TUDelft

Quantum Algorithms and their Implementation on Quantum Computer Simulators

A thesis submitted to the
Faculty of Applied Sciences
Delft Institute of Applied Mathematics

by

Mike van der Lans

in partial fulfilment of the requirements for the degree of

Bachelor of Science

in

Applied Physics

and

Applied Mathematics

Delft, The Netherlands
June 2017

Quantum Algorithms and their Implementation on Quantum Computer Simulators

Mike van der Lans

Delft University of Technology

Supervisors	**Other committee members**
Dr. C.G. Almudever	Dr. W.G.M. Groenevelt
Dr. M. Möller	Dr. D. Elkouss

Delft
June 19, 2018

Abstract

Quantum computation is becoming an increasingly interesting field, especially with the rise of real quantum computers. However, current quantum processors contain a few tens of error-prone qubits and the realization of large-scale quantum computers is still very challenging. Therefore, quantum computer simulators are particularly suitable for testing and analysing quantum algorithms without having a real quantum computer at one's disposal. In this thesis, different quantum algorithms such as Grover's and Shor's algorithm as well as key quantum routines such as the Quantum Fourier Transform (QFT) and a quantum adder/subtractor are described and analysed (optimal number of iterations, time complexity). Some of them have been implemented for an arbitrary number of qubits and have been simulated using two different quantum simulators, the QX simulator developed at QuTech and the Liquid simulator from Microsoft. In addition, how errors affect the success rate of the algorithms has been investigated.

Contents

Introduction

With the rise of real quantum computers, quantum computing is becoming an increasingly interesting field in both physics and mathematics. With large players like Intel, IBM and Microsoft jumping on the hype train, the development is accelerating quickly. Quantum computers with up to 50 qubits are already available at TU Delft and IBM has built a machine with 50 qubits as well.

However, the realization of large-scale physical quantum computers is not only very challenging, but also very costly. Consequently, quantum computers are still scarce to people outside of the research groups developing them and the available quantum computers have at most a few dozen qubits. Besides, the stability of quantum computers and the exclusion of errors is still a real challenge.

For this reasons and because only a few quantum algorithms are available to test these designs, quantum computer simulators are a very good tool for developing and testing quantum algorithms. Several simulators have been developed over the last few years. Microsoft has developed a simulator called Microsoft LIQ$i|\rangle$ in 2016. QuTech, an advanced research centre for quantum computing (a collaboration between TU Delft and TNO) followed later with a simulator called QX Quantum Computing Simulator.

This thesis has two aims, the first one is to understand, analyse and extend two well-known quantum algorithms: Grover's search algorithm (1996) and Shor's algorithm for integer factorization (1994). To this purpose, the quantum circuit for Grover's algorithm and its time complexity as well as its optimal number of iterations are analysed. In addition, the Grover's algorithm is extended for searching multiple values instead of just one. Furthermore, an application of this algorithm to the Boolean satisfiability problem (SAT-problem) is described.

The quantum Fourier transform (QFT), a routine that plays a key role in many quantum algorithms like the fast Fourier transform does in classical algorithms, is analysed in terms of its circuit implementation and time complexity. This QFT is then used for building a quantum adder/subtractor. These two modules, the QFT and the adder/subtractor, are essential blocks of the period finding quantum algorithm on which Shor's algorithm is based. The implementation and the time complexity of Shor's algorithm are analysed as well.

The second aim of this thesis is the creation of a library from which quantum algorithms and subroutines can be called. To accomplish this, Grover's algorithm as well as its extension to multi-search, the QFT and the quantum adder/subtractor have been implemented and simulated (verified) using the QX quantum computer simulator and the Microsoft Liquid simulator. In addition, an error analysis for some of the algorithms has been performed.

Furthermore, the second aim includes gaining experience in using QX Simulator as well as Microsoft Liquid in order to discuss these quantum programming frameworks and formulate requirements for making quantum programming more attractive.

Besides, the extensive introductions and the structure of this thesis make it well suited to serve as a handbook for students and other people that try to find their way into quantum computing and the implementation of quantum algorithms.

