1}^{d} \sum_{k-1}^{d} A_{jk} B_{jk}.$$

Thus, given the vectors $a, b \in \mathbb{R}^{d^2}$corresponding to the matrices A and B expressed in the normalized generalized Gell-Mann basis, we have

$$\langle A, B \rangle = a^T b,$$

since a and b are real-valued vectors.

As a result, a hyperplane in the space of Hermitian matrices over the complex number $\mathbb{C}$ will correspond exactly to a hyperplane in the real vector space $\mathbb{R}^{d^2}$ using the real-valued dot product as the inner product. Thus, given a vector $p \in \mathbb{R}^{d^2}$ corresponding to an entangled Hermitian matrix expressed in the generalized Gell-Mann basis, we can construct a corresponding hyperplane given by a vector $w \in \mathbb{R}^{d^2}$ such that

$$w^T p > 0$$

and

$$w^T x \leq 0$$

for all vectors $x \in \mathbb{R}^{d^2}$ in the convex cone corresponding to all separable Hermitian matrices. We may also restrict our attention to the set of Hermitian matrices with trace 1. In this case, the generalized Gell-Mann basis represents such matrices as vectors in the real vector space $\mathbb{R}^{d^2-1}$, and an entanglement witness can be expressed as a hyperplane given by a vector $w \in \mathbb{R}^{d^2-1}$ and a real scalar $\alpha \in \mathbb{R}$ such that

$$w^T p > \alpha$$

for the vector $p \in \mathbb{R}^{d^2-1}$ corresponding to the density matrix of an entangled quantum state in $\mathcal{H} = \mathcal{H}_1 \otimes \mathcal{H}_2$, and

$$w^T x \leq \alpha$$

for all $x \in \mathcal{S}_{d_1,d_2}$.

If we so desire, we can also consider an entanglement witness for Hermitian matrices corresponding to a hyperplane $w \in \mathbb{R}^{d^2-1}$. This yields an alternative definition of entanglement witnesses, restricted to the set of Hermitian matrices with trace 1, where an entanglement witness for the entangled density matrix ρ is

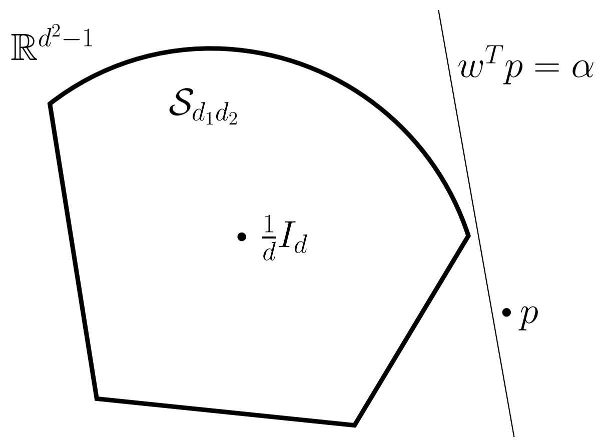


Figure 8.2: $\mathcal{S}_{d_1,d_2}$ and an entangled state ρ separated by $\langle W, x \rangle = \alpha$

defined as a Hermitian operator W with $\mathrm{tr}(W) = 0$ and a real scalar α such that

$$\langle W, \rho \rangle > \alpha$$

and

$$\langle W, \sigma \rangle \leq \alpha$$

for all separable Hermitian matrices σ. We will use this alternative definition for entanglement witnesses in our discussion. It should be noted that although we will be using Hermitian matrices with complex entries in our notation, any actual computation will be done in the normalized generalized Gell-Mann basis, using real vectors from $\mathbb{R}^{d^2-1}$. Figure 8.2 illustrates an entanglement witness separating $\mathcal{S}_{d_1,d_2}$ and an entangled state.

8.4 Search Problems Related to Separability

By formulating entanglement witnesses as hyperplanes which separate points from the convex set $\mathcal{S}_{d_1,d_2}$, we may relate problem of searching for entanglement witnesses to a number of other problems related to convex optimization. First, we shall formally define the Quantum Separability Problem:

Definition 8.4.1 (Quantum Separability Problem)**.** *Given a density matrix ρ corresponding to a quantum state in $\mathcal{H} = \mathcal{H}_1 \otimes \mathcal{H}_2$ and a precision $\delta > 0$, we either assert that there exists a separable state σ such that $\|\rho - \sigma\| < \delta$ or that there exists an entangled state τ such that $\|\rho - \tau\| < \delta$.*

Note that these two options are not mutually exclusive, and that a quantum state which is sufficiently close to the boundary of $\mathcal{S}_{d_1,d_2}$ will satisfy both statements. This is necessary to ensure that the precision requirements for solving the Quantum

Separability Problem remain bounded. This problem has been shown to be NP-hard [Gur03].

In order to find a witness which can verify that a given state is separable, we need to find a set of at least d^2 pure separable states such that σ is a convex combination of these states. In general, there is no efficient method to do this. However, we may apply a basic algorithm derived from the definition of separability by considering the set of pure separable states to precision δ. Formally, such a set would simply include sufficiently many pure separable states so that any pure separable state is within δ of an element of this set. Let Ω_δ be the number of pure separable states required to form such a set. Then, we may consider the $\binom{\Omega_\delta}{d^2}$ different ways of selecting d^2 pure separable states. For each such subset, we need to check whether σ lies within the (d^2-1)-dimensional convex polytope formed by the convex hull of these d^2 states. This can be checked by examining the position of σ with respect to each of the (d^2-2)-dimensional facets on the boundary of this polytope. Since there are d^2 such facets, this task can be accomplished in $\text{poly}(d^2, \log(1/\delta))$ time, giving a total run time of

$$\binom{\Omega_\delta}{d^2} \times \text{poly}(d^2, \log(1/\delta)).$$

In order to verify that a given state is entangled, we need to find an entanglement witness W as defined in Section 8.3. Note that a density matrix τ represents an entangled state if and only if such an entanglement witness exists. Technically, we may restrict our search to entanglement witnesses W with Hilbert-Schmidt norm $\|W\|_{HS} = \text{tr}(W^2) = 1$, since a positive scalar multiple of an entanglement witness remains an entanglement witness. However, in order to formulate the search for an entanglement witness as a search within a convex region, we will define our entanglement witness search space as the (d^2-1)-dimensional unit ball

$$\mathcal{W} = \{A \in \mathbb{C}^{d_1 \times d_2} : A^\dagger = A, \text{tr}(A) = 0, \text{tr}(A^2) \leq 1\}.$$

However, note that this verification process does not place the problem of deciding whether a given state is entangled in the complexity class NP. This is because there is no known polynomial-time process for verifying that the set of separable states is separated from the entangled state by the witness.

Now, we can define the Entanglement Witness Problem:

Definition 8.4.2 (Entanglement Witness Problem)**.** *Given a density matrix ρ corresponding to a quantum state in $\mathcal{H} = \mathcal{H}_1 \otimes \mathcal{H}_2$ and a precision $\delta > 0$, we either assert that there exists a separable state σ such that $\|\rho - \sigma\| < \delta$ or we find an operator $W \in \mathcal{W}$ such that tr$(W\sigma) <$ tr$(W\rho) + \delta$ for every $\sigma \in \mathcal{S}_{d_1,d_2}$.*

Note that because of issues of precision, we are searching for an approximate entanglement witness. In the case that our given state ρ is too close to the boundary of $\mathcal{S}_{d_1,d_2}$, it is permissible to return an incorrect answer. Also, finding a solution to the

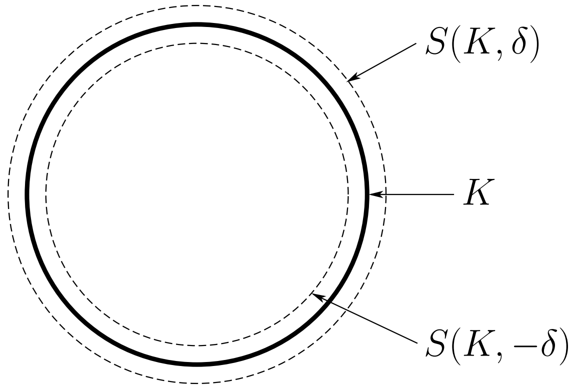


Figure 8.3: A convex set K with $S(K,\delta)$ and $S(K,-\delta)$

Entanglement Witness Problem also gives a solution to the Quantum Separability Problem.

We may relate the Entanglement Witness Problem to a number of convex body problems. Given a bounded convex set K, and some fixed parameter $\delta > 0$ we define $S(K,\delta)$ to be the union of all balls of radius δ with centres belonging to the set K. Conversely, we define $S(K,-\delta)$ to be the union of the centres of all balls of radius δ which are completely contained in the set K. Figure 8.3 gives an illustration of these three sets. We may define the following convex body problems:

Definition 8.4.3 (Weak Membership Problem)**.** *Given a rational vector $p \in \mathbb{R}^n$, either assert that $p \in S(K,\epsilon)$, or assert that $p \notin S(K,-\epsilon)$.*

Definition 8.4.4 (Weak Feasibility Problem)**.** *Given rational $\epsilon > 0$ either find a vector $y \in S(K,\epsilon)$, or assert that $S(K,-\epsilon)$ is empty.*

Definition 8.4.5 (Weak Optimization Problem)**.** *Given a rational vector $c \in \mathbb{R}^n$ and rational $\epsilon > 0$, either find a rational vector $y \in S(K,\epsilon)$ such that $c^T x \leq c^T y + \epsilon$ for every $x \in K$, or assert that $S(K,-\epsilon)$ is empty.*

Definition 8.4.6 (Weak Separation Problem)**.** *Given a rational vector $p \in \mathbb{R}^n$ and rational $\delta > 0$, either assert that $p \in S(K,\delta)$ or find a rational vector $c \in \mathbb{R}^n$, with $\|c\|_\infty = 1$ such that $c^T x < c^T p$ for every $x \in K$.*

We may also define strong versions of each of these problems by setting $\epsilon = 0$ and $S(K,\epsilon) = S(K,-\epsilon) = K$. Furthermore, for quantum states, it may be more appropriate to work with real algebraic numbers, rather than rational numbers, since algebraic numbers are more convenient for the formal definition of these problems as they relate to quantum states. The complexity of representing real numbers within a given precision δ using algebraic numbers is still $O(\log(1/\delta))$. However, from a practical standpoint, such a distinction is unimportant.

We may relate these convex body problems to problems related to quantum state separability and entanglement using the fact that both the set of separable states $\mathcal{S}_{d_1,d_2}$ and the set of potential entanglement witnesses $\mathcal{W}$ are convex. For a given density matrix ρ corresponding to a quantum state in the Hilbert space $\mathcal{H}$, we can also define $\mathcal{W}_\rho$ to be the set of entanglement witnesses which verify that ρ is an entangled state. If ρ is separable, we simply take $\mathcal{W}_\rho = \emptyset$.

The Weak Membership Problem is simply the Quantum Separability Problem applied to the convex set $K = \mathcal{S}_{d_1,d_2}$, while the Weak Separation Problem becomes the Entanglement Witness Problem. The Weak Feasibility Problem applied to $K = \mathcal{W}_\rho$ is also equivalent to the Entanglement Witness Problem. However, these reductions require the use of an oracle which can determine membership in $\mathcal{S}_{d_1,d_2}$ and $\mathcal{W}_\rho$ respectively. For example, verifying that a given Hermitian operator W is an entanglement witness in $\mathcal{W}_\rho$ also requires that we verify that $\langle W, \sigma \rangle$ is bounded by an appropriate constant for all $\sigma \in \mathcal{S}_{d_1,d_2}$. This either requires some structural knowledge concerning $\mathcal{S}_{d_1,d_2}$ or some application of the Weak Optimization Problem.

Chapter 9

Finding Entanglement Witnesses

In this chapter, we will propose a deterministic algorithm for solving the Entanglement Witness Problem for a mixed quantum state, given in terms of a density operator, $\rho \in \mathcal{H} = \mathcal{H}_1 \otimes \mathcal{H}_2$. This also gives us an algorithm for solving the Quantum Separability Problem. These problems were described first in Section 8.4. The algorithm we present will be an efficient reduction from the Weak Separation Problem to the Weak Optimization Problem for $\mathcal{S}_{d_1,d_2}$, which will also give us an algorithm for the Entanglement Witness Problem. Although this algorithm is only polynomial-time relative to a Weak Optimization oracle, such an oracle may be simulated by solving corresponding global optimization problems on functions over $2(d_1 + d_2) - 4$ variables.

9.1 Solving Separation via Optimization

In order to verify that a given Hermitian operator $W \in \mathcal{W}$ is indeed an entanglement witness for a given entangled state ρ, we need to produce a real value α such that the hyperplane $\langle W, x\rangle = \alpha$ separates the set of separable states $\mathcal{S}_{d_1,d_2}$ from the state ρ. We should note that the inner product $\langle A, B\rangle$ we use here is the one given by the dot product of the representations of A and B in $\mathbb{R}^{d^2-1}$ by using the trace 0 elements of the Gell-Mann basis. In particular, the zero element of this space in fact corresponds to the density matrix $\frac{1}{d}I_d$ corresponding to the maximally mixed state. Although we will use retain the Hermitian matrix notation for all of the states and operators, it should be understood that any all computations are actually performed on real vectors in $\mathbb{R}^{d^2-1}$.

We can achieve this verification by solving the related problem of finding the maximum value of $\langle W, \sigma\rangle$ for $\sigma \in \mathcal{S}_{d_1,d_2}$, which, after taking precision considerations into account, becomes an instance of the Weak Optimization Problem. This means that we can use an oracle which solves the Weak Optimization Problem for the set $\mathcal{S}_{d_1,d_2}$ in order to test whether a given operator W is in the set $\mathcal{W}_\rho$.

In order to solve this optimization problem, note that $\langle W, x\rangle$ is a linear function, and that we wish to optimize it over a convex region. This means that there will exist a value of x which optimizes $\langle W, x\rangle$ on the boundary of the set $\mathcal{S}_{d_1,d_2}$. In fact, since $\mathcal{S}_{d_1,d_2}$ is the convex hull of the set of pure separable states on $\mathcal{H}_1 \otimes \mathcal{H}_2$, it is sufficient to restrict our search to these states, since no convex combination of these states will yield a better value of $\langle W, x\rangle$. Pure states on $\mathcal{H}_1$ can be represented by complex vectors of size d_1. However, since pure states are restricted to vectors with norm 1, and since pure states are unaffected by global phase factors, it is in fact possible to parameterize a pure state on $\mathcal{H}_1$ using $d_1 - 1$ complex numbers, or equivalently, $2(d_1 - 1)$ real numbers. Similarly, pure states on $\mathcal{H}_2$ can be parameterized by $2(d_2 - 1)$ real numbers, meaning that separable pure states on $\mathcal{H} = \mathcal{H}_1 \otimes \mathcal{H}_2$ can be parameterized by $2(d_1 + d_2) - 4$ variables. This allows us to reduce the problem of optimizing $\langle W, x\rangle$ on the set $\mathcal{S}_{d_1,d_2}$ to a global optimization problem on a function with $2(d_1 + d_2) - 4$ variables. Although there are no universally efficient techniques for this, there exist heuristic techniques that generally perform well. At the worst case, we require Ω_δ evaluations of $\langle W, x\rangle$, where Ω_δ, defined in Section 8.4, is the number of pure separable states required to form δ-net, where each pure separable set is within δ of an element of the δ-net. This gives us a complexity of

$$\Omega_\delta \times \mathrm{poly}(d^2, \log(1/\delta)).$$

We now have an algorithm which solves the Weak Membership Problem with respect to $\mathcal{W}_\rho$, as well as a method of verifying that there is indeed separation between an entangled state ρ and $\mathcal{S}_{d_1,d_2}$ given an entanglement witness $W \in \mathcal{W}_\rho$.

9.2 Solving the Entanglement Witness Problem

Now, we consider the problem of solving the Entanglement Witness Problem using an oracle which solves the Weak Optimization Problem for $\mathcal{S}_{d_1,d_2}$. Recall that the Entanglement Witness Problem is essentially the Weak Membership Problem applied to the set $\mathcal{W}_\rho$, and an algorithm for performing this task is given in Section 9.1. However, this alone will not give an efficient reduction from the Entanglement Witness Problem to the Weak Optimization Problem. What we need is a method by which we can take each guess $W \in \mathcal{W}$ and use it to reduce our search space. Otherwise, it would be necessary to perform a complete search of set $\mathcal{W}$ for potential elements of $\mathcal{W}_\rho$. Fortunately, such a method exists, and for the rest of this chapter, we will develop this algorithm and give a pseudocode description at the end of Section 9.3. We begin by establishing the following Lemma:

Lemma 9.2.1. *Suppose we have $E \in \mathcal{W}_\rho$ and $W \in \mathcal{W} \setminus \mathcal{W}_\rho$ for a given entangled state ρ. Let $\sigma_W \in \mathcal{S}_{d_1,d_2}$ be the solution to the Weak Optimization Problem for the function $\langle W, x\rangle$ over $\mathcal{S}_{d_1,d_2}$. Define $C = (\rho - \sigma_W) - \mathrm{Proj}_W(\rho - \sigma_W)$. Then, if $\langle E, W\rangle \geq 0$, then we also have $\langle E, C\rangle > 0$.*

Proof. Since $\mathrm{Proj}_W(\rho - \sigma_W) = \langle W, \rho - \sigma_W \rangle W$, we have

$$\langle E, C \rangle = \langle E, \rho - \sigma_W \rangle - \langle W, \rho - \sigma_W \rangle \langle E, W \rangle.$$

Since E is an entanglement witness for ρ, and $\sigma_W \in \mathcal{S}_{d_1,d_2}$, we have

$$\langle E, \rho \rangle > \langle E, \sigma_W \rangle \quad \Rightarrow \quad \langle E, \rho - \sigma_W \rangle > 0.$$

Similarly, since W is not an entanglement witness, and σ_W is maximal, we have

$$\langle W, \rho - \sigma_W \rangle \leq 0.$$

Thus, if $\langle E, W \rangle \geq 0$, we have $\langle E, C \rangle > 0$, as desired. □

This means that if we find a Hermitian operator W which is sufficiently close to the set of entanglement witnesses $\mathcal{W}_\rho$ so that $\langle E, W \rangle \geq 0$ for all $E \in \mathcal{W}_\rho$, the we also get a cutting plane $\langle C, x \rangle = 0$ which guarantees that the set $\mathcal{W}_\rho$ lies on the hemisphere given by $\langle C, x \rangle > 0$. At this point, we need to find an initial vector W_1 which satisfies $\langle E, W_1 \rangle \geq 0$ for all $E \in \mathcal{W}_\rho$, and we need to find a procedure for taking all of the information about $\mathcal{W}_\rho$ derived from a set of guesses and producing a new guess W_k which satisfies $\langle E, W_k \rangle \geq 0$ for all $E \in \mathcal{W}_\rho$.

For the initial vector, we will take $W_1 = \rho / \|\rho\|$. By construction, each element $E \in \mathcal{W}_\rho$ is an entanglement witness for ρ, and thus $\langle E, \rho \rangle > \langle E, \sigma \rangle$ for each $\sigma \in \mathcal{S}_{d_1,d_2}$. In particular, $\sigma = \frac{1}{d} I_d$, which acts as the zero of the space of trace 1 Hermitian operators represented in the Gell-Mann basis, yields $\langle E, \sigma \rangle = 0$, giving us $\langle E, \rho \rangle > 0$ for all $E \in \mathcal{W}_\rho$, as desired.

Finally, let us suppose that at some point during the algorithm, we are able to restrict our search space using n relations of the form $\langle C_j, x \rangle \geq \gamma_j$, where each C_j is obtained by use of Lemma 9.2.1. This gives us the search space

$$\mathcal{W}^{(n)} = \mathcal{W} \cap \left(\bigcap_{j=1}^{n} \{x : \langle C_j, x \rangle \geq \gamma_j\} \right).$$

Since each C_j satisfies $\langle E, C_j \rangle > 0$ for each $E \in \mathcal{W}_\rho$, any conic combination

$$\omega = \sum_{j=1}^{n} \lambda_j C_j$$

where $\lambda_j \geq 0$ for each $j = 1, \ldots, n$ also satisfies $\langle E, \omega \rangle \geq 0$. Any such point ω which also lies within $\mathcal{W}^{(n)}$ would be a valid new guess.

It would be ideal to use the volumetric centre of the region $\mathcal{W}^{(n)}$ as the next guess, since it would guarantee that each successive cut to the search region will decrease its volume by one-half. However, in practice, the volumetric centre cannot be computed efficiently in general. Instead, we will use the analytic centre, which is close to the volumetric centre, but is much easier to compute.

Given a set of n constraints $\langle C_j, x\rangle \geq \gamma_j$, along with the constraint $\langle x, x\rangle \leq 1$, which defines the ball $\mathcal{W}$ of potential entanglement witnesses, we define the analytic centre ω of the feasible region $\mathcal{W}^{(n)}$ to be the unique minimizer of the real convex function

$$F(x) = -\log\left(1 - \langle x, x\rangle\right) - \sum_{j=1}^{n} \log\left(\langle C_j, x\rangle - \gamma_j\right),$$

called the logarithmic barrier function, in the region $\mathcal{W}^{(n)}$. Observe that $F(x)$ approaches infinity when x approaches the boundary of $\mathcal{W}^{(n)}$, as some constraint bounding $\mathcal{W}^{(n)}$ will near equality. We may also show that ω is a conic combination of the vectors C_j by solving $\nabla F(\omega) = 0$, which gives us

$$\omega = \frac{1 - \langle \omega, \omega\rangle}{2} \sum_{j=1}^{n} \frac{C_j}{\langle C_j, \omega\rangle - \gamma_j}.$$

Given that ω must satisfy all of the constraints which define $\mathcal{W}^{(n)}$, the coefficients

$$\frac{1 - \langle \omega, \omega\rangle}{2\left(\langle C_j, \omega\rangle - \gamma_j\right)}$$

must be non-negative, meaning that Lemma 9.2.1 can be applied.

