2004

Improved algorithms for approximate quantum fourier transforms and sparse hamiltonian simulations

Ahokas, Graeme Robert

Ahokas, G. R. (2004). Improved algorithms for approximate quantum fourier transforms and sparse hamiltonian simulations (Unpublished master's thesis). University of Calgary, Calgary, AB. doi:10.11575/PRISM/22839

http://hdl.handle.net/1880/41417

master thesis

THE UNIVERSITY OF CALGARY

Improved Algorithms for Approximate Quantum Fourier Transforms and Sparse Hamiltonian Simulations

by

Graeme Robert Ahokas

A THESIS
SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

April, 2004

UNIVERSITY OF
CALGARY

Abstract

We investigate and improve algorithms for computing the approximate quantum Fourier transform (QFT) and simulating a sparse Hamiltonian.

We present an improved near-linear time algorithm for the approximate QFT modulo 2^n. We give a circuit for the arbitrary modulus version that matches the best currently-known bound. We then relate the difficulty of the arbitrary modulus QFT to classical integer multiplication and division.

We also investigate the problem of simulating an n-qubit sparse Hamiltonian. As input we receive the system's start state, a black box which, via queries, provides the non-zero entries of each row of the Hamiltonian, and a time t. We present an improved polynomial-time quantum algorithm for computing an approximation of the state of the system at time t.

Prior to this, we provide an introduction to quantum computing, and survey some quantum algorithms that are used in our improved algorithms.

Acknowledgements

I would like to thank the Department of Computer Science at the University of Calgary, NSERC, iCORE, and an anonymous donor, who cannot be named, for their generous funding and financial support. Thank you to the Quantum Information Science Research Group, both faculty and grad students, who made research and learning fun and challenging. I wish to thank my supervisor, Richard Cleve, for all the help, support, and encouragement in getting through my degree. I have learned so much, and greatly appreciate your willingness to teach, patience, and the time you invested in me. Finally, I wish to thank my friends and family, who offered constant emotional support in getting to where I am today.

Table of Contents

List of Tables

List of Figures

Chapter 1

Fundamentals of Quantum Information Processing

1.1 Introduction

The field of quantum computing has been growing in popularity over the last two decades. The idea is to use very small physical systems, which obey the postulates of quantum mechanics, to perform computations. The method was suggested back in 1985 by David Deutsch [Deu85], but it was not until nearly 10 years later that the possibilities really became evident. In 1994 Peter Shor [Sho94] presented a fast integer factoring algorithm for quantum computers which intrigued the world of computing. The difficulty of factoring large composite numbers provides much of the basis of modern cryptography. There is no known efficient algorithm to factor, and with a classical computer and the best known techniques, factoring large numbers can take billions of years. With quantum computers and Shor's algorithm, the security of current cryptographic systems is in jeopardy.

Quantum computers are currently in their infancy, with only a handful of examples in the world. Even those that do exist are very primitive, with memory size of only a few "quantum" bits. The computations on these computers do not always work with high probability, due to the nature of the computer: building a quantum computer requires the ability to manipulate and control small systems, on the atomic scale. This in itself is a challenge for physicists. It also requires complete (or as complete as possible) isolation from the surrounding environment in order for the

system to evolve coherently and error free. These and other challenges lead scientists to speculate that full scale quantum computers are still in the distant future.

Exactly what advantages quantum computers have over classical computers has still not been precisely determined. With Shor's algorithm a quantum computer may factor in polynomial-time, but this may be possible, although currently thought to be unlikely, on a classical computer as well. The other well known general purpose quantum algorithm is Grover's database search [Gro96]. It can find an element in an unordered n-element list in time $\Theta(\sqrt{n})$, whereas $\Theta(n)$ time is required classically. There are other specially constructed problems where a quantum computer outperforms a classical device, but in terms of practical algorithms on a quantum computer, they are few and far between.

To begin the development of quantum algorithms, we start with a survey of the postulates of quantum mechanics. Quantum mechanics define the basic operations performable on a quantum computer.

1.2 Postulates of quantum mechanics

Quantum mechanics is a set of rules upon which physical theories are built. The theories developed from quantum mechanics depend on the specific physical system being considered, but all quantum mechanical systems follow the framework of quantum mechanics. From a computer science perspective, the postulates of quantum mechanics act as a hardware abstraction layer. They hide the details of each physical system, and the postulates become a standard interface to the hardware. The abstraction of the physical system is a direct benefit to computer scientists who

are interested in getting involved in the field. To explore the power of quantum computers, we need only explore what is physically possible, according to quantum mechanics. We present the postulates in the style of [NC00].

1.2.1 State space

Postulate 1 An isolated physical system is associated with a Hilbert space known as a *state space*. The state of the system at any moment in time is represented by a unit vector called the *state vector* in the system's state space. Two or more systems may be combined into a single system. The state space of the new composite system is constructed by taking the tensor product of the state spaces of the component systems.

One-qubit space

The simplest physical system is a single qubit. The state space of a single isolated qubit is a two-dimensional Hilbert space. The vectors

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle \quad \text{and} \quad \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle$$

form an orthonormal basis for a one-qubit state space. For convenience we use the *ket* or *Dirac* notation from physics, as writing $|0\rangle$ is simpler than explicitly writing the column vector each time. This notation also allows for writing the conjugate transpose of the vector $|\psi\rangle$ as $\langle\psi|$. Any possible state $|\psi\rangle$ for a one-qubit system may be written as a linear combination of $|0\rangle$ and $|1\rangle$,

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

with $\alpha, \beta \in \mathbb{C}$. Since $|\psi\rangle$ is a unit vector, we have the normalization condition $|\alpha|^2 + |\beta|^2 = 1$ or equivalently $\langle \psi \,|\, \psi \rangle = 1$. The values α and β are the *amplitudes* of the states $|0\rangle$ and $|1\rangle$, respectively.

Multiple-qubit spaces

We can construct an isolated n-qubit system from n one-qubit systems. The state space of the new system will be of dimension 2^n. As before, we need an orthonormal basis to represent any vector in the space. A sample orthonormal basis is the set of vectors $|x\rangle$ for all $x \in \{0,1\}^n$. These vectors are constructed by taking all combinations of the tensor product of n basis vectors for a one-qubit system, $|0\rangle$ and $|1\rangle$. The vector $|x\rangle$ has 2^n elements, all of which are zero except for the x^{th} entry, interpreted as a binary number, is one. A state $|\psi\rangle$ in the new n-qubit state space can then be represented as a linear combination of these basis vectors

$$|\psi\rangle = \sum_{x \in \{0,1\}^n} \alpha_x \, |x\rangle \tag{1.1}$$

with $\alpha_i \in \mathbb{C}$ subject to the normalization condition $\langle \psi \,|\, \psi \rangle$.

The sample orthonormal basis given above was chosen because it has a special name. We call the basis $|x\rangle$ for all $x \in \{0,1\}^n$ the *computational basis*. We often write x in base ten rather than binary for simplicity.

Product states and entangled states

As we have seen, some multi-qubit states can be constructed by taking the tensor product of single qubit states. When writing down such a state, we often eliminate the tensor product symbol, or put the states inside the same ket. That is,

$$|\psi_0\rangle \otimes |\psi_1\rangle = |\psi_0\rangle\,|\psi_1\rangle = |\psi_0\psi_1\rangle\,. \tag{1.2}$$

We call a group of n qubits inside the same ket an n-qubit *quantum register*. An *ancilla* is quantum register used for temporary storage during a computation. Each qubit in the ancilla is assumed to be initialized to $|0\rangle$ and we reset them to this state after the computation.

States that can be written as a tensor product of two (or more, depending on our wishes) quantum registers are called called *product states*. For example, the state

$$|\psi\rangle = \frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle \tag{1.3}$$

can be factored into

$$|\psi\rangle = \left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right) \otimes \left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right). \tag{1.4}$$

So the two qubits are in a product state. On the other hand, a state such as

$$|\psi\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle \tag{1.5}$$

cannot be factored into a product of single qubit registers. A state such as this is called *entangled*.

1.2.2 Evolution of a system

Postulate 2 An isolated physical system's state evolves over time t according to the *Schrödinger equation*,

$$i\frac{d\,|\psi(t)\rangle}{dt} = H\,|\psi(t)\rangle\,. \tag{1.6}$$

In this equation, H is a Hermitian matrix called a *Hamiltonian.* For an n-qubit system, the Hamiltonian is 2^n by 2^n. The Hamiltonian can be either time independent or time dependent; a time-independent Hamiltonian, the case we consider in this thesis, is a constant matrix, while a time-dependent Hamiltonian may be a function of t in the differential equation.

The solution to the Schrödinger equation can be verified to be

$$|\psi(t)\rangle = e^{-iHt} |\psi(0)\rangle . \tag{1.7}$$

From a start state $|\psi(0)\rangle$, evolving the system for time t will give the state $|\psi(t)\rangle$. The operator e^{-iHt} describes the evolution of a state from a known start state. Since the Hamiltonian H is Hermitian, we can show that the evolution operator e^{-iHt} is unitary using the following two facts.

Fact 1.1 (Unitary conjugation) $e^{-itU^\dagger AU} = U^\dagger e^{-itA} U$ *for any matrix* A*, real time* t*, and unitary matrix* U*.*

Proof: The fact can be seen by taking the Taylor series expansion of $e^{-iU^\dagger AUt}$:

$$\begin{aligned} e^{-iU^\dagger AUt} &= I + -iU^\dagger AUt + \left(-iU^\dagger AUt\right)^2/2! + \cdots \\ &= U^\dagger IU + U^\dagger(-iAt)U + U^\dagger((-iAt)^2/2!)U + \cdots \\ &= U^\dagger \left(I + -iAt + (-iAt)^2/2! + \cdots\right) U \\ &= U^\dagger e^{-iAt} U. \end{aligned}$$

■

Fact 1.2 (Exponential of a real diagonal matrix) *For a real diagonal matrix A with entries $\alpha_1, \ldots, \alpha_N$ and real t, e^{-iAt} is a diagonal unitary matrix with entries $e^{-i\alpha_1 t}, \ldots, e^{-i\alpha_N t}$.*

Proof: We use Taylor series again to expand e^{-iAt} yielding

$$\begin{aligned} e^{-iAt} &= I + -iAt + (-iAt)^2/2! + \cdots \\ &= I + \begin{bmatrix} -i\alpha_1 t & & \\ & \ddots & \\ & & -i\alpha_N t \end{bmatrix} + \begin{bmatrix} (-i\alpha_1 t)^2/2! & & \\ & \ddots & \\ & & (-i\alpha_N t)^2/2! \end{bmatrix} + \cdots \\ &= \begin{bmatrix} e^{-i\alpha_1 t} & & \\ & \ddots & \\ & & e^{-i\alpha_N t} \end{bmatrix} \end{aligned}$$

which is a unitary matrix since all α_i and t are real. ∎

Using these two facts, we can show that the evolution operator of a physical system is unitary.

Fact 1.3 (Evolution is unitary) *The evolution operator e^{-iHt} of a system is unitary.*

Proof: Since the Hamiltonian H is Hermitian (and therefore also normal), there exists a unitary matrix U such that $H = U^\dagger A U$ with A a diagonal real matrix. So $e^{-itH} = e^{-itU^\dagger AU} = U^\dagger e^{-itA} U$ by Fact 1.1. From Fact 1.2 e^{-itA} is unitary since t and the entries of A are real, and so the whole expression is unitary as well. ∎

1.2.3 Measurement

To this point we have provided the state space and evolution postulates. An isolated physical system's state can be represented as a vector in its state space, and the system evolves over time according to some unitary operator. It has not been shown, however, how any of this can be used to actually represent or store information, or to do some meaningful computation. Some link is needed to the classical world of representing information as binary strings. The measurement postulate provides this.

Postulate 3 *Measurements* may be performed on a system's state. They can be used to extract classical data values from a quantum state. A set of *measurement operators* $\{M_j\}$ describes the probability of each potential outcome of a measurement, as well as the new state of the system after the measurement. The operators are indexed by the potential outcomes of a measurement of the system. Suppose the state of some quantum system is $|\psi\rangle$. Then the probability of getting outcome j when measuring a system $|\psi\rangle$ is

$$p(j) = \langle\psi| M_j^\dagger M_j |\psi\rangle . \tag{1.8}$$

After the measurement, if the outcome was j, the state of the system is

$$\frac{M_j |\psi\rangle}{\sqrt{\langle\psi| M_j^\dagger M_j |\psi\rangle}}. \tag{1.9}$$

Finally, the measurement operators have the requirement that

$$\sum_j M_j^\dagger M_j = I. \tag{1.10}$$

For the purposes of this thesis, we will only consider the measurement operators in the computational basis. For this basis, the single qubit measurement operators

are $M_0 = |0\rangle\langle 0| = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$ and $M_1 = |1\rangle\langle 1| = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$. The measurement operators for multi-qubit systems are constructed by taking the tensor product of single qubit measurement operators.

Single qubit measurement

To see how a measurement operator works, suppose we have a single qubit in the state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$. Then, by Equation 1.8, the probability of seeing a 0 after the measurement is

$$\begin{aligned} p(0) &= \langle\psi| M_0^\dagger M_0 |\psi\rangle \\ &= \begin{bmatrix} \alpha^* \beta^* \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \\ &= \alpha^* \alpha \\ &= |\alpha|^2. \end{aligned} \tag{1.11}$$

By a similar calculation the probability of measuring a 1 is $|\beta|^2$. If a 0 is measured, by Equation 1.9 the resulting state is

$$\frac{M_0 |\psi\rangle}{\sqrt{\left\langle \psi \middle| M_0^\dagger M_0 \middle| \psi \right\rangle}} = \frac{\alpha}{|\alpha|} |0\rangle . \tag{1.12}$$

The same can be done to determine the state if a 1 were measured. The state after the measurement, Equation 1.12, is a valid quantum state, since for non-zero α, $\left|\frac{\alpha}{|\alpha|}\right|^2 = 1$. The same holds for β.

The value $\frac{\alpha}{|\alpha|}$ is a *global phase*, which we can be ignored since it is unobservable. That is, no measurement can detect it. To see this, the global phase value can

be written as $e^{\delta i}$ for an appropriately chosen δ. Then we see that applying the measurement operators to a state $|\psi\rangle$ and applying them to the same state with a global phase $e^{i\delta}|\psi\rangle$ yield the same outcome probabilities.

$$p(j) = \left\langle \psi \middle| e^{-i\delta} M_j^\dagger M_j e^{i\delta} \middle| \psi \right\rangle = \left\langle \psi \middle| M_j^\dagger M_j \middle| \psi \right\rangle. \tag{1.13}$$

What implications do these measurement operators have? The absolute value squared of the amplitude of each state is the probability that that outcome will occur when the state is measured. The state $|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$ yields 0 and 1 each with probability 1/2. We think of the such a state as being in *superposition*. This is a powerful feature of quantum information that is not possible classically, and is one of the suspected advantages of quantum information. When the measurement occurs, the state *collapses* into either $|0\rangle$ or $|1\rangle$ according to their respective probabilities. Any further measurements on the system will then give the same state.

Partial measurements

It is conceivable that we may wish to measure only some of the qubits in a multi-qubit system. To construct the measurement operators for these partial measurements, we substitute the identity operator for a single qubit measurement operator M_0 or M_1 for all qubits that will not be measured. The identity can be thought of as "doing nothing" to a qubit. The result will be measurement operators that still satisfy the measurement postulate. For example, suppose we have the state

$$|\psi\rangle = \frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle \tag{1.14}$$

and we wish to measure only the first qubit. The measurement operators are defined to be

$$M_0 \otimes I \qquad \text{and} \qquad M_1 \otimes I. \tag{1.15}$$

When the first qubit is measured, we will measure a 0 with probability

$$p(0) = \left\langle \psi \right| (M_0 \otimes I)^\dagger (M_0 \otimes I) \left| \psi \right\rangle = 1/2 \tag{1.16}$$

and measure a 1 with the same probability. If a 0 were measured, the resulting state is

$$\frac{M_0 \otimes I \left|\psi\right\rangle}{\sqrt{\langle \psi \left| (M_0 \otimes I)^\dagger (M_0 \otimes I) \right| \psi \rangle}} = \left|0\right\rangle \left(\frac{1}{\sqrt{2}} \left|0\right\rangle + \frac{1}{\sqrt{2}} \left|1\right\rangle \right).$$

The calculation is similar if a 1 were measured. So the superposition *partially collapsed*. Measuring the first qubit again would always give a 0, but the second qubit remains in superposition.

1.3 Computing with quantum systems

We now show how to use the postulates to perform computations. The general design of a quantum algorithm is as follows:

1. Initialize the system to a known start state.
2. Evolve the system unitarily.
3. Measure the system to extract the output.

1.3.1 Circuit model of computation

In classical computing, circuits are often drawn with the flow of computation going from left to right, or top to bottom. A circuit may have some inputs, set to 0 or 1,

perform some function using an assortment of logic gates, and have some output bits. There is a corresponding model for quantum computing, called a *quantum circuit*, which is the most popular model of computation used. A quantum circuit consists of some input qubits which are arranged on the left. The qubits are drawn as lines; the lines are analogous to wires carrying an electrical signal in a classical circuit. Time flows from left to right; as time passes, we perform unitary operations, denoted as boxes, on the qubits. A qubit's line passing through a box signifies applying the unitary transformation to the qubit. At the end, we may measure the qubits to extract a classical output from the computation. Figure 1.1 shows an example of a quantum circuit.

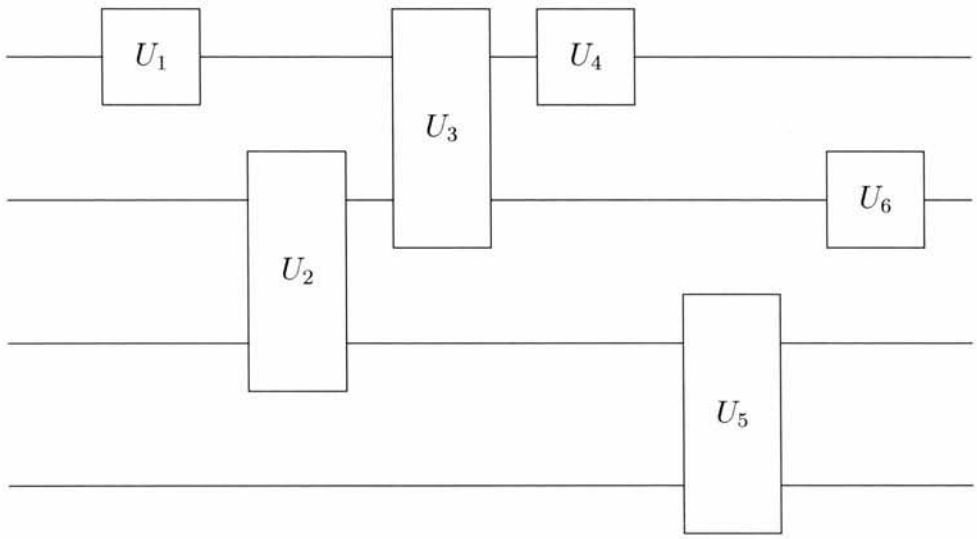

Figure 1.1: A 4-qubit quantum circuit with 6 unitary gates.

In Figure 1.1, we have shown *partial unitary operations*. Instead of applying a unitary operation to the entire system, we can apply a unitary operation to just

some of the qubits, and "do nothing" to the others. Doing nothing can be thought of as applying the identity, since it is unitary and leaves the qubit unchanged. So, in Figure 1.1, the first operator is really $U_1 \otimes I \otimes I \otimes I$, the second is $I \otimes U_2 \otimes I$, and so on. In some cases we group some qubits together in a circuit diagram to reduce clutter. Typically we do this when a group of qubits are used together for some purpose, such as a register to hold a value. To do this, we draw just one line representing the qubits, but draw another diagonal line over it, with an integer indicating how many qubits are grouped together. Figure 1.2 gives an example of this notation.

Figure 1.2: $|x\rangle$ is an n-qubit register.

1.3.2 Unitary gates

Since a quantum system evolves unitarily, we may apply any unitary operation to the qubits. Figure 1.3 lists some common one-qubit unitary gates with their name, circuit symbol, and matrix.

It is easy to verify that each of the operations is unitary. The circuit model that we are presenting, however, may seem completely different than the continuous time unitary evolution operator e^{-iHt} from Postulate 2. In the circuit model, we may think of there being discrete time steps, and at each step we apply some unitary operator to a number of qubits. In the continuous time model presented, there is only the evolution operator for the whole system. It turns out that given a quantum circuit with unitary gates, we can with some work calculate the corresponding Hamiltonian for the system. The problem of going the other direction, that is finding a quantum

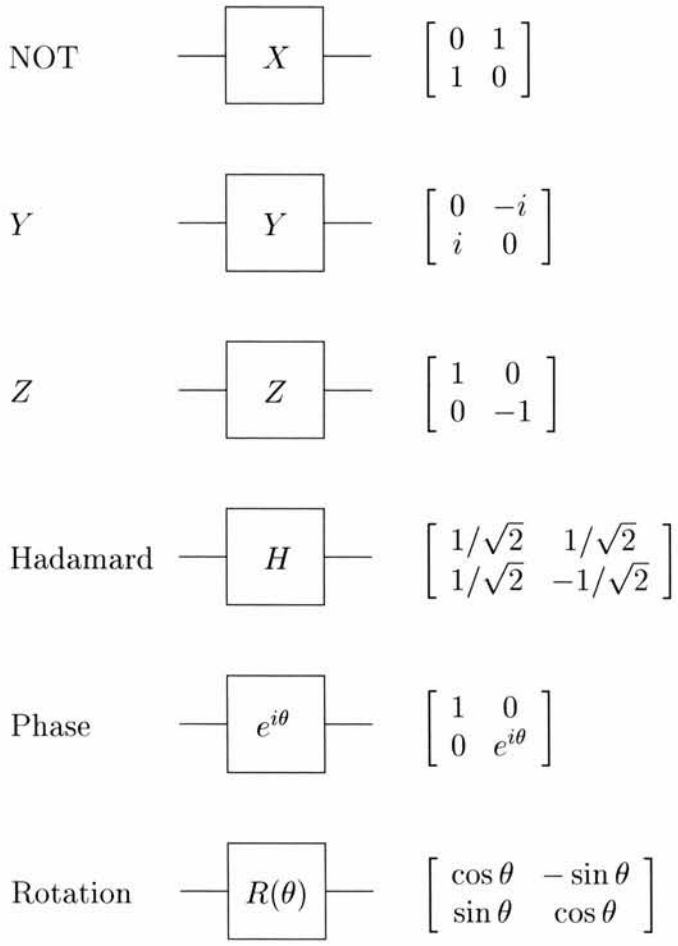

Figure 1.3: Common one-qubit gates.

circuit to implement e^{-iHt} for some t and H, is the subject of Chapter 4. The first point to notice is that the evolution operator describes the evolution of the whole system, while each gate in the circuit model may only apply to a small subset of the qubits. We have already argued that applying a unitary operator to a subset of the qubits just means applying the unitary operator to the subset and the identity to the rest. The tensor product of unitary operators is also unitary, since for any

unitary operators A and B

$$(A \otimes B)^\dagger (A \otimes B) = (A^\dagger \otimes B^\dagger)(A \otimes B) = I \otimes I = I. \tag{1.17}$$

Each time step in the circuit model is unitary, and so the entire evolution is unitary. To find the Hamiltonian associated with a particular circuit requires some calculation. Suppose we have a single qubit and we wish to apply the NOT gate. We then need to find a Hermitian matrix H such that

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = e^{-iHt}.$$

We will fix the value t at 1. Other values of t will just scale the calculated Hamiltonian H. We diagonalize X with two Hadamard gates,

$$X = HZH. \tag{1.18}$$

The derivation is then essentially the same calculation as Fact 1.3 in reverse.

$$\begin{aligned} HZH &= H \begin{bmatrix} e^{-i0} & 0 \\ 0 & e^{-i\pi} \end{bmatrix} H \\ &= He^{-iA}H \quad \text{by Fact 1.2} \\ &= e^{-iHAH} \quad \text{by Fact 1.1} \\ &= e^{-iB}, \end{aligned} \tag{1.19}$$

where $A = \begin{bmatrix} 0 & 0 \\ 0 & \pi \end{bmatrix}$ and $B = HAH = \frac{\pi}{2} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$. The NOT gate was chosen because of its simplicity, but it should be clear that given any unitary matrix, we could derive the corresponding Hamiltonian using the same method. For large systems this may be impractical as the matrices involved grow exponentially in the number of qubits of the system.