In this thesis, some new contributions to knowledge beyond the existing literature are presented. First of all, the formula for determining the optimal number of iterations in Grover's algorithm is refined. Additionally, the extension of Grover's algorithm to a multi-search algorithm and its application to the SAT-problem are new contributions. To the quantum addition subroutine, the time complexity is added and a quantum adder/-subtractor is designed. Finally, the implementation and simulation of the algorithms in QX Simulator and Liquid as well as the error analysis contribute to knowledge beyond the existing literature.

This thesis is organized as follows. In section 1, the basic concepts of quantum computing are introduced. In section 2, Grover's search algorithm is described, analysed and extended to a multi-search algorithm along with an application to the SAT-problem. In section 3, the QFT is introduced. One of its applications, the addition circuit, is described in section 4. This circuit is extended to a quantum adder/subtractor. In section 5, Shor's algorithm for factoring numbers is introduced and analysed. In section 6, the implementation of Grover's algorithm, the QFT and the circuit for adding and subtraction integer numbers in QX Quantum Simulator and Microsoft Liquid is discussed and both simulators are compared. The results of the error analysis on these algorithms is described in section 6 as well. Section 7 gives a summary of the main findings and conclusions as well as some recommendations for further research.

In quantum computation, there is just a small line between physics and mathematics. Quantum algorithms are based on physical properties of qubits, which are usually described mathematically, but attain to solve physical problems. The introductions (section 1) contains quite some physical topics but also introduce the mathematics necessary to describe the physics. The (numerical) analysis of Grover's algorithm and the multi-search algorithm (section 2) and the derivation of the QFT (section 3) mostly contain mathematical topics. The description of quantum addition as well as the quantum adder/subtractor (section 4) and the description of Shor's algorithm (section 5) contain some physical as well as some mathematical topics. The simulations of the algorithms (section 6) mostly contains physical topics, especially the use of an error model to simulate physical errors in the quantum circuits.

This thesis is written in partial fulfilment of the requirements for the degree of Bachelor of Science in Applied Physics and Applied Mathematics.

1 Introduction to Quantum Algorithms

In this chapter, the basic concepts of quantum computing are introduced. First of all, a short introduction into quantum mechanics is given in section 1.1. Then, in section 1.2, quantum computation is introduced. In section 1.3, it is shown how quantum gates can be used to build basic quantum circuits. Finally, quantum algorithms are introduced in section 1.4.

1.1 Quantum Mechanics

The theoretical framework of the behaviour of particles and light on atomic scale is described by the theory of quantum mechanics. In quantum mechanics, the boundary between particles and waves fades. In some cases, atomic objects (electrons, protons, photons etc.) behave like particles, in other cases like waves.

In particular, particles can be described by their wave function. This wave function $\Psi(x,t)$ satisfies the Schrödinger equation, derived by Erwin Schrödinger in 1925.

$$i\hbar\frac{\partial\Psi}{\partial t} = -\frac{\hbar^2}{2m}\frac{\partial^2\Psi}{\partial x^2} + V\Psi \tag{1.1}$$

Here, $\hbar = \frac{h}{2\pi}$ in which h is Planck's constant, t is the time, m is the particle's mass, x is the particle's position and V is some potential. This equation can be solved using separation of variables. This results in some solution $\Psi(x,t) = \psi(x)\phi(t)$. Although the time evolution plays a significant role in quantum computation, it is beyond the scope of this thesis to take it into account. Therefore, only the time-independent solution $\psi(x)$ is of interest. These time-independent solutions satisfy the time-independent Schrödinger equation,

$$-\frac{\hbar^2}{2m}\frac{\partial^2\psi}{\partial x^2} + V\psi = E\psi \tag{1.2}$$

in which E is a constant that is equal to the total energy of the system.

Another result from separation of variables that is well-known is that the general solution of the Schrödinger equation is as linear combination of separable solutions. Therefore, the solution of the time-independent Schrödinger equation can be written in the form

$$\psi(x) = \sum_{n=1}^{\infty} c_n\psi_n(x) \tag{1.3}$$

in which $\psi_n(x)$ is a solution of the time-independent Schrödinger equation with associated energy E_n. Moreover, the ψ_n's are orthogonal, i.e. $\langle\psi_i|\psi_j\rangle = \delta_{ij}$.