9.3 Modified Atkinson-Vaidya Algorithm

Atkinson and Vaidya [AV95] give an algorithm which solves the Weak Membership Problem for a bounded convex set K, given an oracle which takes a given input point x, and either asserts that $x \in K$, within precision, or finds a hyperplane which separates x and K. This algorithm uses the analytic centre of the current bounding constraints, called cutting planes, as a test point at each iteration of the search. In Section 9.2, we established an algorithm which exactly implements such an oracle for the set $\mathcal{W}_\rho$ for a given quantum state expressed as a density matrix $\rho \in \mathcal{H} = \mathcal{H}_1 \otimes \mathcal{H}_2$, with the assumption that the given test point satisfies the preconditions of Lemma 9.2.1. Fortunately, we also established that the analytic centre of the constraints defining $\mathcal{W}^{(n)}$ do indeed satisfy these preconditions. The Atkinson-Vaidya algorithm is efficient with respect to the oracle, in that for a convex set in a d^2-dimensional space, it requires poly$(d^2, \log(1/\delta))$ calls to the oracle, and poly$(d^2, \log(1/\delta))$ arithmetic operations for each iteration associated with an oracle call. The key to achieving this efficient performance is in the management of the cutting planes which bound the search space. Rather than using every cutting plane generated by the oracle, the algorithm occasionally discards cutting planes which are not as useful as the others, in order to place a bound on the total number of cutting planes. Also, instead of using the separating hyperplanes generated by the oracle, which go through the test point x and the origin 0, the algorithm relaxes the cutting plane by shifting it away from the convex set, producing a *shallow cut.*

The shallowness of the cut is determined by a parameter k_0.

However, the Atkinson-Vaidya algorithm, as presented in [AV95], assumes that the region of interest containing the convex set K is bounded within a box, defined using hyperplanes expressed as linear constraints. In our application, the set $\mathcal{W}_\rho$ is bounded by the unit hypersphere in $\mathbb{R}^{d^2-1}$, which gives us one quadratic constraint. Thus, it is necessary to modify the Atkinson-Vaidya algorithm to work with a quadratic constraint.

To initialize the algorithm, recall that the first test point of the algorithm is the vector $W_1 = \rho/\|\rho\|$. This yields the first cutting plane $\langle C_1, x\rangle \geq 0$, where

$$C_1 = (\rho - \sigma_{W_1}) - \mathrm{Proj}_{W_1}(\rho - \sigma_{W_1}),$$

recalling that σ_{W_1} is the value which maximizes the function $\langle W_1, x\rangle$ over the set of separable states $\mathcal{S}_{d_1,d_2}$. We normalize C_1 so that $\|C_1\| = 1$, and set our initial search space

$$\mathcal{W}^{(1)} = \mathcal{W} \cap \{x : \langle C_1, x\rangle \geq 0\}.$$

We can also explicitly find the analytic centre ω of this region as $C_1/\sqrt{3}$. For the purposes of describing this algorithm, we will say that the procedure **Find Cutting Plane** takes a vector W and a precision δ and returns a normalized vector C corresponding to the cutting plane separating W from $\mathcal{W}_\rho$.

Next, recall the definition of the analytic centre ω as the unique value which minimizes the function

$$F(x) = -\log\left(1 - \langle x, x\rangle\right) - \sum_{j=1}^{n} \log\left(\langle C_j, x\rangle - \gamma_j\right)$$

in the region $\mathcal{W}^{(n)}$, or equivalently, the fixed point solution to the equation

$$\omega = \frac{1 - \langle \omega, \omega\rangle}{2} \sum_{j=1}^{n} \frac{C_j}{\langle C_j, \omega\rangle - \gamma_j}.$$

In order to approximate the analytic centre at a given iteration, we use Newton's method, using the previous analytic centre as the initial value. The update step for Newton's method assigns

$$\omega_{new} \leftarrow \omega_{old} - (\nabla^2 F(\omega_{old}))^{-1} \nabla F(\omega_{old}).$$

To determine whether a given approximate analytic centre is sufficiently close to the true analytic centre, we use the function

$$\lambda(x) = \sqrt{\nabla F(x)^T (\nabla^2 F(x))^{-1} \nabla F(x)}.$$

Note that since $\nabla F(\omega) = 0$, we also have $\lambda(\omega) = 0$. Therefore, we will give an upper bound for $\lambda(x)$ to check whether the current Newton iterate is sufficiently close. This upper bound is given as a parameter λ_0 to the algorithm. Also, in order to ensure that we can detect a situation where the feasible region is too small with respect to the precision δ, we continue iterations of Newton's method until

$$1 - \sqrt[3]{1 - 3\lambda(\omega)} \leq \frac{\delta}{5+\delta}\frac{\|\omega\|}{\sqrt{2}}.$$

We will use the name **Update Analytic Centre** to indicate the procedure of updating the analytic centre given an updated set of cutting planes by using Newton's Method.

Finally, we describe how the Atkinson-Vaidya algorithm manages the set of cutting planes. This is done by considering the variational quantities

$$s_j = \frac{\langle C_j, (\nabla^2 F(\omega))^{-1} C_j \rangle}{(\langle C_j, \omega \rangle - \gamma_j)^2}$$

for each cutting plane relation $\langle C_j, x \rangle \geq \gamma_j$, where ω is the current approximate analytic centre. This quantity is related to the distance from the hyperplane $\langle C_j, x \rangle = \gamma_j$ to the *Hessian ellipsoid*,

$$\mathcal{E} = \{x \in \mathbb{R}^{d^2-1} : \langle x - \omega, (\nabla^2 F(\omega))^{-1}(x - \omega) \rangle \leq 1\},$$

which is properly contained in the feasible region $\mathcal{W}^{(n)}$. The smaller s_j is, the farther it is from $\mathcal{E}$. If s_j is smaller than a threshold s_0, given as a parameter to the algorithm, then its effect on the feasible region $\mathcal{W}^{(n)}$ is small, and is discarded. However, checking s_j is a computationally expensive task, and performing this check for each cutting plane at each iteration of the algorithm makes the algorithm inefficient. Atkinson and Vaidya give a simple test which will trigger a check of σ_j for a particular cutting plane $\langle C_j, x \rangle \geq \gamma_j$. For each hyperplane $\langle C_j, x \rangle = \gamma_j$, we associate with it a value κ_j, which is initialized to $\langle C_j, \omega \rangle - \gamma_j$, which is the distance from the current approximate analytic centre to the hyperplane. At each iteration, we check the ratio

$$\mu_j(\omega) = \frac{\langle C_j, \omega \rangle - \gamma_j}{\kappa_j},$$

using the current value of ω for each hyperplane. If the distance between some hyperplane and the analytic centre has more than doubled, that is, if

$$\max_j \mu_j(\omega) > 2,$$

then we check s_j for each hyperplane to see if there is one we can discard. If there is not, we reset the values κ_j which triggered the check.

We now present pseudocode for the whole Modified Atkinson-Vaidya algorithm.

Modified Atkinson-Vaidya Algorithm

```
1. Input: precision $\delta > 0$
2. Parameters: $s_0, k_0, \nu$
3. Initialize:
    • $C_1 \leftarrow$ **Find Cutting Plane** on $\rho$ with precision $\frac{\delta}{5}$
    • Normalize: $C_1 \leftarrow \frac{C_1}{\|C_1\|}$
    • $\omega \leftarrow \frac{C_1}{\sqrt{3}}$
    • $\kappa_1 = \frac{1}{\sqrt{3}}$
4. If $\max_j \mu_j(\omega) > 2$:
    (a) If there is an index $j$ such that $\mu_j(\omega) > 2$ and $s_j(\omega) < s_0$:
        • Discard the hyperplane $\langle C_j, x\rangle \geq \gamma_j$
        • Decrement $n \leftarrow n - 1$ and re-index remaining hyperplanes
        • $\omega \leftarrow$ **Update Analytic Centre**
    (b) Else:
        • For each hyperplane such that $\mu_j(\omega) > 2$: Reset $\kappa_j \leftarrow \langle C_j, \omega\rangle - \gamma_j$
5. Else:
    (a) Increment $n \leftarrow n + 1$
    (b) $C_n \leftarrow$ **Find Cutting Plane** on $\omega$ with precision $\frac{\delta}{5}$
    (c) (If **Find Cutting Plane** returns $E \in \mathcal{W}_\rho$ instead then HALT)
    (d) Find $\gamma_n$ so that
            $$\frac{\langle C_n, (\nabla^2 F(\omega))^{-1} C_n\rangle}{(\langle C_n, \omega\rangle - \gamma)^2} = k_0^2.$$
    (e) Add $\langle C_n, x\rangle \geq \gamma_n$ to the list of cutting planes
    (f) $\omega \leftarrow$ **Update Analytic Centre**
    (g) $\kappa_n \leftarrow \langle C, \omega\rangle - \gamma_n$
6. Stopping Condition: If $n \geq \nu d^2(1/\delta)$ for some parameter $\nu$, (too many hyperplanes), *or* if the volume of an ellipsoid containing the feasible region is too small:
    • Assert: $\rho \in S(\mathcal{S}_{d_1,d_2}, \delta)$ ($\rho$ is within distance $\delta$ of the set of separable states)
    • HALT
```

7. Go to Step 4.

Atkinson and Vaidya show that given proper parameters, this procedure is a polynomial-time algorithm with respect to the oracle **Find Cutting Plane**. Recall that this oracle solves a global optimization problem on the set $\mathcal{S}_{d_1,d_2}$ to precision δ and, in the worst case, checks Ω_δ pure separable states for optimality. This means that the Modified Atkinson-Vaidya Algorithm can solve the Entanglement Witness Problem in

$$\Omega_\delta \times \text{poly}(d^2, \log(1/\delta))$$

steps. We can compare this to the natural algorithm which arises from applying the definition of separability, which runs in

$$\binom{\Omega_\delta}{d^2} \times \text{poly}(d^2, \log(1/\delta)) = O\left(\Omega_d^{d^2}\right)$$

steps. Finally, we can also give an estimate for Ω_δ based on a volumetric argument, where we estimate the volume of the set of pure separable states on the boundary of $\mathcal{S}_{d_1,d_2}$ and divide it by the volume of a hypersphere of diameter δ. This yields

$$\Omega_d = \left(\frac{d}{\delta}\right)^{O(d_1+d_2)},$$

which grows exponentially with the dimension of the Hilbert spaces $\mathcal{H}_1$ and $\mathcal{H}_2$. Thus, the Modified Atkinson-Vaidya algorithm represents an exponential improvement on the order of

$$O\left(\Omega_d^{d^2-1}\right)$$

over the basic search algorithm. Although this still does not yield an efficient algorithm in general, an improvement of this magnitude means that for small, but non-trivial values of d_1 and d_2, this Entanglement Witness Search Algorithm can be of assistance in exploring the properties of separable and entangled states.

Part III

Classical Post-processing for Approximate QFT

Chapter 10

Introduction

Quantum phase estimation is the cornerstone of many of the quantum algorithms that have been discovered to date. In phase estimation, we are given a unitary operator, U, with the ability to perform the controlled-U operation as needed. Then, given an eigenvector of U, we are to estimate the corresponding eigenvalue. The standard algorithm for the quantum phase estimation algorithm makes use of the *Quantum Fourier Transform* (QFT), which is a quantum circuit that performs a Fourier transform on the amplitudes of a given quantum register.

The idea of using an approximate version of the QFT was first introduced by Coppersmith in 1994 [Cop94]. Barenco, *et al.* [BEST96] showed that the logarithmic-depth *Approximate Quantum Fourier Transform* (AQFT) could be used in quantum phase estimation by repeating the estimation procedure polynomially many times, while a more recent analysis by Cheung [Che03] showed that this was unnecessary, as logarithmic-depth AQFT is asymptotically as effective as the full QFT circuit for phase estimation problems. However, there remained the question of how efficient a quantum phase estimation algorithm based on the constant-depth approximate QFT could be.

With QFT-based phase estimation, the final measured result of the quantum algorithm is directly correlated to the desired phase. This is also true with the logarithmic-depth approximate QFT. However with low-depth AQFT, the phase estimation procedure will not naturally yield direct answers, so there must be some amount of classical post-processing involved. In Kitaev's phase estimation algorithm [Kit96], which is equivalent to a regular phase estimation algorithm using a constant-depth approximate QFT, $O(n)$ iterations of the approximate QFT are used to naturally extract the required information, by use of the Chernoff bound.

However, with more involved post-processing, it may be possible to extract the same information with fewer iterations. In Part III of this thesis, we will explore possible techniques to improve the classical post-processing of the results of phase estimation algorithms using low-depth AQFT.

10.1 Summary of Results

We consider in general the problem of retrieving a phase estimate from the final measurement results of a quantum circuit. By constructing the appropriate likelihood function L, which maps possible phase estimates to the probability of obtaining the already-determined final measurement results, we can apply maximum likelihood techniques to obtain good estimates for the original phase.

In the case of n-bit quantum phase estimation, the resulting likelihood function will have at least 2^n individual local maxima, and therefore it is infeasible to search for a maximum likelihood estimate using direct analysis. In the simplest version of the AQFT, where no phase rotations are performed at all, resulting in the Walsh-Hadamard operation. In general, where the AQFT effects a phase rotation of r_k on qubit k, the likelihood function will be of the form

$$L(x) = \prod_{k=1}^{n} \left(\cos^2 \pi(2^{k-1}x - r_k)\right)^{c_k} \left(\sin^2 \pi(2^{k-1} - r_k)\right)^{s_k},$$

where parameters c_k and s_k are determined by the measurement results of the phase estimation circuit. We develop an algorithm which attempts to find a maximum likelihood estimate by bounding the likelihood function by a series of step functions. This algorithm is described in more detail in Chapter 12. Unfortunately, we have not been able to obtain a rigorous analysis of the performance of our algorithm. However, preliminary trials with an implementation of the algorithm suggest that the algorithm does not perform poorly in instances where other efficient algorithms are known. In addition, this classical post-processing algorithm applies to many other situations, and has the potential to perform efficiently in situations where other algorithms are not currently known to work.

Chapter 11 will discuss the QFT and the AQFT in more detail, including previous results related to their use in quantum phase estimation. Chapter 12 will present some techniques which may be used to evaluate the likelihood function $L(x)$, and will include the full description of an algorithm which implements these techniques. Chapter 13 will present the experimental results of one specific implementation of the algorithm discussed in Chapter 12, and finally, Chapter 14 will present some analysis of the running time of various components of the algorithm, and indicate further directions for future research in this area.

Chapter 11

The Approximate Quantum Fourier Transform

The approximate quantum Fourier transform arises naturally from the quantum Fourier transform as a trade-off between accuracy and speed. In the simplest form, as introduced by Coppersmith [Cop94], rotation gates in the QFT that have little effect on the final result can simply be ignored, with the hope that the improvement in the complexity of the circuit offsets the effect of removing these gates. A more complete analysis and further generalizations on the AQFT were explored by Cheung [Che03]. In this chapter, we will introduce quantum phase estimation and the QFT, and develop a number of variations on the idea of an approximate QFT in the context of being a replacement for the full QFT in a quantum phase estimation process.

11.1 Quantum Phase Estimation and QFT

In the quantum eigenvalue estimation problem, we are presented with a unitary operator, U, which acts on an n-qubit register, and an *eigenstate*, $|u\rangle$ of U, which is an n-qubit register containing an eigenvector of U. The unitary operator is given as an arbitrary number of copies of a black-box quantum circuit implementing the controlled-U operation, which maps

$$\text{c-}U : |0\rangle|\psi\rangle \mapsto |0\rangle|\psi\rangle$$

and

$$\text{c-}U : |1\rangle|\psi\rangle \mapsto |1\rangle \otimes (U|\psi\rangle).$$

Given U in this form, and an eigenstate $|u\rangle$, the quantum eigenvalue estimation problem asks for the corresponding eigenvalue λ satisfying $U|u\rangle = \lambda|u\rangle$. Because U is unitary, λ will be of the form $\lambda = e^{2\pi i x}$ for some value $x \in [0, 1)$, and the eigenvalue estimation problem can be solved by estimating x. The eigenvalue estimation

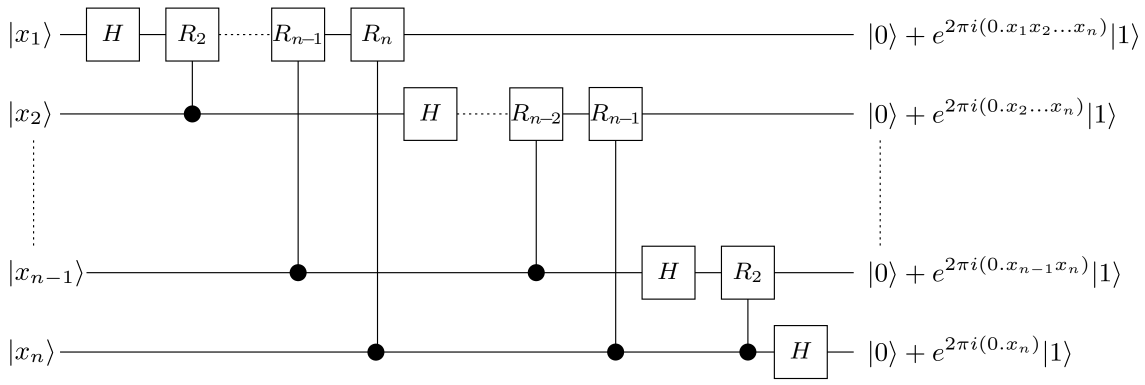

$|x_1\rangle$
$|x_2\rangle$
$|x_{n-1}\rangle$
$|x_n\rangle$
H
R_2
R_{n-1}
R_n
R_{n-2}
$|0\rangle + e^{2\pi i(0.x_1x_2\ldots x_n)}|1\rangle$
$|0\rangle + e^{2\pi i(0.x_2\ldots x_n)}|1\rangle$
$|0\rangle + e^{2\pi i(0.x_{n-1}x_n)}|1\rangle$
$|0\rangle + e^{2\pi i(0.x_n)}|1\rangle$

Figure 11.1: A quantum circuit that performs a quantum Fourier transform

problem, in turn, can be transformed into an instance of the more general quantum phase estimation problem, which is the problem of estimating x given the state

$$\frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} e^{2\pi i k x/2^n} |k\rangle.$$

Quantum phase estimation has numerous applications in terms of quantum algorithms. It is a component in Shor's polynomial-time quantum factoring algorithm, as well as polynomial-time algorithms for solving discrete logarithms. Quantum phase estimation is also the main strategy for solving a class of problems known as the Abelian Hidden Subgroup Problem, in which we are given a Abelian group G and a quantum circuit which has the ability to distinguish different cosets of a fixed subgroup K, and are asked to find a set of generators for K.

The standard strategy for solving the quantum phase estimation problem makes use of the quantum Fourier transform. The QFT is an operation which effectively performs a discrete Fourier transform on the individual amplitudes of a set of basis states. In the specific case of quantum phase estimation, the Fourier transform in question will act on the 2^n computational basis states of an n-qubit register, mapping a given computation basis state $|j\rangle$for $0 \leq j < 2^n$ to

$$\text{QFT}|j\rangle = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} e^{2\pi i j k/2^n} |k\rangle.$$

In order to show that the QFT represents a valid quantum unitary operation, we can simply demonstrate a quantum circuit that performs the QFT. Such a circuit is given in Figure 11.1.

In this circuit, the input register encodes the n-qubit computational basis state $|x\rangle$, setting each individual qubit $|x_k\rangle$ according to the binary expansion:

$$|x\rangle = |x_1 x_2 \ldots x_n\rangle = |x_1\rangle \otimes |x_2\rangle \otimes \cdots \otimes |x_n\rangle.$$

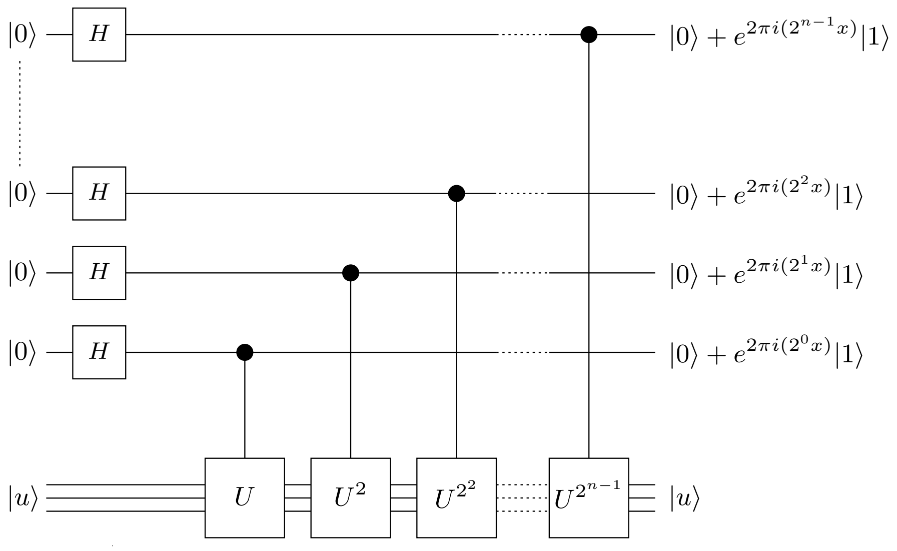

$|0\rangle$
H
$|0\rangle + e^{2\pi i(2^{n-1}x)}|1\rangle$
$|0\rangle + e^{2\pi i(2^{2}x)}|1\rangle$
$|0\rangle + e^{2\pi i(2^{1}x)}|1\rangle$
$|0\rangle + e^{2\pi i(2^{0}x)}|1\rangle$
$|u\rangle$
U
U^2
U^{2^2}
$U^{2^{n-1}}$

Figure 11.2: Solving the eigenvalue estimation problem using phase estimation

The gates labelled H are Hadamard gates, which map $H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, and the gates labelled R_m are phase rotation gates, which map $R_m(\alpha|0\rangle + \beta|1\rangle) = \alpha|0\rangle + e^{2\pi i/2^m}\beta|1\rangle$. Note that in the QFT circuit, each phase rotation R_m is controlled by another qubit, $|x_k\rangle$. This effectively makes R_m a data-dependent phase rotation. That is, given a phase rotation gate R_m controlled by qubit $|x_k\rangle$, the gate effectively maps $R_m(\alpha|0\rangle + \beta|1\rangle) = \alpha|0\rangle + e^{2\pi i x_k/2^m}\beta|1\rangle$. Finally, note that the fractions $0.x_1x_2\ldots$ given in the output states of the QFT are binary expansions.