Two qubit gates

In addition to the one-qubit gates presented in Figure 1.3, we may apply unitary operations on two qubits. The two-qubit gates we use most frequently are the *controlled* versions of one-qubit gates. One of the two qubits acts as a control bit. If the control bit is 0, the gate is not applied to the second (target) qubit. If the control bit is 1, the gate is applied. In a circuit diagram, a controlled gate is drawn with a wire from the control qubit to the target qubit's gate. A controlled-NOT gate is shown in Figure 1.4.

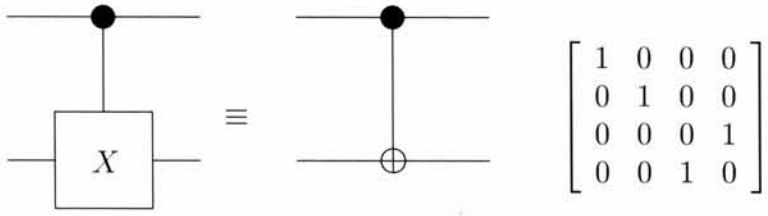

Figure 1.4: The controlled-NOT gate.

It maps

$$|x\rangle\,|y\rangle \mapsto |x\rangle\,|y \oplus x\rangle\,, \tag{1.20}$$

so x is stored on the second qubit. To reverse (erase) the copy, we can perform the controlled-NOT again, giving

$$|x\rangle\,|y \oplus x\rangle \mapsto |x\rangle\,|y \oplus x \oplus x\rangle = |x\rangle\,|y\rangle\,. \tag{1.21}$$

In addition to standard controlled gates, we will use open controlled gates as well. In this case, the gate is applied if the control qubit is 0, and is not applied when the control qubit is 1. An open controlled gate is drawn in the same manner as a

regular controlled gate, except the control is a white-filled circle instead of a black circle. It is easy to implement an open controlled gate from a regular controlled gate and the NOT gate, by applying a NOT gate to control qubit before and after the regular controlled gate.

Multi-qubit gates

The only multi-qubit gate we will mention is the Toffoli gate, which is a "doubly-controlled-NOT" gate. It maps three qubits as

$$|x\rangle\,|y\rangle\,|z\rangle \mapsto |x\rangle\,|y\rangle\,|z \oplus (x \wedge y)\rangle\,. \tag{1.22}$$

More general multi-qubit gates are rarely used in quantum circuits for two reasons. First, in e.g. [BBC+95] it is shown that single qubit gates plus the controlled-NOT gate form a universal set of gates. Second, multi-qubit gates are much more difficult to physically implement on a quantum computer. For gates to be applied, we need complete control of the physical system. The current technological limit allows control of one qubit at a time, and (much more difficultly) two qubits. While the precise limitations are specific to the physical system, it is generally accepted that implementing greater than two-qubit gates is prohibitively difficult. As one and two-qubit gates suffice, this is not a large concern.

The classical analogue to this limitation is, for example, when we wish to compute the bitwise AND of n bits. It would be naïve to assume that we may apply an n-input AND gate and consider it a single logic gate. Rather, to implement the gate, we compute the AND of pairs of bits, compute the AND pairwise again of the results, and continue in this fashion. So computing the AND of n bits would require roughly $n-1$ AND gates. In other words, the FANIN of classical logic gates is usually limited

to one or two bits.

Initialization

To use a quantum system as a computational tool we must have the ability to have the system begin the computation in some known start state. In this thesis, we will assume all qubits are initialized to $|0\rangle$ unless stated otherwise.

1.3.3 Classical computations

We have seen some sample one, two, and three-qubit unitary gates. It follows naturally to question whether we may perform classical computations on a quantum computer. For a nice explanation that quantum computers are at least as powerful as classical computers, see e.g. [NC00]. We briefly review the argument here. As mentioned earlier, any algorithm on a classical computer can be thought of as a classical circuit of logic gates. It is well known that the NAND gate is universal, in that it all other logic gates can be implemented from it. We show that the NAND gate can be implemented with a quantum circuit. The problem is that the NAND (and AND) gates are not reversible. We have not addressed this directly, but all unitary quantum gates are reversible. (Measurements, however, are not reversible). This follows from the fact that all quantum evolution is unitary, and all unitary operators have inverses that are also unitary. If we wish to reverse a computation, we apply the inverse of the original unitary operator. In order to make NAND and AND performable on a quantum computer, we must come up with a unitary version of these logic gates. Luckily, it turns out this is rather easy. Figure 1.5 shows the AND of two qubits $|x\rangle, |y\rangle$ can be implemented reversibly as a Toffoli gate. To get

the and of x and y, we set the third qubit to $|b\rangle = |0\rangle$.

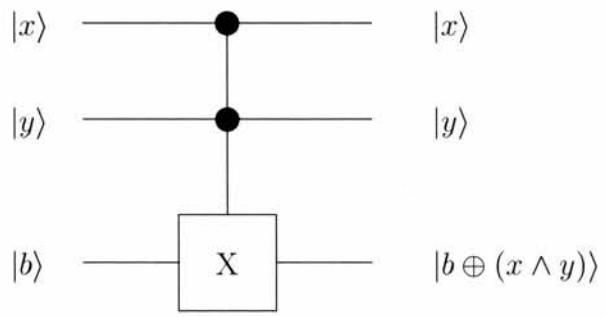

Figure 1.5: The Toffoli gate is a reversible AND gate.

The NAND gate, which is universal, may be created by applying a Toffoli gate and then applying the NOT gate to the third qubit. The Toffoli gate works as a unitary AND gate because the input bits are not changed. Rather, the AND of the input bits is computed and stored on the output qubit. Applying the Toffoli gate again would undo the computation (the Toffoli gate is its own inverse). Since we can implement the NAND gate, any Boolean circuit can be implemented with a quantum computer in a reversible (and therefore unitary) fashion.

There are several results, listed in [NC00], on reversible computation. For our purposes, all we need is the result that any classical algorithm can be transformed into a reversible version with the same asymptotic cost.

1.4 Measures of circuit cost

In quantum computation, as in classical computation, there are various ways of measuring the cost of an algorithm. In the quantum world, since the usual model for quantum algorithms is a quantum circuit, we can measure various properties of

the circuit. The most common are circuit *size*, *depth*, and *width*. The depth of a circuit is defined as follows. We say that a gate U_i *depends on* another gate U_j in the circuit if U_j is to the left of U_i in the circuit and U_i and U_j share at least one qubit. Gates that do not depend on any other gates have depth 1. Other gates have depth 1 plus the maximal depth the gates that they depend on. The depth of the circuit is then the maximal depth of any gate. The width of a circuit is the total number of qubits used. The size of a circuit is simply the total number of (one and two-qubit) gates. Figure 1.6 gives an example of the three measures. When using any of these

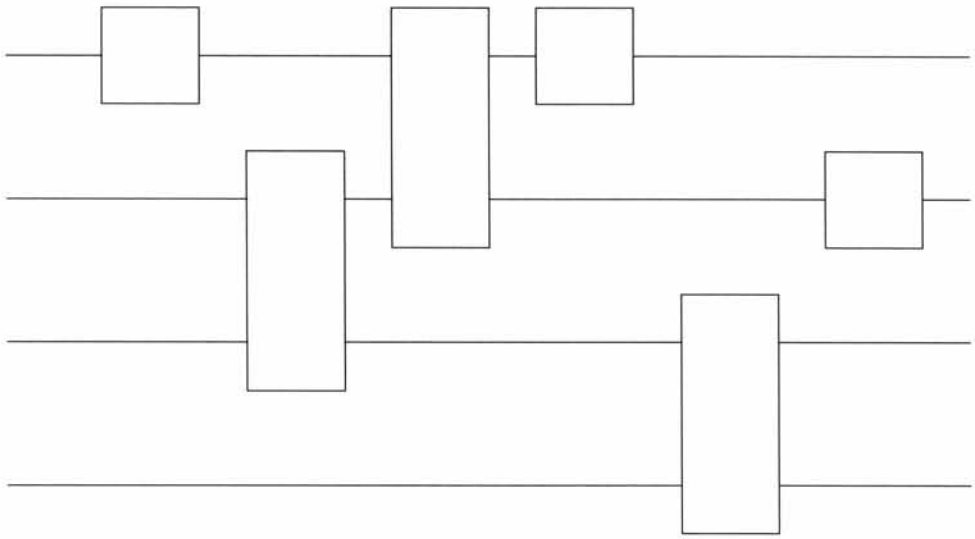

Figure 1.6: A quantum circuit of depth 3, width 4, and size 6.

measures, we assume that each gate in the circuit has unit cost. Of course, the time to implement each gate in the laboratory may vary depending on the type of physical system, apparatus, and so on. The circuit width is of obvious interest, particularly during these early years of quantum computation. Due to technological limitations, we currently only have a handful of qubits with which to perform computations. Circuit width is useful to see what computations are possible within a limited number

of qubits. That is, it is the analogue to classical space complexity. Circuit depth can be considered when we are concerned with *decoherence*, the degradation of a quantum state because of unwanted interactions with the environment. The depth of a circuit indicates how long the system must stay in a coherent state to perform the desired computation. For this thesis, we concentrate on measuring the size of quantum circuits. The size of a circuit corresponds exactly to the classical definition of circuit size. It is a more conservative view of the time required to perform an algorithm than circuit depth. In Figure 1.6, from a circuit-depth point of view, the first two gates (from left to right) both have depth 1. This is implies that these two gates could be applied in parallel, given the ability of the equipment to perform them. The system would have to remain in a coherent state for 3 units of time. From a circuit-size point of view, we simply add the number of gates in the circuit, and this will be an upper bound on the time required for the circuit. We assume each gate has unit cost, and use the terms circuit size and running time interchangeably.

1.5 Approximate versus exact computation

In classical computer science, there are exact, probabilistic, and approximate algorithms. An exact algorithm will give the correct answer with certainty every time the computation is performed. A probabilistic algorithm gives the correct answer with high probability, and an incorrect answer with low probability. An approximate algorithm gives an answer that is close to the correct answer, within some allowed bound. There is interest in approximate and probabilistic algorithms because quite often allowing a small error or failure probability can result in large gains in the

complexity of the problem. The same motivation can be used to argue the need for approximate and probabilistic quantum algorithms. Another argument is based on the realization that in reality quantum computation will not be exact. This is particularly true in these early days of the field. We are not able to completely control quantum systems, and isolate them from external influences. The physical system will introduce some small errors, so if an algorithm produces small errors as well, it will likely not hurt the computation too much. What "small" means and how we measure the error in an approximate quantum algorithm is the topic of the next section.

1.5.1 Measure of distance

The postulates of quantum mechanics tell that the state of a system can be represented as a unit vector in the system's state space. To determine if one state is close to another, we may find the Euclidean distance between the two state vectors. A small Euclidean distance will imply that when the states are measured, we obtain the same result from both measurements with high probability. This must happen because a small Euclidean distance implies that the terms of each of the state vectors are very close. It is also possible to calculate the distance between two unitary operators. Let the unitary operators be A and B (B can be thought of as an approximate version of A). Intuitively, if A and B have a large Euclidean distance, then there is some unit vector $|\psi\rangle$ such that the Euclidean distance between $A|\psi\rangle$ and $B|\psi\rangle$ is large. If A and B have small Euclidean distance, no such vector $|\psi\rangle$ exists. We formalize these arguments and introduce the definitions of Euclidean distance.

For a vector $|\psi\rangle = \sum_{s \in S} \alpha_s |s\rangle$, the Euclidean norm is defined as

$$\| |\psi\rangle \| = \sqrt{\sum_{s \in S} |\alpha_s|^2}. \tag{1.23}$$

The Euclidean distance between two vectors $|\psi\rangle, |\phi\rangle$, is $\| |\psi\rangle - |\phi\rangle \|$. We may extend these definitions to also give norms and distances between operators. For some operator A, the Euclidean norm is defined as

$$\|A\| = \max_{\| |\psi\rangle \| = 1} \|A |\psi\rangle \|. \tag{1.24}$$

The Euclidean distance between two operators A, B is $\|A - B\|$. We use the Euclidean distance of operators to measure the error in a quantum circuit. We prove the following fact, which shows that errors in a quantum circuit add linearly.

Fact 1.4 *For arbitrary unitary matrices A_1, A_2, B_1, B_2, U_1, U_2, U_3,* $\|U_1 A_1 U_2 A_2 U_3 - U_1 B_1 U_2 B_2 U_3\| \leq \|A_1 - B_1\| + \|A_2 - B_2\|$.

Proof:

$$\begin{aligned}
\|U_1 A_1 U_2 A_2 U_3 - U_1 B_1 U_2 B_2 U_3\| &= \|U_1 A_1 U_2 A_2 U_3 - U_1 B_1 U_2 A_2 U_3 + \\
&\quad U_1 B_1 U_2 A_2 U_3 - U_1 B_1 U_2 B_2 U_3\| \\
&\leq \|U_1 A_1 U_2 A_2 U_3 - U_1 B_1 U_2 A_2 U_3\| + \\
&\quad \|U_1 B_1 U_2 A_2 U_3 - U_1 B_1 U_2 B_2 U_3\| \\
&= \|U_1 (A_1 - B_1) U_2 A_2 U_3\| + \|U_1 B_1 U_2 (A_2 - B_2) U_3\| \\
&= \|A_1 - B_1\| + \|A_2 - B_2\|
\end{aligned}$$

■

Some interpretation may help here. Imagine that we have a sequence of unitary operators $U_1 A_1 U_2 A_2 U_3$ that we wish to apply to a system. Instead of applying this sequence exactly, we will approximate the operators A_1 and A_2 with B_1 and B_2, respectively. Fact 1.4 gives an upper bound on the induced error from using approximate unitary operators, rather than the exact operators, between arbitrary unitary operators. It states we may simply add the individual errors each approximate operators induce, and that the sum will be an upper bound on the total error in the circuit. Fact 1.4 can be generalized to k terms:

$$\|U_1 A_1 U_2 \cdots A_k U_{k+1} - U_1 B_1 U_2 \cdots B_k U_{k+1}\| \leq \|A_1 - B_1\| + \cdots + \|A_k - B_k\|. \quad (1.25)$$

A final note on the measure of distance. We have only mentioned the measure of Euclidean distance. There are other more general measures, such as fidelity, that can measure closeness of quantum states. They can be useful when the state is a *mixed state*, which can be viewed in some contexts as a probability distribution of states. However, for our purposes, Euclidean distance suffices. Euclidean distance is a stricter-than-necessary measure, because to achieve small Euclidean distance, states may not have large differences in their global phases, even though those global phases are unobservable. The measure of fidelity disregards global phases, but we use Euclidean distance for its simplicity and intuitive nature. Appendix A shows that the Euclidean distance is sufficient to measure the error of a quantum algorithm when the input state is a mixed state, and that small Euclidean distance implies high fidelity.

Chapter 2

Standard Quantum Algorithms

2.1 Introduction

In this chapter we survey four important quantum algorithms. We begin by introducing the exact and approximate quantum Fourier transforms (QFTs) mod 2^n. These two algorithms will be the basis for our improved algorithms for the QFT presented in Chapter 3. For this reason, we derive the circuits for both. We then outline the phase estimation and amplitude amplification procedures. We present them for completeness, as we use them as black box procedures in our improved approximate QFT algorithms. A more in-depth look at the algorithms is given in [NC00].

2.2 Exact quantum Fourier transform modulo 2^n

The quantum Fourier transform modulo 2^n is a very important algorithm in quantum computing. It is a key ingredient in many quantum algorithms that outperform their classical counterparts. In particular, it is used in phase estimation, which can be used to solve the order finding and integer factoring problems. It was Shor [Sho94] who first showed the power of the QFT and how to compute it for certain large moduli. Cleve [Cle94] and Coppersmith [Cop94] gave quantum circuits to compute the QFT mod 2^n. We introduce the QFT mod N, and show two circuits to implement it for $N = 2^n$. Our explanation is based on the presentation in [NC00].

The quantum Fourier transform modulo N is defined as the unitary operation that takes a basis state $|a\rangle \in \{|0\rangle, \ldots, |N-1\rangle\}$ to the state

$$\frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{2\pi i a y/N} |y\rangle . \tag{2.1}$$

We call the state in 2.1 a *quantum Fourier basis state* for $0 \leq a < N$. For $N = 2^n$, the standard circuit for the QFT can be derived from the definition by factoring the state 2.1. We use the *binary fraction* notation to write the value $a_1/2 + a_2/4 + \cdots + a_n/2^n$ as $0.a_1 a_2 \cdots a_n$. We also may write n-bit integers a as $a = a_1 a_2 \cdots a_n$ or $a = a_1 2^{n-1} + a_2 2^{n-2} + \cdots + a_n 2^0$. The state from 2.1 can be factored as

$$\begin{aligned}
\frac{1}{\sqrt{2^n}} \sum_{y=0}^{2^n-1} e^{2\pi i a y/2^n} |y\rangle &= \frac{1}{\sqrt{2^n}} \sum_{y \in \{0,1\}^n} e^{2\pi i a (y_1 2^{n-1} + y_2 2^{n-2} + \cdots + y_n 2^0)/2^n} |y_1 y_2 \cdots y_n\rangle \\
&= \frac{1}{\sqrt{2^n}} \sum_{y \in \{0,1\}^n} e^{2\pi i a y_1 2^{n-1}/2^n} |y_1\rangle \cdots e^{2\pi i a y_n 2^0/2^n} |y_n\rangle \\
&= \frac{1}{\sqrt{2^n}} \sum_{y \in \{0,1\}^n} e^{2\pi i (0.a_n) y_1} |y_1\rangle \cdots e^{2\pi i (0.a_1 \cdots a_n) y_n} |y_n\rangle \\
&= \frac{1}{\sqrt{2^n}} \sum_{y_1 \in \{0,1\}} e^{2\pi i (0.a_n) y_1} |y_1\rangle \cdots \sum_{y_n \in \{0,1\}} e^{2\pi i (0.a_1 \cdots a_n) y_k} |y_k\rangle \\
&= \frac{1}{\sqrt{2^n}} \left(|0\rangle + e^{2\pi i 0.a_n} |1\rangle\right) \cdots \left(|0\rangle + e^{2\pi i 0.a_1 \cdots a_n} |1\rangle\right) . \qquad (2.2)
\end{aligned}$$

2.2.1 Circuit construction

From the product form of 2.2 it is straightforward to derive a circuit to compute the QFT mod 2^n. We label the phase gate as

$$P_j = \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^j} \end{bmatrix} .$$

A circuit to implement the QFT mod 2^n is given in Figure 2.1.

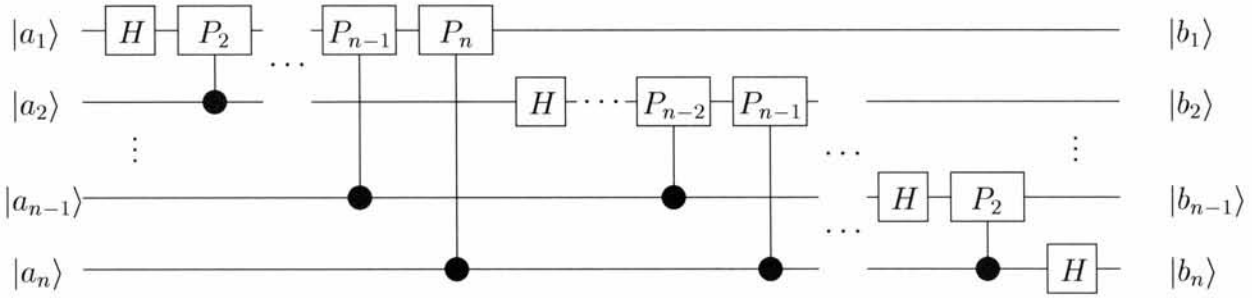

Figure 2.1: Standard circuit for the QFT mod 2^n.

In Figure 2.1 $|b_j\rangle = 1/\sqrt{2^n}\,(|0\rangle + e^{2\pi i 0.a_j \cdots a_n}\,|1\rangle)$. With this circuit construction the order of the output qubits is reversed from the factorization 2.2. This is not a problem as we could either just interpret the output qubits as being reversed or swap the order of the qubits.

To show the correctness of the circuit we step through the application of the gates. On input $|a_1 a_2 \cdots a_n\rangle$, applying a Hadamard to the first qubit gives

$$\frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i 0.a_1}\,|1\rangle\right)|a_2 \cdots a_n\rangle\,.$$

Applying P_2 to the first and second qubits gives

$$\frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i 0.a_1 a_2}\,|1\rangle\right)|a_2 \cdots a_n\rangle\,.$$

Continuing to apply the rest of the phase gates until P_n results in the state

$$\frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i 0.a_1 a_2 \cdots a_n}\,|1\rangle\right)|a_2 \cdots a_n\rangle\,.$$

We now apply the Hadamard to the second qubit, and then again apply the phase gates $P_2, \ldots, P_{n-1}$ to give

$$\frac{1}{\sqrt{4}}\left(|0\rangle + e^{2\pi i 0.a_1 a_2 \cdots a_n}\,|1\rangle\right)\left(|0\rangle + e^{2\pi i 0.a_2 \cdots a_n}\,|1\rangle\right)|a_3 \cdots a_n\rangle\,.$$

Continuing in this fashion for each qubit, the resulting state is

$$\frac{1}{\sqrt{2^n}}\left(|0\rangle + e^{2\pi i 0.a_1 a_2 \cdots a_n}|1\rangle\right)\left(|0\rangle + e^{2\pi i 0.a_2 \cdots a_n}|1\rangle\right)\cdots\left(|0\rangle + e^{2\pi i 0.a_n}|1\rangle\right) \tag{2.4}$$

as desired. The circuit construction above leads to a recursive definition of the QFT mod 2^n.

Standard recursive definition for QFT modulo 2^n

1. Apply H to the first qubit.
2. For $j = 2$ to n apply P_j to the 1^{st} and j^{th} qubits.
3. Apply QFT mod 2^{n-1} to the last $n-1$ qubits.

Figure 2.2 shows the circuit for the QFT mod 2^n resulting from the recursive definition.

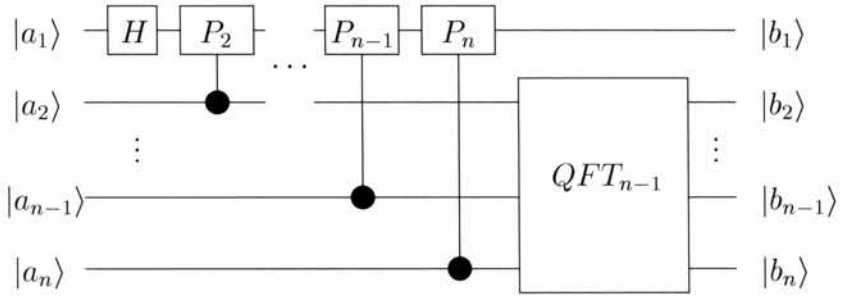

Figure 2.2: Standard recursive circuit for the QFT mod 2^n.

2.2.2 Circuit size

The size of the circuit in Figure 2.2 implementing the QFT mod 2^n is easy to analyze. There are n Hadamard gates, and $\sum_{j=1}^{n-1} j = n(n-1)/2$ phase gates. The total size of the circuit is $O(n^2)$.

2.2.3 Improved circuit construction

The $O(n^2)$ circuit construction for the exact QFT mod 2^n can be improved upon. In [CW00] there is a circuit that improves the bound to $O(n \log^2 n \log\log n)$. To see this improved circuit, we introduce a parameterized definition of the QFT.

Parameterized definition for QFT modulo 2^n

For parameter $m \in \{1, \ldots, n-1\}$

1. Apply QFT mod 2^m to the first m qubits.
2. For $j \in \{1, \ldots, m\}, k \in \{m+1, \ldots, n\}$, apply P_{k-j+1} to the j^{th} and k^{th} qubits.
3. Apply QFT mod 2^{n-m} to the last $n-m$ qubits.