If a measurement is performed on the quantum state, only one of the E_n's will be found. Simultaneously, the quantum state will *collapse* to the ψ_n associated with this E_n. The probability that a measurement yields outcome E_n is given by the squared amplitude of the corresponding coefficient $|c_n|^2$. Of course, the sum of the probabilities should be 1. Because different ψ_n's are orthogonal, this translates into the following condition.

$$\sum_{n=1}^{\infty} |c_n|^2 = 1 \tag{1.4}$$

Therefore, the coefficients c_n are also known as normalization coefficients.

For quantum computation, we consider quantum particles that have two possible quantum states:

$$\psi(x) = \alpha\psi_1(x) + \beta\psi_2(x) \tag{1.5}$$

in which α and β are the normalization constants that satisfy $|\alpha|^2 + |\beta|^2 = 1$. These particles are similar to bits that are used in classical computers. Yet, the strange behaviour of quantum particles can be exploited to achieve some remarkable results.

1.2 Quantum Computation

1.2.1 Single Qubit States

All digital processes are governed by bits. These bits store information and can be used to process this information. One may think of a bit as a small light bulb. It can be either on (commonly represented by 1) or off (commonly represented by 0).

A quantum bit (qubit) is much like a classical bit. Just like a classical bit, which can be 0 or 1, a qubit is also in some state. For example, it could be $|0\rangle$ or $|1\rangle$ (the so-called bra-ket[1] notation used for quantum states was introduced by Dirac[12] in 1939). However, quantum mechanics provides us with more than just those two states - moreover, it provides us with an *infinite* number of states. To be specific, a qubit could be in any linear combination of states,

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{1.6}$$

with $\alpha, \beta \in \mathbb{C}$ the normalization coefficients. In quantum mechanics, a qubit which is not in a single well-defined state but in a linear combination of states, is said to be in a *superposition* of states. When a measurement is performed on a qubit in superposition, it will collapse to one of the well-defined states (in this case $|0\rangle$ or $|1\rangle$). The probability to find $|0\rangle$ is equal to $|\alpha|^2$, the probability to find $|1\rangle$ is equal to $|\beta|^2$. Of course, we always

[1]A quantum state is generally called $|\psi\rangle$ ("ket psi") and its hermitian conjugate $|\psi\rangle^\dagger$ as $\langle\psi|$ ("bra psi").

find one of both states. The probabilities should therefore sum to one, so we must have $|\alpha|^2 + |\beta|^2 = 1$ to have a normalized quantum state.

Every quantum state can be represented in vector notation by defining $|0\rangle \equiv \begin{bmatrix}1\\0\end{bmatrix}$ and $|1\rangle \equiv \begin{bmatrix}0\\1\end{bmatrix}$. This yields

$$|\psi\rangle = \alpha \begin{bmatrix}1\\0\end{bmatrix} + \beta \begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}\alpha\\\beta\end{bmatrix} \tag{1.7}$$

Normalization can now be expressed as an inner product,

$$\langle\psi|\psi\rangle = \begin{bmatrix}\overline{\alpha} & \overline{\beta}\end{bmatrix} \begin{bmatrix}\alpha\\\beta\end{bmatrix} = \alpha^*\alpha + \beta^*\beta = |\alpha|^2 + |\beta|^2 = 1 \tag{1.8}$$

Two quantum states $|\psi_1\rangle = \begin{bmatrix}\alpha_1\\\beta_1\end{bmatrix}$ and $|\psi_2\rangle = \begin{bmatrix}\alpha_2\\\beta_2\end{bmatrix}$ are mutually orthogonal if and only if their inner product is zero,

$$\langle\psi_1|\psi_2\rangle = \begin{bmatrix}\overline{\alpha_1} & \overline{\beta_1}\end{bmatrix} \begin{bmatrix}\alpha_2\\\beta_2\end{bmatrix} = \overline{\alpha_1}\alpha_2 + \overline{\beta_1}\beta_2 = 0 \tag{1.9}$$