In order to solve the quantum eigenvalue estimation problem, we construct an instance of the quantum phase estimation problem by applying the circuit given in Figure 11.2. Note that the controlled-U^{2^k} operation can be implemented simply as 2^k applications of the given black-box controlled-U circuit. However, in most useful applications of quantum eigenvalue estimation, efficient implementations of controlled-U^{2^k} are also available.

Now, assume for the time being that in the quantum phase estimation problem, the desired phase λ is of the form $\lambda = e^{2\pi i x}$, where x is an exact binary fraction of the form $x = 0.x_1x_2\ldots x_n$, that is, x is an integer multiple of $1/2^n$. In this case, the qubits that are output by the circuit in Figure 11.2 are

$$|0\rangle + e^{2\pi i(2^{n-k}x)}|1\rangle = |0\rangle + e^{2\pi i(0.x_{n-k+1}\ldots x_n)}|1\rangle.$$

This is precisely the output of the QFT circuit in Figure 11.1 with the order of qubits at the output reversed. In other words, by applying the inverse of the QFT to the output of Figure 11.2, we obtain the state $|x\rangle$ exactly. The inverse QFT is shown in Figure 11.3.

In the general case, where x is no longer restricted to being an exact binary

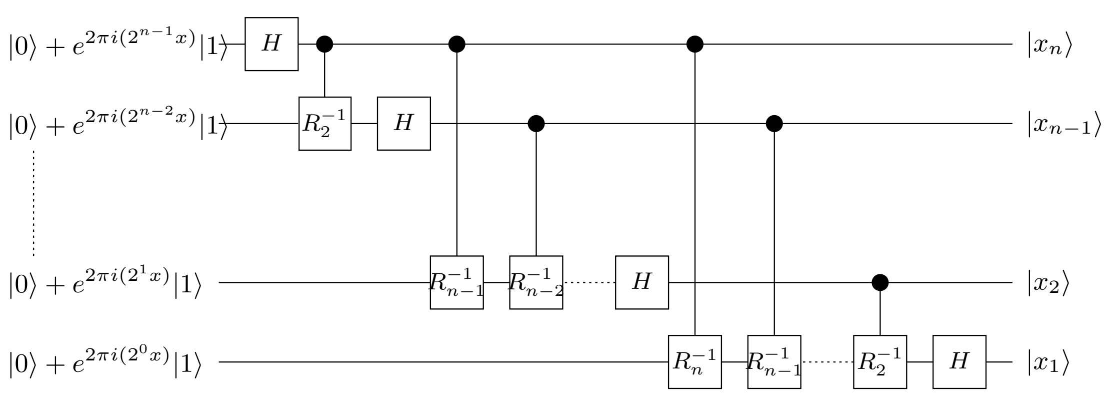

$|0\rangle + e^{2\pi i(2^{n-1}x)}|1\rangle$
$|0\rangle + e^{2\pi i(2^{n-2}x)}|1\rangle$
$|0\rangle + e^{2\pi i(2^{1}x)}|1\rangle$
$|0\rangle + e^{2\pi i(2^{0}x)}|1\rangle$
H
R_2^{-1}
R_{n-1}^{-1}
R_{n-2}^{-1}
R_n^{-1}
$|x_n\rangle$
$|x_{n-1}\rangle$
$|x_2\rangle$
$|x_1\rangle$

Figure 11.3: Solving the quantum phase estimation problem using Inverse QFT

fraction, the phase estimation algorithm no longer gives an exact answer. However, with high probability, the output of the algorithm will be a binary fraction which is close to the value of x. Specifically, it can be shown that given the unitary operator U, the eigenstate $|u\rangle$ and the corresponding eigenvalue $\lambda = e^{2\pi i x}$, where $x \in [0, 1)$, the estimate $\hat{x}$ obtained using the quantum phase estimation algorithm has at least a $\frac{4}{\pi^2} \approx 0.4053$ probability of being the nearest binary fraction estimate, so that $|\hat{x} - x| \leq 2^{-n-1}$. Also, the phase estimation algorithm has at least a $\frac{8}{\pi^2} \approx 0.8106$ probability of being one of the two nearest binary fraction estimates, with $|\hat{x} - x| \leq 2^{-n}$. Obtaining the correct estimate with high probability is a simple matter of repeating the algorithm an appropriate number of times.

11.2 Variations on the QFT

There are a number of variations on the basic concept of the QFT which are potentially useful in either the analysis or practical implementation of quantum phase estimation. One important observation was made by Griffiths and Niu [GN96], who introduced a *semi-classical* version of the inverse quantum Fourier transform.

In order to construct the semi-classical inverse QFT, consider the output register of the inverse QFT in Figure 11.3, which consists of qubits $|x_k\rangle$ for $1 \leq k \leq n$. Observe that in the first half of the phase estimation algorithm, shown in Figure 11.2, the preparation of this qubit in the state $|0\rangle + e^{2\pi i(2^{k-1}x)}|1\rangle$ requires only the eigenstate, and does not require any other qubit in the input register. Furthermore, quantum control of rotation gates followed immediately by measurement in the computation basis can be replaced by measurement followed by classical control. For example, the qubit $|x_n\rangle$ can be measured immediately after the Hadamard operation is applied, and we may then conditionally apply the subsequent rotation gates for the other qubits in the output register based on the result of this measurement. In this way, if we start with the qubit $|x_n\rangle$ and proceed in reverse order down to $|x_1\rangle$, we may construct an alternative quantum phase estimation algorithm composed of circuits as shown in Figure 11.4. In this diagram, R represents the phase

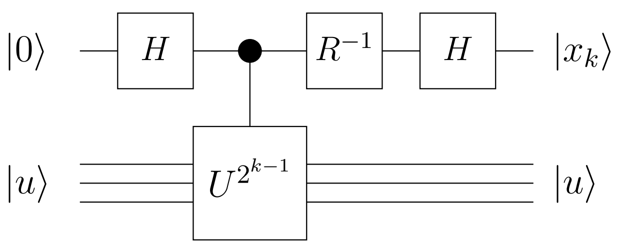
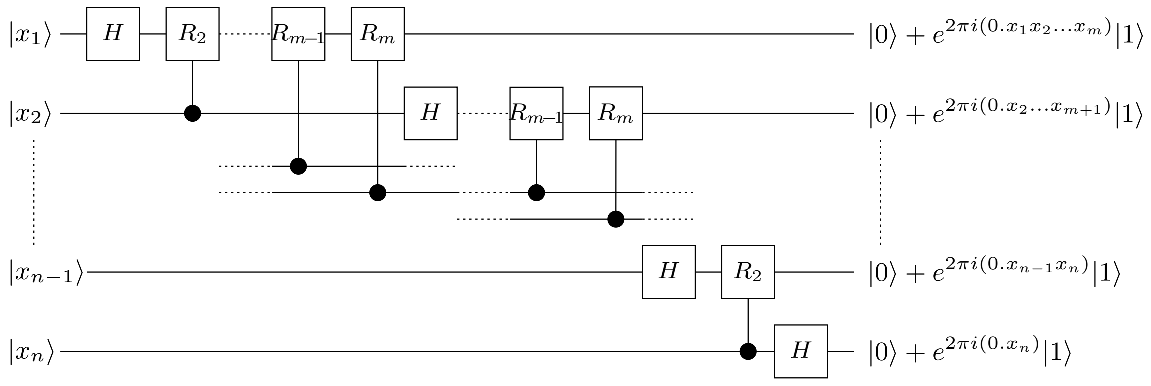


Figure 11.4: Quantum circuit for an individual qubit in phase estimation



$|x_1\rangle$
H
R_2
R_{m-1}
R_m
$|0\rangle + e^{2\pi i(0.x_1x_2\ldots x_m)}|1\rangle$
$|x_2\rangle$
H
R_{m-1}
R_m
$|0\rangle + e^{2\pi i(0.x_2\ldots x_{m+1})}|1\rangle$
$|x_{n-1}\rangle$
H
R_2
$|0\rangle + e^{2\pi i(0.x_{n-1}x_n)}|1\rangle$
$|x_n\rangle$
H
$|0\rangle + e^{2\pi i(0.x_n)}|1\rangle$

Figure 11.5: The Approximate Quantum Fourier Transform AQFT_m

rotation gate which applies a phase shift of $e^{2\pi i(0.0x_{k+1}\cdots x_n)}$. The amount of phase shift is obtained by using the results from the measurements of the higher-order qubits.

11.3 Approximate QFT

The simplest form of approximate QFT is the original scheme by Coppersmith [Cop94], which simply removes some of the controlled rotation gates from the phase estimation algorithm. For an n-qubit register, the quantum phase estimation algorithm using the full QFT requires phase shifts as small as $e^{2\pi i/2^n}$. However these phase rotation gates have only a small effect on the final outcome, and so replacing the QFT with an approximate QFT has the potential to improve the efficiency of the quantum phase estimation algorithm at the cost of losing some accuracy.

We can now define the approximate QFT, parametrized by an integer m, $1 \leq m \leq n$ as the circuit which results from removing any controlled phase rotation gates R_k with $k > m$. A diagram of such a circuit is given in Figure 11.5.

Observe that the circuit AQFT_n is equivalent to the full QFT circuit, as no phase rotation gates are removed. On the other extreme, AQFT_1 is simply the Hadamard transform on an n-qubit register. Since each successive approximate QFT is a poorer approximation, with more phase rotation gates removed, the probability of the output of the phase estimation algorithm using AQFT_m decreases as m decreases. However, it is shown in [Che03] that if $m \geq (\log_2 n) + 2$, the probability P of one iteration of the quantum phase estimation algorithm finding the nearest binary fraction estimate of x, where $\lambda = e^{2\pi i x}$ is the unknown eigenvalue, satisfies

$$P \geq \frac{4}{\pi^2} - \frac{1}{4n} + \frac{m}{4n^2} \geq \frac{4}{\pi^2} - \frac{1}{4n}.$$

In particular, this implies that the difference between phase estimation using the full QFT and phase estimation using a logarithmic-depth AQFT circuit becomes negligible as n becomes large.

11.4 Generalized AQFT

We may also set a semi-classical version of the AQFT-based quantum phase estimation algorithm. In this case, the rotation gate R as represented in Figure 11.4 is adjusted accordingly. Specifically, for qubit $|x_k\rangle$, the rotation gate R applies a phase shift of $e^{2\pi i(0.0x_{k+1}\dots x_{k+m-1})}$, where we define $x_j = 0$ when $j > n$. Alternatively, we may think of this phase shift as an approximation for the phase shift that would be applied at this point by the full QFT-based phase estimation algorithm, which would be $e^{2\pi i(0.0x_{k+1}\dots x_n)}$. However, this phase shift is itself simply an approximation to an ideal phase shift, which is $e^{2\pi i(2^{k-1}x-(0.x_k))}$, where x_k is the correct k-th bit of the nearest binary fraction estimate of x. If we could apply the ideal phase shift in the phase estimation algorithm, we would then obtain the correct answer with probability 1. We can therefore ultimately think of AQFT-based phase estimation as a family of approximations to this ideal process.

If we view the rotation gate in each individual circuit of the semi-classical phase estimation procedure as an estimate for the ideal phase shift amount, then we may generalize the AQFT-based procedure by simply applying a different estimation procedure. In particular, the individual circuits for one particular bit of the final answer x may be performed multiple times in order to refine our estimate of the ideal phase shift amount before proceeding to the next bit. Also, if implementation details allow, the phase shift applied by the rotation gate R can be selected from a continuous range. As long as individual phase shifts obtained for the individual circuits are not worse approximations for the ideal phase shift than those given by the logarithmic-depth AQFT (with $m = \log_2 n + 2$), the same lower-bound result for the success probability applies.

We may also calculate the probability of obtaining the correct bit x_k given the phase shift amount applied. In an individual circuit for computing the bit x_k, we begin by constructing the state $|0\rangle + e^{2\pi i(2^{k-1}x)}|1\rangle$. Note that with the ideal phase shift amount of $e^{2\pi i(2^{k-1}x-0.x_k)}$, applying R^{-1} yields the state $|0\rangle + e^{2\pi i(0.x_k)}|1\rangle$. Applying the Hadamard operation H on this state yields $|x_k\rangle$, which gives us the

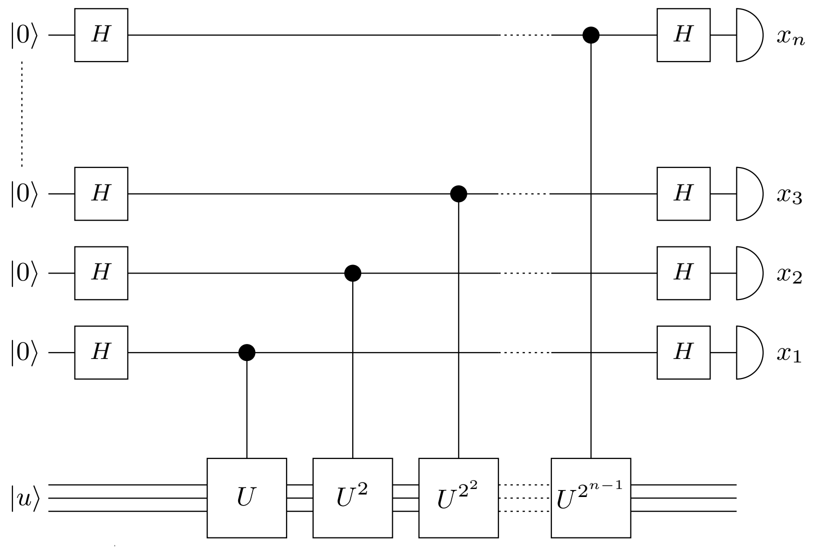


Figure 11.6: One iteration of Kitaev's phase estimation algorithm

correct result x_k with certainty upon measurement in the computational basis. With an approximation to the ideal phase shift amount, applying R^{-1} yields the state $|0\rangle + e^{2\pi i(0.x_k+\delta_k)}|1\rangle$ for some amount δ_k, and the correct result x_k is measured with probability $\cos^2 \pi\delta_k$.

The success probability of the phase estimation algorithm will be the combined product of these individual success probabilities. That is, given the offsets δ_k corresponding to each qubit $|x_k\rangle$ in the phase estimation procedure, the total success probability will be

$$P = \prod_{k=1}^{n} \cos^2 \pi\delta_k.$$

11.5 Low-depth AQFT

A natural question to explore next is lower-depth AQFT, using the circuit AQFT_m with parameter $m < \log_2 n + 2$. In particular, constant-depth AQFT may be of particular interest in addressing concerns regarding physical implementation, as only a fixed set of rotation gates may be available or efficiently constructed.

The immediate problem with low-depth AQFT is that with less accurate phase rotation corrections, the success probability P can become quite low. However, the success probability is calculated under the assumption that the phase estimate x is given directly as the measurement result of the algorithm. Kitaev [Kit96] gives an algorithm which in effect uses the circuit AQFT_1, which has no phase rotation correction. Kitaev's algorithm repeats this procedure, which is illustrated in Figure 11.6, $O(\log n)$ times. Rather than using the measurement results for any particular bit directly, the ratio of the number of times $|0\rangle$ is measured and the number of times $|1\rangle$ is measured is used to estimate the offset δ_k. Since no rotation gates are used for phase correction in this algorithm, these offsets essentially estimate $2^{k-1}x - \lfloor 2^{k-1}x \rfloor$ for $1 \leq k \leq n$. If these estimates are sufficiently consistent with each other, they may be combined into an estimate of x.

In the next chapter, we will explore alternatives to Kitaev's procedure for using low-depth AQFT in quantum phase estimation.

Chapter 12

Classical Post-processing

In this chapter, we will develop algorithmic techniques for working with the likelihood functions which arise from AQFT-based quantum phase estimation. We will also present a complete algorithm for finding local maxima for such functions.

12.1 Maximum Likelihood Estimation

Maximum likelihood estimation is a statistical method for finding the value of a set of parameters from a given set of data generated by those parameters. It is a widely-used method as the estimates produced have good statistical properties. In the context of quantum phase estimation, maximum likelihood estimation is a method for finding likely input parameters to a quantum circuit given a set of measurement data from that circuit.

The *likelihood* of an event causing a given specific observed outcome is simply the probability of the outcome being observed given the occurrence of that event. However, likelihood differs from simple probability in that while probability is concerned with assigning a distribution to the possible future outcomes of one given event, likelihood considers the reverse, where a fixed outcome is already known, and we are concerned with the set of possible events which could have occurred to produce the observed outcome. Given an outcome, the likelihoods of the possible events which could have produced the outcome do not need to sum to 1.

Specifically, given a procedure which depends on an unknown parameter, x, and a set of measurement results from that procedure, we may define a function $L(x)$ which gives the likelihood of any particular value of the parameter x yielding the observed outcome. The maximum likelihood estimate $\hat{x}$ is then defined as the value of x which maximizes the likelihood function $L(x)$. Furthermore, the correct parameter x is likely to be found in a region where $L(x)$ is relatively large compared to the global maximum. Thus, searching for the correct parameter x can be accomplished by searching for parameters with high values of $L(x)$.

Unfortunately, in general, likelihood functions can be difficult to work with analytically. We will develop techniques for handling the likelihood functions which arise from the quantum phase estimation algorithm.

12.2 A Likelihood Function for Phase Estimation

As we observed with Kitaev's phase estimation procedure, even though phase rotation corrections are not being applied before measurement, some information about the phase $2^{k-1}x - \lfloor 2^{k-1}x \rfloor$ can still be obtained with repetition. The maximum likelihood method represents a method of extracting this information with strong asymptotic statistical properties, in that it is optimally unbiased, meaning that asymptotically, no other unbiased estimator gives a lower expected error. However, it does require that the problem of finding input parameters which maximize or nearly maximize the likelihood function be computationally feasible.

We will now derive a formula for the likelihood function $L(x)$ which arises from the quantum phase estimation algorithm. In general, given a qubit in the state $|0\rangle + e^{2\pi i\theta}|1\rangle$, the process of performing a Hadamard gate and a measurement in the computational basis will yield the measurement result $|0\rangle$ with probability $\cos^2 \pi\theta$, and the result $|1\rangle$ with probability $\sin^2 \pi\theta$. If the phase rotation gate in the individual circuit given in Figure 11.4 performs a phase shift of $e^{2\pi i r_k}$ for some constant r_k, then in the individual phase estimation circuit for the bit x_k, the probability of measuring $|0\rangle$ is $\cos^2 \pi(2^{k-1}x - r_k)$ and the probability of measuring $|1\rangle$ is $\sin^2 \pi(2^{k-1}x - r_k)$. We can now compute the likelihood function corresponding to the process of performing this individual circuit a number of times. Suppose that in the course of repeating this measurement, we obtain $|0\rangle$ as the result c_k times, and $|1\rangle$ as the result s_k times. The likelihood function for this measurement result will be the probability of obtaining it, given the input phase x as a variable parameter, which is

$$\binom{c_k + s_k}{c_k} \left(\cos^2 \pi(2^{k-1}x - r_k)\right)^{c_k} \left(\sin^2 \pi(2^{k-1}x - r_k)\right)^{s_k}.$$

As we are only concerned with finding maxima, we may ignore the constant factor. We therefore define the likelihood function for the measurement results for bit x_k as

$$f_k(x) = \left(\cos^2 \pi(2^{k-1}x - r_k)\right)^{c_k} \left(\sin^2 \pi(2^{k-1}x - r_k)\right)^{s_k}.$$

Then the likelihood function for obtaining all of the measurement results for the entire phase estimation procedure will be

$$L(x) = \prod_{k=1}^{n} f_k(x).$$

Applying the maximum likelihood method is now a search for maximum values of the likelihood function $L(x)$. Note that as this function will have at least 2^{n-1}

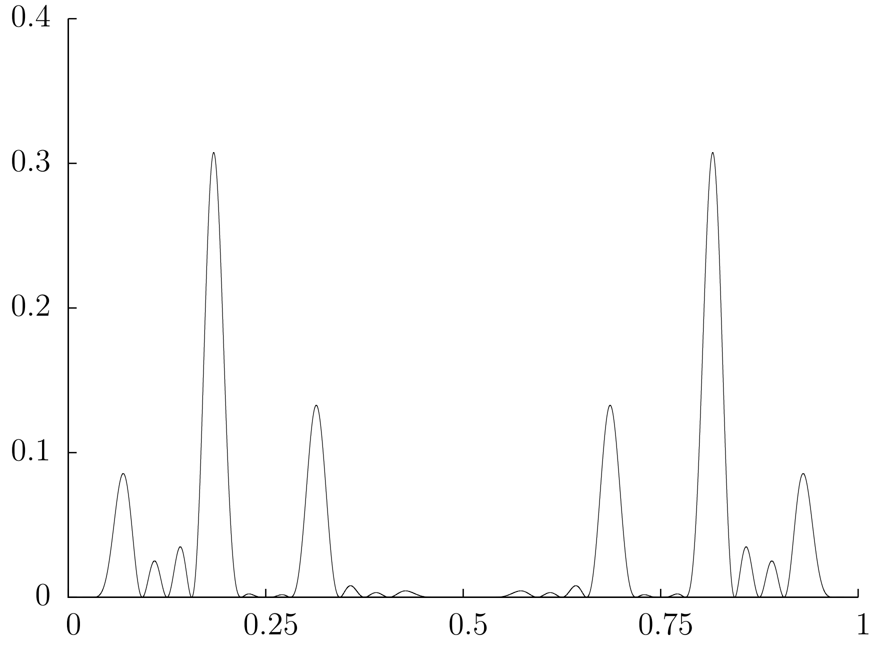


Figure 12.1: A typical likelihood function for phase estimation, with $n = 5$

distinct roots, and therefore as many local maxima, an analytical search using calculus will not be efficient. Since it is likely that with this method, the correct phase will be close to one of the largest maxima of this function, we would like to be able to produce a list of the largest local maxima of this function in descending order. This procedure will work as long as the task of producing the list is not too inefficient, and as long as the correct answer is not too far from the top of the list.