The standard recursive definition for the QFT mod 2^n is just the parameterized definition with parameter $m = 1$. Figure 2.3 shows the circuit construction for the parameterized QFT mod 2^n. We now argue the correctness of the param-

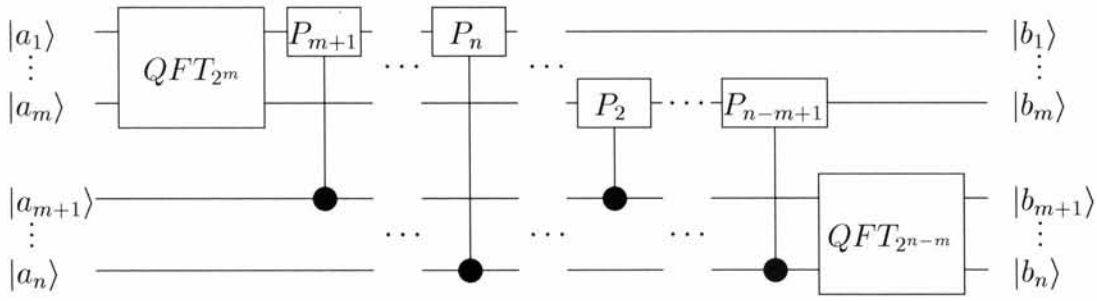

Figure 2.3: Parameterized circuit for the QFT mod 2^n.

eterized construction for the QFT mod 2^n. We do this by re-ordering some of the phase and Hadamard gates from the standard recursive construction. Pick

any $m \in \{1, \ldots, n-1\}$. In the standard recursive circuit (Figure 2.2), there will be a QFT mod 2^{n-m} on the last $n-m$ qubits at the $(m-1)^{th}$ level of the recursion. This is identical to the QFT mod 2^{n-m} on the last $n-m$ qubits in the parameterized construction (Figure 2.3). So all that remains is to show that the gates before the QFT mod 2^{n-m} in the standard recursive construction can be re-ordered to give the first half of the parameterized circuit. Figure 2.4 shows the difference between the standard and parameterized constructions for $n = 4$ and $m = 2$.

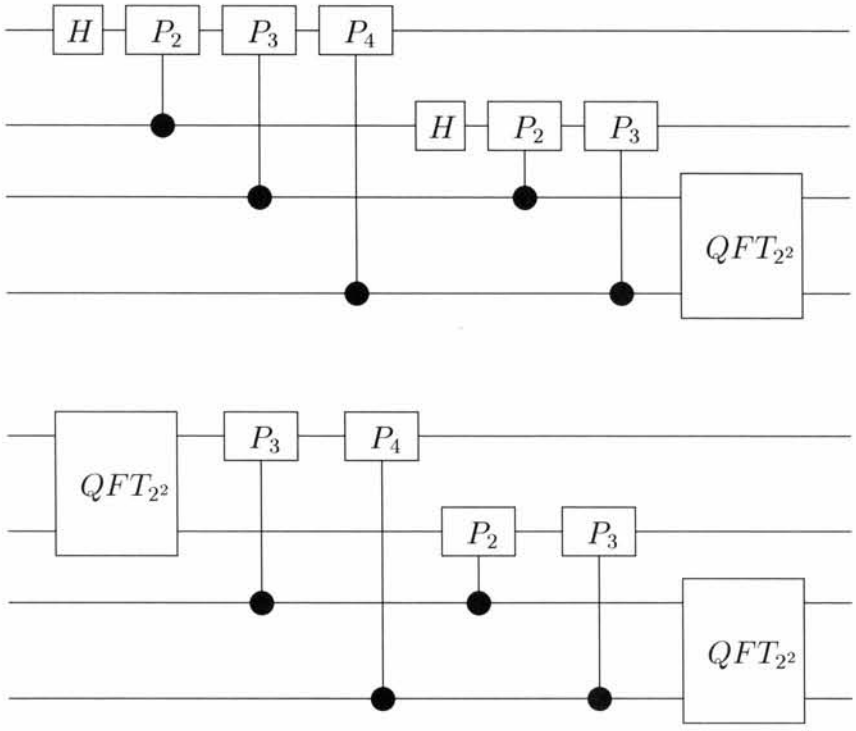

Figure 2.4: The standard recursion construction (above) and the parameterized construction (below) for the QFT mod 2^4 with parameter $m = 2$.

To re-arrange the gates from the standard recursive construction to give the parameterized version, we use the fact that gates which operate on different qubits commute. Look at all gates in the standard recursive construction that are only

applied to the first m qubits. There will be m Hadamard gates, and $\binom{m}{2}$ phase gates, one for each pair of qubits. In Figure 2.4, these are the first two H gates and the first P_2 phase gate. These gates by themselves form a QFT mod 2^m on the first m qubits. Further, if we keep the relative order of these gates, we may shift them to the very left (beginning) of the circuit. When we shift these gates, they are only ever shifted past gates that operate on other qubits. In Figure 2.4, we shift the two H gates, plus the P_2 gate, to the beginning of the circuit, keeping their relative order. (The first H gate and P_2 gate are already there). The result of this shift for our example QFT mod 2^4 is shown in Figure 2.5.

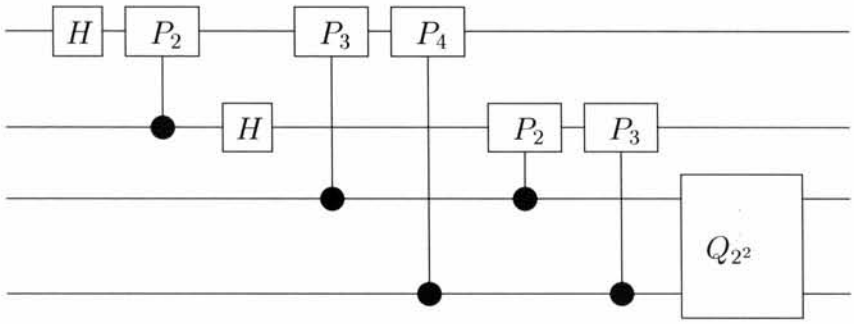

Figure 2.5: The standard recursive QFT mod 2^n with all gates operating on only the first m qubits shifted to the beginning of the circuit. The resulting circuit is identical to the parameterized QFT mod 2^n circuit.

After the shift, there is a QFT mod 2^m on the first m qubits. The phase gates that remain in the middle are phase gates between all pairs of qubits i, j where $i \in \{0, \ldots, m\}$ and $j \in \{m+1, \ldots, n\}$. Then, there is a QFT mod 2^{n-m} on the last $n-m$ qubits. This layout exactly matches the definition of the parameterized circuit construction (Figure 2.3). So we conclude the standard recursive construction and the parameterized construction are equivalent.

The parameterization of the QFT mod 2^n does not by itself improve the circuit size. Recall from the definition of the parameterized construction, we have in Step 2 the phase gates

2. For $j \in \{1, \ldots, m\}, k \in \{m+1, \ldots, n\}$, apply P_{k-j+1} to the j^{th} and k^{th} qubits.

We can treat the first m qubits as register $|x\rangle$ with the m^{th} qubit the most significant, and the next $n-m$ qubits as register $|y\rangle$, with the $(m+1)^{th}$ qubit the most significant. The phase gates in Step 2 then performs the mapping

$$|x\rangle |y\rangle \rightarrow e^{2\pi i xy/2^n} |x\rangle |y\rangle . \tag{2.5}$$

That is, Step 2 essentially multiplies the m-bit integer x and $(n-m)$-bit integer y and stores the result in the phase of the state. So, we may use any integer multiplication algorithm to replace these phase gates and perform the multiplication. Currently the best known integer multiplication algorithm is that of Schönhage and Strassen [SS71], which can multiply two n-bit integers in time $O(n \log n \log\log n)$. We can use this algorithm, plus an n-qubit ancilla register, and in place of the $m(n-m)$ phase gates in Step 2 of the parameterized definition, we do the following:

2.1 Map $|x\rangle |y\rangle |0\rangle$ to $|x\rangle |y\rangle |xy\rangle$.

2.2 Map $|x\rangle |y\rangle |xy\rangle$ to $e^{2\pi i xy/2^n} |x\rangle |y\rangle |xy\rangle$.

2.3 Map $e^{2\pi i xy/2^n} |x\rangle |y\rangle |xy\rangle$ to $e^{2\pi i xy/2^n} |x\rangle |y\rangle |0\rangle$.

Steps 2.1 and 2.3 require a reversible multiplication, mapping $|x\rangle |y\rangle |b\rangle \rightarrow |x\rangle |y\rangle |b \oplus xy\rangle$. Because x is m bits, and y is $(n-m)$ bits, their product will be at most $m+n-m = n$

bits, so an n-qubit ancilla is sufficient. To perform Step 2.2, we need the mapping

$$|xy\rangle \mapsto e^{2\pi ixy/2^n} |xy\rangle . \tag{2.6}$$

This can be easily performed with n phase gates. For each $j \in \{1, \ldots, n\}$, apply P_j to the $(n-j+1)^{th}$ qubit of $|xy\rangle$, where the 1^{st} qubit is the least significant, and the n^{th} the most significant.

If we set the parameter $m = \lfloor \frac{n}{2} \rfloor$, Steps 2.1 and 2.3 can be accomplished in time $O(n \log n \log\log n)$, and Step 2.2 in time $O(n)$. The total cost of the parameterized circuit is given by the recurrence relation

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + O(n \log n \log\log n). \tag{2.7}$$

Solving the recurrence relation gives a circuit size of

$$\begin{aligned} T(n) &= T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + O(n \log n \log\log n) \\ &\leq \sum_{i=1}^{\lceil \log n \rceil} 2^{i-1} O\left(\frac{n}{2^{i-1}} \log \frac{n}{2^{i-1}} \log\log \frac{n}{2^{i-1}} \right) \\ &< \sum_{i=1}^{\log n+1} O\left(n \log \frac{n}{2^{i-1}} \log\log \frac{n}{2^{i-1}} \right) \\ &< O(n \log n \log\log n) \sum_{i=1}^{\log n+1} 1 \\ &\in O(n \log^2 n \log\log n). \end{aligned} \tag{2.8}$$

2.3 Approximate QFT modulo 2^n

As we argued earlier, for real world quantum computations good approximate algorithms will suffice. In [Cop94], Coppersmith shows how to approximate F_{2^n} within ϵ by a quantum circuit of size $O(n \log(n/\epsilon))$. Intuitively, the idea is start with the

$O(n^2)$ standard recursive definition for the QFT mod 2^n in Section 2.2 and then eliminate all c-P_j (controlled-phase) gates where j exceeds some threshold value. As j grows large, the controlled phase gate approaches the identity, so removing the gate will not hurt the computation too much. We formalize the argument below.

2.3.1 Effect of discarded controlled-phase gate

We first establish the error induced by the deletion of a single controlled-phase gate in the QFT computation. To do this we calculate the Euclidean distance between the phase gate and the identity. Let the phase gate be c-P_j, and the error ϵ (induced Euclidean distance) is then

$$\begin{aligned}
\epsilon &= \|I - \text{c-}P_j\| \\
&= \left\| \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{2\pi i/2^j} \end{bmatrix} \right\| \\
&= \left\| \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 - e^{2\pi i/2^j} \end{bmatrix} \right\|.
\end{aligned}$$

The Euclidean distance is maximized for this matrix when $|\psi\rangle = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$, and so

$$\begin{aligned}
\epsilon &= \left\| \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 - e^{2\pi i/2^j} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \right\| \\
&= \left\| \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 - e^{2\pi i/2^j} \end{bmatrix} \right\| \\
&= \left| 1 - e^{2\pi i/2^j} \right| \\
&\leq 2\pi/2^j.
\end{aligned}$$

The last step in the derivation is justified by the fact that the value is just the length of the chord with end points 1 and $e^{2\pi i/2^j}$ on the unit circle. This chord length is upper bounded by the length of the arc between these two points, which is $2\pi/2^j$.

2.3.2 Derivation of approximate circuit

To achieve the improved circuit size and error bound we may sum the induced error from each discarded controlled-phase gates. Table 2.6 shows the frequency of each phase gate in the standard QFT, and the error each induces if we were to eliminate it.

Controlled-Phase Gate	Induced Error	Frequency in Circuit
c-P_n	$2\pi/2^n$	1
c-P_{n-1}	$2\pi/2^{n-1}$	2
c-P_{n-2}	$2\pi/2^{n-2}$	3
$\vdots$	$\vdots$	$\vdots$
c-P_1	$2\pi/2$	$n-1$

Figure 2.6: Frequency and error of each controlled phase gate in the standard QFT mod 2^n.

We set a threshold l, and keep all phase gates c-P_1, c-P_2,...,c-P_l. We discard all phase gates c-P_{l+1},...,c-P_n. Discarding these phase gates induces an error upper bounded by

$$\begin{aligned} \epsilon &\leq \sum_{j=1}^{n-l} j2\pi/2^{n-j+1} \\ &< \pi n/2^n \sum_{j=1}^{n-l} 2^j \\ &< \frac{2\pi n}{2^l}. \end{aligned} \tag{2.9}$$

Solving 2.9 for l yields

$$\begin{aligned} 2^l &< \frac{2\pi n}{\epsilon} \\ l &\in O\left(\log \frac{n}{\epsilon}\right). \end{aligned} \tag{2.10}$$

With an error ϵ of any constant or as small as an inverse polynomial $\frac{1}{n^c}$ for constant c, $l \in O(\log n)$. To find the size of the approximate QFT circuit, we sum up

the n Hadamard gates plus the number remaining phase gates.

$$\begin{aligned} n + \sum_{j=n-O(\log \frac{n}{\epsilon})}^{n-1} j &< n + \sum_{j=1}^{n-\left(n-O(\log \frac{n}{\epsilon})\right)} j + n - O\left(\log \frac{n}{\epsilon}\right) \\ &\leq n + \sum_{j=1}^{O(\log \frac{n}{\epsilon})} j + \sum_{j=1}^{O(\log \frac{n}{\epsilon})} n + \sum_{j=1}^{O(\log \frac{n}{\epsilon})} O\left(\log \frac{n}{\epsilon}\right) \\ &\in O\left(n \log \frac{n}{\epsilon}\right) \end{aligned} \tag{2.11}$$

So the total circuit size is $O(n \log (n/\epsilon))$ to approximate the QFT mod 2^n within ϵ, which corresponds to the results in [Cop94].

2.4 Phase estimation

The phase estimation algorithm is a very important tool in quantum computing. It was introduced in [Kit95] and [CEMM98] contains a good overview of the algorithm. In conjunction with the QFT mod 2^n, it can be used to solve integer factoring, order finding, and many other problems. We provide a brief survey of the algorithm here.

The phase estimation procedure computes the eigenvalue corresponding to an eigenvector of a unitary operator.

Phase Estimation

Input: An n-qubit unitary operator U, and an n-qubit eigenvector $|\psi\rangle$ of U.

Output: An m-bit approximation to θ, where the corresponding eigenvector of $|\psi\rangle$ is $e^{2\pi i\theta}$. The algorithm computes the mapping

$$|0\rangle^{\otimes m} |\psi\rangle \mapsto |\theta\rangle |\psi\rangle \tag{2.12}$$

with high probability, and includes a measurement of the first register at the end. From [CEMM98], setting $m = 2n$ will guarantee the first n-bits of θ are correct with

probability $1 - O(1/2^n)$. The cost of the algorithm is the sum of the cost of an inverse QFT mod 2^m, and the cost of performing the controlled-U operation

$$|x\rangle |y\rangle \mapsto |x\rangle U^x |y\rangle \tag{2.13}$$

for an m-qubit register $|x\rangle$ and n-qubit register $|y\rangle$. That is, applying the unitary U x times to $|y\rangle$ for any x. In general this may be difficult, but for certain unitary operators U this is possible.

For our purposes we need to use a unitary version of phase estimation without the measurement. From ideas in [BBBV97], the measurement can be removed and it can be shown that the phase estimation algorithm will give a good approximation to (2.12). That is, if we let PE be the phase estimation algorithm without measurement, $\| PE |0\rangle^{\otimes m} |\psi\rangle - |\theta\rangle |\psi\rangle \| \leq O(1/2^n)$.

2.5 Amplitude amplification

One of the two general purpose algorithms in quantum computing is the so-called database search algorithm of Grover [Gro96] The other is of course Shor's factoring algorithm [Sho94]. In a more general form it is known as amplitude amplification [BH97]. In this thesis we require a limited version of the algorithm only. The problem may be described as follows. We are given a quantum register holding an equally weighted superposition of N elements (from 0 to $N-1$). There are M "good" elements in the superposition, and $N - M$ "bad" elements. We do not know which of the elements are good or bad, but we have the ability to recognize a good element with the use of an oracle. We wish to measure the register and get a good state with certainty (or high probability). We let the initial superposition be $|\psi\rangle = U |0\rangle =$

$\frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle$ (the unitary operator U is used to create the superposition). Let the set $A \subseteq N$ be all good states, and $B = N - A$ be all bad states. Then define two states, one an evenly weighed superposition of all good states, and another of all the bad states, as

$$|a\rangle = \frac{1}{\sqrt{M}} \sum_{s \in A} |s\rangle \tag{2.14}$$

$$|b\rangle = \frac{1}{\sqrt{N-M}} \sum_{t \in B} |t\rangle \,. \tag{2.15}$$

It is possible to rewrite the initial state $|\psi\rangle$ in terms of $|a\rangle$ and $|b\rangle$:

$$|\psi\rangle = \frac{1}{\sqrt{N/M}} |a\rangle + \frac{1}{\sqrt{N/(N-M)}} |b\rangle \tag{2.16}$$

We then define the unitary operation I_f, our oracle which recognizes good solutions, as the mapping:

$$I_f |x\rangle = \begin{cases} -|x\rangle & \text{if} \quad x \in A \\ |x\rangle & \text{otherwise} \end{cases}$$

and define the mapping I_0 as

$$I_0 |x\rangle = \begin{cases} -|x\rangle & \text{if} \quad x = 0 \\ |x\rangle & \text{otherwise} \end{cases} .$$

We let θ be the angle between $|\psi\rangle$ and $|b\rangle$, then

$$\langle b \,|\, \psi \rangle = \frac{1}{\sqrt{N/(N-M)}} = \cos\theta$$

and

$$\langle a \,|\, \psi \rangle = \frac{1}{\sqrt{N/M}} = \sin\theta.$$

Define a Grover iterate as

$$G = -U I_0 U^\dagger I_f. \tag{2.17}$$

Each application of G to $|\psi\rangle$ rotates $|\psi\rangle$ toward $|a\rangle$ by an angle of 2θ. That is,

$$\langle a \,|\, (-UI_oU^\dagger I_f)^k \,|\, \psi \rangle = \sin((2k+1)\theta). \tag{2.18}$$

Of particular interest to the algorithms in this paper is when $N/M = 4$. Then $\langle a \,|\, \psi \rangle = 1/2 = \sin \pi/6$. So one application of the Grover iterate will rotate the state such that $\langle a \,|\, \psi \rangle = \sin \pi/2 = 1$. That is, when there is a probability of $M/N = 1/4$ of measuring a "good" state, one application of the Grover iterate will rotate the state such that the probability of measuring a "good" state is now 1. The cost for the amplitude amplification procedure is just the cost of the Grover iterate G.

Chapter 3

Improved Circuits for Approximate Quantum Fourier Transforms

In this chapter we present our improved algorithm for the approximate quantum Fourier transform (QFT) mod 2^n, which outperforms the previously best known circuit in [Cop94]. We then present a different algorithm which computes the approximate QFT for an arbitrary modulus q with a circuit size equal to the best known bound from [HH00]. We then relate the difficulty of computing the QFT mod q with the classical problems of integer multiplication and division. We do this by giving two reductions: a reduction from integer multiplication to computing the approximate QFT mod q, and a reduction from division to constructing a quantum Fourier basis state, for a modulus q.

3.1 Improved approximate QFT modulo 2^n

We now introduce our improved approximate QFT mod 2^n. We show the following theorem.

Theorem 3.1 *The approximate QFT modulo 2^n on n qubits can be performed by a quantum circuit of size $O(n(\log\log(n/\epsilon))^2 \log\log\log(n/\epsilon))$ with error at most ϵ.*

We prove the theorem by first describing the algorithm, and then analyzing both the error and circuit size in Sections 3.1.2 and 3.1.3.

Our approach is a combination of the ideas behind the constructions in Sections 2.2.3 and 2.3. Specifically, we start with the parameterized recursive construction of the exact QFT mod 2^n from Section 2.2.3. In Steps 2.1 and 2.3, instead of performing an exact multiplication of the registers $|x\rangle$ and $|y\rangle$, we approximately compute the product. This reduces the cost of the multiplication but also introduces an error. We show that the trade-off can be adjusted to yield a circuit of size $O(n(\log\log(n/\epsilon))^2 \log\log\log(n/\epsilon))$. While the improvement over the $O(n\log(n/\epsilon))$ version is small, it is a step closer to the goal of computing the QFT in linear time. Currently, no known lower bound exists for the QFT, other than perhaps the obvious one of $\Omega(n)$, since we need to look at all input qubits.

3.1.1 Approximate multiplication

At the top level of the recursion we multiply x, an $m = \lfloor \frac{n}{2} \rfloor$-bit number with y, an $n - m = \lceil \frac{n}{2} \rceil$-bit number. We set a threshold value l, and we use only the most significant l bits of x and y, as in Figure 3.1. This value l is constant for all recursive computations and is not recalculated. In the recursive calls when the lengths of x and y become less than l, we simply use the exact QFT constructions, multiplying all bits of x and y. Figure 3.2 shows the improved QFT construction and the approximate multiplication at the top level of the recursion.

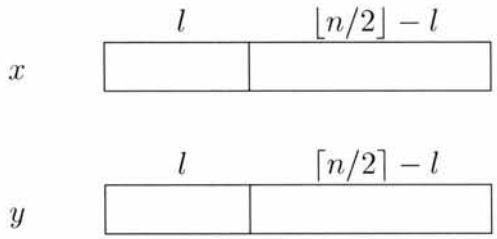

Figure 3.1: The l highest-order bits of x and y are used.

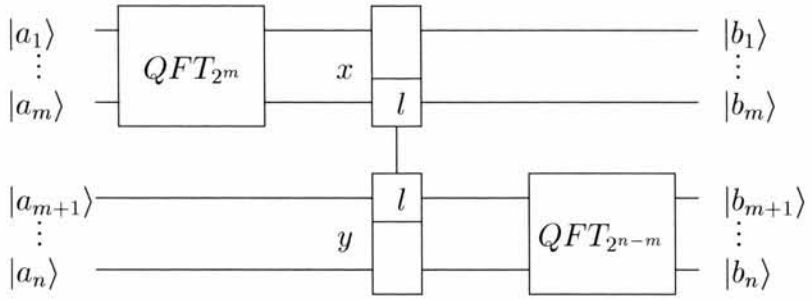

Figure 3.2: The improved QFT construction with an approximation multiplication, represented by the middle gate, on the l highest-order bits of each register.

To calculate the error incurred by these approximate multiplications, let x and y be the two $n/2$-bit numbers to be multiplied at the highest level of the recursion. We assume for simplicity that n is even. Let $\tilde{x}$ be x with all but its l high order bits truncated to 0 (thus, $|x - \tilde{x}| < 2^{n/2-l}$). Define $\tilde{y}$ similarly. Then

$$|xy - \tilde{x}\tilde{y}| = |x(y - \tilde{y}) + (x - \tilde{x})\tilde{y}| < (|x| + |y|)2^{n/2-l} < 2^{n-l+1}. \tag{3.1}$$

It follows that if only l high-order bits are used in Step 2, which maps $|x\rangle\,|y\rangle$ to $e^{2\pi i xy/2^n}\,|x\rangle\,|y\rangle$, then the maximum error in the phase is

$$e^{2\pi i 2^{n-l+1}/2^n} = e^{4\pi i/2^l}. \tag{3.2}$$

This implies that the approximation induces a Euclidean distance error of at most

$$4\pi/2^l. \tag{3.3}$$

3.1.2 Error analysis

The error bound 3.3 also applies to subsequent occurrences of Step 2 in the recursive execution of the algorithm. In each such occurrence, the integers x and y are smaller than $n/2$ bits. Since l is fixed, the error incurred by approximating the product can

only decrease, because we discard fewer and fewer bits of x and y. The recursion is $\lceil \log n/2 \rceil$ levels deep, and at the i^{th} level of the recursion there are 2^i multiplications. Therefore, the cumulative error ϵ is upper bounded by

$$\epsilon = \sum_{i=0}^{\lceil \log (n/2) \rceil} 2^i \frac{4\pi}{2^l} < \frac{4\pi n}{2^l}. \tag{3.4}$$

Solving 3.4 for l results in

$$l = \lceil \log(4\pi n/\epsilon) \rceil. \tag{3.5}$$

Setting $l \in O(\log n/\epsilon)$ suffices to approximate within ϵ.