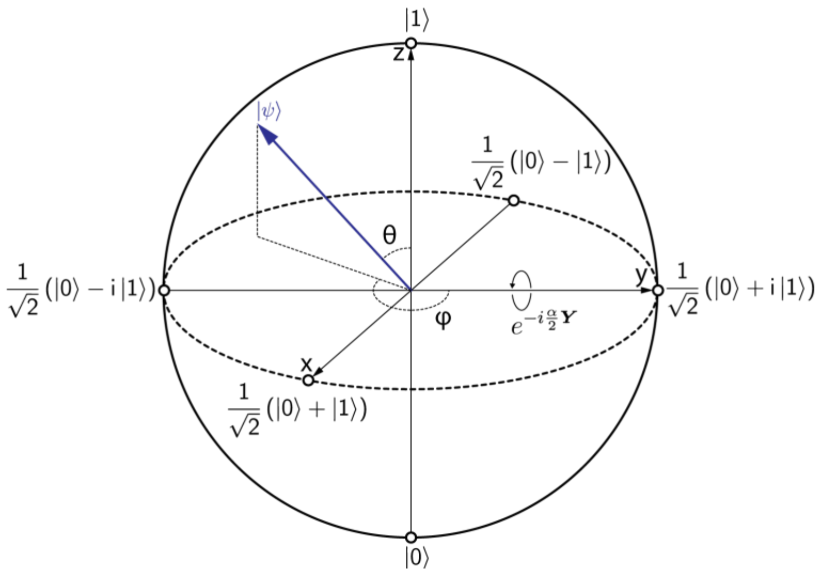
Clearly, $|0\rangle$ and $|1\rangle$ are orthogonal states. Therefore, $|0\rangle, |1\rangle$ forms a basis. This is called the computational basis, since 0 and 1 are often used to perform computations in computers[26].

Since $|\psi\rangle$ is normalized, we can express it as

$$|\psi\rangle = e^{i\delta}\left(\cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle\right) \tag{1.10}$$

with $\theta \in [0, \pi]$, $\phi \in [0, 2\pi]$ and δ some real number. The value of δ has no physical consequences, because its probability amplitude squared equals one and its contribution to both $|0\rangle$ and $|1\rangle$ is equal. The second part is the reason that $e^{i\delta}$ is referred to as the global phase.

All single qubit states can be visualized in spherical coordinates with θ the polar angle and φ the azimuthal angle (δ is not included, since it has no physical consequences). This is the so-called Bloch sphere representation. Quantum states that lie on the axes have their own representation and are known as Clifford states. In figure 1.1, these states are displayed in the Bloch sphere[26].



$|1\rangle$
z
$|\psi\rangle$
$\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$
θ
$\frac{1}{\sqrt{2}}(|0\rangle - i\,|1\rangle)$
y
$\frac{1}{\sqrt{2}}(|0\rangle + i\,|1\rangle)$
φ
$e^{-i\frac{\alpha}{2}\mathbf{Y}}$
x
$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$
$|0\rangle$

Figure 1.1. The Clifford states in the Bloch sphere. Reprinted from *Programming for the quantum computer* (Dickel, 2016).

The Bloch sphere is a nice way to visualize quantum states and to identify orthogonal states. Furthermore, because diametrically opposite states in the Bloch sphere are orthogonal it also gives insight in which particular states are orthogonal. However, it is only possible to do this for single qubit states. When multiple qubit states are considered, it is not possible to visualize states in such a way[26].

1.2.2 Multiple Qubit States

Multiple qubits can be combined into a so-called qubit register. For example, if a pair of qubits is taken together they could be in any linear combination of states. Two qubits can constitute four different states,

$$|\psi\rangle = \alpha\,|00\rangle + \beta\,|01\rangle + \gamma\,|10\rangle + \delta\,|11\rangle \tag{1.11}$$

with $\alpha, \beta, \gamma, \delta \in \mathbb{C}$ the normalization coefficients: $|\alpha|^2 + |\beta|^2 + |\gamma|^2 + |\delta|^2 = 1$. Again, the probability of finding one specific state if a measurement is performed is equal to the square of the amplitude of the corresponding normalization coefficient.

There is one particular interesting example of a two qubit state.

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \tag{1.12}$$

This state is one of the four Bell states which plays a central role in the famous Einstein-Podolsky-Rosen paper[14], because the qubits are *entangled*. If the first qubit is measured, the outcome is either $|0\rangle$ or $|1\rangle$. If the outcome is, for example, $|0\rangle$ the state collapses to a linear combination of all quantum states in which the first qubit is $|0\rangle$. This results in $|\psi\rangle = |00\rangle$. On the other hand, if the outcome is $|1\rangle$ the quantum state will collapse to $|\psi\rangle = |11\rangle$. So if only the first qubit is measured, we know for sure that the second qubit is in the same state. Thus, the state of the second qubit cannot be described independently of the state of the first qubit. This interesting property has several applications, for example in the development of a quantum internet that cannot be eavesdropped on (see [22] for more on this subject).