12.3 Algorithmic Approach

The general approach that we will take in order to search for maxima is to construct a step function which bounds the likelihood function $L(x)$ from above, and then to continually refine this bounding function until we find a maximum. A step function can be represented simply as a list of intervals subdividing the range $[0, 1]$, with a value associated with each interval. In our approach, we may refine a bounding step function by selecting the highest step and attempting to refine that upper bound, until we find that the highest step is a tight bound for the function. At that point, the interval corresponding to that step contains the global maximum. After finding the global maximum, we may continue this refinement process to find other local maxima in order of size.

In order to construct these step functions, we introduce the idea of *subfunctions*

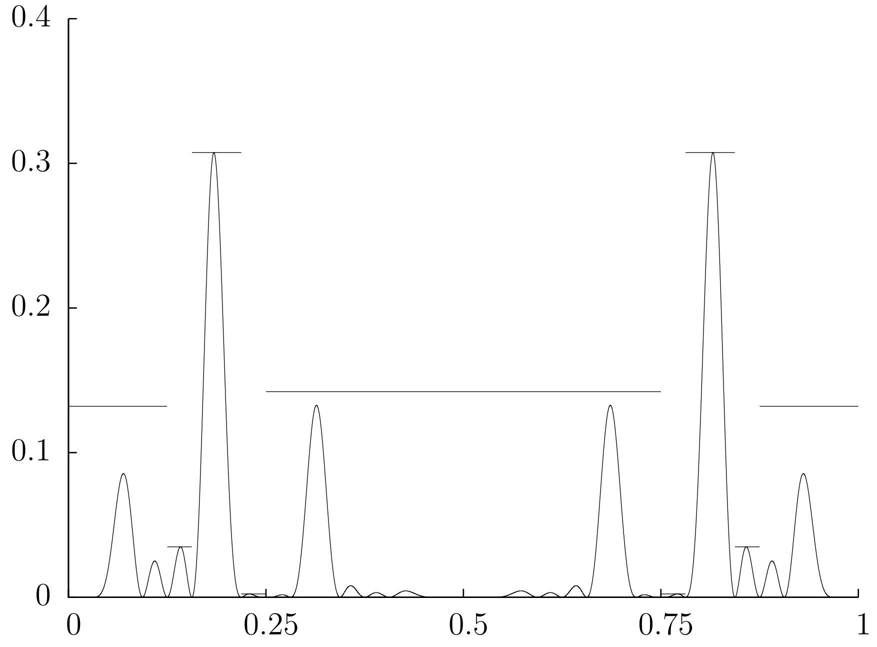


Figure 12.2: A likelihood function $L(x)$ bounded by a step function

of the likelihood function. We define the subfunction $L_j(x)$ as

$$L_j(x) = \prod_{k=1}^{j} f_k(x).$$

Given an interval bounded by two consecutive roots of $L_j(x)$, it is easy to find the single maximum within that interval. We will also define the subfunction $S_j(x)$ as the product of the remainder of the terms,

$$S_j(x) = \prod_{k=j+1}^{n} f_k(x).$$

With the bound on $L_j(x)$ on a given interval, and a global bound for $S_j(x)$, we can form a bound for the likelihood function $L(x) = L_j(x)S_j(x)$ on that interval. In order to refine an estimate, in the form of an interval, and an upper bound, we simply consider the intervals and bounds given by the next indexed subfunction, $L_{j+1}(x)$.

One final observation we can make is that the function $S_j(x)$ is of the same general form as the likelihood functions, $L(x)$. Specifically, $S_j(2^{-j}x)$ could itself be a likelihood function of the same form as $L(x)$. In the refinement procedure where a step in the step function corresponding to the subfunction $L_j(x)$ is replaced by several steps which refine the upper bound on $L(x)$, instead of simply finding a more refined estimate using a different subfunction $L_{j+1}(x)$, we can take a step function which forms an upper bound for $S_j(x)$ and multiply it with the maximum value of $L_j(x)$ on the original interval. This forms a step function which bounds $L(x) = L_j(x)S_j(x)$ on the original interval.

12.4 A Post-processing Algorithm

In our algorithm, we will represent step functions as a list of individual steps, consisting of an interval, and a value. We will store one more piece of information with each step. In the algorithm, each step arises from an upper bound for some subfunction $L_j(x)$. We will store this value j with each step.

Before we begin the post-processing algorithm, we need to find bounds for the subfunctions $S_j(x)$. To do this, we can simply apply the post-processing algorithm to the function $S_j(2^{-j}x)$, as observed Section 12.3. Note that the functions $S_k(2^{-k}x)$ for $j < k \leq n$ form the subfunctions of $S_j(2^{-j}x)$. Thus, as long as we start with the subfunction $S_n(x)$ and work down to $S_1(x)$ as we compute bounds, we will not find a situation where we depend on a bound for a subfunction that we have not yet computed.

We first present this basic interval update procedure, which uses upper bounds that have been derived for the subfunctions $S_j(x)$, but does not use any step function bounds for the subfunctions.

Basic Interval Update Procedure

```
1. Input: a likelihood function $L(x)$, and a step, $(I, p, j)$ in the form of an interval
   $I$, a value $p$, and an index $j$

2. Output: a set $s$, of steps that will replace $(I, p, j)$ in the step function

3. Precondition: Either $I = [a, b]$ where $a$ and $b$ are consecutive roots of $L_j(x)$
   or $I = \mathbb{R}$, $p = 1$, and $L_j(x) = 1$.

4. Precondition: $L(x) \leq p$ for all $x \in I$

5. If $I = \mathbb{R}$:

     • If $L_{j+1}(x)$ has no roots:
         – Return $s = \{(\mathbb{R}, 1, j + 1)\}$ and exit
     • Else let $a$ be any root of $L_{j+1}(x)$
     • Set $b \leftarrow a + 1$

6. Initialize: $s \leftarrow \emptyset$
```

```
7. Let $a = r_1 < r_2 < \ldots < r_t = b$ be the set of roots of $L_{j+1}(x)$ in the interval
   $[a, b]$
8. For each interval $[r_k, r_{k+1}]$ with $1 \leq k < t$:
    • Set $q_1$ to be the maximum value of $L_{j+1}(x)$ on the interval $[r_k, r_{k+1}]$
    • Set $q_2 \leftarrow bound[j+1]$, the upper bound for $S_{j+1}(x)$
    • Add the step $([r_k, r_{k+1}], q_1 q_2, j+1)$ to the set $s$
9. Return: $s$, the set of steps that will replace $(I, p, j)$ in the step function
```

To compute the subfunction bounds, we use the basic interval update procedure, being careful to compute the subfunctions starting with $S_{n-1}(x)$ down to $S_0(x)$, so that upper bounds for $S_j(x)$ required in Step 8 will always be available. Note that the number of times that the basic interval update function is run is a parameter to the entire post-processing algorithm. We will discuss various strategies for setting this parameter in Chapter 14.

Compute Subfunction Bounds

```
1. Initialize: Set $bound[n] \leftarrow 1$
2. For each subfunction $S_j(x)$ with $n > j \geq 0$, starting at $j = n - 1$ down to
   $j = 0$:
    • Initialize: $stepfunction[j] \leftarrow \{(\mathbb{R}, 1, 0)\}$
    • Repeat some fixed number of times:
        – Find $(I, p, k) \in stepfunction[j]$ such that $p$ is maximal
        – If $k = j$ and the bound $S_j(2^{-j}x) \leq p$ on $x \in I$ is tight:
            ∗ Exit loop (since $p$ is a tight bound for all of $S_j(x)$)
        – Run: **Basic Interval Update Function** with input function
          $S_j(2^{-j}x)$ and input step $(I, p, k)$
    • Find $(I, p, k) \in stepfunction[j]$ such that $p$ is maximal
    • Set $bound[j] \leftarrow p$
    • For each $(I, p, k) \in stepfunction[j]$ with $I = [a, b]$:
        – Set $a \leftarrow 2^{-j}a$
        – Set $b \leftarrow 2^{-j}b$
    • Postcondition: $stepfunction[j]$ describes a step function bounding $S_j(x)$
      from above on an interval of size $2^{-j}$
```

    - Optional: Continue to repeat the **Basic Interval Update Function** on a copy of $stepfunction[j]$ to improve the step bounds (possibly including $bound[j]$) without increasing the number of steps in $stepfunction[j]$

3. Postcondition: The two lists $bound[0..n]$ and $stepfunction[0..n-1]$ are initialized

Note that in addition to computing a global bound for each subfunction $S_j(x)$, we also construct a step function bound for each subfunction. We will use these step function bounds in our enhanced interval update procedure below. Note that we have the option to improve the step bounds for a particular subfunction $S_j(x)$ without adding extra steps to that bound. In particular, we can use this strategy to build a step function with better bounds, but without adding extra steps in the enhanced interval update procedure.

Enhanced Interval Update Procedure

1. Input: a step, (I, p, j) in the form of an interval, I, a value p, and an index j
2. Output: a set, s, of steps that will replace (I, p, j) in the step function
3. Precondition: Lists $bound[0..n]$ and $stepfunction[0..n-1]$ have been initialized
4. Precondition: Either $I = [a, b]$ where a and b are consecutive roots of $L_j(x)$ or $I = \mathbb{R}$, $p = 1$, and $L_j(x) = 1$.
5. Precondition: $L(x) \leq p$ for all $x \in I$
6. If $I = \mathbb{R}$:
    - If $L_{j+1}(x)$ has no roots:
        - Return $s = \{(\mathbb{R}, 1, j+1)\}$ and exit
    - Else let a be any root of $L_{j+1}(x)$
    - Set $b \leftarrow a + 1$
7. Initialize: $s \leftarrow \emptyset$
8. Set m to be the maximum value of $L_j(x)$ on the interval I
9. For each interval $(J, q, k) \in stepfunction[j]$:
    - For each $l \in \mathbb{Z}$ such that $I \bigcap \left(\frac{l}{2^j} + J\right) \neq \emptyset$:
        - Add the step $\left(I \bigcap \left(\frac{l}{2^j} + J\right), mq, j+k\right)$ to the set s

```
10. Return: $s$, the set of steps that will replace $(I, p, j)$ in the step function
```

Next, we will describe the final post-processing algorithm.

Classical Post-processing Algorithm

```
1. Input: A likelihood function, $L(x)$ corresponding to measurement results from
   the quantum phase estimation algorithm
2. Run: Compute Subfunction Bounds
3. Initialize: $s \leftarrow \{(\mathbb{R}, 1, 0)\}$
4. Initialize: $maxlist \leftarrow \emptyset$, which will be a list of intervals containing global
   maxima
5. Repeat until $maxlist$ contains the desired interval, or too many intervals:
     • Find $(I, p, j) \in s$ such that $p$ is maximal
         – If $j = n$ and the bound $L(x) \leq p$ is tight on interval $I$:
             ∗ Add $(I, p, j)$ to $maxlist$
         – If $j = n$ and the bound $L(x) \leq p$ is not tight on interval $I$:
             ∗ Set $p$ to be the maximum of $L(x)$ on interval $I$ in the step $(I, p, j)$
               in the set $s$
         – If $j < n$:
             ∗ Run: Enhanced Interval Update Function with input step
               $(I, p, j)$
6. Return: $maxlist$
```

We also have the option in the post-processing algorithm of taking every interval with $j = n$ that we find, rather than waiting for them to emerge in order. This may be preferable in applications where verifying whether or not a given phase is the correct one can be done very quickly.

12.5 Sample Algorithm Execution

In this section, we will walk through a sample run of the algorithm on the likelihood function

$$L(x) = \cos^2(\pi x)\sin^2(2\pi x)\sin^2(2^2\pi x)\sin^2(2^3\pi x)\cos^2(2^4\pi x),$$

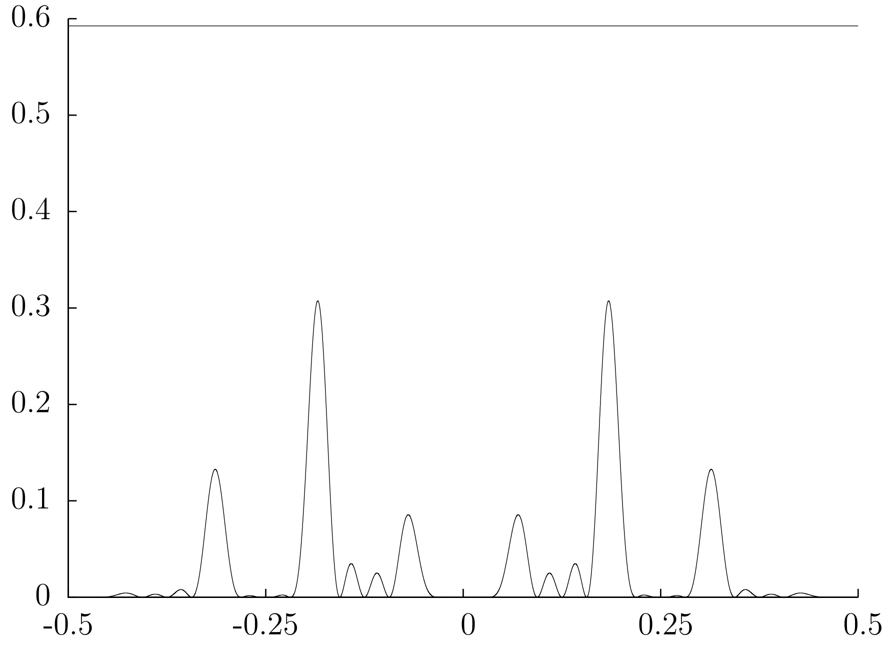


Figure 12.3: A first step function bounding $L(x)$

with $n = 5$. In order to simplify the exposition, we will use the Basic Interval Update Procedure, and we will set the subfunction bounds to the trivial values $bound[j] = 1$, for each j. This will allow us to demonstrate the basic idea of interval updating.

- Initialize: $s \leftarrow \{(\mathbb{R}, 1, 0)\}$ and $maxlist \leftarrow \emptyset$
- $(I, p, j) \leftarrow (\mathbb{R}, 1, 0)$ is maximal in s. We have $j < n$ so we update this interval:
  - $I = \mathbb{R}$, and $L_1(x) = \cos^2(\pi x)$ has a root at $x = -\frac{1}{2}$, so set $[a, b] = [-\frac{1}{2}, \frac{1}{2}]$
  - $-\frac{1}{2}$ and $\frac{1}{2}$ are the only roots of $L_1(x)$ in the interval $[a, b]$, so we consider the interval $[-\frac{1}{2}, \frac{1}{2}]$:
    - $q_1 \leftarrow \max_{[-\frac{1}{2},\frac{1}{2}]} L_1(x) = 1$, and $q_2 = 1$
    - We obtain the step $([-\frac{1}{2}, \frac{1}{2}], 1, 1)$
  - The set of steps replacing $(\mathbb{R}, 1, 0)$ is $\{([-\frac{1}{2}, \frac{1}{2}], 1, 1)\}$
- We replace $(\mathbb{R}, 1, 0)$ in s to get $s \leftarrow \{([-\frac{1}{2}, \frac{1}{2}], 1, 1)\}$
- $(I, p, j) \leftarrow ([-\frac{1}{2}, \frac{1}{2}], 1, 1)$ is maximal in s. We update this interval:
  - Roots of $L_2(x) = \cos^2(\pi x) \sin^2(2\pi x)$ in the interval $[a, b]$ are $\{-\frac{1}{2}, 0, \frac{1}{2}\}$
  - Consider the interval $[-\frac{1}{2}, 0]$:
    - $q_1 \leftarrow \max_{[-\frac{1}{2},0]} L_2(x) = 0.592593$, and $q_2 = 1$

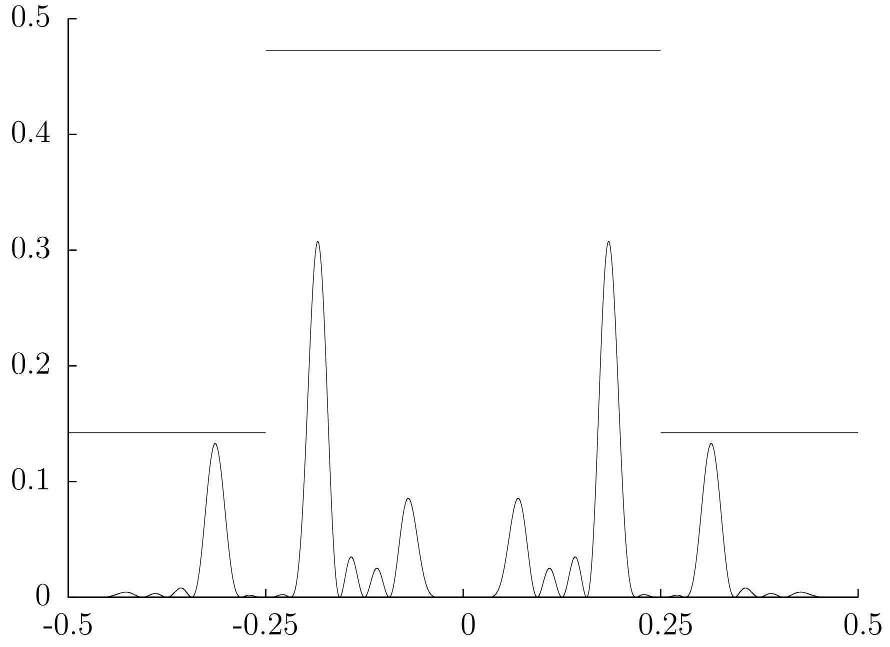


Figure 12.4: A second step function bounding $L(x)$

  - $*$ We obtain the step $([-\frac{1}{2}, 0], 0.592593, 2)$
- $-$ Consider the interval $[0, \frac{1}{2}]$:
  - $*$ $q_1 \leftarrow \max_{[0,\frac{1}{2}]} L_2(x) = 0.592593$, and $q_2 = 1$
  - $*$ We obtain the step $([0, \frac{1}{2}], 0.592593, 2)$

- We replace $([-\frac{1}{2}, \frac{1}{2}], 1, 1)$ in s to get

$$s \leftarrow \{([-\tfrac{1}{2}, 0], 0.592593, 2), ([0, \tfrac{1}{2}], 0.592593, 2)\}$$

(see Figure 12.3)

- $(I, p, j) \leftarrow ([-\frac{1}{2}, 0], 0.592593, 2)$ is maximal in s. We update this interval:
  - $-$ Roots of $L_3(x) = \cos^2(\pi x)\sin^2(2\pi x)\sin^2(2^2\pi x)$ in the interval $[-\frac{1}{2}, 0]$ are $\{-\frac{1}{2}, -\frac{1}{4}, 0\}$
  - $-$ For $[-\frac{1}{2}, -\frac{1}{4}]$, we obtain the step $([-\frac{1}{2}, -\frac{1}{4}], 0.142138, 3)$
  - $-$ For $[-\frac{1}{4}, 0]$, we obtain the step $([-\frac{1}{4}, 0], 0.472388, 3)$

- We update

$$s \leftarrow \{([0, \tfrac{1}{2}], 0.592593, 2), ([-\tfrac{1}{4}, 0], 0.472388, 3), ([-\tfrac{1}{2}, -\tfrac{1}{4}], 0.142138, 3)\}$$

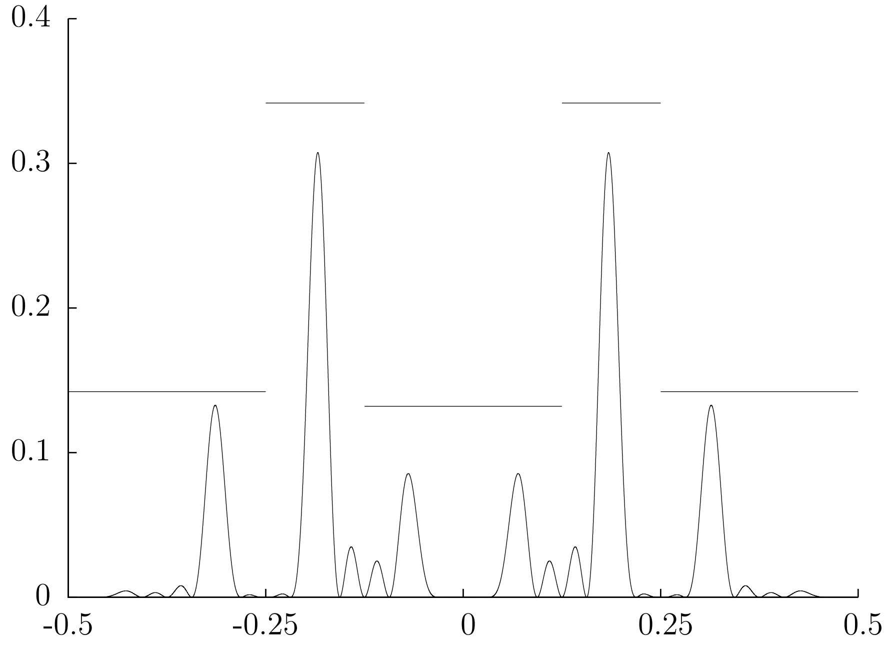


Figure 12.5: A third step function bounding $L(x)$

- We similarly update $([0, \frac{1}{2}], 0.592593, 2)$ to get

$$\begin{aligned} s \;\leftarrow\; & \{([-\tfrac{1}{4}, 0], 0.472388, 3), ([0, \tfrac{1}{4}], 0.472388, 3), \\ & ([-\tfrac{1}{2}, -\tfrac{1}{4}], 0.142138, 3), ([\tfrac{1}{4}, \tfrac{1}{2}], 0.142138, 3)\} \end{aligned}$$