3.1.3 Circuit size

We next analyze the size of our improved circuit when $l = O(\log(n/\epsilon))$. This analysis will be broken into two parts: the first $\lceil \log (n/2) \rceil - \lceil \log l \rceil$ levels of the recursion, where l-bit multiplications occur; and the last $\lceil \log l \rceil$ levels of the recursion, where $n/2 < l$ and so $n/2$-bit multiplications occur. The cost of the first part is

$$\begin{aligned} \sum_{i=0}^{\lceil \log (n/2) \rceil - \lceil \log l \rceil} 2^i \cdot O(l \log l \log\log l) &\leq (n/l) \cdot O(l \log l \log\log l) \\ &= O\left(n \log\log(n/\epsilon) \log\log\log(n/\epsilon)\right). \end{aligned}$$

The cost of the second part is

$$\begin{aligned} \sum_{i=0}^{\lceil \log l \rceil} 2^{i+\lceil \log (n/2) \rceil - \lceil \log l \rceil} \cdot O\left(\frac{l}{2^i} \log \frac{l}{2^i} \log\log \frac{l}{2^i}\right) &\leq \frac{n}{l} \sum_{i=1}^{\lceil \log l \rceil} 2^i \cdot O\left(\frac{l}{2^i} \log \frac{l}{2^i} \log\log \frac{l}{2^i}\right) \\ &= n \sum_{i=1}^{\lceil \log l \rceil} O\left(\log \frac{l}{2^i} \log\log \frac{l}{2^i}\right) \\ &\leq n \sum_{i=1}^{\log l + 1} O(\log l \log\log l) \\ &= O\left(n \left(\log\log(n/\epsilon)\right)^2 \log\log\log(n/\epsilon)\right). \end{aligned}$$

Thus, the total circuit size is $O\left(n\left(\log\log(n/\epsilon)\right)^2 \log\log\log(n/\epsilon)\right)$.

3.2 Approximate QFT modulo q

We now focus our attention on the approximate QFT mod q. Hales and Hallgren [HH00] have a construction that requires $O(n \log n \log\log n)$ gates when $\epsilon \geq 1/n^{O(1)}$. We provide a construction based on the method of Kitaev [Kit95] which gives the same bound. In algorithms we often abbreviate the QFT mod q as F_q.

Theorem 3.2 *The approximate QFT modulo q on n qubits (for $q < 2^n$) may be performed with a quantum circuit of size $O(n(\log\log(n/\epsilon))^2 \log\log\log(n/\epsilon))$ plus the cost of an n-bit integer multiplication and division, with error at most ϵ*

We show the theorem by describing the algorithm and analyzing the error in Sections 3.2.1, 3.2.2, and 3.2.3. To begin, we divide the problem into two halves. The first will be called the quantum Fourier state (QFS) calculation, and will map an input vector $|j\rangle$ and an ancilla register $|0\rangle$ as

$$|j\rangle\,|0\rangle \mapsto |j\rangle\,|\psi_j\rangle \tag{3.6}$$

where

$$|\psi_j\rangle = F_q(|j\rangle) = \frac{1}{\sqrt{q}} \sum_{k=0}^{q-1} e^{2\pi i j k/q}\,|k\rangle\,. \tag{3.7}$$

The second half will be called the quantum Fourier phase (QFP) calculation, and will map the input vector and a Fourier state as

$$|j\rangle\,|\psi_j\rangle \mapsto |0\rangle\,|\psi_j\rangle\,. \tag{3.8}$$

3.2.1 Quantum Fourier state construction

As input we receive a state $|j\rangle$, which we will assume to be some basis vector in $|0\rangle, \ldots, |q-1\rangle$ for simplicity. By linearity we may extend the argument to include any input state. The first step will involve constructing an evenly weighted superposition over q elements.

To do this, we use an n-qubit quantum register, initialized to $|0\rangle^{\otimes n}$, and assume $2^{n-1} < q < 2^n$. In addition we will require three ancilla qubits, two assumed to be in state $|0\rangle$ and one in state $|-\rangle = (|0\rangle - |1\rangle)/\sqrt{2}$.

$$|0\rangle^{\otimes n} |00\rangle |-\rangle . \tag{3.9}$$

We then apply a Hadamard gate to each of the first n bits, yielding the state

$$H^{\otimes n} |0\rangle^{\otimes n} |00\rangle |-\rangle = \left(\frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} |k\rangle \right) |00\rangle |-\rangle . \tag{3.10}$$

The superposition is now equally weighted over $2^n > q$ elements. To transform this into a superposition of size q, we use the amplitude amplification technique reviewed in Section 2.5. If we were to measure the first register, the probability of measuring a state $|k\rangle$ with $0 \leq k < q$ is

$$q \frac{1}{2^n} \tag{3.11}$$

which is greater than $1/2$ and less than 1. We adjust the probability with a rotation gate, so the probability of measuring a good state is exactly $1/4$. A single application of amplitude amplification will then yield the equally weighted superposition of q elements with certainty.

The first ancilla qubit will be used to mark if k is smaller than q. This is performable in linear time, by subtracting k from q and checking if the result is positive

or negative. Doing this check yields the state

$$\left(\frac{1}{\sqrt{2^n}}\sum_{k=0}^{q-1}|k\rangle|1\rangle+\frac{1}{\sqrt{2^n}}\sum_{k=q}^{2^n-1}|k\rangle|0\rangle\right)|0\rangle|-\rangle\,. \tag{3.12}$$

We now apply a rotation gate to the second ancilla qubit. This produces the state

$$\left(\frac{1}{\sqrt{2^n}}\sum_{k=0}^{q-1}|k\rangle|1\rangle+\frac{1}{\sqrt{2^n}}\sum_{k=q}^{2^n-1}|k\rangle|0\rangle\right)(\alpha|1\rangle+\beta|0\rangle)|-\rangle\,. \tag{3.13}$$

The values of α and β above still must be calculated. We wish to use one Grover iteration to find the states whose ancilla qubits are $|11\rangle$, as those are the "good" states in that they are an evenly weighted superposition over q elements. One Grover iteration will produce these "good" states with certainty when the probability of measuring a "good" state is 1/4. The probability of measuring a $|11\rangle$ in the state (3.13) is $q\cdot\frac{\alpha^2}{2^n}$. Setting this equal to 1/4 and solving for α gives

$$\alpha=\sqrt{\frac{2^n}{4q}} \tag{3.14}$$

which can be verified to be greater than 0 and less than 1. We then set $\beta=\sqrt{1-\alpha^2}$ in the rotation matrix above.

For convenience we will label the transformation W that creates the state (3.13) from $|0\rangle^{\otimes n}|00\rangle|-\rangle$, and note that W requires number of gates linear in n. The values of α and β in the rotation matrix can be calculated classically beforehand, since they are fixed constants in the circuit.

A Grover iteration has the form $G=-WI_0W^\dagger I_F$. We have already defined W above. The transformation I_F is the oracle that tests if a particular state is desired or not, and injects that value into the phase of the state.

$$I_F|k\rangle=\begin{cases}-|k\rangle & \text{if the first two ancilla are } \quad |11\rangle\\ |k\rangle & \text{otherwise}\end{cases}$$

This is easy to accomplish by using a Toffoli whose two inputs are the first two ancilla, and whose output inverts the third ancilla which is in the $|-\rangle$ state.

Finally, the I_0 transformation flips the phase of the state $|k\rangle$ if and only if $k = 0$, i.e. $I_0 |k\rangle = -|k\rangle$ iff $k = 0$. Checking if an n-bit register is 0 is possible using a linear number of gates in n, since each bit must be checked.

With each transformation in the Grover iteration defined, we now apply the iteration, which will find all states with $|11\rangle$ as their ancilla bits with certainty. Each operator in the Grover iteration is implementable in at most $O(n)$ gates, and so the entire transformation takes $O(n)$ gates as well. At the end, the two ancilla qubits are in state $|11\rangle$, and so the first two registers are in the state

$$|j\rangle \frac{1}{\sqrt{q}} \sum_{k=0}^{q-1} |k\rangle . \tag{3.15}$$

All the remains for the quantum Fourier state calculation is to inject the phase $e^{2\pi ijk/q}$ into the state. We use the classical integer multiplication algorithm of Schönhage and Strassen [SS71] to do this in the same fashion as the QFT mod 2^n, using an $2n$-qubit ancilla:

1. Map $|j\rangle |k\rangle |0\rangle$ to $|j\rangle |k\rangle |jk\rangle$.

2. Apply a phase gate of $e^{2\pi i(2^{2n-m}/q)}$ to the m^{th} qubit of the ancilla for $1 \leq m \leq 2n$ to map $|j\rangle |k\rangle |jk\rangle$ to $e^{2\pi ijk/q} |j\rangle |k\rangle |jk\rangle$.

3. Map $|j\rangle |k\rangle |jk\rangle$ to $|j\rangle |k\rangle |0\rangle$.

We now have the desired quantum Fourier state

$$|j\rangle \frac{1}{\sqrt{q}} \sum_{k=0}^{q-1} e^{2\pi ijk/q} |k\rangle . \tag{3.16}$$

The cost for this last step is just the cost of classical n-bit integer multiplication. So we conclude the total cost for the quantum Fourier state calculation is $O(n \log n \log\log n)$, with the limiting step being classical multiplication. We note that the QFS calculation is exact (no error) and succeeds with certainty.

3.2.2 Quantum Fourier phase

The quantum Fourier phase transformation maps the state $|j\rangle |\psi_j\rangle$ as follows:

$$|j\rangle |\psi_j\rangle \mapsto |0\rangle |\psi_j\rangle . \tag{3.17}$$

Looking at the transformation in reverse, we show that it is essentially just phase estimation. The algorithm for phase estimation may then be used in reverse for our purposes as a quantum Fourier phase transformation.

From Section 2.4, the phase estimation procedure takes an n-qubit unitary operator U, an eigenvector $|\psi\rangle$ of U, and a parameter m. For our purposes we set the parameter $m = 2n$. It produces, with error at most $O(1/2^n)$, the first n bits of an estimate to θ (where $e^{2\pi i\theta}$ is the corresponding eigenvalue of $|\psi\rangle$). The states $|\psi_j\rangle$ are eigenvectors of U, where

$$U |k\rangle = |k - 1 \mod q\rangle , \tag{3.18}$$

with corresponding eigenvalues $e^{2\pi ij/q}$. This can be verified by simply applying U

to $|\psi_j\rangle$:

$$\begin{aligned} U|\psi_j\rangle &= U\frac{1}{\sqrt{q}}\sum_{k=0}^{q-1} e^{2\pi ijk/q}|k\rangle \\ &= \frac{1}{\sqrt{q}}\sum_{k=0}^{q-1} e^{2\pi ijk/q}|k-1\rangle \\ &= \frac{1}{\sqrt{q}}\sum_{k=0}^{q-1} e^{2\pi ij(k+1)/q}|k\rangle \\ &= \frac{1}{\sqrt{q}}\sum_{k=0}^{q-1} e^{2\pi ij/q}e^{2\pi ijk/q}|k\rangle \\ &= e^{2\pi ij/q}|\psi_j\rangle\,. \end{aligned} \tag{3.19}$$

So, if we input U and $|\psi_j\rangle$ into the phase estimation algorithm, we will get a good approximation to j/q. We multiply by j/q by q to obtain j. With U and $|\psi_j\rangle$ as defined, the phase estimation procedure unitarily performs the mapping

$$|0\rangle|\psi_j\rangle \mapsto |j/q\rangle|\psi_j\rangle \tag{3.20}$$

with error at most $O(1/2^n)$. So multiplying the first register by q and then performing phase estimation in reverse will perform the desired quantum Fourier phase mapping 3.17.

The cost of the quantum Fourier phase calculation is then the sum of a multiplication by q and the phase estimation procedure. Phase estimation costs include performing the controlled-U operation,

$$U|x\rangle|y\rangle \mapsto |x\rangle U^x|y\rangle \tag{3.21}$$

for $m = 2n$-qubit register x and n-qubit register y, and the inverse QFT mod 2^m. Since one application of U subtracts 1 from a register, applying U to $|y\rangle$ x times is

easy: we subtract x from y, modulo q. Because x is $2n$-bits and y and q only n-bits, we perform a division to determine $x \mod q$, and then subtract it from y modulo q, to perform the controlled-U. From [vzGG99] division can be performed in time equal to multiplication: $O(n \log n \log\log n)$. For the inverse QFT mod 2^m, we use our improved approximate QFT mod 2^n presented earlier. Setting the error for this to $1/n^{O(1)}$, the approximate QFT mod 2^m costs $O\left(n\left(\log\log n\right)^2 \log\log\log n\right)$.

So we perform the quantum Fourier phase transformation in time

$$O\left(n\left(\log\log n\right)^2 \log\log\log n + n \log n \log\log n\right) = O(n \log n \log\log n) \tag{3.22}$$

with error at most $\epsilon \leq O(1/2^n) + O(1/n^{O(1)}) \in O(1/n^{O(1)})$.

3.2.3 Complete approximate QFT modulo q

To perform the approximate QFT mod q, we just perform the quantum Fourier state calculation, followed by the quantum Fourier phase calculation. The size of the complete circuit for the QFT mod q is bounded by the size of the approximate QFT mod 2^n plus the cost of an n-bit multiplication and an $2n$-bit division. For error $\in O(1/n^{O(1)})$, the total size of the circuit is

$$O\left(\left(n\left(\log\log n\right)^2 \log\log\log n\right) + n \log n \log\log n\right). \tag{3.23}$$

which is $O(n \log n \log\log n)$. We remark that the integer multiplication and division steps are the limiting factors in performing the approximate QFT mod q.

3.3 Approximate QFTs modulo q and integer multiplication

With a near linear time algorithm for the approximate QFT mod 2^n, it is natural to ask whether these bounds extend to the arbitrary modulus QFT. We have just

provided an algorithm for the approximate QFT mod q that performs the transform in time $O(n \log n \log\log n)$, but the question remains: can we do better? In this section, we provide a reduction from integer multiplication to computing the QFT mod q, where q is an arbitrary n-bit modulus. We show the following theorem.

Theorem 3.3 *If there is a quantum circuit of size $o(n \log n \log\log n)$ that computes the approximate QFT mod q (for $2^{n-1} < q < 2^n$) then there is also a quantum circuit of size $o(n \log n \log\log n)$ that multiplies two n-bit integers with success probability $1 - 1/n^{O(1)}$.*

Such a quantum circuit would be smaller than the most efficient classical multiplication circuit that is known [SS71], which dates back to 1971. This can be either interpreted as evidence that a circuit size of $o(n \log n \log\log n)$ is unlikely for the arbitrary modulus case, or that there may be improved multiplication algorithms based on quantum information. We show the theorem by describing the reduction and its correctness in the following section.

3.3.1 Reduction to the QFT modulo q

Suppose we have two n-bit integers, a and b, and we wish to compute their product ab. Perform the following:

On input of three n-qubit registers $|a\rangle |b\rangle |0\rangle$,

1. Perform the inverse QFT mod 2^n on $|b\rangle$.

2. Create an equally weighted superposition of non-negative integers less than a.

3. Map the last two registers $|j\rangle |k\rangle$ to $|k2^n + j\rangle$.

4. Perform the QFT mod $a2^n$ on the now-combined second and third register.

We show that after performing these steps, we will get $|ab\rangle$ with certainty. Step 1 gives the state

$$|a\rangle \left(\frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} e^{-2\pi ijb/2^n} |j\rangle \right) |0\rangle . \tag{3.24}$$

We now create the superposition over a elements on the last register. This may be accomplished in linear time using amplitude amplification, as was done in Section 3.2.1 for the construction of a quantum Fourier basis state.

$$|a\rangle \left(\frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} e^{-2\pi ijb/2^n} |j\rangle \right) \left(\frac{1}{\sqrt{a}} \sum_{k=0}^{a-1} |k\rangle \right) . \tag{3.25}$$

Now combine the last two n-qubit registers into one $2n$-qubit register. This step is easy. We can either just interpret the second register holding $|k\rangle$ as the most significant, or swap the two registers so $|k\rangle$ comes first, and its bits are from $n+1$ to $2n$. Either way, the desired effect is achieved.

$$|a\rangle \left(\frac{1}{\sqrt{a2^n}} \sum_{j=0}^{2^n-1} \sum_{k=0}^{a-1} e^{-2\pi ijb/2^n} |k2^n + j\rangle \right) \tag{3.26}$$

Now substituting $l = k2^n + j$ to combine the summations gives $j = l - k2^n$ and 3.26 is equal to

$$\begin{aligned}
|a\rangle \left(\frac{1}{\sqrt{a2^n}} \sum_{l=0}^{a2^n-1} e^{-2\pi i(l-k2^n)b/2^n} |l\rangle \right) &= |a\rangle \left(\frac{1}{\sqrt{a2^n}} \sum_{l=0}^{a2^n-1} e^{2\pi ik2^n b/2^n} e^{-2\pi ilb/2^n} |l\rangle \right) \\
&= |a\rangle \left(\frac{1}{\sqrt{a2^n}} \sum_{l=0}^{a2^n-1} e^{2\pi ikb} e^{-2\pi ilb/2^n} \cdot |l\rangle \right) \\
&= |a\rangle \left(\frac{1}{\sqrt{a2^n}} \sum_{l=0}^{a2^n-1} e^{-2\pi ilb/2^n} |l\rangle \right) \\
&= |a\rangle \left(\frac{1}{\sqrt{a2^n}} \sum_{l=0}^{a2^n-1} e^{-2\pi ilab/(a2^n)} |l\rangle \right) . \qquad (3.27)
\end{aligned}$$

Now look carefully at the state of the second register in 3.27. It is just the inverse QFT mod $a2^n$ applied to $|ab\rangle$. So if we apply the QFT mod $a2^n$ on the state, we recover $|ab\rangle$ exactly.

$$|a\rangle\,|ab\rangle \tag{3.28}$$

To multiply with this method, we used an inverse QFT mod 2^n, amplitude amplification, and the QFT mod $a2^n$. As we stated, amplitude amplification runs in time $O(n)$. For the uses of the QFT, if we use an approximate QFT for both, we will not get $|ab\rangle$ with certainty, but will do so with high probability. So if we can perform the approximate QFT mod q in time $o(n \log n \log\log n)$, we can use this method to multiply integers more quickly than the best known classical method.

3.4 Quantum Fourier state construction and integer division

In the previous section, we provided a reduction from integer multiplication to the approximate QFT mod q. We show in this section that a full approximate QFT mod q is not necessary to for a reduction from multiplication. Rather, it is the case that the problem of constructing a quantum Fourier basis state is sufficient. We show the following theorem.

Theorem 3.4 *If there is a quantum circuit of size $o(n \log n \log\log n)$ that constructs the quantum Fourier basis state $|\psi_a\rangle = F_q\,|a\rangle$ (for $2^{n-1} < q < 2^n$) then there is also a quantum circuit of size $o(n \log n \log\log n)$ that divides two n-bit integers with success probability $1 - 1/n^{O(1)}$.*

This is significant because it could be that constructing a quantum Fourier basis state is an easier problem than computing a full QFT. We show the theorem by two

remarkably different reductions. In Appendix B we show that division can be used to multiply.

3.4.1 Reduction with phase estimation

The first reduction from division to quantum Fourier basis state construction uses the phase estimation algorithm. This is very similar to the quantum Fourier phase calculation of Section 3.2.2.

Suppose we have two n-bit integers a, b, and we wish to compute an m-bit approximation to a/b (assume for now that $a < b$). We begin by constructing the quantum Fourier basis state $|\psi_a\rangle$.

$$\begin{aligned} |\psi_a\rangle &= F_{b2^{m'-n}} \left| a \cdot 2^{m'-n} \right\rangle \\ &= \frac{1}{\sqrt{b2^{m'-n}}} \sum_{k=0}^{b2^{m'-n}-1} e^{2\pi i k a 2^{m'-n}/b2^{m'-n}} |k\rangle \\ &= \frac{1}{\sqrt{b2^{m'-n}}} \sum_{k=0}^{b2^{m'-n}-1} e^{2\pi i k a/b} |k\rangle \end{aligned} \tag{3.29}$$

We define m' later. We now use $|\psi_a\rangle$ as the eigenvector in the input of the phase estimation algorithm. The unitary operator

$$U|k\rangle = \left| k - 1 \mod b2^{m'-n} \right\rangle \quad \text{for} \quad 0 \leq k < b2^{m'-n} \tag{3.30}$$

has eigenvectors $|\psi_a\rangle$ with corresponding eigenvalues $e^{2\pi ia/b}$, since

$$\begin{aligned}
U|\psi_a\rangle &= U\frac{1}{\sqrt{b2^{m'-n}}}\sum_{k=0}^{b2^{m'-n}-1} e^{2\pi ika/b}|k\rangle \\
&= \frac{1}{\sqrt{b2^{m'-n}}}\sum_{k=0}^{b2^{m'-n}-1} e^{2\pi ika/b}U|k\rangle \\
&= \frac{1}{\sqrt{b2^{m'-n}}}\sum_{k=0}^{b2^{m'-n}-1} e^{2\pi ika/b}\left|k-1 \mod b2^{m'-n}\right\rangle \\
&= \frac{1}{\sqrt{b2^{m'-n}}}\sum_{k=0}^{b2^{m'-n}-1} e^{2\pi i(k+1)a/b}|k\rangle \\
&= e^{2\pi ia/b}|\psi_a\rangle .
\end{aligned}$$

This operator will be used as the controlled-U operator in the phase estimation algorithm. If we use m' bits as the control for phase estimation, the controlled-U can be performed by subtracting the control register from the target register modulo $b2^{m'-n}$, since

$$|x\rangle U^x|y\rangle = |x\rangle\left|y-x \mod b2^{m'-n}\right\rangle . \tag{3.31}$$

The control register, the target register, and the modulus are all m' bits, so the subtraction can be performed in time linear in m'. From the statement of phase estimation in Section 2.4, we set $m' = 2m$, and as output we receive an m-bit approximation to a/b with probability $1 - O\left(\frac{1}{2^m}\right)$. The phase estimation algorithm also requires an inverse QFT mod $2^{m'}$. If we use our improved approximate QFT mod 2^n for this, and set $m', m \in O(n)$, we get an m-bit approximation to a/b with probability $1 - O(1/n^{O(1)})$.

The restriction requiring $a < b$ was introduced since the QFT mod b can only accept inputs from 0 to $b-1$. If a happens to be larger than b, we can simply

multiply b by 2^n to guarantee $2^n b > a$, compute the division, and then divide the result by 2^n to recover the answer. These multiplications do not cost anything, since they can just be interpreted as a shift the radix point. Further, there is no additional asymptotic cost in using $2n$ bits instead of n in the phase estimation algorithm.

3.4.2 Reduction with QFTs

The second reduction is more direct, in that it does not use the phase estimation algorithm. Let a and b be non-zero n-bit integers with $a < b$. The algorithm for computing an n-bit approximation of their quotient, a/b, is as follows.

1. Construct the quantum Fourier basis state $F^{\dagger}_{b2^{2n}} |a2^{2n}\rangle$.
2. Apply $F_{2^{4n}}$.
3. Measure the register, yielding c, and output $c/2^{3n}$ rounded off to the nearest integer.