Similar to single qubit states, different two qubit states can be defined as vectors to ease up the calculations in a later stage.

$$|00\rangle \equiv \begin{bmatrix}1\\0\\0\\0\end{bmatrix} \quad |01\rangle \equiv \begin{bmatrix}0\\1\\0\\0\end{bmatrix} \quad |10\rangle \equiv \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \quad |11\rangle \equiv \begin{bmatrix}0\\0\\0\\1\end{bmatrix} \tag{1.13}$$

Thus, $|\psi\rangle$ can be rewritten as

$$|\psi\rangle = \alpha \begin{bmatrix}1\\0\\0\\0\end{bmatrix} + \beta \begin{bmatrix}0\\1\\0\\0\end{bmatrix} + \gamma \begin{bmatrix}0\\0\\1\\0\end{bmatrix} + \delta \begin{bmatrix}0\\0\\0\\1\end{bmatrix} = \begin{bmatrix}\alpha\\\beta\\\gamma\\\delta\end{bmatrix} \tag{1.14}$$

This notation can be generalized to a register containing any number of qubits. Note that the number of states doubles with each qubit that is added. Therefore the total number of states scales as 2^n, where n is the number of qubits in the register. At any time, the quantum register can be in a linear superposition of all of these states. On the contrary, a register containing classical bits can only be in one particular state at a time. This is one of the two main advantages that qubits have over classical bits.

Because the quantum register can be in a linear combination of all possible qubit states, multiple states can be evaluated and modified simultaneously. Therefore, computations can be run in parallel without any extra cost. Matching the performance of a quantum computer containing n qubits with a classical computer requires exponentially more bits.

Because of these advantages that quantum computers have over classical devices, quantum computers are being developed. With a quantum computer, these advantages can be exploited in a to speed up several classical algorithms. This results in some very interesting *quantum algorithms.* A quantum algorithm takes a qubit register and modifies the qubit values and coefficients of the qubit states in a clever way using several *quantum operators.*

There is one property of qubits that can be a drawback if it is not dealt well with. This is the fact that quantum computation is often non-deterministic. There is a certain probability of finding the qubits in some state. Therefore, it can occur that a wrong outcome is obtained after performing a measurement. For that reason, one should design a quantum algorithm in such a way that this probability is minimized.

1.2.3 Single Qubit Operations

In short, there are two types of operations that can be carried out on a single qubit. The first one is measurement. A measurement on a qubit can be performed in any orthogonal basis (diametrically opposite states on the Bloch sphere). After measuring a qubit, superposition is destroyed. Its wave function will collapse to one of the states in the orthogonal measuring basis. The outcome of the measurement is the eigenvalue belonging to the state it collapsed to. If we measure $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ in the computational basis, $\{|0\rangle, |1\rangle\}$, the qubit will either collapse to $|0\rangle$ (with probability $|\alpha|^2$) or to $|1\rangle$ (with probability $|\beta|^2$).

In addition, a basis transformation can be carried out. For example, the state can be expressed in the Hadamard basis, $\{|+\rangle, |-\rangle\}$, instead of the computational basis. It can be easily verified that $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \frac{\alpha+\beta}{\sqrt{2}}|+\rangle + \frac{\alpha-\beta}{\sqrt{2}}|-\rangle$. After measuring, the wave function either collapses to $|+\rangle$ (with probability $\frac{|\alpha+\beta|^2}{2}$) or to $|-\rangle$ with probability $\frac{|\alpha-\beta|^2}{2}$. However, throughout this thesis, measurements are usually performed in the computational basis, unless specified otherwise.