  (see Figure 12.4)

- $(I, p, j) \leftarrow ([-\frac{1}{4}, 0], 0.472388, 3)$ is maximal in s. We update this interval:
  - Roots of $L_4(x) = \cos^2(\pi x)\sin^2(2\pi x)\sin^2(2^2\pi x)\sin^2(2^3\pi x)$ in the interval $[-\frac{1}{4}, 0]$ are $\{-\frac{1}{4}, -\frac{1}{8}, 0\}$
  - For $[-\frac{1}{4}, -\frac{1}{8}]$, we obtain the step $([-\frac{1}{4}, -\frac{1}{8}], 0.341715, 4)$
  - For $[-\frac{1}{8}, 0]$, we obtain the step $([-\frac{1}{8}, 0], 0.132031, 4)$

- We similarly update $([0, \frac{1}{4}], 0.472388, 3)$ to get

$$\begin{aligned} s \;\leftarrow\; & \{([-\tfrac{1}{4}, -\tfrac{1}{8}], 0.341715, 4), ([\tfrac{1}{8}, \tfrac{1}{4}], 0.341715, 4), \\ & ([-\tfrac{1}{2}, -\tfrac{1}{4}], 0.142138, 3), ([\tfrac{1}{4}, \tfrac{1}{2}], 0.142138, 3), \\ & ([-\tfrac{1}{8}, 0], 0.132031, 4), ([0, \tfrac{1}{8}], 0.132031, 4)\} \end{aligned}$$

  (see Figure 12.5)

- $(I, p, j) \leftarrow ([-\frac{1}{4}, -\frac{1}{8}], 0.341715, 4)$ is maximal in s. We update this interval:

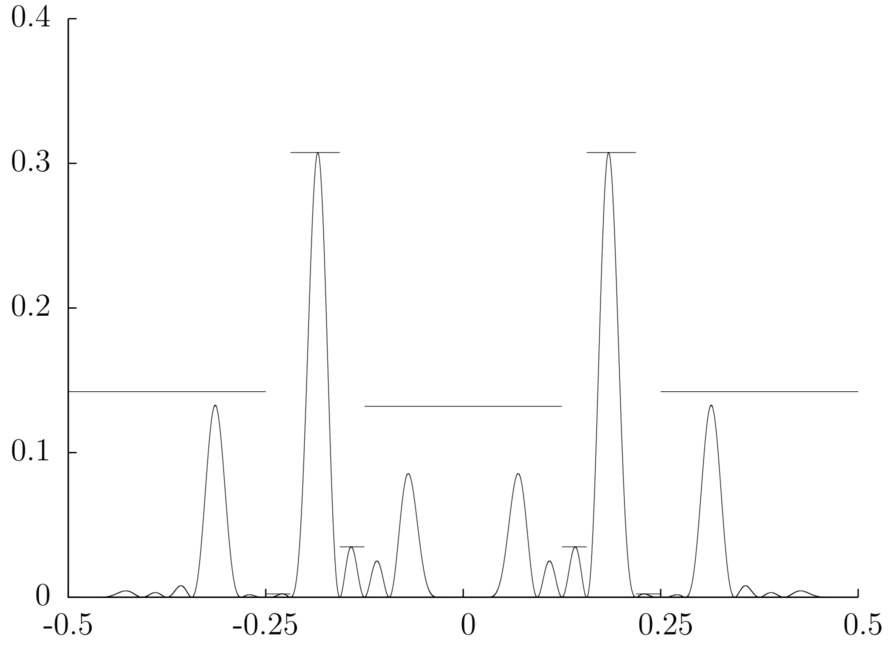


Figure 12.6: A fourth step function bounding $L(x)$

– Roots of $L_5(x) = \cos^2(\pi x)\sin^2(2\pi x)\sin^2(2^2\pi x)\sin^2(2^3\pi x)\cos^2(2^4\pi x)$ in the interval $[-\frac{1}{4}, -\frac{1}{8}]$ are $\{-\frac{1}{4}, -\frac{7}{32}, -\frac{5}{32}, -\frac{1}{8}\}$

– For $[-\frac{1}{4}, -\frac{7}{32}]$, we obtain the step $([-\frac{1}{4}, -\frac{7}{32}], 0.002331, 5)$

– For $[-\frac{7}{32}, -\frac{5}{32}]$, we obtain the step $([-\frac{7}{32}, -\frac{5}{32}], 0.307432, 5)$

– For $[-\frac{5}{32}, -\frac{1}{8}]$, we obtain the step $([-\frac{5}{32}, -\frac{1}{8}], 0.141825, 5)$

- We similarly update $([\frac{1}{8}, \frac{1}{4}], 0.341715, 4)$ to get

$$\begin{aligned} s \quad \leftarrow \quad & \{([-\tfrac{7}{32}, -\tfrac{5}{32}], 0.307432, 5), ([\tfrac{5}{32}, \tfrac{7}{32}], 0.307432, 5), \\ & ([-\tfrac{1}{2}, -\tfrac{1}{4}], 0.142138, 3), ([\tfrac{1}{4}, \tfrac{1}{2}], 0.142138, 3), \\ & ([-\tfrac{5}{32}, -\tfrac{1}{8}], 0.141825, 5), ([\tfrac{1}{8}, \tfrac{5}{32}], 0.141825, 5), \\ & ([-\tfrac{1}{8}, 0], 0.132031, 4), ([0, \tfrac{1}{8}], 0.132031, 4) \\ & ([-\tfrac{1}{4}, -\tfrac{7}{32}], 0.002331, 5), ([\tfrac{7}{32}, \tfrac{1}{4}], 0.002331, 5)\} \end{aligned}$$

(see Figure 12.6)

- $(I, p, j) \leftarrow ([-\frac{7}{32}, -\frac{5}{32}], 0.307432, 5)$ is maximal in s. Since $j = n$, the interval $[-\frac{7}{32}, -\frac{5}{32}]$ contains a maximum

- $(I, p, j) \leftarrow ([\frac{5}{32}, \frac{7}{32}], 0.307432, 5)$ is maximal in s. Since $j = n$, the interval $[\frac{5}{32}, \frac{7}{32}]$ contains a maximum

- The two largest maxima of $L(x)$ can be found in the intervals $[-\frac{7}{32}, -\frac{5}{32}]$ and $[\frac{5}{32}, \frac{7}{32}]$

Chapter 13

Implementation and Experimental Results

In this chapter, we will discuss some of the details involved in implementing the classical post-processing algorithm discussed in Chapter 12. The source code for an implementation of the algorithm is available in Appendix A. We will also present experimental results from trial runs of this implementation.

13.1 Implementation Details

In practice, a major bottleneck in the algorithm is the evaluation of the function $L(x)$ and its subfunctions $L_j(x)$. Indeed, if we are also evaluating derivatives $L_j'(x)$ in order to maximize $L_j(x)$ over an interval, the interval updating procedure can become very slow. To address this, instead of trying to maximize

$$L(x) = \prod_{k=1}^{n} \left(\cos^2 \pi(2^{k-1}x - r_k)\right)^{c_k} \left(\sin^2 \pi(2^{k-1}x - r_k)\right)^{s_k},$$

we instead maximize the function

$$\begin{aligned}\ell(x) &= \log L(x) \\ &= 2\sum_{k=1}^{n} c_k \log\cos\pi(2^{k-1}x - r_k) + s_k \log\sin\pi(2^{k-1}x - r_k),\end{aligned}$$

also known as the log-likelihood function. The function $\ell(x)$ has the derivative

$$\ell'(x) = 2\pi \sum_{k=1}^{n} 2^{k-1} \left(-c_k \tan\pi(2^{k-1}x - r_k) + s_k \cot\pi(2^{k-1}x - r_k)\right),$$

which is much easier to compute than $L'(x)$. Of course, the constant factors may be safely ignored for both maximization and zero-finding.

The step functions themselves are stored in a priority queue implemented on a *binary heap*, which is a data structure which inserts and retrieves in $O(\log n)$ time, where n is the number of the items in the queue.

In order to find local maxima for both the basic and enhanced interval update procedures, we use the algorithm known as *Brent's Method*, as presented in [PTVF92] to find a root of the derivative $\ell'(x)$. Brent's Method attempts to use *inverse quadratic interpolation*, where given three points which lie on the function $y = f(x)$, we use the polynomial $x = (y - r_1)(y - r_2)$ running through these points as an approximation of $y = f(x)$ in order to find the next estimate. Finally, Brent gives an additional condition, described in the pseudocode algorithm below, which much be satisfied in order for the algorithm to accept the interpolated estimate. If the condition is not satisfied, the new estimate is simply the midpoint of the current interval bounding the desired root.

Brent's Method

```
1. Input: $[a_0, b_0]$ containing a root of a continuous function $f(x)$.
2. Precondition: Either $f(a_0) < 0 < f(b_0)$ or $f(b_0) < 0 < f(a_0)$
3. Set $b_{-1} \leftarrow b_0$
4. Set $k \leftarrow 0$
5. Set $bisection \leftarrow TRUE$
6. Repeat until $f(b_k) = 0$ or $|b_k - a_k|$ is sufficiently small (below a preset error tolerance)
    • If $|f(a_k)| < |f(b_k)|$, swap $a_k$ and $b_k$
    • If $f(a_k) \neq f(b_k)$ and $f(a_k) \neq f(b_{k-1})$, set
          $b_{k+1} \leftarrow \frac{a_k f(b_k) f(b_{k-1})}{(f(a_k) - f(b_k))(f(a_k) - f(b_{k-1}))} + \frac{b_k f(a_k) f(b_{k-1})}{(f(b_k) - f(a_k))(f(b_k) - f(b_{k-1}))} + \frac{b_{k-1} f(a_k) f(b_k)}{(f(b_{k-1}) - f(a_k))(f(b_{k-1}) - f(b_k))}$
      (inverse quadratic interpolation)
    • Else, set
          $b_{k+1} \leftarrow \frac{b_k f(a_k) - a_k f(b_k)}{f(b_k) - f(a_k)}$
      (linear interpolation)
```

- **Brent's condition:** If $b_{k+1}$is between $\frac{3a_k+b_k}{4}$ and b_k, and either ($bisection = TRUE$ and $|b_{k+1} - b_k| < |b_k - b_{k-1}|$) or ($bisection = FALSE$ and $|b_{k+1} - b_k| < |b_{k-1} - b_{k-2}|$) then:
    - Accept interpolation: Set $bisection \leftarrow FALSE$
- Else:
    - Perform bisection: Set $b_{k+1} \leftarrow \frac{a_k+b_k}{2}$
    - Set $bisection \leftarrow TRUE$
- Increment counter: Set $k \leftarrow k+1$

7. Return: b_k, which will be within the preset error tolerance to a root of $f(x)$ with $x \in [a_0, b_0]$

13.2 Algorithmic Parameters

There are a number of parameters to the classical post-processing algorithm that we did not elaborate on in Chapter 12. These are the parameters for the step function bounds constructed for the subfunctions $S_j(x)$ in the procedure **Compute Subfunction Bounds**. The two parameters that we may adjust here are the number of steps we want in a step function bound, and the degree of accuracy in that bound. We can express the degree of accuracy in terms of an optional step in the **Compute Subfunction Bounds** procedure, in which we continue to update an fixed additional number of intervals after initially finding the step function bound, and then adjusting the bounds for each individual step of the step function bound based on the updated intervals. Intuitively, a step function which provides a tighter bound on a subfunction $S_j(x)$ should more accurately determine which intervals can be safely ignored, and which require further processing. However, the **Extended Interval Update Procedure** requires that for each interval we wish to update, we potentially add as many steps to our list of intervals as there are in our step function bound for subfunction $S_j(x)$. As a result, having too many steps in a bound for a subfunction may add extra work for the algorithm later. It would also increase the memory requirements of the algorithm.

There is also a question of the amount of floating-point precision required for the algorithm. In terms of finding bounds for the functions, the interval endpoints can be expressed as binary fractions using $n+1$ bits. Within an interval, the bounds are to be used to compare the interval to all other intervals. With $O(2^n)$ potential intervals in the very worst case, we should use $O(n)$ bits of precision to compute this bound. However, we do not need to distinguish each interval bound perfectly if we assume that the correct interval is among the ones with the highest likelihood. In such a case, it would be sufficient to compute the interval bounds to a constant precision. However, the inputs will still require $O(n)$ bits to represent. If the number of interval bounds required is super-polynomial, then the complexity of computing the bounds to $O(n)$ bits of precision would be dominated by other super-polynomial terms.

13.3 Post-processing and Phase Estimation

Recall that in the semi-classical version of the quantum phase estimation procedure, we attempt to estimate each bit starting with the final bit, x_n, working our way back to the first bit, x_1. In the course of actually performing this procedure, we require estimates for the phase rotation correction for each bit that we are attempting to find. In order to find a good rotation correction for each new bit, the post-processing algorithm needs to be performed for the data that has already been measured.

Specifically, suppose that we have already made a collection of measurements for the bits $x_{j+1}, \ldots, x_n$, and we wish to find a rotation correction r_j for the individual circuit for measuring bit x_j. The likelihood function for this partial collection of measurements is simply the subfunction

$$S_j(x) = \prod_{k=j+1}^{n} \left(\cos^2 \pi(2^{k-1}x - r_k)\right)^{c_k} \left(\sin^2 \pi(2^{k-1}x - r_k)\right)^{s_k}.$$

The phase rotation correction r_j is an estimate of the fractional part of $2^j x$, which simply requires applying the post-processing algorithm on the function $S_j(2^{-j}x)$. However, note that in the precomputation phase of the post-processing algorithm on the final likelihood function $L(x)$, the subfunctions $S_j(2^{-j}x)$ are processed starting with $j = n$, working our way down to $j = 1$. Thus, obtaining the corrections r_j during the quantum phase estimation procedure can be done in tandem with computing the subfunction bounds. Furthermore, if the phase rotation correction r_j only needs to be determined up to some error δ, as is the case for the AQFT-based phase estimation procedure, we do not necessarily need to find the global maximum of $S_j(2^{-j}x)$ to determine r_j. It is sufficient to find an interval of size δ which most likely contains the exact phase.

If we are able, we may also add a degree of interaction to the phase estimation procedure. If we find at a certain stage of phase estimation that we obtain measurement data that does not fit well with the hypothesis that the rotation correction r_j for the current bit x_j is close to the ideal phase correction, we may go back and select another likely phase correction from the previous measurement data. Although this will increase the amount of work required for quantum phase estimation, it may provide the best method for finding the correct phase efficiently in the event that we obtain improbable measurement results.

One final consideration for performing phase estimation is the number of samples required for each bit in order to retrieve the correct phase with high probability. In Kitaev's procedure, $O(\log n)$ samples are obtained for each qubit. This quantity is necessary in order to bound the probability that the sampling procedure will produce an incorrect result for any of the n bits, since Kitaev's algorithm involves the "stitching" of these n combined sampling results for each bit into an answer. If any of these results are inconsistent, then this stitching procedure will fail. However, the maximum likelihood method gives us a reasonable strategy for handling inconsistent data, so we may expect that using our classical post-processing method will allow us to use fewer samples. In particular, a constant number of samples for each qubit would produce a constant expected ratio of inconsistent results. From the analysis of Kitaev's algorithm, we know that with a sufficient number of samples, the data will be consistent and combined sampling results also gives the most likely phase estimate. At this point, the performance of our post-processing method depends only on the efficiency of the bounding procedure, since we need only find the most likely phase.

13.4 Experimental Results

The goal of trial runs of the post-processing algorithm on sample data is twofold. First, we would like some indication of how the running time of the algorithm grows as a function on the input size, n. Secondly, we would like to see how the adjustable parameters of the algorithm affect the running time, in order to find strategies for selecting good parameters.

It is important, in constructing sample input data sets for testing the post-processing algorithm, that we generate data that corresponds to the likely output of the quantum phase estimation algorithm. While it is possible to construct pathological data which may reduce the algorithm's ability to find good bounds, such data do not affect the usefulness of the algorithm if it is unlikely that it should correspond to the output of the phase estimation algorithm for any input phase. In particular, there should be some amount of correlation between the measurement statistics (c_k, s_k) for a particular bit x_k and the corresponding statistics for nearby bits, owing to the direct relationship between the phases being sampled.

In order to construct proper data sets, we simulate the quantum phase estimation algorithm for a randomly selected phase by computing the probability distribution for the measurement results, and then sampling from that distribution. In this way, we may ensure that the data generated represent typical data sets which the algorithm may encounter. In a practical application, it is possible that an oracle will be available which allows us to verify a potential solution in this way. However, if such an oracle were not available, we would simply use the most likely solutions as estimates.

In general, the data indicate a strong dependence on the accuracy of the data, as one would expect. If we obtain data which a lower likelihood, given the original phase, then the algorithm can potentially be forced to systematically check a large number of more likely solutions before finding the correct one. In the data tables, we have included both the arithmetic mean and geometric mean of the number of interval updates required to find a phase over several trials of the algorithm with random data. If we expect that the number of interval updates grows exponentially, then the geometric mean is the most appropriate for combining these results. If the growth is linear, then the arithmetic mean is the most appropriate. Since we suspect that the growth maybe somewhere in between, we include both.

Size	Samples	Step Function	Mean (A)	Mean (G)
15	10	30	40.1	21.4
15	100	30	14.6	13.4
15	1000	30	6.1	4.6
15	10000	30	3.6	3.2
20	10	40	49.4	33.5
20	100	40	13.9	10.7
20	1000	40	4.1	3.3
20	10000	40	5.2	3.8
25	10	50	166.3	91.8
25	100	50	18.3	13.7
25	1000	50	8.4	7.2
25	10000	50	5.4	4.5
30	100	60	16.3	15.4
30	1000	60	8.6	6.4
30	10000	60	4.0	3.4

Table 13.1: Average interval updates for various sample sizes

Table 13.1 shows the performance of the post-processing algorithm in relation to the consistency of the data, using step function bounds with $2n$ steps. We can see that with highly consistent data, the algorithm requires very few interval updates, and that as the data becomes less consistent, the number of interval updates required increases. This provides us with some evidence that the algorithm does not perform poorly in cases when other, more efficient algorithms could be applied, meaning that the overhead associated with using maximum likelihood techniques is not excessive.

Table 13.2 shows the performance of the post-processing algorithm in relation to how well we construct a bound for the subfunctions

Size	Samples	Step Function	Mean (A)	Mean (G)
30	10	60	307.2	207.0
30	10	70	500.1	219.8
30	10	80	395.5	231.1
30	10	90	228.8	194.7
30	10	100	242.1	164.8
30	10	120	306.1	169.9
30	10	140	457.3	217.3
30	10	160	235.0	191.8
30	10	180	141.9	121.5
30	10	200	199	146.7

Table 13.2: Average interval updates for various step function parameters

$$S_j(x) = \prod_{k=j+1}^{n} \left(\cos^2 \pi(2^{k-1}x - r_k)\right)^{c_k} \left(\sin^2 \pi(2^{k-1}x - r_k)\right)^{s_k}.$$

The number of samples is fixed as a small constant. The precomputation phase of the post-processing algorithm runs in polynomial time relative to this bound. Note that the performance of the algorithm varies significantly, as we can observe from the arithmetic mean data. Again, this indicates that errors have a fairly large effect on the number of interval updates needed. However, AQFT-based phase estimation reduces to Kitaev's algorithm only in the worst case. In general, we may also consider circuits with low constant depth. These circuits are much more likely to produce consistent data, since an attempt is being made to approximate the ideal phase correction amount, and since inconsistent data is required to make bad approximations to the ideal phase correction amount. Thus, using the classical post-processing algorithm may provide a reasonable solution to the problem of recovering the solution to the phase estimation problem using with a low-depth circuit using only a constant number of samples for each bit.

Chapter 14

Analysis and Discussion

In this chapter, we will present an analysis on the running time of various components of the classical post-processing algorithm. We will also summarize some heuristic and experimental results concerning the number of intervals which the algorithm has to process. This number appears to be difficult to bound properly for several reasons which will also be discussed in this chapter.

We will also discuss the possibility of applying the techniques used in the post-processing algorithm for other problems which reduce to analyzing likelihood functions arising from quantum circuits. In particular, we will look at adapting the subfunction bounding techniques developed in previous chapters to the Dihedral Hidden Subgroup Problem.

14.1 Updating Intervals

We now analyze the asymptotic run time of various components of the post-processing algorithm as a function of the input size, n. The goal of this section is to find the run time of the basic and enhanced interval update procedures.

We begin by looking at the task of evaluating the log-likelihood function $\ell(x)$, and its derivative $\ell(x)$. Recall that

$$\ell(x) = 2\sum_{k=1}^{n} c_k \log\cos\pi(2^{k-1}x - r_k) + s_k \log\sin\pi(2^{k-1}x - r_k)$$

and

$$\ell'(x) = 2\pi\sum_{k=1}^{n} 2^{k-1}\left(-c_k \tan\pi(2^{k-1}x - r_k) + s_k \cot\pi(2^{k-1}x - r_k)\right).$$

We can evaluate functions such as $\log x$, $\sin x$, $\cos x$, $\tan x$ and $\cot x$ to a constant precision by evaluating a fixed Taylor series expansion to a constant precision, which would require constant $O(1)$ time asymptotically. Since both $\ell(x)$ and $\ell'(x)$ require summing n terms together, they can each be evaluated in $O(n)$ time.