The output is correct if the result c from step 3 satisfies $|c - \frac{a}{b}2^{4n}| < 2^{3n-1}$. After Step 2, the state of the register is

$$\left|\psi_{a/b}\right\rangle = \frac{1}{\sqrt{b2^{6n}}} \sum_{y=0}^{2^{4n}-1} \left(\sum_{x=0}^{b2^{2n}-1} \left(e^{2\pi i/2^{4n}}\right)^{x\cdot\left(y-\frac{a}{b}2^{4n}\right)} \right) |y\rangle \,. \tag{3.32}$$

For any $y \neq \frac{a}{b}2^{4n}$, the probability that measuring $\left|\psi_{a/b}\right\rangle$ yields y is bounded by

$$\begin{aligned} \left| \frac{1}{\sqrt{b2^{6n}}} \sum_{x=0}^{b2^{2n}-1} \left(e^{2\pi i/2^{4n}}\right)^{x\cdot\left(y-\frac{a}{b}2^{4n}\right)} \right|^2 &= \frac{1}{b2^{6n}} \left| \frac{e^{\left(2\pi i/2^{4n}\right)\left(y-\frac{a}{b}2^{4n}\right)\left(b2^{2n}\right)} - 1}{e^{\left(2\pi i/2^{4n}\right)\left(y-\frac{a}{b}2^{4n}\right)} - 1} \right|^2 \\ &< \frac{1}{b2^{6n}} \left(\frac{2}{\left(2\pi/2^{4n}\right)\left(y-\frac{a}{b}2^{4n}\right)\left(\frac{2}{\pi}\right)} \right)^2 \\ &< \frac{2^{2n}}{4b\left(y-\frac{a}{b}2^{4n}\right)^2}. \end{aligned} \tag{3.33}$$

We sum this probability over all values of y where $|y - \frac{a}{b}2^{4n}| \geq 2^{3n-1}$, or equivalently when $y \geq 2^{3n-1} + \frac{a}{b}2^{4n}$ or $y \leq \frac{a}{b}2^{4n} - 2^{3n-1}$. In the first case we obtain

$$\begin{aligned}
\sum_{y \geq 2^{3n-1} + \frac{a}{b}2^{4n}} \frac{2^{2n}}{4b\left(y - \frac{a}{b}2^{4n}\right)^2} &< \int_{2^{3n-1} + \frac{a}{b}2^{4n} - 1}^{\infty} \frac{2^{2n}}{4b\left(y - \frac{a}{b}2^{4n}\right)^2} \mathrm{d}y \\
&= \frac{2^{2n}}{4b}\left(\frac{1}{2^{3n-1} - 1}\right) \\
&\in O(1/2^n). \qquad (3.34)
\end{aligned}$$

Similarly the second case yields

$$\begin{aligned}
\sum_{y \leq \frac{a}{b}2^{4n} - 2^{3n-1}} \frac{2^{2n}}{4b\left(y - \frac{a}{b}2^{4n}\right)^2} &< \int_{-\infty}^{\frac{a}{b}2^{4n} - 2^{3n-1} + 1} \frac{2^{2n}}{4b\left(y - \frac{a}{b}2^{4n}\right)^2} \mathrm{d}y \\
&= \frac{2^{2n}}{4b}\left(\frac{1}{2^{3n-1} - 1}\right) \\
&\in O(1/2^n). \qquad (3.35)
\end{aligned}$$

Again, we remove the restriction of $a < b$ by multiplying b by 2^n if $a \geq b$, performing the division, and then dividing by 2^n to recover the answer. As before, for Step 2 we may use our approximate QFT mod 2^n presented earlier, so the algorithm has cost $O(n(\log\log n)^2 \log\log\log n)$ plus the cost of constructing the quantum Fourier basis state mod q with precision $1/n^{O(1)}$, and succeeds with probability $1 - 1/n^{O(1)}$.

Chapter 4

Simulations of Sparse Hamiltonians

4.1 Introduction

In Chapter 1 we saw that the postulates of quantum mechanics state that the Hamiltonian of a system governs the system's evolution over time according to the Schrödinger equation. The question follows then, that if we know the Hamiltonian and the start state of a some system, what is the state of the system after time t? It turns out that this question can be answered classically (given sufficient computing power) in the same manner as the arguments used to show that evolution is unitary. Given some Hamiltonian H, system start state $|\psi(0)\rangle$, and time t, we wish to calculate $|\psi(t)\rangle = e^{-iHt}|\psi(0)\rangle$. Because H is Hermitian, we can diagonalize H into $U^\dagger DU$, where U is some unitary matrix, and D is a real diagonal matrix. We then have

$$\begin{aligned} e^{-iHt} &= e^{-iU^\dagger DUt} \\ &= U^\dagger e^{-iDt} U. \end{aligned} \tag{4.1}$$

From Fact 1.2, e^{-iDt} equals a diagonal unitary matrix B with diagonal entries $e^{-id_j t}$, where d_j are the diagonal entries of D. So to calculate $|\psi(t)\rangle$, we need to merely calculate the matrix product

$$|\psi(t)\rangle = U^\dagger BU\,|\psi(0)\rangle\,.$$

Unfortunately, this approach is not realistic for large systems and for arbitrary Hamiltonians. For an n-qubit system, the size of the Hamiltonian H will be 2^n by 2^n, exponential in the input size. To diagonalize H and compute the matrix products, we need access all the entries of H, which would require exponential time. Further, if we wished to actually store H in memory, we would need exponential storage space. For small values of n, this is of course possible, but as n grows the requirements quickly exceed what is possible classically.

We instead follow the idea in [ATS03] and restrict the input Hamiltonian. They suggest that a natural restriction is to require that the Hamiltonian of the system we wish to simulate be *sparse*. For an n-qubit system, each row and column will have only a polynomial (in n) number of non-zero entries. Because the Hamiltonian is 2^n by 2^n, there will still be an exponential number of non-zero entries. We also require that, given a row, we have a function or black box that gives access to the non-zero entries of that row.

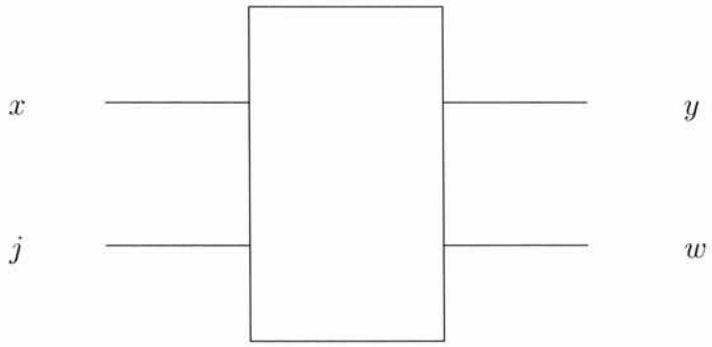

Figure 4.1: The Hamiltonian black box.

As input to the black box or function we input x and j, meaning we wish to know the j^{th} non-zero entry of row x. As output, we receive y and w, meaning the j^{th} non-zero entry of row x is in column y and has value w. Since H is Hermitian,

the ability to query only the elements of a row in the Hamiltonian, and not column, is sufficient. If row x does not have a j^{th} non-zero entry, we assume the black box outputs $y = x$ and $w = 0$. Note that the order in which the black box lists the row entries is completely arbitrary. We only require that when asked for the j^{th} non-zero entry of row x, we get the same result from the black box each time. For example, it could be that the j^{th} non-zero entry of row x is in column y, and the k^{th} non-zero entry of row x is in column w, with $j < k$ and $w < y$.

We remark that in general, diagonalizing even sparse Hamiltonians is still a difficult problem, because there are still an exponential number of non-zero entries in the Hamiltonian. Certainly if some information about the structure of the Hamiltonian were known before-hand it may be easier, but we do not assume any such knowledge.

4.2 General simulation technique

We now give a general overview of the technique for simulating a Hamiltonian. To simulate

$$e^{-iHt} \tag{4.2}$$

we try to find some sequence of matrices where $U^\dagger V U = H$ and $U, U^\dagger$ are unitary. We can then use $U^\dagger V U$ in place of H, and use the power of unitary conjugation, Fact 1.1, to move $U, U^\dagger$ out of the exponent:

$$e^{-iHt} = e^{-iU^\dagger V U t} = U^\dagger e^{-iVt} U. \tag{4.3}$$

We are then left to simulate e^{-iVt} and to implement $U, U^\dagger$ unitarily in the circuit model. The process is repeated on V.

When convenient, we will draw these sequences of matrices in circuit diagrams to help show correctness. The drawing of Hermitian circuits is an adjustment from the usual unitary-only circuit model of quantum computation. While we will mention when circuits are Hermitian and when they are unitary, we advise the reader to remember that these circuits will not be implemented directly, but in the exponent.

Finding a sequence of matrices with $U^\dagger VU = H$ is in general a difficult problem. For suitable H however, it is possible, and usually requires *ancilla* registers. An ancilla is one or more qubits used for temporary work space during an algorithm. So we actually try to find a sequence $U^\dagger VU$ such that $U^\dagger VU = H \otimes I$. It now may not be immediately clear that we may substitute $U^\dagger VU$ for H. If $U^\dagger VU$ uses an ancilla, it operates on a larger space than H. Fortunately, the following fact allows us to make the substitution.

Fact 4.1 *For a Hermitian operator H, $e^{-i(H\otimes I)t} = e^{-iHt} \otimes I$.*

Proof: *The equality may be verified by taking the Taylor series expansion of $e^{-iH\otimes It}$.*

$$\begin{aligned} e^{-iH\otimes It} &= \sum_{j=0}^{\infty} (-iH \otimes It)^j / j! \\ &= I \otimes I + -itH \otimes I + (-itH \otimes I)^2/2 + (-itH \otimes I)^3/6 + \ldots \\ &= I \otimes I + (-itH) \otimes I + ((-itH)^2/2) \otimes I + ((-itH)^3/6) \otimes I + \ldots \\ &= (I + -itH + (-itH)^2/2 + (-itH)^3/6 + \ldots) \otimes I \\ &= e^{-iHt} \otimes I. \end{aligned}$$

■

So if $U^\dagger VU = H \otimes I$ we may use $U^\dagger VU$ in place of H. Fact 4.1 assures us that

this will be the same as applying $e^{-iHt} \otimes I$: e^{-iHt} to the input qubits and I to an ancilla.

4.2.1 Preliminaries

Before beginning we define some functions and notation. The Hamiltonian H of an n-qubit system can be considered to be an *adjacency matrix* of a *graph*. A graph $G = (V, E)$ is a set of *vertices* V, and a set of *edges* E. A graph can be *directed* or *undirected*. In a directed graph, each edge $e = (u, v) \in E$ is comprised of an ordered pair of vertices (u, v), with each vertex $u, v \in V$. (The edge goes from vertex u to v.) In an undirected graph, the edge is comprised of an unordered pair of vertices. (The edge goes in both directions between vertices u and v). If there is an edge $(u, v) \in E$, vertex u is *adjacent* to vertex v, and vertex v is a neighbour of vertex u.

Further, for undirected graphs we say the edge $e = (u, v)$ is *incident* to vertices u, v. Edges can be *weighted*, where the edge e is also assigned a *weight*. The *degree* of a vertex is the number of vertices adjacent to it. The degree of the graph is the maximum degree of all vertices. One way to represent a graph is to store it in an adjacency matrix. For an n-vertex graph, the adjacency matrix is n by n, with entry at row j and column k indicating the weight of the edge from vertex j to vertex k. In undirected graphs, the adjacency matrix must be symmetric. If no such edge exists in the graph, the entry (and therefore the weight) is defined to be 0. It is possible to have *loops* in the graph, an edge from vertex v to itself. Edge direction is irrelevant in this case.

In the case of the Hamiltonian H of an n-qubit system, H is the adjacency matrix of a $2n$-vertex directed graph, with vertices labeled $0, \ldots, 2^n - 1$. Because

H is Hermitian, if the edge $(u, v) \in E$, then $(v, u) \in E$ as well, and the weight of edge (u, v) is the complex conjugate of the weight of (v, u), and vice-versa. We define $m_j(x)$ to be the j^{th} neighbour (adjacent vertex) of vertex x in the graph, or equivalently the j^{th} non-zero entry in row x of the Hamiltonian. We stress the ordering of the neighbours of x is arbitrary. The weight of this edge in the graph is defined to be $w_j(x)$, and we assume for now it is an n-bit integer in the range $0, \ldots, 2^n - 1$. We lift this restriction later.

Corresponding to the Hamiltonian black box in Figure 4.1 we use a unitary version of the black box for use in quantum circuits. M operating four n-qubit registers as

$$M |x\rangle |j\rangle |y\rangle |r\rangle = |x\rangle |j\rangle |y \oplus m_j(x)\rangle |r \oplus w_j(x)\rangle . \tag{4.4}$$

M copies the name of the j^{th} neighbour of x onto the third register, and the weight of the edge from x to $m_j(x)$ onto the fourth register. In the case that x is guaranteed to have no more than one neighbour, we drop the subscript from the functions $m(x)$ and $w(x)$, and leave out the second register of M as they are implied. We set M's input registers $|y\rangle$ and $|r\rangle$ to $|0\rangle$ so M's output values $m_j(x)$ and $w_j(x)$ are available. Finally, the unitary swap operation is denoted T; it naturally swaps two registers as $T |x\rangle |y\rangle = |y\rangle |x\rangle$.

4.3 The simulation problem

We formally define the simulation problem.

Hamiltonian Simulation:

Input: The start state $|\psi(0)\rangle$ of an n-qubit system, time t, Hamiltonian H with norm $\|H\| = \Delta$ and with at most d non-zero entries in each row, unitary Hamiltonian black

box M that gives access to the entries of H, error tolerance ϵ.

Output: The system state $|\psi(t)\rangle = e^{-iHt}|\psi(0)\rangle$ with error at most ϵ. We wish to count the number of 1 and 2-qubit gates, as well as the number of calls to the unitary Hamiltonian black box M.

From [ATS03], their simulation algorithm performs the simulation of e^{-iHt} in time

$$O\left(\frac{n^{10}d^3t^{3/2}\Delta^{3/2}}{\sqrt{\epsilon}}\right) \tag{4.5}$$

and

$$O\left(\frac{n^{9}d^3t^{3/2}\Delta^{3/2}}{\sqrt{\epsilon}}\right) \tag{4.6}$$

queries to the Hamiltonian black box, with error at most ϵ[1]. We improve upon this bound with the more efficient algorithm for colouring the Hamiltonian, and show the following theorem.

Theorem 4.2 *For an n-qubit Hamiltonian H with norm at most Δ and at most d non-zero entries in each row, e^{-iHt} can be simulated in time*

$$O\left(\frac{n(\lg^* n)^2d^3t^{3/2}\Delta^{3/2}}{\sqrt{\epsilon}}\right)$$

with error $\epsilon \in 1/n^{O(1)}$. In addition, the simulation uses

$$O\left(\frac{(\lg^* n)d^3t^{3/2}\Delta^{3/2}}{\sqrt{\epsilon}}\right)$$

queries to the Hamiltonian black box M.

[1]There may be additional costs in [ATS03] not specified in (4.5) and (4.6).

4.4 Simulating real very sparse Hamiltonians

We first consider the task of simulating *very sparse* Hamiltonians. We define very sparse to mean at most one non-zero entry in each row and column.

4.4.1 Unweighted degree 1 Hamiltonians

We begin by considering a very restricted sparse Hamiltonian H. We will require that H has exactly one 1 in each row and column. The graph represented by H is therefore unweighted and all vertices are of degree 1. H is then a permutation matrix, Hermitian and unitary. This section shows the following lemma.

Lemma 4.1 *For an n-qubit Hamiltonian H with exactly one 1 in each row and column, e^{-iHt} may be simulated with a quantum circuit of size $O(n)$ plus two calls to the Hamiltonian black box.*

Consider the action of H on a n-qubit basis vector $|x\rangle$:

$$H|x\rangle = |m(x)\rangle\,. \tag{4.7}$$

We can mimic the behaviour of H, using an n-qubit ancilla register, by the Hermitian (and unitary) operator

$$M^{\dagger}TM. \tag{4.8}$$

In this case, the restrictions imposed on H allow us to ignore the second and fourth registers of M. Applying 4.8 to a basis vector $|x\rangle$ and ancilla gives

$$\begin{aligned} M^\dagger TM\,|x\rangle\,|0\rangle &= M^\dagger T\,|x\rangle\,|m(x)\rangle \\ &= M^\dagger\,|m(x)\rangle\,|x\rangle \\ &= |m(x)\rangle\,|x \oplus m(m(x))\rangle \\ &= |m(x)\rangle\,|0\rangle\,. \end{aligned} \tag{4.9}$$

Note that $M^\dagger = M$, although we continue to write $M^\dagger$ for clarity. So we use $M^\dagger TM$ in place of H since $M^\dagger TM = H \otimes I$, and $M^\dagger TM$ is Hermitian. We can then use unitary conjugation, Fact 1.1, to get

$$e^{-iM^\dagger TMt} = M^\dagger e^{-iTt} M. \tag{4.10}$$

We are now left to simulate e^{-iTt}, and to perform $M, M^\dagger$ in the circuit model. We repeat the process to find a sequence of matrices that equal T. In this case, this sequence will be unitary (because T is unitary), although the sequence is only required to be Hermitian.

Since the swap operation

$$\begin{aligned} T\,|x\rangle\,|y\rangle &= T\,|x_{n-1}\cdots x_1x_0\rangle\,|y_{n-1}\cdots y_1y_0\rangle \\ &= |y_{n-1}\cdots y_1y_0\rangle\,|x_{n-1}\cdots x_1x_0\rangle \\ &= |y\rangle\,|x\rangle \end{aligned}$$

merely swaps the bits of x and y, x_1 with y_1, x_2 with y_2, and so on, T may be written as

$$T = \bigotimes_{l=1}^{n} S^{(l,n+l)} \tag{4.11}$$

where S is the two-qubit swap gate, with the exponent indicating which qubits are being swapped. It is possible to diagonalize the S operator by defining

$$W = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad E = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

We may write S as $S = WEW$ and

$$T = \bigotimes_{l=1}^{n} (WEW)^{l,n+l} = \bigotimes_{l=1}^{n} W^{l,n+l} \bigotimes_{l=1}^{n} E^{l,n+l} \bigotimes_{l=1}^{n} W^{l,n+l}. \tag{4.12}$$

Consider the action of the matrix $\bigotimes_{l=1}^{n} E^{l,n+l}$. It operates on a $2n$-qubit vector $|x\rangle |y\rangle = |x_{n-1} \cdots x_1 x_0\rangle |y_{n-1} \cdots y_1 y_0\rangle$ and maps $|x\rangle |y\rangle$ to:

$$\begin{cases} |x\rangle |y\rangle & \text{if} \quad x_j = 1 \quad \text{and} \quad y_j = 0 \quad \text{an even number of times for all } j \\ -|x\rangle |y\rangle & \text{otherwise} \end{cases} \tag{4.13}$$

One way to implement $\bigotimes_{l=1}^{n} E^{l,n+l}$ is to compute the parity of the number of times $x_j y_j = 10$, and add a phase of 1 for even parity, and -1 for odd parity. We use n Toffoli-like gates to compute the parity, and store it on an ancilla qubit. A Z gate on that ancilla qubit will add the correct phase, and we use the n Toffoli-like gates again to clean up. These Toffoli-like gates and the Z gates are Hermitian so they may be used. Figure 4.2 shows the final Hermitian circuit for T.

The circuit in Figure 4.2 may seem more complicated than necessary to implement T. However, it has a very useful structure. It is of the form $U^\dagger V U$, with U unitary and V Hermitian, and further, U (the W and Toffoli-like gates) is easily implementable unitarily in the circuit model. W is a two-qubit gate, and the Toffoli-like gate can be implemented with a Toffoli gate and two X gates. So both are

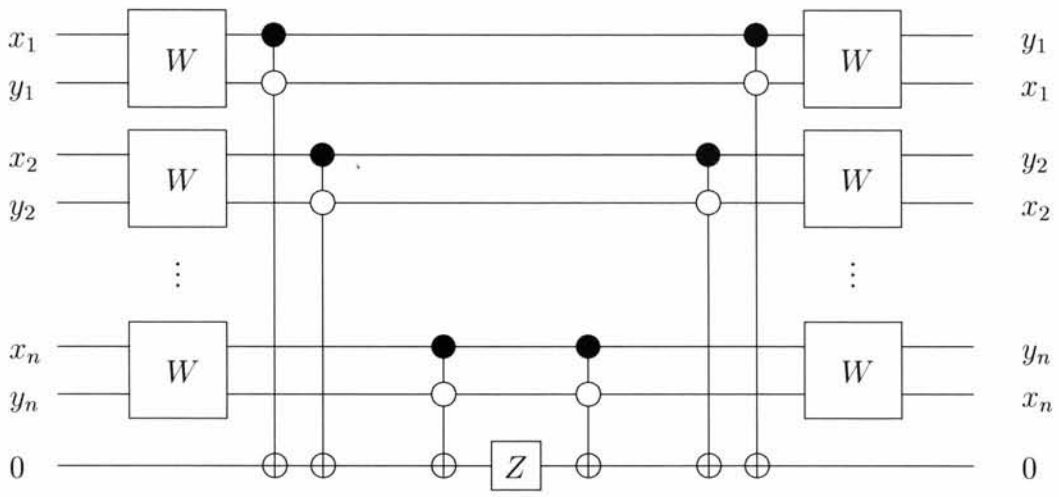

Figure 4.2: Hermitian circuit for T, the swap operation.

standard gates allowed in our unitary circuit model. Since the circuit in Figure 4.2 performs T, we may use it for the swap circuit when we simulate e^{-iTt}. As the W and Toffoli-like gates are unitary and self-inverse, unitary conjugation 1.1 allows us to move those gates out of the exponent and perform them unitarily. What remains is to simulate e^{-iZt}. Luckily, Fact 1.2 gives that

$$e^{-iZt} = \begin{bmatrix} e^{-it} & \\ & e^{it} \end{bmatrix} \tag{4.14}$$

is just a one-qubit phase gate, which may be implemented in the unitary circuit model.

Figure 4.3 shows the final circuit to simulate e^{-iHt}. We input the n-qubits of the start state $|\psi(0)\rangle$, and apply the unitary operations in the circuit. As output, we receive $|\psi(t)\rangle$, the state after evolving the system for time t.

To the simplify upcoming circuit diagrams, we introduce a new circuit diagram in Figure 4.4. The A gate will take the place one group of the W and Toffoli-like gates

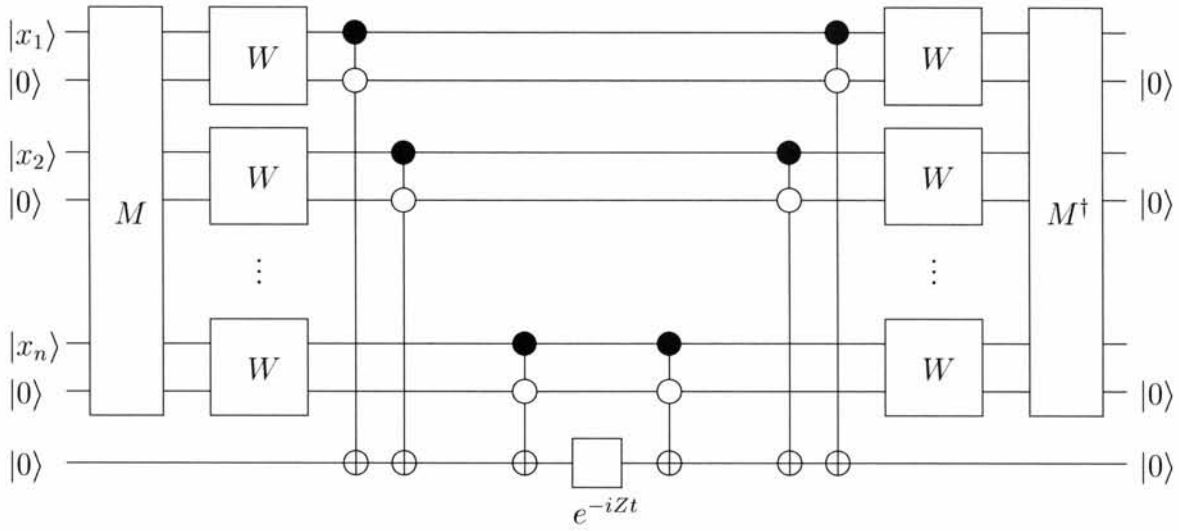

Figure 4.3: Unitary circuit for e^{-iHt} for H unweighted, degree 1.

in the circuit. We also group all the bits of $|x\rangle$ and the n-qubit ancilla together to reduce clutter. Functionally the circuits are the same and we could always translate between the two by rearranging the qubits.