The second type of operations are linear operators. In computer science, these linear operators are referred to as logic gates. Logic gates are also used in classical computing. For example, there is the NOT gate. The NOT gate flips the bit it is applied to, so 0 becomes 1 and 1 becomes 0. The quantum analogue of the NOT gate should take $|0\rangle$ to $|1\rangle$ and $|1\rangle$ to $|0\rangle$, so if the qubit is in a superposition state, the quantum NOT gate transforms the input state $\alpha|0\rangle + \beta|1\rangle$ to $\alpha|1\rangle + \beta|0\rangle$. Since all qubit gates are linear operators, they can be denoted as a transition matrix. The quantum analogue of the NOT gate is for historical reasons referred to as the Pauli-X gate, after the Austrian physicist Wolfgang Pauli who was one of the pioneers in quantum physics,

$$X \equiv \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{1.15}$$

It is not hard to see that this operator serves its purpose, indeed if $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ then

$$X|\psi\rangle \equiv \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \alpha \end{bmatrix} \equiv \beta|0\rangle + \alpha|1\rangle \tag{1.16}$$

Qubit gates should maintain the normalization of the wave function, else the operator maps the input quantum state to an invalid quantum state. Therefore, the operators should be unitary. Consequently, all quantum gates are reversible as well so they obey time reversal symmetry[7]. The transition matrices that represent a quantum gate should therefore be unitary as well.

Definition 1.1. *Let U be some invertible matrix and denote its conjugate transpose by $U^\dagger$. Then U is a unitary matrix if and only if $UU^\dagger = U^\dagger U = I$.*

From this, it also follows that $U^{-1} = U^\dagger$. So if an operator U is applied to some state $|\psi\rangle$ and afterwards $U^\dagger$ is applied, the output state is again $|\psi\rangle$.

To show that unitary matrices indeed preserve normalization, let $|\psi\rangle$ be some state that is normalized (i.e. $\langle\psi|\psi\rangle = 1$). Then

$$\langle U\psi|U\psi\rangle = \langle\psi| U^\dagger U |\psi\rangle = \langle\psi|\psi\rangle = 1 \tag{1.17}$$

It is easily verified that the Pauli-X gate is indeed unitary.

Moreover, if a unitary operator is real and symmetric (as is the case for X), $U^\dagger = U$. Thus if we apply U twice to $|\psi\rangle$, we end up with $UU|\psi\rangle = UU^\dagger|\psi\rangle = I|\psi\rangle = |\psi\rangle$.

Besides the Pauli-X gate, there are the Pauli-Y gate and Pauli-Z gate. The Pauli-Y gate is not used often, but it is included for completeness. The Pauli-Y gate is given by

$$Y \equiv \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \tag{1.18}$$

And the Pauli-Z gate is given by

$$Z \equiv \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{1.19}$$

This gate takes the input state $\alpha|0\rangle + \beta|1\rangle$ to $\alpha|0\rangle - \beta|1\rangle$. Note that it can be verified that both of these gates are unitary. Also, it turns out that $Y = XZ = ZX$.

One may wonder why these gates are called X, Y and Z you need to go back to the Bloch sphere from figure 1.1. The Pauli gates rotate a single qubit state by 180° around the x-, y- and z-axis of the Bloch sphere respectively.

The Hadamard gate, which is named after the French mathematician Jacques Hadamard, is used in virtually any quantum algorithm. The Hadamard gate (H) is given by

$$H \equiv \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{1.20}$$

This gate is important because it takes $|0\rangle$ or $|1\rangle$ to an equal superposition of both. To be precise, it takes $|0\rangle$ to $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and $|1\rangle$ to $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$. You may recognize these states, these are exactly the $|+\rangle$ and $|-\rangle$ states we encountered while discussing the Bloch sphere. Thus it turns out that the Hadamard gate can also be described by a rotation on the Bloch sphere. In particular, it is a 180° rotation around the line $x = z$. Note that this is more obvious than it seems, because $H = \frac{X+Z}{\sqrt{2}}$.

Finally, there are two gates that are closely related to the Pauli-Z gate. The S gate is defined as

$$S \equiv \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \tag{1.21}$$

and the T gate as

$$T \equiv \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} \tag{1.22}$$

Similar to the Pauli-Z gate, the S gate and T gate rotate a single qubit state around the z-axis. Whereas the Pauli-Z gate rotates the state by 180°, the S gate rotates the state by 90° and the T gate by 45°. Clearly, applying the S gate twice results in a 180° rotation. This implies that $S^2 = Z$. Similarly, $T^2 = S$ (so $T^4 = Z$).

Of course, there are many more single qubit gates, in fact there is an infinite number of them. The only requirement is that the operator is unitary. However, these are the single qubit gates that are of most interest in most quantum algorithms.