Finding the maximum value of $\ell(x)$ on a given interval requires finding the corresponding root of $\ell'(x)$ on that interval, then evaluating $\ell(x)$ at that point. Since we only need to evaluate this value to a fixed precision, the number of times $\ell'(x)$ needs to be evaluated is bounded by a constant.

The final tasks which we need to consider from those performed during the interval update procedure are the removal of the interval being processed from the priority queue storing the step function, and the insertion of the intervals which will replace it. Using a binary heap implementation of a priority queue, both insertions to and deletions from the queue require $O(\log s)$ time, where s is the number of steps in the priority queue. However, since each step in the priority queue is bounded by roots of $L(x)$, there can never be more steps in the queue than there are roots of $L(x)$ in the interval $[0, 1)$, that is, one period of $L(x)$. Since $L(x)$ has $O(2^n)$ roots, we can conclude that insertions to and deletions from the priority queue require $O(n)$ time. While $O(2^n)$ represents the very worst case in the number of steps in the priority queue, we will attempt to give a better sense of how many steps there could actually be in the queue in the next section.

While the Basic Interval Update Procedure requires only a constant-bounded number of insertions to the priority queue, in the Extended Interval Update Procedure, the number of steps we insert into the priority queue is given as the number of steps in our step function bound of the subfunctions $S_j(x)$. This quantity is given as a parameter to the full post-processing algorithm, and it would be preferable if we did not require that this parameter be constant-bounded. Fortunately, we can implement the priority queue using a Fibonacci heap, rather than a binary heap. While this data structure is slightly more complex, insertions into a Fibonacci heap can be accomplished in $O(1)$ amortized time. Deletions still require $O(\log s)$ time, where s is the number of steps in the priority queue. With this implementation of the priority queue, we may construct step function estimates of the subfunctions $S_j(x)$ using $O(n)$ steps, while still guaranteeing that the Extended Interval Update Procedure will not require more than $O(n)$ time in order to insert the new intervals into the priority queue.

We can now finally conclude that updating an interval for a likelihood function $L(x)$ with n terms requires $O(n)$ time, as long as the parameter for the size of the subfunction estimates is bounded by $O(n)$ as well.

14.2 Constructing the Step Function

We will now consider the question of how many interval updates are needed in order to construct a step function bound for $L(x)$ which successfully finds the solution to the phase estimation problem. Unfortunately, the question of bounding this quantity exactly seems to be a difficult one, due to the number of unknown variables involved. However, there are a number of remarks we can make for certain cases.

In the optimal case, there exists at least one value of x for which $L(x) = 1$. This will occur, for example, when the full QFT is used in one iteration of the quantum phase estimation algorithm. When this is the case, the priority queue of intervals to be processed will always produce an interval with bound 1 until each such value x is found. At the very maximum, the number of intervals processed will be the number of such values, multiplied by n. In other words, the number of intervals processed will be $O(n)$.

Using AQFT-based phase estimation, it has been shown in [Che03] that with the logarithmic-depth AQFT on $n \geq 3$ qubits and using AQFT_m with $m \geq \log_2 n + 2$, the estimate constructed by directly using the measurement results as with the QFT is also the maximum likelihood estimate, with $L(x) \geq 1 - \frac{\pi^2}{16n} > \frac{3}{4}$, while $L(x) \leq \frac{3}{4}$ for other estimates. This means that once an interval is bounded by a constant which is at most $\frac{3}{4}$, it will not be processed again before we find the correct estimate. Since the total number of intervals we need to process is directly related to the number of intervals which are processed which do not contain the answer, the algorithm should run efficiently when the number of false intervals is kept to a minimum. The number of such intervals which contain some value x with $L(x) > \frac{3}{4}$ should be fairly small.

In the very worst-case scenario, it is possible that we face the extremely unlikely possibility of our measurement data being exactly negatively correlated to the correct phase parameter. In this case, the desired interval could be the smallest local maximum, requiring that each and every other local maximum of $L(x)$ be checked first. This clearly requires $O(2^n)$ steps to complete. It should be noted that the likelihood associated with the correct phase cannot ever be exactly 0, as it is impossible for such observed data to be produced by this phase.

For a typical case, it would appear that the number of interval updates required depends strongly on the ability to bound false intervals quickly. This, in turn, would depend on the amount of information the measurement results contain about the correct phase. Measurement data containing more information about the hidden phase will provide a larger separation between the value of $L(x)$ corresponding to the correct answer and the likelihoods of most of the other incorrect answers. This makes it more likely for false intervals to be discarded sooner, which keeps the total number of interval updates lower.

14.3 The Dihedral Hidden Subgroup Problem

The *Dihedral Hidden Subgroup Problem* is a problem for which the post-processing techniques may also be applicable. In general, in a Hidden Subgroup Problem (HSP), we are given a group G, which contains a hidden subgroup, $K \leq G$. To retrieve the hidden subgroup, we are given a function $f : G \to \mathbb{Z}$ which has the

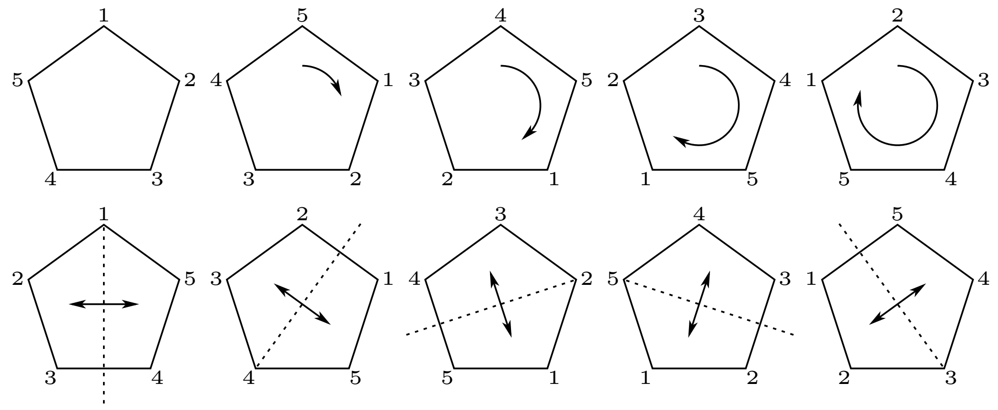


Figure 14.1: The Dihedral Group D_5 as rotations and reflections on a pentagon

property that $f(g_1) = f(g_2)$ if and only if $g_1 \in g_2K$, that is, if g_1 and g_2 are in the same (left) coset of K. We are given the function f in the form of a quantum black-box circuit U_f which takes an input register $|g\rangle|x\rangle$ for some $g \in G$ and ancillary register x and outputs the state $|g\rangle|x \oplus f(g)\rangle$. The Hidden Subgroup Problem for Abelian groups G is solved using quantum phase estimation, and was discussed in Chapter 11.

In the Dihedral Hidden Subgroup Problem, the group G is a dihedral group, D_N, which consists of ordered pairs $(a, b) \in \mathbb{Z}_N \times \mathbb{Z}_2$, and the group operation $(a_1, b_1) \cdot (a_2, b_2) = (a_1 + (-1)^{b_1} a_2, b_1 + b_2)$. In general, for $N \geq 2$, this is a non-commutative operation. The subgroups of D_n consist of the subgroup $Z_N = \{(a, 0) : a \in \mathbb{Z}_N\}$, subgroups of Z_N which are of the form Z_m where $m \mid N$, the dihedral subgroups D_m where $m \mid N$, and finally, subgroups of order 2 of the form $\{(0, 0), (a, 1)\}$, for each $a \in \mathbb{Z}_N$.

In 2004, Kuperberg [Kup03] presented a quantum algorithm which solves the dihedral HSP in sub-exponential time. This was improved upon by Regev [Reg04], who adapted the algorithm to use only polynomial space.

Ettinger and Hyer [EH99] give a quantum polynomial-time algorithm which produces data which contain enough information to find the solution to the dihedral HSP. However, determining the solution of an instance of the dihedral HSP from the output of the algorithm requires the classical post-processing of a function with exponentially many local maxima. We will outline the algorithm given by Ettinger and Hyer, and then propose some possible ideas which may aid with the post-processing of the output data from their algorithm.

In the Ettinger-Hyer algorithm, we begin by considering $K \cap Z_N$ as a subgroup of Z_N. Since Z_N is Abelian, finding $K \cap Z_N$ is simply an application of the Abelian HSP algorithm. We may also use the given black-box function U_f to determine whether the elements $(0, 0)$ and $(0, 1)$ are in the same coset. These two pieces of information are sufficient to distinguish all the subgroups of D_N except for the trivial subgroup $\{(0, 0)\}$ and the order-2 subgroups $\{(0, 0), (a, 1)\}$, where $a \in \mathbb{Z}_N$,

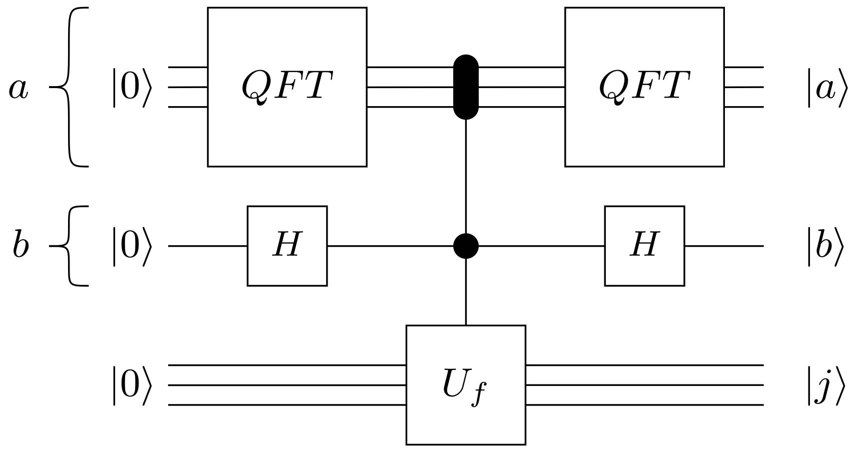


Figure 14.2: Ettinger-Hyer algorithm for the Dihedral HSP problem

and $a \neq 0$. Thus, at this point, we simply need to give an algorithm capable of distinguishing between these possibilities.

We proceed by applying the black-box function U_f to an equal superposition of the elements of D_N, then performing the quantum Fourier transform and Hadamard operation $QFT_N \otimes H$ on the first two registers, respectively.

Using the state $|a\rangle|b\rangle$ to represent the element $(a, b) \in D_N$, we begin with the equal superposition

$$\frac{1}{\sqrt{2N}}\left(\sum_{j=0}^{N-1}|j\rangle\right) \otimes (|0\rangle + |1\rangle) \otimes |00\ldots 0\rangle$$

consisting of each element of D_N, with an ancillary register set to $|00\ldots 0\rangle$. We first need to apply the black-box controlled U_f operation. For a given subgroup $\{(0,0),(a,1)\}$ of D_N, the left cosets will be $\{(j,0),(j+a,1)\}$ for each $j \in Z_N$. We can say that the function f maps each such left coset to $f(j) = f(j,0) = f(j+a,1)$. Note that we also have $f(j,1) = f(j-a)$. Applying U_f for this particular function f yields

$$\begin{aligned}
& \frac{1}{\sqrt{2N}}\sum_{j=0}^{N-1}(|j\rangle|0\rangle|f(j)\rangle + |j\rangle|1\rangle|f(j-a)\rangle) \\
= \; & \frac{1}{\sqrt{2N}}\sum_{j=0}^{N-1}(|j\rangle|0\rangle|f(j)\rangle + |j+a\rangle|1\rangle|f(j)\rangle) \\
= \; & \frac{1}{\sqrt{2N}}\sum_{j=0}^{N-1}(|j\rangle|0\rangle + |j+a\rangle|1\rangle) \otimes |f(j)\rangle.
\end{aligned}$$

If we then apply $QFT_N \otimes H$ to this state, we get

$$\begin{aligned} & \frac{1}{2N}\sum_{j=0}^{N-1}\sum_{k=0}^{N-1}\left(e^{2\pi ijk/N}|k\rangle\otimes(|0\rangle+|1\rangle)+e^{2\pi i(j+a)k/N}|k\rangle\otimes(|0\rangle-|1\rangle)\right)\otimes|f(j)\rangle \\ = & \frac{1}{2N}\sum_{j=0}^{N-1}\sum_{k=0}^{N-1}e^{2\pi ijk/N}\left((1+e^{2\pi iak/N})|k\rangle|0\rangle+(1-e^{2\pi iak/N})|k\rangle|1\rangle\right)\otimes|f(j)\rangle. \end{aligned}$$

Note that the value of the ancillary register $|j\rangle$ at this point in time will not affect the probability distribution of outcomes for the first two registers. Measuring in the computational basis yields $|k\rangle|0\rangle$ with probability

$$\begin{aligned} P_a(k,0) &= \sum_{j=0}^{N-1}\left\|\frac{1}{2N}e^{2\pi ijk/N}(1+e^{2\pi iak/N})\right\|^2 \\ &= N\left\|\frac{1}{2N}(1+e^{2\pi iak/N})\right\|^2 \\ &= \frac{1}{2N}\left(1+\cos(2\pi ak/N)\right) \\ &= \frac{1}{N}\cos^2(\pi ak/N). \end{aligned}$$

Similarly, measurement in the computational basis yields $|k\rangle|1\rangle$ with probability

$$P_a(k,1)=\frac{1}{N}\sin^2(\pi ak/N).$$

Ettinger and Hyer proceed by repeating this procedure $m = O(\log N)$ times and recording the values k whenever a measurement result $|k\rangle|0\rangle$ is obtained. They then consider the function

$$f(x)=\sum_{j=1}^{m}P_x(k_j,0)=\frac{1}{N}\sum_{j=1}^{m}\cos^2(\pi xk_j/N)$$

and the two integer values $\hat{a}$ and $N-\hat{a}$ which maximize $f(x)$ on the range $1 \leq x \leq N-1$. They show that with $O(\log N)$ such samples, then with probability at least $1-\frac{1}{2N}$, the correct value of a is either $\hat{a}$ or $N-\hat{a}$. Finally, to distinguish between the order-2 subgroups $\{(0,0),(\hat{a},1)\}$ and $\{(0,0),(N-\hat{a},1)\}$ and the trivial subgroup $\{(0,0)\}$, we test whether $(\hat{a},1)$ or $(N-\hat{a},1)$ are indeed in the hidden subgroup.

While Ettinger and Hyer show that this procedure produces enough measurement data in order to recover the hidden subgroup, the search for the maximum for the function $f(x)$ still appears to require $O(N)$ steps.

14.4 Subfunction Bounding for Dihedral HSP

In this section, we suggest a method for adapting the subfunction bounding techniques which have been developed in this thesis to the post-processing problem presented by the Ettinger-Hyer method for solving the Dihedral HSP. In Section 14.3, we established that applying the Ettinger-Hyer algorithm for the Dihedral HSP for the group D_N yields the outcome $(k, b) \in \mathbb{Z}_N \times \mathbb{Z}_2$ with probability

$$P_a(k,b) = \frac{1}{N}\cos^2\left(\frac{\pi a k}{N} + \frac{\pi b}{2}\right),$$

when $\{(0,0),(a,1)\}$ is the hidden subgroup. At this point, Ettinger and Hyer consider the function

$$f(x) = \sum_{j=1}^{m} P_x(k_j, 0),$$

taking only the measurement results (k, b) with $b = 0$. However, we may also use the likelihood function generated by this procedure,

$$L(x) = \prod_{j=1}^{m} P_x(k_j, b_j) = \prod_{j=1}^{m} \cos^2\left(\frac{\pi x k_j}{N} + \frac{\pi b_j}{2}\right),$$

ignoring the constant factor of N^{-m}. Then, the two values $\hat{a}$ and $N - \hat{a}$ which maximize the function $L(x)$ on the range $1 \leq x \leq N-1$ are the maximum likelihood estimates for the parameter a which indicates the desired hidden subgroup.

In order to apply the post-processing techniques developed in Chapter 12, we first need to define the subfunctions which are to be bounded. Since the values k_j are sampled randomly from the integers from 0 to $N-1$, we could choose to fix the ordering of these terms so that the values k_j are given in ascending order. We define the subfunctions of $L(x)$ as

$$L_p(x) = \prod_{j=1}^{p} P_x(k_j, b_j) = \prod_{j=1}^{p} \cos^2\left(\frac{\pi x k_j}{N} + \frac{\pi b_j}{2}\right),$$

for $1 \leq p \leq m$, and give our initial interval as $[0, N]$. However, note that the subfunction

$$L_1(x) = P_x(k_1, b_1) = \cos^2\left(\frac{\pi x k_1}{N} + \frac{\pi b_1}{2}\right)$$

will have $k_1 = O(N)$ roots on the interval $[0, N]$, where the local maximum on each of the k_1 subintervals will be 1.

We may remedy this situation by considering the subfunction

$$L_2(x) = P_x(k_1, b_1)P_x(k_2, b_2) = \cos^2\left(\frac{\pi x k_1}{N} + \frac{\pi b_1}{2}\right)\cos^2\left(\frac{\pi x k_2}{N} + \frac{\pi b_2}{2}\right),$$

which has at most $k_1 + k_2$ roots on the interval $[0, N]$. However, the local maxima on the subintervals corresponding to $L_2(x)$ will not all be the same. Furthermore, we can retrieve the subintervals in descending order of the corresponding local maxima by observing that the values x at which local maxima for $P_x(k_1, b_1)$ and $P_x(k_2, b_2)$ occur are exact integer multiples of $\frac{N}{2k_1k_2}$. This means that we can give a modular equation for the local maxima for $P_x(k_1, b_1)$ in terms of multiples of $\frac{N}{2k_1k_2}$, and similarly, a modular equation for $P_x(k_2, b_2)$. Finally, subintervals of $[0, N]$ with higher local maxima for $L_2(x)$ correspond to local maxima of $P_x(k_1, b_1)$ and $P_x(k_2, b_2)$ which are closer together. Given an integer multiple of $\frac{N}{2k_1k_2}$, we may find all pairs of local maxima from $P_x(k_1, b_1)$ and $P_x(k_2, b_2)$ which are at that exact distance apart by using the modular equations for these local maxima, and solving the resulting system of modular equations using the Chinese Remainder Theorem. By going through multiples of $\frac{N}{2k_1k_2}$ in ascending order, we find the corresponding subintervals for $L_2(x)$ in descending order of local maximum. Now, instead of finding all subintervals of $L_2(x)$ first, we may start with only the best subintervals, adding more only when necessary.

Although this strategy represents an improvement over a simple linear search through the interval $[0, N]$, it is not clear that this improvement represents any significant gain. However, it does illustrate a situation in which the subfunction bounding techniques that we have developed can be applied to a more complex post-processing problem.

14.5 Further Directions

Finally, we will outline a number of possible directions for future research and exploration. Of course, it is always desirable to find new applications for the post-processing techniques we have developed. Any quantum algorithm which can be expressed as the statistical sampling of unknown parameters can be analyzed in terms of likelihood. However, without some convenient underlying structure to the likelihood function, the subfunction bounding techniques may not give any advantage over a brute-force analysis.

There is also the question of analyzing the performance of the post-processing algorithm for the Quantum Phase Estimation Problem. The exact relationship between the parameters of the post-processing algorithm and the expected average-case performance of the algorithm is still not known.

Finally, there may be ways to improve the performance of the algorithm itself, possibly by using functions other than step functions to bound the subfunctions. The goal would be to bound the subfunction as well as possible, without increasing the number of times the derivative of either the likelihood function $L'(x)$ or the log likelihood function $\ell'(x)$ need to be evaluated. Specifically, while a step function provides a good bound for $\ell(x)$ near the middle of an interval, this function approaches $-\infty$ at both endpoints of the interval. It is possible that by improving the modelling of this behaviour, we can avoid analyzing the likelihood function over even more regions which are unlikely to contain relatively large local maxima.

Appendix A

Classical Post-processor Implementation

This appendix contains the C++ source code to an implementation of the classical post-processing algorithm described in Part I of this thesis. Multi-precision arithmetic was performed using the *MPFR* library, available at `http://www.mpfr.org/`.