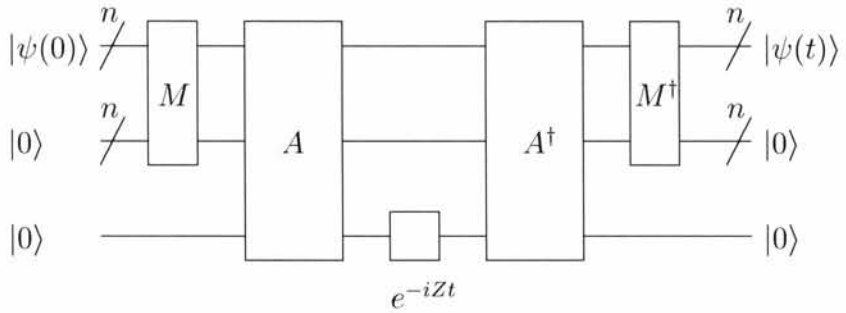

Figure 4.4: Simplified circuit for e^{-iHt} for H unweighted, degree 1.

4.4.2 Weighted degree 0 or 1 Hamiltonians

We now relax the restrictions on H, and allow H to have at most one non-zero entry in each row and column. Each entry may be an n-bit integer from 0 to $2^n - 1$. This

section shows the following lemma.

Lemma 4.2 *For an n-qubit Hamiltonian H with at most one non-negative n-bit integer in each row and column, e^{-iHt} may be simulated with a quantum circuit of size $O(n)$ plus two calls to the Hamiltonian black box.*

H acts on a basis state $|x\rangle$ as

$$H|x\rangle = w(x)|m(x)\rangle . \tag{4.15}$$

Recall when row x does not have a non-zero entry, we assume $w(x) = 0$, so $H|x\rangle = 0$. So we must find a Hermitian circuit to map $|x\rangle$ as 4.15. One such circuit is given in Figure 4.5.

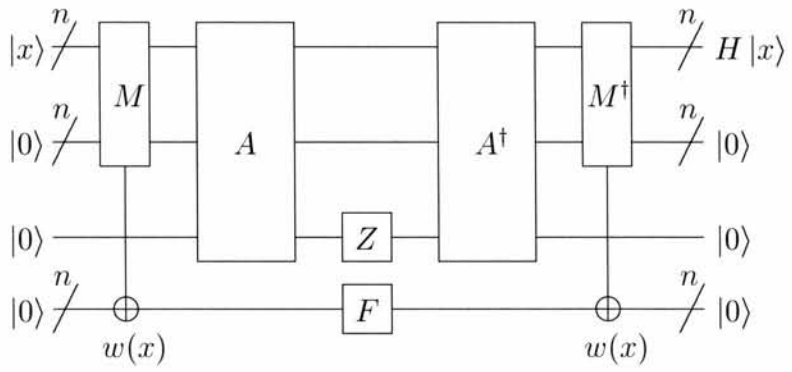

Figure 4.5: Hermitian circuit for H weighted, degree 0 or 1.

In Figure 4.5, we use M to save the weight of the non-zero entry of row x to an ancilla. The F gate is the Hermitian operator

$$F = \begin{bmatrix} 0 & & & & \\ & 1 & & & \\ & & 2 & & \\ & & & \ddots & \\ & & & & 2^n - 1 \end{bmatrix}.$$

It maps $|w(x)\rangle$ to $w(x)\,|w(x)\rangle$. We now apply the circuit to a basis state to show it acts as H. On input $|x\rangle$,

1. Apply M to $|x\rangle$ and the ancilla:

$$|x\rangle\,|0\rangle\,|0\rangle\,|0\rangle \mapsto |x\rangle\,|m(x)\rangle\,|0\rangle\,|w(x)\rangle\,.$$

2. Apply $T = A^\dagger Z A$:

$$\mapsto |m(x)\rangle\,|x\rangle\,|0\rangle\,|w(x)\rangle\,.$$

3. Apply F:

$$\mapsto w(x)\,|m(x)\rangle\,|x\rangle\,|0\rangle\,|w(x)\rangle\,.$$

4. Apply $M^\dagger$:

$$\mapsto w(x)\,|m(x)\rangle\,|x \oplus m(m(x))\rangle\,|0\rangle\,|w(x) \oplus w(m(x))\rangle = w(x)\,|m(x)\rangle\,|0\rangle\,|0\rangle\,|0\rangle\,.$$

Since the circuit in Figure 4.5 implements 4.15, we use it in place of H in e^{-iHt}. Again the special structure of the circuit allows the use of unitary conjugation to

move the M, $M^\dagger$, A,and $A^\dagger$ gates out of the exponent; we can perform them as regular unitary operations in the circuit model. All that remains to implement is

$$e^{-iZ\otimes Ft}. \tag{4.16}$$

The matrix $Z \otimes F$ is F when the first qubit is 0, and $-F$ when it is 1. The unitary operation e^{-iFt} maps $|w(x)\rangle \mapsto e^{-iw(x)t}|w(x)\rangle$. We perform 4.16 with a series of controlled-phase gates as follows. Let $|p\rangle$ be the qubit to which we apply Z in the Hermitian circuit in Figure 4.5. Let $|a\rangle = |a_{n-1}a_{n-2}\ldots a_0\rangle$ be the ancilla register holding $w(x)$, with a_{n-1} the most significant bit. If $|p\rangle = |0\rangle$, for each j, apply

$$\begin{bmatrix} 1 & 0 \\ 0 & e^{-i2^j t} \end{bmatrix}$$

to the j^{th} qubit of $|a\rangle$. If $|p\rangle = |1\rangle$, apply

$$\begin{bmatrix} 1 & 0 \\ 0 & e^{i2^j t} \end{bmatrix}$$

to the j^{th} qubit of the $|a\rangle$. This is easy to perform with a sequence of $2n$ controlled-phase gates. Figure 4.6 shows the unitary circuit to implement (4.16).

4.4.3 Handling real Hamiltonian entries

To this point we have assumed that the entries (weights) of the Hamiltonian are n-bit integers. If that were the case, and t is a value such that we can implement the controlled phase gates in Figure 4.6 without error, the circuit to simulate e^{-iHt} does so without error. Given $|\psi(0)\rangle$, we can calculate $|\psi(t)\rangle$ exactly. Unfortunately, this

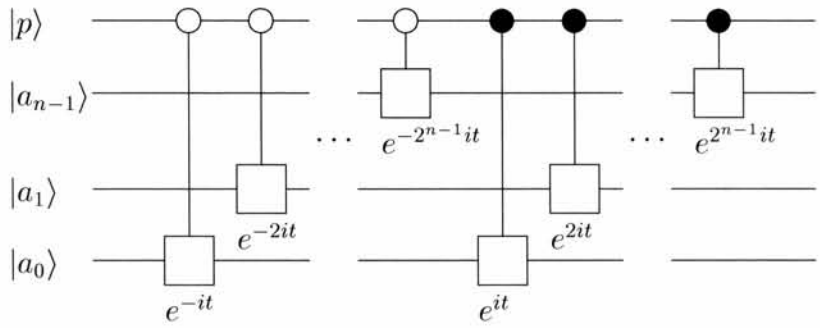

Figure 4.6: Unitary circuit for $e^{-iZ\otimes Ft}$.

may not be a realistic assumption. The entries of H are, in general, real numbers. In this section, we introduce a circuit to handle real-valued Hamiltonians, and analyze the circuit in in Sections 4.4.4 and 4.4.5 to show the following lemma.

Lemma 4.3 *For an n-qubit Hamiltonian H with at most one real-valued entry in each row and column, e^{-iHt} may be simulated with a quantum circuit of size $O(n+r)$ plus two calls to the Hamiltonian black box with error at most $O(1/2^r)$.*

We need to impose some sort of representation in which we will accept the entries from the unitary black box. We introduce the parameter r to be the precision with which we store both weights of the Hamiltonian and the time t. We store them as $3r$-bit binary numbers, plus an additional bit for the sign. The first r bits are the most significant and represent the bits to the left of the fixed radix point. The last $2r$ bits store any fractional portion of the weight. So, we can interpret the weights as rational numbers with a signed $3r$-bit numerator and a denominator of 2^{2r}.

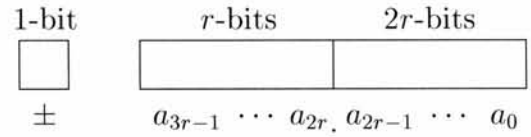

Figure 4.7: We use $(3r+1)$-bit integers to store t and the weights.

We now modify the circuits in Figures 4.5 and 4.6 to handle $3r+1$ bit weights with denominator 2^{2r}. The F gate Figure in 4.5 maps $|w(x)\rangle \mapsto w(x)\,|w(x)\rangle$. To do this, we make F a $3r$-bit operator, with diagonal values $0, \ldots, 2^{3r} - 1$ and apply $F/2^{2r}$.

Finally, we handle the case when $w(x)$ is signed by storing the sign of $w(x)$ onto another ancilla qubit. We define $s(w(x))$ to be sign of $w(x)$, and $s(w(x)) = 1$ iff $w(x)$ is negative. Furthermore, we define $v(x) = |w(x)|$. Once the $s(w(x))$ is stored on an ancilla, applying the Z operator to the qubit will add a phase of -1 if $w(x)$ is negative, as desired. The complete Hermitian circuit to simulate H is given in Figure 4.8.

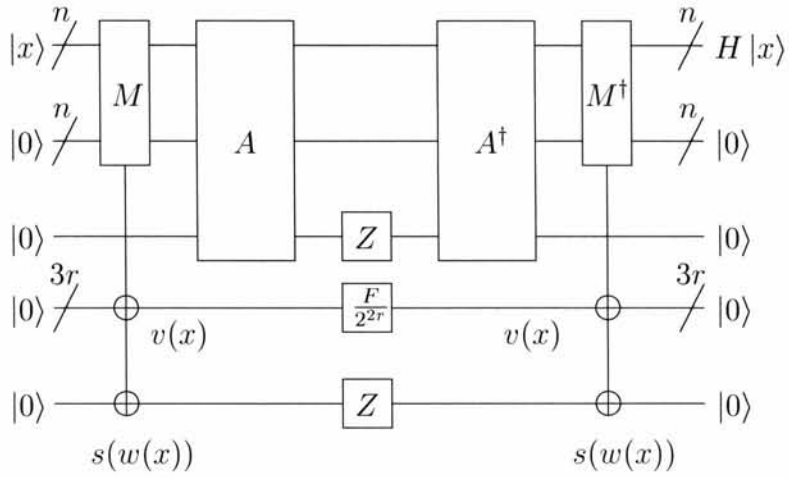

Figure 4.8: Hermitian circuit for H real, degree 0, 1.

In Figure 4.8 we abuse the definition of M, and use it to save $v(x)$ and $s(w(x))$ onto separate ancilla registers. We re-define M in this manner for convenience. To simulate e^{-iHt}, we use this circuit in place of H. Using unitary conjugation leaves

$$e^{-iZ\otimes F/2^{2r}\otimes Zt}, \tag{4.17}$$

which may be simulating in a manner similar to the circuit in Figure 4.6. Since $Z \otimes F/2^{2r} \otimes Z$ is equivalent to $I \otimes F/2^{2r} \otimes I$ when the parity of the first and last qubits is 0, and $I \otimes (-F/2^{2r}) \otimes I$ when the parity is 1, simulating (4.17) may be performed by computing the parity of the first and last qubits and applying a series of controlled phase gates as in Figure 4.6. To apply $F/2^{2r}$ instead of F, we subtract $2r$ from the exponent of the power of two in each phase gate.

4.4.4 Simulation performance

We can very readily analyze the performance of the weighted very sparse Hamiltonian simulation in terms of the number of quantum gates required, as well as the number of calls to the graph (Hamiltonian) black box. The algorithm uses the unitary black box M twice. The circuit contains $2n$ W gates, $2n$ Toffoli-like gates, $6r$-controlled phase gates, and a constant number of other gates (for computing the parity of two qubits). In total there are two black box calls plus $O(n + r)$ unitary gates.

4.4.5 Error analysis

As we mentioned, when all entries in the Hamiltonian and t can be exactly represented as $3r$-bit numbers, the simulation of e^{-iHt} is without error. When they cannot, there will be some error in the circuit. The controlled phase gates implementing (4.17) will introduce the error if $w(x)$ and t cannot be stored exactly. The parameter r scales with the size of the entries in the Hamiltonian. Because we use $2r$ bits to store the decimal portion approximation to $w(x)$ and t, the error introduced by approximating each will be less than $1/2^{2r}$. The phase gates map

$$|w(x)\rangle \mapsto e^{-iw(x)t} |w(x)\rangle , \tag{4.18}$$

so we can calculate the maximum error in the phase in the same manner as Section 3.1.1. From Section 2.3.1, the deletion of a phase gate $e^{-i\theta}$ induces a Euclidean distance error of at most θ. So we can calculate the error by determining the difference between the exact and approximate product of $w(x)t$. Let $\widetilde{w(x)}$ and $\widetilde{t}$ be the $3r$-bit approximations to $w(x)$ and t respectively. Then $|w(x) - \widetilde{w(x)}| < 1/2^{2r}$ and $|t - \tilde{t}| < 1/2^{2r}$, and using the approximations results in a maximum induced Euclidean distance of

$$\begin{aligned} |w(x)t - \widetilde{w(x)}\widetilde{t}| &= |w(x)(t - \widetilde{t}) + (w(x) - \widetilde{w(x)})\widetilde{t}| \\ &\leq (|w(x)| + |t|)\,\frac{1}{2^{2r}} \\ &\leq \frac{1}{2^{r-1}} \\ &\in O\left(\frac{1}{2^r}\right) \end{aligned} \tag{4.19}$$

The approximation has an exponentially small error in r. This is the case even for very large (up to 2^r) values of $w(x)$ and t. Setting $r \in \Theta(n)$ gives exponentially small error in the number of qubits of the system, and a linear circuit size, $O(n+r) = O(n)$, for the whole simulation.

4.5 Simulating imaginary very sparse Hamiltonians

We now turn our attention to simulating imaginary very sparse Hamiltonians. As before, we begin be restricting the values in the Hamiltonian.

4.5.1 Unweighted degree 1 Hamiltonians

To simulate Hamiltonians with imaginary values (of the form ib with b real) we will use a circuit similar in spirit to the real case. To start we require that H has exactly

one non-zero entry in each row and column, and each non-zero entry is either i or $-i$. We show the following lemma.

Lemma 4.4 *For an n-qubit Hamiltonian H with exactly one i or $-i$ in each row and column, e^{-iHt} may be simulated with a quantum circuit of size $O(n)$ plus two calls to the Hamiltonian black box.*

Again, H acts on a basis state $|x\rangle$ as

$$H|x\rangle = w(x)|m(x)\rangle . \tag{4.20}$$

We can simulate this action with the Hermitian circuit in Figure 4.9.

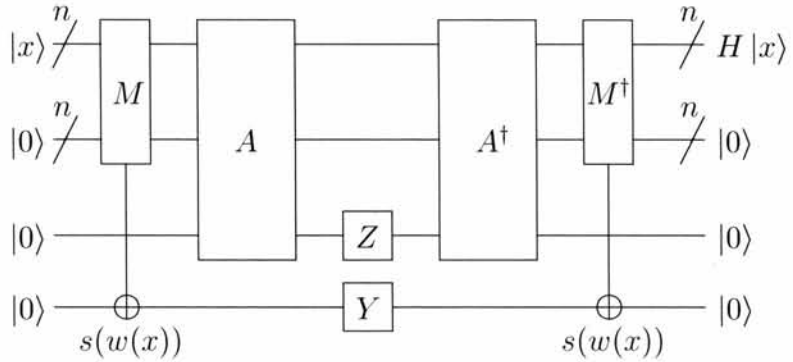

Figure 4.9: Hermitian circuit for H imaginary, unweighted, degree 1.

Because all non-zero entries in H are either i or $-i$, we need only one qubit to store $w(x)$. As in the real case, we let M save the sign of $w(x)$ onto its third register; it will flip the qubit if the sign is negative, and do nothing if the sign is positive. When $s(w(x)) = 0$, corresponding to $w(x) = i$, the first application of M leaves the third register unchanged. We then apply Y, introducing a phase of $i = w(x)$, and flipping the qubit. The second application of M then flips the third qubit, reseting it to zero, since $w(m(x)) = -i$ so $s(w(m(x)) = 1$. In the case that $s(w(x)) = 1$, the

first (but not the last) application of M flips the target qubit, so the first M gate and the Y gate ensure the ancilla will be reset to zero. In this case the Y gate adds the correct phase of $-i$. We can verify that the circuit in Figure 4.9 implements H. On input of a basis state $|x\rangle$ (we assume $w(x) = i$, the other case is similar):

1. Apply M ($s(w(x)) = 0$) to the input and ancilla:

$$|x\rangle |0\rangle |0\rangle |0\rangle \mapsto |x\rangle |m(x)\rangle |0\rangle |0\rangle .$$

2. Apply $T = A^\dagger Z A$:

$$\mapsto |m(x)\rangle |x\rangle |0\rangle |0\rangle .$$

3. Apply Y:

$$\mapsto i |m(x)\rangle |x\rangle |0\rangle |1\rangle .$$

4. Apply $M^\dagger$ (here $s(w(x)) = -1$ so M's third register gets flipped):

$$\mapsto i |m(x)\rangle |x \oplus m(m(x))\rangle |0\rangle |1 \oplus 1\rangle = i |m(x)\rangle |0\rangle |0\rangle |0\rangle = w(x) |m(x)\rangle .$$

Since the circuit in Figure 4.9 implements H, we use it in place of H in e^{-iHt}. Again, unitary conjugation moves the M and A gates out of the exponent, and are left with

$$e^{-iZ \otimes Yt}. \tag{4.21}$$

This operator is easy to diagonalize. If we let

$$G = \begin{bmatrix} -i/\sqrt{2} & -1/\sqrt{2} \\ -1/\sqrt{2} & -i/\sqrt{2} \end{bmatrix} = \frac{-1}{\sqrt{2}} \begin{bmatrix} i & 1 \\ 1 & i \end{bmatrix}$$

then Y may be written as

$$Y = G^\dagger Z G. \tag{4.22}$$

Substituting $G^\dagger ZG$ for Y in the circuit in Figure 4.9, we can move G and $G^\dagger$ out of the exponent since G is unitary, and we are left to simulate

$$e^{-iZ\otimes Zt}. \tag{4.23}$$

By Fact 1.2,

$$e^{-iZ\otimes Zt} = \begin{bmatrix} e^{-it} & & & \\ & e^{it} & & \\ & & e^{it} & \\ & & & e^{-it} \end{bmatrix}.$$

To implement $e^{-iZ\otimes Zt}$ we compute the parity of the two qubits, add a phase of e^{-it} for even parity, e^{it} for odd parity, and uncompute the parity to clean up. Figure 4.10 shows the final unitary circuit to simulate e^{-iHt}.

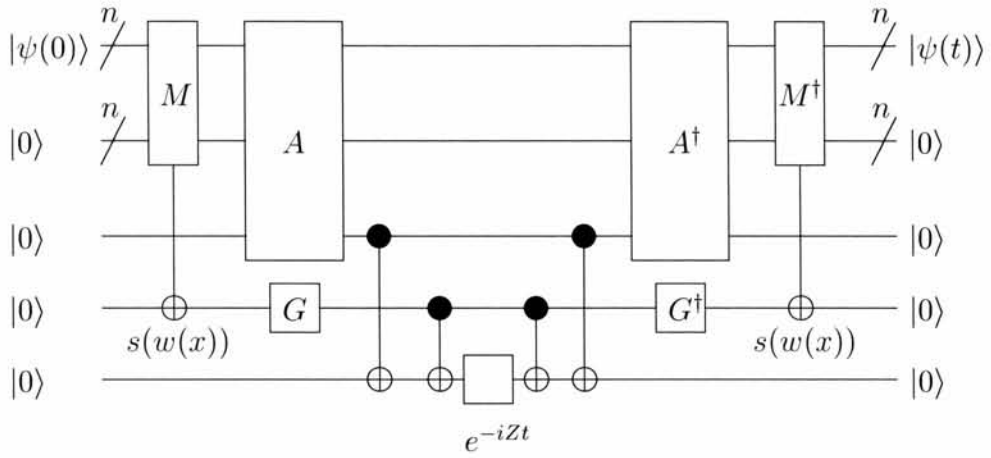

Figure 4.10: Circuit to simulate e^{-iHt} for H imaginary, unweighted, degree 1.

4.5.2 Weighted degree 0 or 1 Hamiltonians

We now allow at most one non-zero entry in each row and column of the form ib, where b is an signed n-bit integer. This section gives a circuit to simulate such a Hamiltonian. Section 4.5.3 expands to arbitrary imaginary entries and analyzes the error and size to show the following lemma.

Lemma 4.5 *For an n-qubit Hamiltonian H with at most one imaginary entry in each row and column, e^{-iHt} may be simulated with a quantum circuit of size $O(n+r)$ plus two calls to the Hamiltonian black box with error at most $O(1/2^r)$.*

We again use $v(x) = |w(x)|$. If row x has no non-zero entry, $w(x) = 0$. Now H on a basis vector maps $|x\rangle$ as

$$H|x\rangle = w(x)|m(x)\rangle = (-1)^{s(w(x))} iv(x)|w(x)\rangle . \tag{4.24}$$

We can take the circuit in Figure 4.9 and modify it to handle the fewer restrictions on H. Instead of saving only the sign of $w(x)$, we save the sign and also the unsigned weight $v(x)$ on an ancilla register. As with the real valued Hamiltonians, we can apply the F operation to the ancilla to map $|v(x)\rangle \mapsto v(x)|v(x)\rangle$ and then erase the ancilla.

In Figure 4.11, we substitute Y with $G^\dagger ZG$. The M operator continues to map $|x\rangle|0\rangle$ to $|x\rangle|m(x)\rangle$. We also use it to store the sign of $w(x)$ on the first ancilla qubit, flipping it to 1 if the sign is negative, and to store $v(x)$ on the second ancilla register. This is a slight abuse of the M operator as we originally defined it, but we use the same name for convenience.

We can verify that the circuit in Figure 4.11 implements H. On input of a basis vector $|x\rangle$ (we assume $w(x) = ib$ with $b \geq 0$; the case when $b < 0$ is similar):

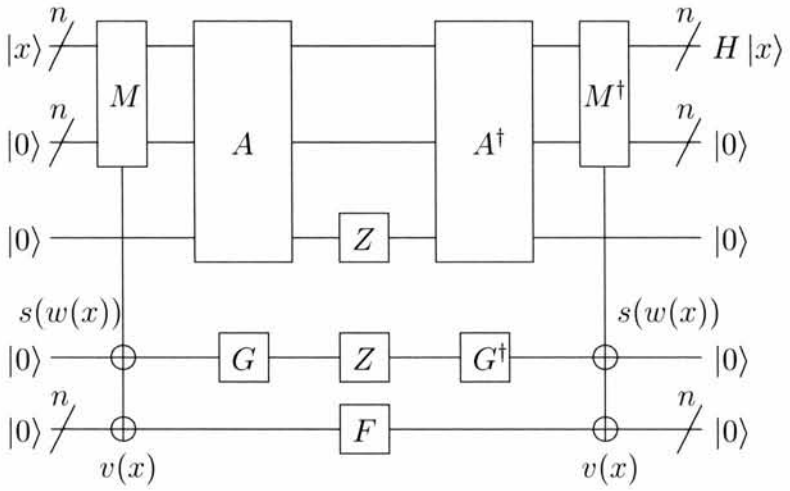

Figure 4.11: Hermitian circuit for H imaginary, weighted, degree 0 or 1.