A.1 interval.h

```
// interval.h
//
// Interval class for managing intervals

#ifndef __interval_h__
#define __interval_h__

#include <vector>
#include <functional>

#include <mpfr.h>

using namespace std;

// The Interval class can also maintain a tree hierarchy with intervals
// matched to their respective subintervals

class Interval
{
 public:
  Interval( unsigned int level_in, mp_prec_t range_prec );  // Constructor
  ~Interval();                                              // Destructor

  void refine();      // Improve current bound based on children's bounds
 public:
  mpfr_t left;        // Left endpoint of the interval
  mpfr_t right;       // Right endpoint of the interval
  double bound;       // Bound on the function
  unsigned int level; // Level of the subfunction
  bool tight;         // Set true if the interval bound is known to be tight

  vector<Interval *> child;    // List of children
};

// Interval_lt is a function object comparing two intervals given by pointers
// This function is used by the STL priority queue implementation

struct Interval_lt : public binary_function<Interval *, Interval *, bool>
{
  bool operator() ( Interval *a, Interval *b ) { return a->bound < b->bound; }
};

#endif
```

A.2 interval.cpp

```
// interval.cpp
//
// Implementation of Interval class for managing intervals

#include <vector>

#include <mpfr.h>

#include "interval.h"

using namespace std;

// Interval Constructor
//
// range_prec is the desired precision for the endpoint values

Interval::Interval( unsigned int level_in, mp_prec_t range_prec )
  : level( level_in )
{
  mpfr_init2( left, range_prec );
  mpfr_init2( right, range_prec );
}

// Interval Destructor

Interval::~Interval()
{
  // Delete children (if any)
  vector<Interval *>::iterator it;
  for( it = child.begin(); it != child.end(); it++ ) delete (*it);

  // Delete data

  mpfr_clear( left );
  mpfr_clear( right );
}

// Interval::refine
//
// Improve current interval bound based on children's bounds

void Interval::refine()
{
  if( child.empty() ) return;

  vector<Interval *>::iterator it = child.begin();

  bound = (*it)->bound;

  for( ; it != child.end(); it++ )
  {
    (*it)->refine();
    if( (*it)->bound > bound ) bound = (*it)->bound;
  }
}
```

A.3 function.h

```
// File: function.h
//
// Function class for managing likelihood functions

#ifndef __function_h__
#define __function_h__

#include <vector>
#include <queue>

#include <mpfr.h>

#include "interval.h"

using namespace std;

// Function class

class Function
```

```
{
 public:
  Function();         // Constructor
  ~Function();        // Destructor

  void set_size( const unsigned int n_in );

  void set_data( const unsigned int index, mpfr_t r_in,
                 const unsigned int c_in, const unsigned int s_in );

  void evaluate( mpfr_t rop, mpfr_t op, const int index );
  void eval_deriv( mpfr_t rop, mpfr_t op, const int index );

 protected:
  double compute_subbound( Interval *intptr );
  void update_interval( Interval *intptr );

 protected:
  unsigned int n;     // Number of stages; n = 0 if function uninitialized

  mpfr_t *r;          // Stores the rotation amount for each stage
  mpfr_t *r_pi;       // Rotation amounts multiplied by pi
  unsigned int *c;    // Stores the number of 0 results measured for each stage
  unsigned int *s;    // Stores the number of 1 results measured for each stage

 private:
  mpfr_t pi;          // Stored value of pi
  mpfr_t tol;         // Error tolerance

  // We store temporary multi-precision variables within the class
  // so they need to be initialized only once

  mpfr_t _k, _l, _m, _n;  // Pre-initialized temporary variables (extra prec)
  mpfr_t _u, _v, _w;      // Pre-initialized temporary variables (regular prec)

  mpfr_t _a, _b, _c, _d, _e;  // Extra prec variables for compute_subbound
  mpfr_t _ab;
  mpfr_t _p, _q, _r, _s, _t;
  mpfr_t _f, _fa, _fb, _fc;   // Regular prec variables for compute_subbound
};

// Subfunction class, subclass of Function class, with added functionality
// for handling subfunctions

class Subfunction : public Function
{
 public:
  Subfunction();      // Constructor
  ~Subfunction();     // Destructor

 protected:
  vector<Interval *> intlist;

  double bound;
 private:
  Interval *interval;

  friend class Postprocessor;
};

#endif
```

A.4 function.cpp

```
// File: function.cpp
//
// Implementation of Function class for managing likelihood functions

#include <cassert>
#include <cmath>
#include <iostream>

#include <mpfr.h>

#include "interval.h"
#include "function.h"

using namespace std;

// Function Constructor

Function::Function()
  : n( 0 )
{
}

// Function Destructor

Function::~Function()
{
  if( n == 0 ) return;                    // The function is uninitialized

  for( unsigned int k = 0; k < n; k++ )
  {
    mpfr_clear( r[k] );
    mpfr_clear( r_pi[k] );
  }

  delete[] r;
  delete[] r_pi;
  delete[] c;
  delete[] s;
  mpfr_clear( pi );
  mpfr_clear( tol );

  mpfr_clear( _a );
  mpfr_clear( _b );
  mpfr_clear( _c );
  mpfr_clear( _d );
  mpfr_clear( _e );
  mpfr_clear( _ab );
  mpfr_clear( _p );
  mpfr_clear( _q );
  mpfr_clear( _r );
  mpfr_clear( _s );
  mpfr_clear( _t );
  mpfr_clear( _f );
  mpfr_clear( _fa );
  mpfr_clear( _fb );
  mpfr_clear( _fc );

  mpfr_clear( _k );
  mpfr_clear( _l );
  mpfr_clear( _m );
  mpfr_clear( _n );
  mpfr_clear( _u );
  mpfr_clear( _v );
  mpfr_clear( _w );

  n = 0;
}

// Function::set_size
//
// Sets the size of the function to n, and initializes data

void Function::set_size( const unsigned int n_in )
{
  if( n != 0 ) return;            // This procedure should not be run twice

  n = n_in;

  // Initialize rotation and measurement data to 0

  r = new mpfr_t [n];
  r_pi = new mpfr_t [n];
  c = new unsigned int [n];
  s = new unsigned int [n];

  for( unsigned int k = 0; k < n; k++ )
  {
    mpfr_init2( r[k], n+50 );
    mpfr_set_ui( r[k], 0, GMP_RNDN );
    mpfr_init2( r_pi[k], n+50 );
    mpfr_set_ui( r_pi[k], 0, GMP_RNDN );
    c[k] = s[k] = 0;
  }

  // Initialize constants and temporary variables

  mpfr_init2( pi, n+50 );
  mpfr_const_pi( pi, GMP_RNDN );

  mpfr_init2( tol, n+50 );
  mpfr_set_ui( tol, 1, GMP_RNDN );
  mpfr_div_2ui( tol, tol, mpfr_get_prec( tol ) - 2, GMP_RNDN );

  mpfr_init2( _a, n+50 );
  mpfr_init2( _b, n+50 );
  mpfr_init2( _c, n+50 );
  mpfr_init2( _d, n+50 );
  mpfr_init2( _e, n+50 );
  mpfr_init2( _ab, n+50 );
  mpfr_init2( _p, n+50 );
  mpfr_init2( _q, n+50 );
  mpfr_init2( _r, n+50 );
  mpfr_init2( _s, n+50 );
  mpfr_init2( _t, n+50 );
  mpfr_init2( _f, 75 );
  mpfr_init2( _fa, 75 );
  mpfr_init2( _fb, 75 );
  mpfr_init2( _fc, 75 );

  mpfr_init2( _k, n+50 );
  mpfr_init2( _l, n+50 );
  mpfr_init2( _m, n+50 );
  mpfr_init2( _n, n+50 );
  mpfr_init2( _u, 75 );
  mpfr_init2( _v, 75 );
  mpfr_init2( _w, 75 );
}

// Function::set_data
//
// Sets rotation and measurement data corresponding to a given index

void Function::set_data( const unsigned int index, mpfr_t r_in,
                         const unsigned int c_in,
                         const unsigned int s_in )
{
  if( index < n )     // Range check
  {
    mpfr_set( r[index], r_in, GMP_RNDN );
    mpfr_mul( r_pi[index], r_in, pi, GMP_RNDN );
    c[index] = c_in;
    s[index] = s_in;
  }
}

// Function::evaluate
```

```
//
// Evaluates the function up to index terms, at op, and returns answer in rop

void Function::evaluate( mpfr_t rop, mpfr_t op, const int index )
{
  mpfr_set_ui( rop, 0, GMP_RNDN );

  mpfr_mul( _n, op, pi, GMP_RNDN );           // invariant: _n = 2^k x * pi

  for( int k = 0; k < index; k++ )
  {
    mpfr_sub( _m, _n, r_pi[k], GMP_RNDN );    // _m = pi*( 2^k x - r_k )

    // double _n for next iteration
    mpfr_mul_2ui( _n, _n, 1, GMP_RNDN );

    if( c[k] || s[k] )    // if either c[k] or s[k] are non-zero
    {
      mpfr_sin_cos( _u, _v, _m, GMP_RNDN );

      if( s[k] )
      {
        mpfr_abs( _u, _u, GMP_RNDN );
        mpfr_log( _u, _u, GMP_RNDN );
        mpfr_mul_ui( _u, _u, s[k], GMP_RNDN );
        mpfr_add( rop, rop, _u, GMP_RNDN );   // _u = s[k] * log | sin (_m) |
      }

      if( c[k] )
      {
        mpfr_abs( _v, _v, GMP_RNDN );
        mpfr_log( _v, _v, GMP_RNDN );
        mpfr_mul_ui( _v, _v, c[k], GMP_RNDN );
        mpfr_add( rop, rop, _v, GMP_RNDN );   // _v = c[k] * log | cos (_m) |
      }
    }
  }
}

// Function::eval_deriv
//
// Evaluates the derivative of the function up to index terms

void Function::eval_deriv( mpfr_t rop, mpfr_t op, const int index )
{
  mpfr_set_ui( rop, 0, GMP_RNDN );

  mpfr_mul( _n, op, pi, GMP_RNDN );           // invariant: _n = 2^k x * pi

  for( int k = 0; k < index; k++ )
  {
    mpfr_sub( _m, _n, r_pi[k], GMP_RNDN );    // _m = pi*( 2^k x - r_k )

    // double _n for next iteration
    mpfr_mul_2ui( _n, _n, 1, GMP_RNDN );

    if( c[k] || s[k] )    // if either c[k] or s[k] are non-zero
    {
      mpfr_sin_cos( _u, _v, _m, GMP_RNDN );

      if( s[k] )
      {
        mpfr_div( _w, _v, _u, GMP_RNDN );
        mpfr_mul_ui( _w, _w, s[k], GMP_RNDN );
        mpfr_mul_2ui( _w, _w, k, GMP_RNDN );
        mpfr_add( rop, rop, _w, GMP_RNDN );   // _w = 2^k * s[k] * cot (_m)
      }

      if( c[k] )
      {
        mpfr_div( _w, _u, _v, GMP_RNDN );
        mpfr_mul_ui( _w, _w, c[k], GMP_RNDN );
        mpfr_mul_2ui( _w, _w, k, GMP_RNDN );
        mpfr_sub( rop, rop, _w, GMP_RNDN );   // _w = 2^k * c[k] * tan (_m)
      }
    }
  }
}

// Function::compute_subbound
//
// Computes an interval's subfunction bound, as the maximum value of the
// subfunction (indicated by level) on the interval.  At the bottommost level
// this just computes the function bound
//
// The interval is assumed to be part of the function's interval tree
//
// We assume that the endpoints of the interval are two consecutive roots
//
// We currently use Brent's method for finding the zero of the derivative
//
// compute_subbound must not use the same temporary variables as eval_deriv

double Function::compute_subbound( Interval *intptr )
{
  assert( intptr != NULL );

  mpfr_set( _a, intptr->left, GMP_RNDU );
  mpfr_set( _b, intptr->right, GMP_RNDD );
  mpfr_set_inf( _fa, 1 );
  mpfr_set_inf( _fb, -1 );

  if( mpfr_nan_p( _a ) ) // The interval is R, so value should be 0
    return 0.0;

  int index = intptr->level;
  bool interpolate;
  int i = 0;
```

```
// _b is the estimate
// _a is the counterpoint
// _c is the previous estimate
// _e is used in Brent's condition, and _d is used to update _e

mpfr_set( _c, _a, GMP_RNDN );
mpfr_set( _fc, _fa, GMP_RNDN );

mpfr_sub( _d, _b, _a, GMP_RNDN );
mpfr_set( _e, _d, GMP_RNDN );

do {
  // Swap _a and _b if _a seems to be a better guess

  if( mpfr_cmpabs( _fb, _fa ) > 0 )
  {
    mpfr_swap( _a, _b );
    mpfr_swap( _fa, _fb );
  }

  mpfr_sub( _ab, _a, _b, GMP_RNDN );

  interpolate = !( mpfr_inf_p( _fa ) || mpfr_inf_p( _fb ) );

  if( interpolate )
  {
    // Attempt inverse quadratic interpolation

    // We compute in a way that does not subtract evaluations of the function
    // as this may be unstable

    mpfr_div( _s, _fb, _fa, GMP_RNDN );

    if( mpfr_equal_p( _a, _c ) || mpfr_equal_p( _b, _c ) )
    {
      mpfr_mul( _p, _ab, _s, GMP_RNDN );  // Linear Interpolation
      mpfr_sub_ui( _q, _s, 1, GMP_RNDN );
    } else {
      mpfr_div( _r, _fb, _fc, GMP_RNDN );
      mpfr_div( _t, _fa, _fc, GMP_RNDN );

      mpfr_sub( _p, _c, _b, GMP_RNDN );
      mpfr_sub( _q, _r, _t, GMP_RNDN );
      mpfr_mul( _p, _p, _q, GMP_RNDN );
      mpfr_mul( _p, _p, _t, GMP_RNDN );
      mpfr_sub_ui( _r, _r, 1, GMP_RNDN );
      mpfr_mul( _q, _ab, _r, GMP_RNDN );
      mpfr_sub( _p, _p, _q, GMP_RNDN );
      mpfr_mul( _p, _p, _s, GMP_RNDN );

      mpfr_sub_ui( _s, _s, 1, GMP_RNDN );
      mpfr_sub_ui( _t, _t, 1, GMP_RNDN );
      mpfr_mul( _q, _r, _s, GMP_RNDN );
      mpfr_mul( _q, _q, _t, GMP_RNDN );
    }

    if( mpfr_zero_p( _q ) ) interpolate = false;
    else
    {
      mpfr_div( _p, _p, _q, GMP_RNDN );

      if( mpfr_cmpabs( _p, tol ) < 0 )
      {
        mpfr_set( _q, _b, GMP_RNDN );
        break;
      }

      // The interpolated estimate is _b + _p

      mpfr_mul_ui( _q, _ab, 3, GMP_RNDN );
      mpfr_div_2ui( _q, _q, 2, GMP_RNDN );    // _q = 3(_a-_b)/4
    }
  }

  // Check Brent's condition for accepting interpolated estimate

  if( interpolate
      && ( mpfr_sgn( _p ) == mpfr_sgn( _q ) )
      && ( mpfr_cmpabs( _q, _p ) >= 0 )
      && ( mpfr_cmpabs( _e, _p ) >= 0 ) )
  {
    mpfr_set( _e, _d, GMP_RNDN );
    mpfr_set( _d, _p, GMP_RNDN );
  } else {                                 // Take bisection step
    mpfr_div_2ui( _d, _ab, 1, GMP_RNDN );
    mpfr_set( _e, _d, GMP_RNDN );
  }

  mpfr_set( _c, _b, GMP_RNDN );            // Current guess becomes previous
  mpfr_set( _fc, _fb, GMP_RNDN );

  mpfr_add( _q, _b, _d, GMP_RNDN );

  eval_deriv( _f, _q, index );
  if( mpfr_zero_p( _f ) ) break;

  if( mpfr_sgn( _f ) == mpfr_sgn( _fa ) )  // Update bounds
  {
    mpfr_set( _a, _q, GMP_RNDN );
    mpfr_set( _fa, _f, GMP_RNDN );
  } else {
    mpfr_set( _b, _q, GMP_RNDN );
    mpfr_set( _fb, _f, GMP_RNDN );
  }

  i++;
  } while( mpfr_cmpabs( _ab, tol ) > 0 );

  evaluate( _f, _q, index );
  return mpfr_get_d( _f, GMP_RNDN );
}

// Function::update_interval
//
// Expands one single interval in the interval tree
// Function and subfunction bounds are not computed

void Function::update_interval( Interval *intptr )
{
  assert( intptr != NULL );

  unsigned int level = intptr->level;

  if( level == n ) return;                    // Last level

  if( !intptr->child.empty() ) return;        // Already expanded

  Interval *newintptr;

  if( c[level] == 0 && s[level] == 0 )        // No data for current level
  {
    newintptr = new Interval( level+1, n+50 );
    mpfr_set( newintptr->left, intptr->left, GMP_RNDN );
    mpfr_set( newintptr->right, intptr->right, GMP_RNDN );

    intptr->child.push_back( newintptr );

    return;
  }

  // Compute constants which will help us subdivide interval

  unsigned int k = level+1;

  mpfr_set( _m, r[level], GMP_RNDN );

  mpfr_set_ui( _n, 1, GMP_RNDN );
  mpfr_div_2ui( _n, _n, k, GMP_RNDN );        // _n = 1/2^(level+1)

  if( s[level] == 0 )
  {
    mpfr_add( _m, _m, _n, GMP_RNDN );         // _m = r[level] + 1/2^(level+1)

    mpfr_mul_2ui( _n, _n, 1, GMP_RNDN );
    k--;
  }

  if( c[level] == 0 )
  {
    mpfr_mul_2ui( _n, _n, 1, GMP_RNDN );
    k--;
  }

  // _m is guaranteed to be a root of term level+1
  // _n = 1/2^k is the distance between consecutive roots

  if( mpfr_number_p( intptr->left ) )
  {
    mpfr_set( _k, intptr->left, GMP_RNDN );
    mpfr_set( _l, intptr->right, GMP_RNDN );
  } else {                                    // Interval is R
    mpfr_set( _k, _m, GMP_RNDN );
    mpfr_add_ui( _l, _m, 1, GMP_RNDN );
  }

  // [_k,_l] is the interval to be subdivided

  mpfr_sub( _w, _k, _m, GMP_RNDN );
  mpfr_mul_2ui( _w, _w, k, GMP_RNDN );
  mpfr_floor( _w, _w );
  mpfr_div_2ui( _w, _w, k, GMP_RNDN );
  mpfr_add( _m, _m, _w, GMP_RNDN );

  // _m is the largest root <= intptr->left

  bool finished = false;

  // Subdivide the interval

  while( !finished )
  {
    mpfr_add( _m, _m, _n, GMP_RNDN );

    if( mpfr_cmp( _m, _l ) >= 0 )    // last subinterval
    {
      mpfr_set( _m, _l, GMP_RNDN );
      finished = true;
    }

    newintptr = new Interval( level+1, n+50 );
    mpfr_set( newintptr->left, _k, GMP_RNDN );
    mpfr_set( newintptr->right, _m, GMP_RNDN );

    intptr->child.push_back( newintptr );

    mpfr_set( _k, _m, GMP_RNDN );
  }
}

// Subfunction Constructor

Subfunction::Subfunction()
  : interval( NULL )
{
}

// Subfunction Destructor

Subfunction::~Subfunction()
{
  if( interval != NULL ) delete interval;
}
```

A.5 postproc.h

```
// File: postproc.h
//
// Postprocessor class for post-processing algorithm

#ifndef __postproc_h__
#define __postproc_h__

#include <queue>

#include <mpfr.h>

#include "function.h"

using namespace std;

class Postprocessor : public Subfunction
{
 public:
  Postprocessor();          // Constructor
  ~Postprocessor();         // Destructor

  void set_estimate_size( unsigned int subsize_in, unsigned int subdepth_in );
  void build_subfunctions();
  void build_subfunction( unsigned int j );

  Interval *find_maximum();

 public:
  unsigned int intcount;    // Number of intervals processed

 private:
  void initialize();

 private:
  Subfunction *subfunction;
  double *subbound;

  unsigned int subsize;     // Number of steps in subfunction estimate
  unsigned int subdepth;    // Intervals used for subfunction estimate
  priority_queue<Interval *,vector<Interval *>,Interval_lt> pqueue;

  mpfr_t _a, _b, _c, _d, _e, _f;  // Extra precision temporary variables
};

#endif
```