1. Apply M:

$$|x\rangle |0\rangle |0\rangle |0\rangle |0\rangle \mapsto |x\rangle |m(x)\rangle |0\rangle |0\rangle |v(x)\rangle .$$

2. Apply $T = A^\dagger Z A$:

$$\mapsto |m(x)\rangle |x\rangle |0\rangle |0\rangle |v(x)\rangle .$$

3. Apply $Y = G^\dagger Z G$:

$$\mapsto i |m(x)\rangle |x\rangle |0\rangle |1\rangle |v(x)\rangle .$$

4. Apply F:

$$\mapsto ib |m(x)\rangle |x\rangle |0\rangle |1\rangle |v(x)\rangle .$$

5. Apply $M^\dagger$ (here $w(m(x)) = -ib$ so M's third register gets flipped):

$$\mapsto ib |m(x)\rangle |x \oplus m(m(x))\rangle |0\rangle |1 \oplus 1\rangle |v(x) \oplus v(x)\rangle = w(x) |m(x)\rangle |0\rangle |0\rangle |0\rangle |0\rangle .$$

To simulate e^{-iHt}, we use the familiar method of unitary conjugation. The circuit in Figure 4.11 implements H, and we move the M, A, and G gates out of the

exponent, and are left with $e^{-iZ\otimes Z\otimes Ft}$. This may be performed in a manner similar to Figure 4.6, using a sequence of controlled-phase gates. Since

$$Z \otimes Z \otimes F = \begin{bmatrix} F & & & \\ & -F & & \\ & & -F & \\ & & & F \end{bmatrix},$$

we compute the parity of the first two qubits, and apply e^{-iFt} for even parity, and e^{iFt} for odd parity. Figure 4.12 leaves out the details of this calculation.

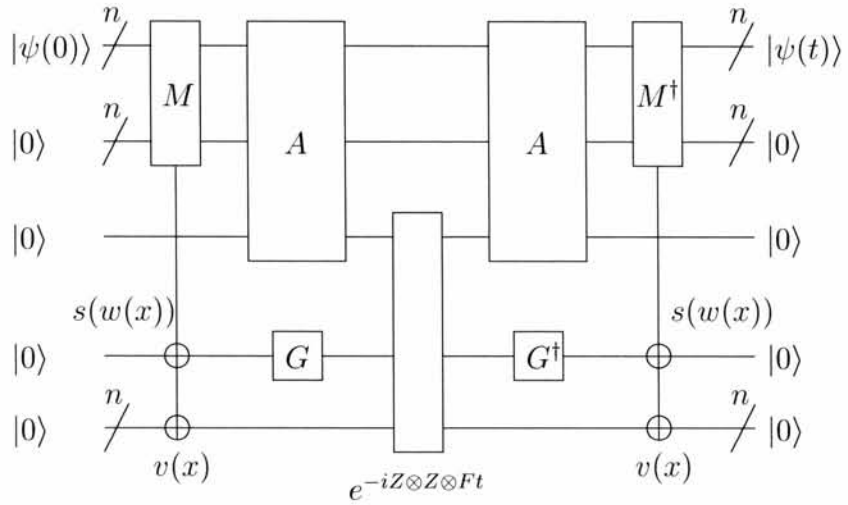

Figure 4.12: Circuit to simulate e^{-iHt} for H imaginary, weighted, degree 0 or 1.

4.5.3 Non-integer weights, error analysis, and performance

The handling of non-integer Hamiltonian entries follows directly from the real case in Section 4.4.5. We use $3r$ bits to store an approximation to $v(x)$ with an additional bit for the sign. The maximum error induced is in $O(1/2^r)$. The circuit uses two two calls to the unitary black box M, $2n$ W and Toffoli-like gates, and $6r$ controlled

phase gates. There are a constant number of other gates, so like the real valued Hamiltonian in total there are 2 black box calls plus $O(n + r)$ gates in the circuit with error at most $O(1/2^r)$.

4.6 Colouring sparse Hamiltonians

To this point we have seen that we can simulate e^{-iHt} for very restricted cases of H:

1. H has at most one non-zero real entry in each row.

2. H has at most one non-zero imaginary entry in each row.

We would like to simulate any general sparse Hamiltonian; for a Hamiltonian H on n qubits, simulate e^{-iHt} where H has a polynomial (in n) number of non-zero entries in each row. These entries are, in general, complex numbers.

Remarkably, it is possible to simulate any sparse Hamiltonian using the circuits we have constructed to handle Cases 1 and 2 above, plus some other classical processing. The idea is used in [NC00, ATS03] and is as follows. We divide the original sparse Hamiltonian H into a sum of sub-Hamiltonians, all of which conform to Case 1 or Case 2.

$$H = \sum_c H_c \tag{4.25}$$

We then simulate each individual H_c in turn for some fraction of the total time t, say t/p. If we repeat this process p times, each H_c will have been simulated for time t. Given the right value for p, if H can be divided into a manageable number of sub-Hamiltonians, it can be shown that the result will be a very good approximation to simulating the original Hamiltonian H.

4.6.1 Colouring the Hamiltonian

The first task at hand is to partition H into a sum of sub-Hamiltonians. We do this by computing an *edge colouring* of the graph represented by H. An edge colouring is a labeling (colouring) assigned to the edges of the graph, such that, for each vertex v, all edges incident to v have a different colour. Suppose we have an edge colouring for H, and we choose some colour (say red). We keep all red edges of H and throw away all other edges. The subgraph H_{red} that remains will be very sparse. Each row of H_{red} has at most one non-zero entry, because each vertex in the graph has at most one red edge incident to it. So we can use the circuits from the previous sections to simulate $e^{-iH_{red}t}$.

Requirements of the colouring algorithm

The colouring algorithm must meet certain requirements for it to be suitable for our purposes. The first is that the colouring be *symmetric*. The edge from vertex x to y in the graph must be the same colour as the edge from y to x. When we simulate for some colour c, the sub-Hamiltonian H_c induced by that colour must still be Hermitian. If the entry (x, y) is in H_c, so must (y, x). This means there will be two adjacent edges of the same colour incident to both x and y. But, for the purposes of the colouring the Hamiltonian, we can think of the pair of edges (x, y) and (y, x) as being combined into one undirected edge. Similarly, when loops (edges from vertex x to itself) are coloured, we consider vertex x to have only one incident edge of that colour.

The second requirement is that the colouring algorithm be *local*. The graph H has a (possibly) exponential number of edges, and 2^n vertices. Therefore the algorithm

does not have the luxury of examining the entire graph to determine the colour of each edge. It must give an answer by looking at only a small portion of the graph.

Another consequence of the exponential size of the graph is that there is no way to store the edge colouring computed by the algorithm. Certainly making a list of edges that are red, edges that are blue, and so on is not feasible. Instead we will use the algorithm in a query fashion. We implement a new black box M', where M' acts as follows:

$$M' |x\rangle |c\rangle |y\rangle |r\rangle = |x\rangle |c\rangle |y \oplus m(x)\rangle |r \oplus w(x)\rangle . \tag{4.26}$$

M' accepts a vertex x and a colour c, and returns the neighbour of x that has an incident edge of colour c with x (if it exists), and the weight of that edge. If x has no incident edge of colour c, M' returns $m(x) = x$ and $w(x) = 0$ as usual. To describe the colouring algorithm, we modify the problem slightly, and show how to implement M' with this modified version later.

The algorithm can only use the graph structure as provided by the Hamiltonian black box M to compute the colour: the labels of the vertices, the number of edges incident to vertices, the neighbour vertices, and so on. It cannot rely on previously computed edge colours.

The problem of local edge colouring (as well as the similar problem of local vertex colouring) is a well studied problem in computer science. The main results that interest us are the upper and lower bounds on the number of colours required to locally colour a graph. The fewer the colours, the fewer sub-Hamiltonians, and the more quickly the simulation will run. There is a trade-off between the number of colours used in the edge colouring, and the time required to compute the colouring. We leave this for later during the analysis of the algorithm. Define d to be the

maximum degree of all vertices in H. The best known upper bound on a local colouring computable in polynomial time is $O(d)$ colours [DMP01]. Vizing's theorem [Viz64] states that every graph of maximum degree d can be edge coloured with d or $d+1$ colours. (A side note: the problem of *deciding* if a graph requires d or $d+1$ edge colours is NP-complete [Hol81].) So, the algorithm of [DMP01] is asymptotically tight. However, it turns out this algorithm is not directly useful in this context. We instead will use an algorithm give in [Lin92, SV93], which can locally edge colour with $O(d^2)$ colours.

The colouring algorithm

We define the edge colouring problem as follows:

Edge colouring:

Input: Vertices x and y in the graph.

Output: A colour c for the edge (x, y) that is guaranteed to differ from the colours of all other edges incident to x or y.

We show the following lemma.

Lemma 4.6 *An edge (x, y) of a 2^n-vertex graph with vertex labels $0, \ldots, 2^n - 1$ whose entries in its adjacency matrix are given by a black box M (as defined in Figure 4.1) may be edge-coloured in time $O(n(\lg^* n)^2)$.*

We consider the number of Hamiltonian black box calls to M later. This is an improvement over the colouring provided in [ATS03], which used d^2n^6 colours.

Using the Hamiltonian black box M, we determine that y is the j^{th} neighbour of x, and that x is the k^{th} neighbour of y. It is possible that $x = y$, in which case $j = k$. Because the maximum degree of H is d, $j, k \leq d$. As a first step, we colour the edge

with the tuple

$$Colour(x, y) = \begin{cases} (j, k) & \text{if} \quad x \leq y \\ (k, j) & \text{if} \quad y > x \end{cases} \tag{4.27}$$

The edge colour records the neighbour rank of the numerically smaller vertex first. If the $x = y$ then of course the order does not matter. There are d^2 possible edge colours, as there are d choices for both j and k. What does this colouring scheme guarantee? The colour's first component is j, so no edge from x to another vertex w may have a colour beginning with j, as long as $w > x$. This is because x cannot have two j^{th} neighbours, and for all edges from x to another vertex w where $w > x$, the edge starts with x's neighbour information. Similarly, the colour's second component is k, so no edge from y to another vertex z may have a second component of k when $z < y$. If $x = y$, then the edge is coloured (j, j), and no other edge incident to x can be coloured with j.

There are two cases when this colouring algorithm can fail, shown in Figure 4.13. Suppose there is an edge from x to some vertex $w \neq y$ with $w < x$, and x is w's j^{th} neighbour, and w is $x's$ k^{th} neighbour. In this case, the edges (x, y) and (w, x) would both be assigned the colour (j, k). The second case is similar, involving the vertex y. If there is an edge (y, z) in the graph with $y < z$, and $y's$ j^{th} neighbour is z, and $z's$ k^{th} neighbour is y, then the edge (y, z) would be coloured (j, k) as well.

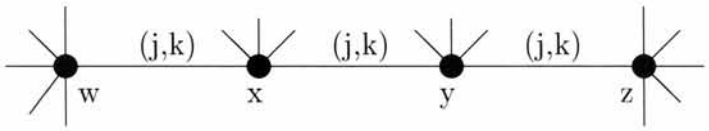

Figure 4.13: Problem cases for the first colouring algorithm. When $w < x < y$ or $x < y < z$ it is possible to have adjacent edges with the same colour.

With this colouring algorithm the two problem cases demonstrate the potential

for there to be paths in the graph, where each edge in the path has the same colour. More information is needed in the edge colour to separate the edges in these potential paths. Ideally, the path can be coloured with two colours. We could append to the colour (j, k) a single bit, and the edge colours in the path could alternate as $(j, k, 0)$ and $(j, k, 1)$. Unfortunately, it is appears difficult, given an edge (x, y) in the graph, to determine if the edge should be coloured $(j, k, 0)$ or $(j, k, 1)$. Traversing the path in the graph to find the beginning is not possible, since it may be exponentially long. A technique called "deterministic coin tossing" (or deterministic symmetry breaking) [CV86a, CV86b, CLR90] can be used to eliminate paths of one colour in the graph. It will not use the the minimum of two colours, but will colour the path with at most six colours, which is asymptotically equivalent.

The deterministic coin tossing method

If it is the case that we are trying to colour the edge (x, y) and $x = y$, then we do not need to use the deterministic coin tossing method, and we colour the edge $(j, j, 000)$, where "000" are three bits set to 0.

So assume now we are trying to colour the edge (x, y) in the graph with $x < y$, and that the edge has been already coloured (j, k). If we were to append to this colour the name of the edge's numerically smaller vertex, the colour becomes (j, k, x). Using this rule throughout the graph would result in a valid edge colouring. It solves the two problem cases in Figure 4.13. The edges (w, x), (x, y), and (y, z) would be coloured (j, k, w), (j, k, x), and (j, k, y) respectively. The rule by itself, however, results in too many colours. The vertex labels are n-bit integers, so there are $d^2 2^n$ possible colours for the edges. The deterministic coin tossing method re-labels (for the purposes

of computing a valid colouring) the vertices along the (j,k)-coloured path with a constant number of labels such that each vertex's new label is guaranteed to be different than its neighbours new labels. We then append the numerically smaller vertex's new label to the colour to complete the edge colouring.

The algorithm uses a dynamic programming approach. We define $d(x,y)$ to be the index of the first bit, starting from the least significant, that $x = x_{m-1} \cdots x_1\, x_0$ and $y = y_{m-1} \cdots y_1\, y_0$ differ. If x and y are m bits, $d(x,y) \in \{0, \ldots, m-1\}$. Note that $d(x,y)$ is itself a $\lceil \lg m \rceil$-bit integer. We assume we are trying to colour the edge (x,y), and so trying to compute a new label for x. For the purposes of demonstrating the algorithm, we assume that there are also edges (y,z), and (z,a) in the graph, all coloured (j,k), but no edge (a,b) with colour (j,k) for $x < y < z < a < b$. The initial label for x is x^0, the next x^1, and so on.

In Step 1, we calculate x^1, by concatenating $d(x^0,y^0)$ and the bit of x^0 at position $d(x^0,y^0)$. We calculate y^1 and z^1 similarly. There is no edge (a,b), because the (j,k)-coloured path ends at a, so we set $a_1 = 0 \cdot a_0^0$. We use the symbol $\circ$ to denote concatenation of strings.

Initial Label	Label 1
x^0	$x^1 = d(x^0,y^0) \circ x^0_{d(x^0,y^0)}$
y^0	$y^1 = d(y^0,z^0) \circ y^0_{d(y^0,z^0)}$
z^0	$z^1 = d(z^0,a^0) \circ z^0_{d(z^0,a^0)}$
a^0	$a^1 = 0 \circ a^0_0$

Table 4.1: Vertex labels after 1 round of deterministic coin tossing.

Consider the new labels x^1, y^1, z^1 and a^1. We claim that the new labels are distinct for adjacent vertices along the path. Look at labels x^1 and y^1. There are two cases. If $d(x^0,y^0) \neq d(y^0,z^0)$, then $x^1 \neq y^1$ since x^1 contains $d(x^0,y^0)$ and y^1

contains $d(y^0, z^0)$. If $d(x^0, y^0) = d(y^0, z^0)$, then x^0 and y^0 differ in the same location as y^0 and z^0. But since the labels x^1 and y^1 include the bits of x^0 and y^0 at that location, which are different, $x^1 \neq y^1$. So we can conclude that $x^1 \neq y^1$ and $y^1 \neq z^1$. The case that remains is showing $z^1 \neq a^1$. There are again two cases. If $d(z^0, a^0) \neq 0$, then $z^1 \neq a^1$ because z^1 contains $d(z^0, a^0)$ and a^1 contains 0. If $d(z^0, a^0) = 0$, then z^0 and a^0 differ in position 0. But then z^1 and a^1 contain z_0^0 and a_0^0 respectively, so $z^1 \neq a^1$.

Now look at the size of the new labels. We stated that $d(x, y)$ is an $(\lceil \lg n \rceil)$-bit number when x and y are n-bits, so each new label is an $(\lceil \lg n \rceil + 1)$-bit number. We have therefore replaced an n-bit labeling with a $(\lceil \lg n \rceil + 1)$-bit labeling, such that each vertex's label along the path is different than its adjacent vertices' labels. We now simply repeat this process, calculating new labels from the previously computed labels in the same manner, and each label will be distinct from its two neighbours. Table 4.2 shows the computed labels after two rounds of deterministic coin tossing.

Initial Label	Label 1	Label 2
x^0	x^1	$x^2 = d(x^1, y^1) \circ x^1_{d(x^1, y^1)}$
y^0	y^1	$y^2 = d(y^1, z^1) \circ y^1_{d(y^1, z^1)}$
z^0	z^1	$z^2 = d(z^1, a^1) \circ z^1_{d(z^1, a^1)}$
a^0	a^1	$a^2 = 0 \circ a^1_0$

Table 4.2: Vertex labels after 2 rounds of deterministic coin tossing.

We repeat until a constant number of labels are used. Because we repeatedly replace n-bit labels with $(\lceil \lg n \rceil + 1)$ bit labels, the number of bits decreases *extremely* rapidly. The function $\lg^* x$ is defined to be the minimum $i \geq 0$ such that $\lg^i x \leq 1$. We show that only $O(\lg^* n)$ steps are needed for the deterministic coin tossing algorithm to reach a constant number of labels. Our argument follows that of [CLR90]. We

define r_i to be the number of bits in the label right before the i^{th} iteration of the algorithm. So $r_1 = \lceil \lg x \rceil$ and $r_{i+1} = \lceil \lg r_i \rceil + 1$. The initial label x is n bits. We show the following lemma.

Lemma 4.7 ([CLR90]) *If* $\lceil \lg^i x \rceil \geq 2$, *then* $r_i \leq \lceil \lg^i x \rceil + 2$.

Proof: The proof is by induction on i. When $i = 1$, $\lceil \lg x \rceil \geq 2$ as long as $x > 4$. It must be that $x > 4$, or else we already have reached a constant number of labels for the vertices, and don't need the deterministic coin tossing method. Now $r_1 = \lceil \lg x \rceil < \lceil \lg x \rceil + 2$, so the base case of $i = 1$ is true. Now assume that the statement is true for some $j < i$. We show it is also true for $j + 1$. We know that $r_{j+1} = \lceil \lg r_j \rceil + 1$, so

$$\begin{aligned}
r_j &= \lceil \lg r_{j-1} \rceil + 1 \\
&\leq \lceil (\lceil \lg^{j-1} x \rceil + 2) \rceil + 1 \quad \text{by I.H.} \\
&\leq \lceil \lg (\lg^{j-1} x + 3) \rceil + 1 \\
&\leq \lceil \lg (2 \lg^{j-1} x) \rceil + 1 \quad \text{since } \lceil \lg^j x \rceil \geq 2 \Rightarrow \lceil \lg^{j-1} x \rceil \geq 3 \\
&= \lceil \lg (\lg^{j-1} x) + 1 \rceil + 1 \\
&= \lceil \lg^j x \rceil + 2.
\end{aligned}$$

■

Now define $m = \lg^* x$, so $1 < \lg^{m-1} x \leq 2$ and $2 < \lg^{m-2} x \leq 4$. Since $\lceil \lg^{m-2} x \rceil > 2$, by Lemma 4.7

$$r_{m-2} \leq \lceil \lg^{m-2} x \rceil + 2 \leq 6. \tag{4.28}$$

So with $m - 2$ repetitions, we have computed labels with at most 6 bits. Another iteration of the algorithm will reduce this to at most $(\lceil \lg 6 \rceil + 1) = 4$-bit integers.

One more will reduce the labels to 3-bit integers. Now the 3-bit labels x and y may differ in positions $00, 01$ or 10, and for each of these x at that position may be 0 or 1. So an additional iteration of the algorithm when will give one of a possible 6 labels for each vertex. In total, at most

$$m - 2 + 3 = m + 1 \in O(\lg^* n) \tag{4.29}$$

iterations lead to a constant number (6) of labels. Note that for practical purposes, $O(\lg^* n)$ is as good as a constant, because for all real-world values of n, $\lg^*(n) \leq 7$.

At the end of the algorithm, when the final computed label for x, called (say) x^p, is one of six possible labels, we colour the edge (x, y) with the colour (j, k, x^p). Similarly, if we were colouring the edge (y, z) it would be coloured (j, k, y^p). Because $x^p \neq y^p$, the colours are different and the edge colouring is valid.

Now consider the dependencies when we are trying to compute a new label for x. To calculate x^1 we only use x^0 and y^0. To calculate x^2, we only need x^1 and y^1. To calculate x^3, we only need x^2 and y^2, but y^2 itself needs z^1. So at each step in the algorithm, we must traverse one more edge along the path. As there are only $O(\lg^* n)$ steps, we traverse a very short distance, and we compute at most $O((\lg^* n)^2)$ intermediate labels to compute the final label for x.

Recall in the case when we are colouring the edge (x, x), instead of using the deterministic coin tossing method, we just assigned the edge the colour $(j, j, 000)$. Any value could be used here, since a vertex x with a loop labeled (j, j) cannot have another j^{th} edge incident to it. In total, the colouring algorithm ensures each edge is assigned one of $6d^2 \in O(d^2)$ colours.

Performance of deterministic coin tossing

To calculate the final label for a vertex, we compute $O((\lg^* n)^2)$ intermediate labels. Each label takes $O(n)$ bit comparisons to compute, to find a bit where two labels differ. So, the total time is

$$O(n(\lg^* n)^2). \tag{4.30}$$

We consider the number of black box calls later.

4.7 Complete sparse Hamiltonian simulation algorithm

We now describe the complete simulation algorithm for a general sparse Hamiltonian. We first show a simple fact about matrix norms, and then introduce the Trotter formula, which shows that if $\sum_c H_c = H$ then e^{-iHt} may be approximated by interleaving short simulations of Hamiltonians $e^{-iH_c t}$.

Fact 4.3 *For matrices A, B C, if $\|A-B\| \leq \epsilon$ and $\|B-C\| \leq \delta$, then $\|A-C\| \leq \epsilon+\delta$.*

Proof: *Assume $\|A-B\| \leq \epsilon$ and $\|B-C\| \leq \delta$. So $\|A-C\| = \|A-B+B-C\| \leq \|A-B\| + \|B-C\| \leq \epsilon+\delta$ as desired.* ∎

4.7.1 Trotter formula

The standard 3^{rd} order Trotter formula as given in [NC00] shows that when H is a sum of two Hermitian matrices, $H = A + B$, that e^{-iHt} can be approximated by evolving each e^{-iA} and e^{-iB} for some time separately.

Lemma 4.8 (Trotter Formula) *Let $H = A + B$ with A, B, H Hermitian. Let $\|H\|, \|A\|, \|B\| \leq \Delta$. Then*

$$\|e^{-iH\delta} - e^{-iA\delta/2}e^{-iB\delta}e^{-iA\delta/2}\| \leq O(\delta\Delta)^3$$

provided $\delta\Delta < 1$.

We interpret the lemma as stating that simulating $e^{-iH\delta}$ with $e^{-iA\delta/2}e^{-iB\delta}e^{-iA\delta/2}$ induces an error of at most $O((\delta\Delta)^3)$. This can be verified (somewhat tediously) by taking Taylor series approximations. The lower order terms cancel, leaving only terms with $(\delta\Delta)^3$ and higher exponents. There are improvements which give a higher order Trotter formula in [Suz93], but we use this standard 3^{rd} order version in this thesis.

When H is the sum of more than two Hermitian matrices, we repeat applications of the Trotter formula. Suppose $H = \sum_{c=1}^{D} H_c$ with all H_c Hermitian and H, H_c all have norm bounded by Δ. We define U_δ to be the sequence

$$U_\delta = \left[e^{-iH_1\delta/2} \cdots e^{-iH_{D-1}\delta/2}e^{-iH_D\delta/2}\right]\left[e^{-iH_D\delta/2}e^{-iH_{D-1}\delta/2} \cdots e^{-iH_1\delta/2}\right], \tag{4.31}$$

and have

$$\|e^{-iH\delta} - e^{-iH_1\delta/2}e^{-i\sum_{c=2}^{D} H_c\delta}e^{-iH_1\delta/2}\| \leq O((\delta\Delta)^3) \tag{4.32}$$

Now we recursively use the Trotter formula to simulate $e^{-i\sum_{c=2}^{D} H_c\delta}$. From Fact 4.3, each use of the Trotter formula adds an error of $O(D(\delta\Delta)^3)$. One recursive use of the Trotter formula gives

$$\|e^{-iH\delta} - e^{-iH_1\delta/2}e^{-iH_2\delta/2}e^{-i\sum_{c=3}^{D} H_c\delta}e^{-iH_2\delta/2}e^{-iH_1\delta/2}\| \leq 2O((\delta\Delta)^3) \tag{4.33}$$

and $(D-1)$ uses gives

$$\begin{aligned} \|e^{-iH\delta} - U_\delta\| &\leq (D-1)O((\delta\Delta)^3) \\ &\in O(D(\delta\Delta)^3). \end{aligned} \tag{4.34}$$

Usually we want the error to be smaller than a constant, specifically on the order of $O(1/n^{O(1)})$. To achieve this, we will find that δ will have to scale inverse polynomially with D and Δ.

To simulate e^{-iHt}, we set $\delta = t/k$, and we repeat the simulation of $e^{-iH\delta}$ a total of k times, since

$$\left(e^{-iHt/k}\right)^k = e^{-iHt}. \tag{4.35}$$

Instead of applying k times $e^{-iHt/k}$, we apply k times the approximation U_δ. From Fact 1.4 in Chapter 1, each substitution of U_δ in place of $e^{-iH\delta}$ adds an error of $O\left(D(\delta\Delta)^3\right)$. So applying k times U_δ results in a total error of

$$\|e^{-iHt} - (U_\delta)^k\| \leq O\left(kD(\delta\Delta)^3\right) = O\left(\frac{Dt^3\Delta^3}{k^2}\right). \tag{4.36}$$

We set k later.

4.7.2 The algorithm

We are now able to give the complete simulation algorithm for simulating e^{-iHt}. We first review the pieces of the algorithm and then glue them together. From Sections 4.4 and 4.5 we can efficiently simulate very sparse Hamiltonians with all entries real or all entries imaginary. The colouring algorithm shows we can efficiently divide the Hamiltonian into pieces such that each is very sparse. The Trotter formula shows

that simulating each piece of a Hamiltonian gives a good approximation to simulating the whole Hamiltonian. The details follow.

The sparse Hamiltonian may be coloured in $6d^2$ colours. However, the circuits in Sections 4.4 and 4.5 can only simulate very sparse Hamiltonians that have either all real or all imaginary components. In general a Hamiltonian may have complex entries, of the form $a + ib$ with a, b real. We further divide each very sparse Hamiltonian into a sum of two Hamiltonians: one with only real entries and the other with only imaginary entries. We append an extra bit to the colour to indicate whether the weight is the imaginary or real component of the complete weight. As a result of the extra bit, we use at most $12d^2$ colours to completely colour the Hamiltonian. So each colour is a tuple of the form (j, k, f, g). Entries j and k store the neighbour ranks, f stores one of 6 possible colours as determined by deterministic coin tossing, and g indicates if the weight refers to the real or imaginary component.

We now show how to implement the M' black box we alluded to earlier, using the colouring algorithm and the Hamiltonian black box M. The M' black box operates as

$$M' |x\rangle |c\rangle |y\rangle |r\rangle = |x\rangle |c\rangle |y \oplus m(x)\rangle |r \oplus w(x)\rangle . \tag{4.37}$$

We implement M' as follows.

Algorithm for the unitary M' operator:

Input: vertex x, colour $c = (j, k, f, g)$.

Output: $m(x), w(x)$, where $m(x)$ is the vertex y that shares an edge of colour c with x, the weight $w(x)$ of that edge, or $m(x) = x$ and $w(x) = 0$ if the edge does not exist.

1. Check if vertex x has a j^{th} neighbour, and if that j^{th} neighbour's k^{th} neighbour is x. If so, call the neighbour y, otherwise return $m(x) = x$ and $w(x) = 0$.

2. Finish colouring the edge (x, y) using the deterministic coin tossing method, or append to the colour "000" if $x = y$. If the computed colour does not match the input colour f, return $m(x) = x$ and $w(x) = 0$.

3. Let the weight of the edge (x, y) be $a + ib$. If $g = 0$ and $a \neq 0$, return $m(x) = y$ and $w(x) = a$.

4. Otherwise if $g = 0$ and $a = 0$, return $m(x) = x$ and $w(x) = 0$.

5. Otherwise if $g = 1$ and $b \neq 0$, return $m(x) = y$ and $w(x) = b$.

6. Otherwise $g = 1$ and $b = 0$ so return $m(x) = x$ and $w(x) = 0$.

M' only returns vertices and weights corresponding to a colour c. In effect, given a colour, M' returns only vertices and weights in the subgraph induced by colour c.

We now analyze the time to implement M'. It first uses the black box M twice to find y, if it exists. It then must use deterministic coin tossing to compute a new label for x, and see if it matches with c. To do this, we have seen that the coin tossing method requires time $O(n(\lg^* n)^2)$. It also requires $O(\lg^* n)$ black box queries to M, to find up to $O(\lg^* n)$ vertices along the (j, k)-coloured path. Note that if we had used the $O(d)$ colouring algorithm of [DMP01], implementing M' would require more work, because the neighbour information j, k is not embedded in the colour. With M' defined, it is easy to simulate $e^{-iH_c\delta}$.

Algorithm to simulate $e^{-iH_c\delta}$:

Input: Hamiltonian H, colour c, time δ, and start state $|\psi(0)\rangle$.

Output: Approximation of $e^{-iH_c\delta}|\psi(0)\rangle$.

We assume $|\psi(0)\rangle = |x\rangle$. States other than basis states work by linearity.

1. If c refers to the real weight component then

   (a) Apply the very sparse real Hamiltonian simulation circuit from Section 4.4.3 to simulate $e^{-iH\delta}$ on input $|x\rangle$, where the two unitary black box M calls in the circuit are replaced with M' as defined above.

2. If c refers to the imaginary weight component then

   (a) Apply the very sparse imaginary Hamiltonian simulation circuit from Section 4.5.2 to simulate $e^{-iH\delta}$ on input $|x\rangle$, where the two unitary black box M calls in the circuit are replaced with M' as define above.

The algorithm works as follows. Depending on the colour, we use either the real or imaginary circuit to simulate the Hamiltonian. Because M' only returns entries of the subgraph induced by colour c from H, M' in effect returns only entries of H_c, so $e^{-iH_c\delta}$ is simulated. The complete algorithm to simulate e^{-iHt} is as follows.

Algorithm to simulate e^{-iHt}:

Input: Hamiltonian H, time t, error tolerance ϵ, and start state $|\psi(0)\rangle$.

Output: Approximation of $|\psi(t)\rangle = e^{-iHt}|\psi(0)\rangle$ to within ϵ accuracy.

1. Repeat k times (we set k in the next section):

   (a) For colour $c = 1, \ldots, 12d^2$ simulate $e^{-iH_ct/2k}$.

(b) For colour $c = 12d^2, \ldots, 1$ simulate $e^{-iH_c t/2k}$.

4.7.3 Performance and error analysis

We now analyze the performance of the simulation. To simulate each $e^{-iH_c t/2k}$, from Lemmas 4.3 and 4.5 the circuits use $O(n + r)$ unitary gates, plus two calls to the M' black box, and each induce an error of $O(1/2^r)$. As we saw previously, each application of M' uses time $O(n(\lg^* n)^2)$ plus $O(\lg^* n)$ black box queries to M. We colour the Hamiltonian with $D = O(d^2)$ colours, simulate $e^{-iH_c t/2k}$ for each colour, and repeat k times. The total time for the simulation is then

$$O(d^2 k \left(n + r + n(\lg^* n)^2\right)) \tag{4.38}$$

plus

$$O(d^2 k(\lg^* n)) \tag{4.39}$$

queries to M. The total error in the circuit is

$$O\left(\frac{d^2 k}{2^r} + \frac{d^2 t^3 \Delta^3}{k^2}\right). \tag{4.40}$$

We set (4.40) to be less than ϵ, and solve for k. So

$$O\left(\frac{d^2 k}{2^r} + \frac{d^2 t^3 \Delta^3}{k^2}\right) < \epsilon. \tag{4.41}$$

To achieve an error ϵ of an inverse polynomial in n, the first term disappears if we set $r = n$ and k and d are polynomial in n. We are then left with

$$O\left(\frac{d^2 t^3 \Delta^3}{k^2}\right) < \epsilon. \tag{4.42}$$

Setting k to

$$k = O\left(\frac{d t^{3/2} \Delta^{3/2}}{\sqrt{\epsilon}}\right) \tag{4.43}$$

satisfies this, and we find that the complete simulation takes time

$$O\left(\frac{d^3 t^{3/2} \Delta^{3/2} n (\lg^* n)^2}{\sqrt{\epsilon}}\right) \tag{4.44}$$

plus

$$O\left(\frac{d^3 t^{3/2} \Delta^{3/2} (\lg^* n)^2}{\sqrt{\epsilon}}\right) \tag{4.45}$$

queries to the Hamiltonian black box, for error $\epsilon \in 1/n^{O(1)}$. This completes the proof of Theorem 4.2.

Chapter 5

Conclusion

In this thesis we have given an improved algorithm for the approximate quantum Fourier transform mod 2^n. In time $O(n(\log\log(n/\epsilon))^2 \log\log\log(n/\epsilon))$ it computes the approximate QFT mod 2^n with error at most ϵ. We have also seen that the arbitrary modulus approximate QFT seems to be a more difficult problem. We gave an algorithm that computes the approximate QFT mod q in time equal to that of the 2^n case plus the cost of an n-bit multiplication and division. We presented a reduction from integer multiplication to computing the approximate QFT mod q. The best known algorithm for integer multiplication is that of Schönhage and Strassen [SS71] from 1971, which runs in time $O(n \log n \log\log n)$. The reduction from multiplication to the QFT mod q it hints that an algorithm for the QFT mod q with running time equal to that of the 2^n case (or at least better than $O(n \log n \log\log n)$) may be difficult to find. In fact, we showed the complete QFT mod q is not even required for multiplication. Rather, integer division reduces to constructing a quantum Fourier basis state, a seemingly easier problem. Still, constructing this state seems difficult without a multiplication. On the other hand, the reduction should provide motivation for studying the QFT and basis state construction. If a quantum Fourier basis state for arbitrary modulus can be constructed more quickly than classical multiplication, we immediately have a faster-than-classical quantum multiplication algorithm. This would raise eyebrows in the quantum community for a couple reasons. First, it would give another quantum algorithm that solves a real-life

problem more quickly than classically. It would also finally improve upon the bound of multiplication, which has not seen improvement for over 30 years. Of course, the improvement is by way of a quantum algorithm, not a classical one.

In the second part of the thesis, we showed an improved algorithm for simulating, with low error, the evolution of a quantum system when the system's Hamiltonian is sparse. For a sparse n-qubit Hamiltonian, simulating it takes time polynomial in n and t. One place for future investigation would be to try to incorporate the $O(d)$ colouring algorithm. As we speculated, this may or may not lead to a faster overall simulation time. It appears that, in general, at least time t is required using the Trotter formula method for simulation. That is, we cannot with this method predict the future state of a system at time t more quickly than t. This is because if we were to use an improved Trotter formula, the exponent of t in (4.44), the simulation time, would approach 1 as higher order Trotter formulae were used. It is also interesting that the best we can achieve with this method appears to be an error of an inverse polynomial in n. To get exponential precision, we must compute for an exponential amount of time. Further, the norm of the Hamiltonian factors into the running time of the simulation if it is not normalized. Because of this, it is not possible to "cheat" and simulate the system for a large t by multiplying the Hamiltonian by a large value, and thus have t small. Our final remark is that this method seems unable to be generalized to non-sparse Hamiltonians. In particular, it is certainly not possible to edge colour a graph with an exponential maximum degree. So, for non-sparse Hamiltonians a decidedly different strategy is likely needed.

Bibliography

[ATS03] D. Aharonov and A. Ta-Shma. Adiabatic quantum state generation and statistical zero knowledge. *Los Alamos Preprint Archive quant-ph/0301023*, 2003.

[BBBV97] C. Bennett, E. Bernstein, G. Brassard, and U. Vazirani. Strengths and weaknesses of quantum computing. *SIAM J. Comput.*, 26(5):1510–1523, 1997.

[BBC+95] A. Barenco, C. Bennett, R. Cleve, D. DiVincenzo, N. Margolus, S. Shor, T. Sleator, J. Smolin, and H. Weinfurter. Elementary gates for quantum computation. *Physical Review A*, 52(5):3457–3467, 1995.

[BH97] G. Brassard and P. Høyer. An exact quantum polynomial-time algorithm for Simon's problem. *Proc. of Fifth Israeli Symposium on Theory of Computing and Systems (ISTCS'97)*, pages 12–23, 1997.

[CEMM98] R. Cleve, A. Ekert, C. Macchiavello, and M. Mosca. Quantum algorithms revisited. *Proc. R. Soc. Lond. A*, pages 339–354, 1998.

[Cle94] R. Cleve. A note on computing quantum Fourier transforms by quantum programs. Manuscript. Available at http://www.cpsc.ucalgary.ca/~cleve/papers.html, 1994.

[CLR90] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms.* The MIT Press, first edition, 1990.

[Cop94] D. Coppersmith. An approximate Fourier transform useful in quantum factoring. Technical Report RC19642, IBM, 1994.

[CV86a] R. Cole and U. Vishkin. Deterministic coin tossing and accelerating cascades: micro and macro techniques for designing parallel algorithms. *Proceedings of the 18th Annual ACM Symposium on Theory of Computing (STOC)*, pages 206–219, 1986.

[CV86b] R. Cole and U. Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Information and Control*, 70:32–53, 1986.

[CW00] R. Cleve and J. Watrous. Fast parallel circuits for the quantum Fourier transform. *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS 00)*, pages 526–536, 2000.

[Deu85] D. Deutsch. Quantum theory, the church-turing principle and the universal quantum computer. *Proceedings Royal Society London*, A(400):96–117, 1985.

[DMP01] G. De Marco and A. Pelc. Fast distributed graph coloring with $o(\delta)$ colors. *Proc. 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'2001)*, pages 630–635, 2001.

[Gro96] L. Grover. A fast quantum mechanical algorithm for database search. *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 212–219, 1996.

[HH00] L. Hales and S. Hallgren. Improved quantum fourier transform algorithm

and applications. *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science (FOCS 00)*, 2000.

[Hol81] I. Holyer. The NP-completeness of edge-coloring. *SIAM J. Comput.*, 10(4):718–720, 1981.

[Kit95] A. Kitaev. Quantum measurements and the abelian stabilizer problem. *Los Alamos Preprint Archive quant-ph/9511026*, 1995.

[Lin92] N. Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, 1992.

[NC00] M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, 2000.

[Sho94] P. Shor. Algorithms for quantum computation: discrete logarithms and factoring. *Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS 94)*, pages 124–134, 1994.

[SS71] A. Schönhage and V. Strassen. Schnelle multiplikation großer zahlen. *Computing*, 7:281–292, 1971.

[Suz93] M. Suzuki. Improved Trotter-like formula. *Phys. Lett. A*, 180:232–234, 1993.

[SV93] M. Szegedy and S. Vishwanathan. Locality based graph coloring. *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 201–207, 1993.

[Viz64] V.G. Vizing. On an estimate of the chromatic class of a p-graph. *Diskret. Analiz*, 3(23), 1964.

[vzGG99] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.

Appendix A

Approximate Algorithms and Mixed States

From the postulates as stated we have that the state of a quantum system is represented as a unit vector in a state space. When the state is of this form we call the state a *pure state*. It turns out there is a more general representation available called the *mixed state* representation. Mixed states allow the state to be in a *probability distribution* of pure states. This could occur if, for example, we were to measure some of the qubits of a system, and then with some probability "throw away" or "forget" the measurement result. Suppose there is some collection $|\psi_i\rangle$ of pure states that the current system could be in. Then we write the state of the system ρ in the mixed state representation as

$$\rho = \sum_i p_i \, |\psi_i\rangle \, \langle\psi_i|$$

where $\sum_i p_i = 1$ with all p_i non negatives reals, and $|\psi_i\rangle \langle\psi_i|$ is the *outer product*, the matrix obtained by matrix multiplying the column vector $|\psi_i\rangle$ with its conjugate transpose, the row vector $\langle\psi_i|$.

Mixed states tend to be used more in the area of quantum information theory. They tend to carry with them additional mathematical baggage, which for our purposes, the development of quantum algorithms, is unnecessary. We mention them only for completeness, and to show that our method of using Euclidean distance of operators is sufficient to develop approximate quantum algorithms. We show that if an approximate quantum algorithm with a pure state as input produces an output

state with error (Euclidean distance) of at most ϵ, then it will with a mixed state as input produce an output mixed state very close to correct as well. This should be intuitively apparent, since mixed states are simply convex combinations of pure states, and so a convex combination of small errors will be small. To do this, we introduce *fidelity*, a way to measure closeness of mixed states.

For mixed states ρ, ξ, the fidelity F is defined as

$$F(\rho, \xi) = \mathrm{tr}\sqrt{\sqrt{\rho}\xi\sqrt{\rho}}$$

which ranges in value from 0 to 1. A fidelity of 1 indicates that the states are indistinguishable, and of 0 that the states can be distinguished with certainty. For ρ, ξ both pure states, the fidelity simplifies to $F(\rho, \xi) = |\langle\rho|\,|\xi\rangle|$. For a more thorough review of fidelity, see e.g. [NC00]. We require the following fact about inner products and Uhlmann's theorem to complete our argument. We also require the concept of a *purification*. A purification is just a conceptual tool for dealing with mixed states. It says that a mixed state in some Hilbert space H may be thought of as a pure state in a larger Hilbert space $H \otimes K$. This larger space does not need to be in our possession, nor do we care where it actually is. All we need to know is that it exists, and that when we perform operations on our space H, it behaves exactly like the mixed state.

Fact A.1 *For pure states* $|\psi\rangle$, $|\phi\rangle$, $|\langle\psi\,|\,\phi\rangle - 1| \leq \| \,|\psi\rangle - |\phi\rangle\,\|$.

Proof: $|\langle\psi\,|\,\phi\rangle - 1| = |\langle\psi|\,(|\psi\rangle + |\phi\rangle - |\psi\rangle) - 1| = |\langle\psi\,|\,\psi\rangle + \langle\psi|\,(|\phi\rangle - |\psi\rangle) - 1| =$ $|\langle\psi|\,(|\phi\rangle - |\psi\rangle)\,| \leq \| \,|\psi\rangle\,\| \| \,|\phi\rangle - |\psi\rangle\,\|$ (by Cauchy-Schwarz)$= \| \,|\psi\rangle - |\phi\rangle\,\|$. ∎

Theorem A.2 (Uhlmann's theorem) *For mixed states ρ, ξ in the Hilbert space $\mathcal{H}$, and another Hilbert space $\mathcal{K} = \mathcal{H}$,*

$$F(\rho, \xi) = \max |\langle \psi | \delta \rangle|$$

with the maximum taken over all purifications $|\psi\rangle, |\delta\rangle \in \mathcal{H} \otimes \mathcal{K}$ that purify ρ and ξ respectively.

Fact A.3 *For unitary operations U, V, a mixed state ρ, and $0 \leq \epsilon < 1$, if $\|U - V\| \leq \epsilon$ then the fidelity $F(U\rho U^\dagger, V\rho V^\dagger) \geq 1 - \epsilon$.*

Proof: Although U, V may be any unitary matrices, it may help to think of U as some exact algorithm, and V as an approximate version. We assume $\|U - V\| \leq \epsilon$, and so $\|U \otimes I - V \otimes I\| \leq \epsilon$ as well. So for any vector $|\psi\rangle$ of appropriate dimension, $\|(U \otimes I - V \otimes I)|\psi\rangle\| = \|U \otimes I |\psi\rangle - V \otimes I |\psi\rangle\| \leq \epsilon$.

Now let the mixed state ρ be in the Hilbert space $\mathcal{H}$, and let another Hilbert space $\mathcal{K} = \mathcal{H}$. Consider the purification $|\phi\rangle$ of ρ in $\mathcal{H} \otimes \mathcal{K}$. This purification exists such that $\rho = \mathrm{tr}_{\mathcal{K}} |\phi\rangle \langle\phi|$. From Fact A.1, we have that $|\langle\phi| (U^\dagger \otimes I)(V \otimes I) |\phi\rangle - 1| \leq \|U \otimes I |\phi\rangle - V \otimes I |\phi\rangle\| \leq \epsilon$. This implies that $1 - \epsilon \leq 1 - |\langle\phi| (U^\dagger \otimes I)(V \otimes I) |\phi\rangle - 1| \leq |\langle\phi| (U^\dagger \otimes I)(V \otimes I) |\phi\rangle|$.

Putting everything together, we establish the desired lower bound on the fidelity.

$$\begin{aligned} F(U\rho U^\dagger, V\rho V^\dagger) &= F(U \otimes I |\phi\rangle, V \otimes I |\phi\rangle) \\ &\geq |\langle\phi| (U^\dagger \otimes I)(V \otimes I) |\phi\rangle| \quad \text{by Uhlmann's theorem} \\ &\geq 1 - \epsilon. \end{aligned}$$

■

We conclude that to develop an approximate quantum algorithm, it is sufficient to use the measure of Euclidean distance.

Appendix B

Reduction of Multiplication to Division

Suppose we wish to calculate the product of two n-bit integers x and y. If we had a computer with infinite precision, we could calculate the product without error using only three division as follows:

1. $R \leftarrow 1/x$
2. $S \leftarrow R/y$
3. $T \leftarrow 1/S$

We may still use this method to calculate the product, with suitably accurate precision. To do this, we first calculate the precision required to represent inverses.

Let x be an n-bit integer. We wish to calculate how many bits are required to store x as $1/x$ such that x can be recovered with certainty. Let $\tilde{x}$ be the stored approximation to $1/x$, so

$$\tilde{x} = \frac{1}{x} + \delta \tag{B.1}$$

with $|\delta| < 1/2^k$ for some k. Taking the inverse of $\tilde{x}$ to recover x gives

$$\begin{aligned}
\frac{1}{\tilde{x}} &= \frac{1}{\frac{1}{x}+\delta} \\
&= \frac{1}{\frac{1+x\delta}{x}} \\
&= \frac{x}{1+x\delta} \\
&= \frac{\frac{x}{\delta}}{x+\frac{1}{\delta}} \\
&= \frac{\frac{x}{\delta}+x^2}{\frac{1}{\delta}+x} - \frac{x^2}{\frac{1}{\delta}+x} \\
&= x - \frac{x^2}{\frac{1}{\delta}+x}
\end{aligned} \tag{B.2}$$

The error term is $x^2/(1/\delta + x)$, which is always positive. If this error term is less than $1/2$, we may extract x using this method by simply rounding the inverse of $\tilde{x}$.

$$\frac{x^2}{\frac{1}{\delta}+x} < \frac{1}{2} \tag{B.3}$$

Solving B.3 for δ gives the constraint on δ that

$$\delta < \frac{1}{2x^2 - x}. \tag{B.4}$$

Setting $\delta < 1/(2^{2n+1})$ satisfies B.4. So using $k = 2n + 1$ bits to represent $1/x$ guarantees that we may recover x with certainty.

We use a similar calculation to determine the number of bits required to perform a multiplication with the above algorithm. We again let $\tilde{x}$ be our approximation to $1/x$ as in B.1. Then

$$S = \frac{R}{y} = \frac{\tilde{x}}{y} + \delta = \frac{1}{xy} + \frac{\delta}{y} + \delta. \tag{B.5}$$

The product $T = xy$ is

$$\begin{aligned} T = \frac{1}{S} &= \frac{1}{\frac{1}{xy} + \delta\left(\frac{1}{y}+1\right)} \\ &= \frac{1}{\frac{1+xy\delta\left(\frac{1}{y}+1\right)}{xy}} \\ &= \frac{xy}{1+xy\delta\left(\frac{1}{y}+1\right)} \\ &= \frac{\frac{xy}{\delta\left(\frac{1}{y}+1\right)}}{xy+\frac{1}{\delta\left(\frac{1}{y}+1\right)}} \\ &= \frac{\frac{xy}{\delta\left(\frac{1}{y}+1\right)} + (xy)^2 - (xy)^2}{xy+\frac{1}{\delta\left(\frac{1}{y}+1\right)}} \\ &= xy - \frac{(xy)^2}{xy+\frac{1}{\delta\left(\frac{1}{y}+1\right)}}. \end{aligned} \tag{B.6}$$

The error term is $(xy)^2 / \left(xy + \frac{1}{\delta((1/y)+1)}\right)$. Again we require this term to be less than $1/2$. Setting up this inequality and solving for δ gives

$$\begin{aligned} \frac{(xy)^2}{xy+\frac{1}{\delta\left(\frac{1}{y}+1\right)}} &< \frac{1}{2} \\ \delta &< \frac{1}{\left(2(xy)^2 - xy\right)\left(\frac{1}{y}+1\right)}. \end{aligned} \tag{B.7}$$

Setting $\delta < 1/2^{4n+2}$ satisfies this inequality. We conclude that to use the three-division algorithm above to perform multiplication, we must compute each step with $4n + 2$ bits of precision.