A.6 postproc.cpp

```
// File: postproc.cpp
//
// Implementation of Postprocessor class for post-processing algorithm

#include <queue>
#include <vector>

#include <mpfr.h>

#include "interval.h"
#include "function.h"
#include "postproc.h"

using namespace std;

// Postprocessor Constructor

Postprocessor::Postprocessor()
  : subfunction( NULL ), subsize( 2 )
{
}

// Postprocessor Destructor

Postprocessor::~Postprocessor()
{
  if( subfunction == NULL ) return;      // This postprocessor was not used

  delete[] subfunction;
  delete[] subbound;

  mpfr_clear( _a );
  mpfr_clear( _b );
  mpfr_clear( _c );
  mpfr_clear( _d );
  mpfr_clear( _e );
  mpfr_clear( _f );
}

// Postprocessor::set_estimate_size
```

```
//
// Sets the two parameters for computing subfunction estimates
// subsize_in gives the number of steps to use in the estimate
// subdepth_in gives the number of intervals to process for the estimate

void Postprocessor::set_estimate_size( unsigned int subsize_in,
                                       unsigned int subdepth_in )
{
  subsize = subsize_in;
  subdepth = subdepth_in;

  if( subsize < 2 ) subsize = 2;                    // Minimum size
  if( subdepth < subsize ) subdepth = subsize;
}

// Postprocessor::initialize
//
// Initialize subfunctions and priority queue
// subfunction != NULL if and only if initialize has been called

void Postprocessor::initialize()
{
  if( n == 0 ) return;                      // Function data not initialized

  if( subfunction != NULL ) return;         // Computation already performed

  // Initialize subfunctions and temporary variables

  subfunction = new Subfunction[n];
  subbound = new double[n+1];

  mpfr_init2( _a, n+50 );
  mpfr_init2( _b, n+50 );
  mpfr_init2( _c, n+50 );
  mpfr_init2( _d, n+50 );
  mpfr_init2( _e, n+50 );
  mpfr_init2( _f, n+50 );

  for( unsigned int j = 0; j < n; j++ )
    subfunction[j].set_size( n-j );

  subbound[n] = 0;

  // Initialize the priority queue

  Interval *intptr = new Interval( 0, n+50 );
  intptr->bound = 0;
  pqueue.push( intptr );
  intcount = 0;
}

// Postprocessor::build_subfunctions
//
// Builds step function estimates for all subfunctions
```

```
void Postprocessor::build_subfunctions()
{
  if( n == 0 ) return;                      // Function data not initialized

  if( subfunction == NULL ) initialize();   // Subfunctions not initialized

  for( int j = n-1; j >= 0; j-- )
    build_subfunction( j );
}

// Postprocessor::build_subfunction
//
// Builds step function estimates for a single subfunction
// The subfunction's interval pointer is NULL if subfunction is uninitialized

void Postprocessor::build_subfunction( unsigned int j )
{
  if( n == 0 ) return;                      // Function data not initialized

  if( subfunction == NULL ) initialize();   // Subfunctions not initialized

  if( subfunction[j].interval != NULL ) return;
                                            // Computation already performed
  Interval *intptr;
  vector<Interval *>::iterator it;

  for( unsigned int i = 0; i < n-j; i++ )
    subfunction[j].set_data( i, r[i+j], c[i+j], s[i+j] );

  priority_queue<Interval *,vector<Interval *>,Interval_lt> pqueue1, pqueue2;

  // Initialize data for the first interval (which should be R)

  subfunction[j].interval = new Interval( 0, n+50 );
  subfunction[j].interval->bound = 0;

  // Update intervals until we have subdepth of them

  pqueue1.push( subfunction[j].interval );

  for( unsigned int d = 1; d < subdepth; )
  {
    if( pqueue1.empty() ) break;

    intptr = pqueue1.top();
    pqueue1.pop();

    if( intptr->level < subfunction[j].n ) d--;

    subfunction[j].update_interval( intptr );

    for( it = intptr->child.begin(); it != intptr->child.end(); it++ )
    {
```

```
    (*it)->bound = subfunction[j].compute_subbound( *it ) +
                   subbound[j+(*it)->level];
    pqueue1.push( *it );
    d++;
  }
}

subfunction[j].interval->refine();

// Find a step function with subsize steps to estimate the subfunction

pqueue2.push( subfunction[j].interval );

for( unsigned int d = 1; d < subsize; )
{
  if( pqueue2.empty() ) break;

  intptr = pqueue2.top();
  pqueue2.pop();

  if( intptr->child.empty() )  // Add interval to estimate
  {
    // Scale interval

    mpfr_div_2si( intptr->left, intptr->left, j, GMP_RNDN );
    mpfr_div_2si( intptr->right, intptr->right, j, GMP_RNDN );

    subfunction[j].intlist.push_back( intptr );
    continue;
  }
  else
    d--;

  for( it = intptr->child.begin(); it != intptr->child.end(); it++ )
  {
    pqueue2.push( *it );
    d++;
  }
}

// Add the rest of the intervals from the priority queue to the estimate

while( !pqueue2.empty() )
{
  intptr = pqueue2.top();
  pqueue2.pop();

  // Scale interval

  mpfr_div_2si( intptr->left, intptr->left, j, GMP_RNDN );
  mpfr_div_2si( intptr->right, intptr->right, j, GMP_RNDN );

  subfunction[j].intlist.push_back( intptr );
}
```

```
  // Set the global subfunction bound

  subbound[j] = subfunction[j].interval->bound;
}

// Postprocessor::find_maximum
//
// Finds the highest local maximum not yet found
// Returns the appropriate interval, and stores the result in a list

Interval* Postprocessor::find_maximum()
{
  if( subfunction == NULL ) return NULL;    // Precomputation not performed

  Interval *intptr, *newintptr;
  vector<Interval *>::iterator it;
  unsigned int j;

  while( !pqueue.empty() )
  {
    intptr = pqueue.top();
    pqueue.pop();
    intcount++;

    j = intptr->level;      // Index of the appropriate subfunction

    if( j == n )
    {
      if( intptr->tight )   // If the bound is tight, then return the interval
      {
        // Adjust interval so that left endpoint is in [0,1)

        while( mpfr_cmp_ui( intptr->left, 1 ) >= 0 )
        {
          mpfr_sub_ui( intptr->left, intptr->left, 1, GMP_RNDN );
          mpfr_sub_ui( intptr->right, intptr->right, 1, GMP_RNDN );
        }

        while( mpfr_sgn( intptr->left ) < 0 )
        {
          mpfr_add_ui( intptr->left, intptr->left, 1, GMP_RNDN );
          mpfr_add_ui( intptr->right, intptr->right, 1, GMP_RNDN );
        }

        intlist.push_back( intptr );
        return intptr;
      }

      // Otherwise, tighten the bound and push it back on the queue

      intptr->bound = compute_subbound( intptr ) + subbound[j];
      intptr->tight = true;
      pqueue.push( intptr );
      continue;
    }

    double intbound = compute_subbound( intptr );
    intptr->bound = intbound + subbound[j];

    mpfr_set_ui( _e, 1, GMP_RNDN );
    mpfr_div_2ui( _e, _e, j, GMP_RNDN );

    it = subfunction[j].intlist.begin();

    // [_a,_b] is the interval we are subdividing

    if( mpfr_number_p( intptr->left ) )
    {
      mpfr_set( _a, intptr->left, GMP_RNDN );
      mpfr_set( _b, intptr->right, GMP_RNDN );
    } else {                                  // Interval is R
      mpfr_set( _a, (*it)->left, GMP_RNDN );  // Grab left range from interval
      mpfr_add_ui( _b, _a, 1, GMP_RNDN );
    }

    // Loop through all the intervals in the subfunction

    for( it = subfunction[j].intlist.begin();
         it != subfunction[j].intlist.end(); it++ )
    {
      // [_c,_d] is the shift of the current interval satisfying _c <= _a

      mpfr_sub( _f, _a, (*it)->left, GMP_RNDN );
      mpfr_mul_2si( _f, _f, j, GMP_RNDN );
      mpfr_floor( _f, _f );
      mpfr_div_2si( _f, _f, j, GMP_RNDN );
      mpfr_add( _c, (*it)->left, _f, GMP_RNDN );
      mpfr_add( _d, (*it)->right, _f, GMP_RNDN );

      if( mpfr_cmp( _d, _a ) > 0 )
      {
        newintptr = new Interval( j+(*it)->level, n+50 );
        mpfr_set( newintptr->left, _a, GMP_RNDN );
        mpfr_set( newintptr->right, _d, GMP_RNDN );
        newintptr->bound = intbound + (*it)->bound;

        pqueue.push( newintptr );
      }

      mpfr_add( _c, _c, _e, GMP_RNDN );
      mpfr_add( _d, _d, _e, GMP_RNDN );

      while( mpfr_cmp( _c, _b ) < 0 )
      {
        newintptr = new Interval( j+(*it)->level, n+50 );
        mpfr_set( newintptr->left, _c, GMP_RNDN );
        mpfr_min( newintptr->right, _d, _b, GMP_RNDN );
```

```
        newintptr->bound = intbound + (*it)->bound;

        pqueue.push( newintptr );
        mpfr_add( _c, _c, _e, GMP_RNDN );
        mpfr_add( _d, _d, _e, GMP_RNDN );
      }
    }

    // Delete the old interval

    delete intptr;
  }

  // The priority queue is empty, so all intervals have been found

  return NULL;
}
```

A.7 simulate.h

```
// File: simulate.h
//
// Simulation class for simulating phase estimation

#include <mpfr.h>

using namespace std;

class Simulation
{
 public:
  Simulation( unsigned int bits_in );
  ~Simulation();

  void set_bits( unsigned int bits_in );
  void set_phase( mpfr_t phase_in );
  void get_phase( mpfr_t phase_out );
  void set_random_phase();

  void measure( unsigned int iterations, unsigned int multiple,
                mpfr_t rotation, unsigned int &c, unsigned int &s );
  void measure_2( unsigned int iterations, unsigned int power,
                  mpfr_t rotation, unsigned int &c, unsigned int &s );

 private:
  unsigned int bits;            // Number of bits to be estimated

  mpfr_t phase;
  gmp_randstate_t randstate;
};
```

A.8 simulate.cpp

```
// File: simulate.cpp
//
// Implementation of Simulation class for simulating phase estimation

#include <ctime>

#include <gmp.h>
#include <mpfr.h>

#include "simulate.h"

using namespace std;

// Simulation Constructor

Simulation::Simulation( unsigned int bits_in )
  : bits( bits_in )
{
  mpfr_init2( phase, bits+50 );
  mpfr_set_ui( phase, 0, GMP_RNDN );

  // Initialize random number seed based on current time

  gmp_randinit_default( randstate );
  gmp_randseed_ui( randstate, time(0) );
}

// Simulation Destructor

Simulation::~Simulation()
{
  mpfr_clear( phase );
  gmp_randclear( randstate );
}

// Simulation::set_bits
//
// Sets the number of bits to be estimated

void Simulation::set_bits( unsigned int bits_in )
{
  bits = bits_in;
  mpfr_set_prec( phase, bits+50 );
}

// Simulation::set_phase
//
// Sets the hidden phase

void Simulation::set_phase( mpfr_t phase_in )
```

```
{
  mpfr_set( phase, phase_in, GMP_RNDN );
}

// Simulation::get_phase
//
// Returns the hidden phase

void Simulation::get_phase( mpfr_t phase_out )
{
  mpfr_set( phase_out, phase, GMP_RNDN );
}

// Simulation::set_random_phase
//
// Sets the phase to something random in the interval [0,1), with the
// appropriate number of bits

void Simulation::set_random_phase()
{
  mpfr_urandomb( phase, randstate );
}

// Simulation::measure
//
// Simulates the quantum circuit which performs the hidden phase shift
// multiple times, rotates by -rotation, and repeats this "iteration" times.
// c is set to be the number of measurements of 0
// s is set to be the number of measurements of 1

void Simulation::measure( unsigned int iterations, unsigned int multiple,
                          mpfr_t rotation, unsigned int &c, unsigned int &s )
{
  mpfr_t angle, prob, rand;

  mpfr_init2( angle, bits+50 );
  mpfr_init2( prob, bits+50 );
  mpfr_init2( rand, bits+50 );

  // Compute the effective phase that we are sampling

  mpfr_mul_ui( angle, phase, multiple, GMP_RNDN );
  mpfr_sub( angle, angle, rotation, GMP_RNDN );
  mpfr_frac( angle, angle, GMP_RNDN );

  // Compute probability of measuring 0

  mpfr_const_pi( prob, GMP_RNDN );
  mpfr_mul( prob, prob, angle, GMP_RNDN );
  mpfr_cos( prob, prob, GMP_RNDN );
  mpfr_sqr( prob, prob, GMP_RNDN );

  c = s = 0;
  for( unsigned int i = 0; i < iterations; i++ )
  {
    mpfr_urandomb( rand, randstate );
    if( mpfr_cmp( rand, prob ) < 0 ) c++;
    else s++;
  }

  mpfr_clear( angle );
  mpfr_clear( prob );
  mpfr_clear( rand );
}

// Simulation::measure_2
//
// Simulations the quantum circuit which performs the hidden phase shift
// 2^power times, rotates by -rotation, and repeats this "iteration" times.
// c is set to be the number of measurements of 0
// s is set to be the number of measurements of 1

void Simulation::measure_2( unsigned int iterations, unsigned int power,
                            mpfr_t rotation, unsigned int &c, unsigned int &s )
{
  mpfr_t angle, prob, rand;

  mpfr_init2( angle, bits+50 );
  mpfr_init2( prob, bits+50 );
  mpfr_init2( rand, bits+50 );

  // Compute the effective phase that we are sampling

  mpfr_mul_2ui( angle, phase, power, GMP_RNDN );
  mpfr_sub( angle, angle, rotation, GMP_RNDN );
  mpfr_frac( angle, angle, GMP_RNDN );

  // Compute probability of measuring 0

  mpfr_const_pi( prob, GMP_RNDN );
  mpfr_mul( prob, prob, angle, GMP_RNDN );
  mpfr_cos( prob, prob, GMP_RNDN );
  mpfr_sqr( prob, prob, GMP_RNDN );

  c = s = 0;

  for( unsigned int i = 0; i < iterations; i++ )
  {
    mpfr_urandomb( rand, randstate );
    if( mpfr_cmp( rand, prob ) < 0 ) c++;
    else s++;
  }

  mpfr_clear( angle );
  mpfr_clear( prob );
  mpfr_clear( rand );
}
```

A.9 timer.h

```
// File: timer.h
//
// Timer class for timing-related functions

#include <sys/times.h>

using namespace std;

class Timer
{
 public:
  void start() { times( &s ); }
  void stop() { times( &e ); }
  double elapsed() { return (double)( e.tms_utime - s.tms_utime )/100.0; }
  double split() { stop(); return elapsed(); }

 private:
  struct tms s, e;
};
```

A.10 process.cpp

```
// File: process.cpp
//
// This performs the postprocessing algorithm using a given data file

#include <iostream>
#include <iomanip>

#include "timer.h"
#include "interval.h"
#include "postproc.h"

using namespace std;

int main( int argc, char **argv )
{
  // Output usage information

  if( argc != 4 )
  {
    cout << "Usage: " << argv[0] << " [size] [subsize] [subdepth]" << endl;
    return 1;
  }

  // Read parameters from command line
  //
```

```
// size = size of likelihood function
// subsize = number of steps in a subfunction estimate
// subdepth = number of intervals to process to make subfunction estimates

unsigned int size, subsize, subdepth;

size = atoi( argv[1] );
subsize = atoi( argv[2] );
subdepth = atoi( argv[3] );

// Initialize the Postprocessor class

Postprocessor P;

P.set_size( size );

// Read phase estimation data from input

mpfr_t phase, rotation;
unsigned int c, s;

mpfr_init2( phase, size+50 );
mpfr_inp_str( phase, NULL, 10, GMP_RNDN );

mpfr_init2( rotation, size+50 );

for( unsigned int i = 0; i < size; i++ )
{
  mpfr_inp_str( rotation, NULL, 10, GMP_RNDN );
  cin >> c >> s;
  P.set_data( i, rotation, c, s );
}

mpfr_clear( rotation );

// Build subfunctions

P.set_estimate_size( subsize, subdepth );

Timer timer;     // Initialize timer

timer.start();
P.build_subfunctions();
timer.stop();

double subtime = timer.elapsed();

// Perform postprocessing algorithm

Interval *intptr;

timer.start();
while(1)
{
    intptr = P.find_maximum();

    if( intptr == NULL ) break;        // We should never have to do this

    // Adjust interval to check for wrap-around

    if( mpfr_cmp_ui( intptr->right, 1 ) >= 0
        && mpfr_cmp( phase, intptr->left ) < 0 )
    {
      mpfr_sub_ui( intptr->left, intptr->left, 1, GMP_RNDN );
      mpfr_sub_ui( intptr->right, intptr->right, 1, GMP_RNDN );
    }

    if( mpfr_cmp( intptr->left, phase ) <= 0
        && mpfr_cmp( intptr->right, phase ) >= 0 ) break;
  }
  timer.stop();

  // Output statistics
  //
  // size      subsize   subdepth  subtime   time      intervals

  cout << setw(10) << size << setw(10) << subsize;
  cout << setw(10) << subdepth << setw(10) << subtime;
  cout << setw(10) << timer.elapsed() << setw(10) << P.intcount << endl;

  return 0;
}
```

A.11 generate.cpp

```
// File: generate.cpp
//
// Generates simulated measurement data for a given or a random phase

#include <cstdio>
#include <iostream>

#include "simulate.h"

using namespace std;

int main( int argc, char **argv )
{
  // Output usage information

  if( argc != 4 && argc != 5 )
  {
    cout << "Usage: " << argv[0] << " [size] [reps] [depth] [(phase)]" << endl;
    cout << endl;
    cout << "To generate a random phase, leave optional parameter out" << endl;
    return 1;
  }

  // Read parameters from command line
  //
  // size = size of phase estimation problem (bits)
  // reps = number of reps for each bit
  // depth = depth of rotation gate corrections
  // phase (optional) = hidden phase to be estimated
  //
  // Note that the simulated rotation correction will always be the nearest one

  unsigned int size, reps, depth;

  size = atoi( argv[1] );
  reps = atoi( argv[2] );
  depth = atoi( argv[3] );

  // Initialize the simuation

  Simulation sim( size );
  mpfr_t phase, rotation;
  unsigned int c, s;

  mpfr_init2( phase, size+50 );
  mpfr_init2( rotation, size+50 );

  // If the phase data is missing, we need to generate a random phase
  // Otherwise, read the phase from the command-line parameters

  if( argc == 4 )
  {
    sim.set_random_phase();
    sim.get_phase( phase );

    // Make sure phase is in the range [0,1)

    mpfr_frac( phase, phase, GMP_RNDN );
    if( mpfr_sgn( phase ) < 0 )
      mpfr_add_ui( phase, phase, 1, GMP_RNDN );
  }
  else
  {
    mpfr_set_str( phase, argv[4], 10, GMP_RNDN );
    sim.set_phase( phase );
  }

  // Output phase
  // Data will be output in a format compatible with the "process" program

  mpfr_out_str( NULL, 10, 0, phase, GMP_RNDN );
  cout << endl;
```

```
  // Simulate phase estimation algorithm

  mpfr_set_ui( rotation, 0, GMP_RNDN );

  for( unsigned int power = 0; power < size; power++ )
  {
    // Find the nearest rotation correction

    if( depth > 1 )
    {
      mpfr_mul_2ui( rotation, phase, power+depth, GMP_RNDN );
      mpfr_round( rotation, rotation );
      mpfr_div_2ui( rotation, rotation, depth-1, GMP_RNDN );
      mpfr_frac( rotation, rotation, GMP_RNDN );
      mpfr_div_2ui( rotation, rotation, 1, GMP_RNDN );
    }

    // Simulate the actual measurement (reps) times

    sim.measure_2( reps, power, rotation, c, s );

    // Output measurement data

    mpfr_out_str( NULL,10,0, rotation, GMP_RNDN );
    cout << " " << c << " " << s << endl;
  }

  mpfr_clear( phase );
  mpfr_clear( rotation );

  return 0;
}
```

Bibliography

[AAKV01] D. Aharonov, A. Ambainis, J. Kempe, and U. Vazirani. Quantum walks on graphs. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 50–59, 2001. 35

[AV95] D. Atkinson and P. Vaidya. A cutting plane algorithm for convex programming that uses analytic centers. *Mathematical Programming*, 69:1–43, 1995. 54, 55

[BEST96] A. Barenco, A. Ekert, K.-A. Suominen, and P. Törmä. Approximate quantum fourier transform and decoherence. *Physical Review A*, 54(1):139–146, 1996. 60

[Che03] D. Cheung. Using generalized quantum fourier transforms in quantum phase estimation algorithms. Master's thesis, University of Waterloo, 2003. 1, 60, 62, 67, 92

[Che04] D. Cheung. Improved bounds for the approximate QFT. In *Proceedings of the Winter International Symposium of Information and Communication Technologies (WISICT)*, pages 192–197, 2004. 1

[Cop94] D. Coppersmith. An approximate Fourier transform useful in quantum factoring. Research Report RC19642, IBM, 1994. 60, 62, 66

[DLS97] C. Dürr, H. LêThanh, and M. Santha. A decision procedure for well-formed linear quantum cellular automata. *Random Structures and Algorithms*, 11:381–394, 1997. 12

[DS96] C. Dürr and M. Santha. A decision procedure for unitary linear quantum cellular automata. In *Proceeding of the 37th IEEE Symposium on Foundations of Computer Science*, pages 38–45, 1996. 12

[EH99] M. Ettinger and P. Høyer. On quantum algorithms for noncommutative hidden subgroups. *Lecture Notes in Computer Science*, 1563:478–487, 1999. 93

[EHGC04] J. Eisert, P. Hyllus, O. Guhne, and M. Curty. Complete hierarchies of efficient approximations to problems in entanglement theory. *Physical Review A*, 70:062317, 2004. 44

[Gác83] P. Gács. Reliable computation with cellular automata. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC)*, pages 32–41, 1983. 37

[GB02] L. Gurvits and H. Barnum. Largest separable balls around the maximally mixed bipartite quantum state. *Physical Review A*, 66:062311, 2002. 44

[GN96] R. B. Griffiths and C.-S. Niu. Semiclassical Fourier transform for quantum computation. *Physical Review Letters*, 76:3228–3231, 1996. 65

[Gur03] L. Gurvits. Classical deterministic complexity of Edmonds' problem and quantum entanglement. In *Proceedings of the 35th ACM Symposium on Theory of Computing (STOC)*, pages 10–19, 2003. 48

[HHH96] M. Horodecki, P. Horodecki, and R. Horodecki. Separability of mixed states: necessary and sufficient conditions. *Physics Letters A*, 223:1–8, 1996. 44

[HN75] M. Harao and S. Noguchi. Fault tolerant cellular automata. *Journal of Computer and System Sciences*, 11:171–185, 1975. 37

[ITC06] L. M. Ioannou, B. C. Travaglione, and D. Cheung. Convex separation from optimization via heuristics. `arXiv.org:cs.DS/0603089`, 2006. 1

[ITCE04] L. M. Ioannou, B. C. Travaglione, D. Cheung, and A. K. Ekert. Improved algorithm for quantum separability and entanglement detection. *Physical Review A*, 70(6):060303, 2004. 1

[Kit96] A. Kitaev. Quantum measurements and the abelian stabilizer problem. *Electronic Colloquium on Computational Complexity*, 3(3), 1996. 60, 68

[Kup03] G. Kuperberg. A subexponential-time quantum algorithm for the dihedral hidden subgroup problem. `arXiv.org:quant-ph/0302112`, 2003. 93

[Llo93] S. Lloyd. A potentially realizable quantum computer. *Science*, 261:1569–1571, 1993. 15, 35

[Mey96] D. Meyer. From quantum cellular automata to quantum lattice gases. *Journal of Statistical Physics*, 85:551–574, 1996. 15, 35

[Per96] A. Peres. Separability criterion for density matrices. *Physical Review Letters*, 77:1413–1415, 1996. 44

[PTVF92] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C, 2nd ed.* Cambridge University Press, 1992. 84

[Reg04] O. Regev. A subexponential time algorithm for the dihedral hidden subgroup problem with polynomial space. `arXiv.org:quant-ph/0406151`, 2004. 93

[Sho96] P. Shor. Fault-tolerant quantum computation. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 56–65, 1996. 37

[SW04] B. Schumacher and R. F. Werner. Reversible quantum cellular automata. `arXiv.org:quant-ph/0405174`, 2004. 10, 13, 14, 27

[TM87] T. Toffoli and N. Margolus. *Cellular Automata Machines*. MIT Press, 1987. 8

[Tof77] T. Toffoli. *Cellular Automata Mechanics*. PhD thesis, University of Michigan, 1977. 6

[Too76] A. Toom. Monotonic binary cellular automata. *Problems of Information Transmission*, 12:33–37, 1976. 37

[vD96] W. van Dam. Quantum cellular automata. Master's thesis, University of Nijmegen, 1996. 11

[Wat95] J. Watrous. On one-dimensional quantum cellular automata. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 528–537, 1995. 10, 11, 27