1.2.4 Multiple Qubit Operations

A single qubit gate can also be applied to an arbitrary register of qubits. First, some notation is introduced. If A and B are $n \times n$ matrices, the *Kronecker product* $A \otimes B$ is defined as

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{n1}B & \cdots & a_{nn}B \end{bmatrix} \tag{1.23}$$

For example, consider the X gate that is applied the two qubit register $|01\rangle$.

$$(X \otimes X)\,|01\rangle \equiv \left(\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \right) \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \equiv |10\rangle \tag{1.24}$$

Similarly, if it is desirable to only apply the X gate to the second qubit and leave the first qubit unchanged,

$$(I \otimes X)\,|01\rangle \equiv \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \right) \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \equiv |00\rangle \tag{1.25}$$

Besides single qubit gates, there are also a number of qubit gates that require at least two qubits. These gates are controlled by one or multiple qubits, and executed on only a single qubit. The easiest multiple qubit gate is the controlled not gate (CNOT, also referred to as CX). There is one control qubit and one qubit on which the gate may be executed depending on the value of the first qubit. If qubit one is $|0\rangle$, nothing happens. However, when qubit one is $|1\rangle$ the Pauli-X gate (see equation (1.15)) is executed on the second qubit. The control qubit always goes through unchanged. So, if the input state is $\alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle$ in which the first qubit is the control and the second qubit is the target, the output is $\alpha|00\rangle + \beta|01\rangle + \gamma|11\rangle + \delta|10\rangle$. Therefore, the CNOT gate is given by

$$\text{CNOT} \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{1.26}$$

In fact, a control qubit can be added to any single qubit gate. If the control qubit is $|0\rangle$, the gate is not executed. On the other hand, if it is $|1\rangle$ the single qubit gate is executed on the target qubit. As an example, consider the controlled Z gate. If the input state is $\alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle$ in which the first qubit is the control and the second qubit is the target, the output is $\alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle - \delta|11\rangle$. The matrix of the controlled Z gate is given by

$$\text{C(Z)} \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \tag{1.27}$$

In general, if A is any unitary operator a control qubit can be added. Let $C(A)$ denote the operator that results from this addition. If $C(A)$ is controlled by the first qubit and targeted at the second qubit, its matrix is given by

$$C(A) = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \otimes I + \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \otimes A = \begin{bmatrix} I & \begin{matrix} 0 & 0 \\ 0 & 0 \end{matrix} \\ \begin{matrix} 0 & 0 \\ 0 & 0 \end{matrix} & A \end{bmatrix} \tag{1.28}$$

Remember if A and B are 2×2 matrices then $A \otimes B \equiv \begin{bmatrix} a_{11}B & a_{12}B \\ a_{21}B & a_{22}B \end{bmatrix}$.

There are two qubit gates other than controlled single qubit gates. An example is the SWAP gate that switches the position of two qubits. $|00\rangle$ and $|11\rangle$ are thus unchanged under the swap operation. $|01\rangle$ and $|10\rangle$, on the other hand, are mapped to $|10\rangle$ and $|01\rangle$

respectively. Its matrix representation is therefore given as

$$\text{SWAP} \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{1.29}$$

The swap gate does not really change the state of a quantum register. It merely changes the order of the qubits in the register.

Another gate that is almost analogous to the CNOT gate is the Toffoli gate, also known as CCNOT. This is a three qubit gate that applies the Pauli-X gate to a qubit only if *two* control qubits are both $|1\rangle$.

In table 1.1 on page 14, an overview of all gates discussed in this section is given. Along with the matrix representation that can be used to calculate the output state from a given input, the representation of each gate in a circuit is included. These circuit representations will be used throughout this thesis in a multitude of quantum circuits.

It turns out that all qubit gates can be described by only four different gates. This is a so-called universal set of quantum gates. One such set consists of the Hadamard gate, the S gate, the CNOT gate and the Toffoli gate[30].

1.3 Quantum Circuits

Quantum gates on their own are not of much use. But by coupling quantum gates together, a more complex *quantum circuit* can be build. Quantum circuits are composed by qubits and gates operating on them. These gates can manipulate the value of the qubit (e.g. the X gate) or the phase of a qubit (e.g. the Z gate), or both (e.g. the Y gate). The most basic quantum circuit, which can be used as random number generator, is the quantum equivalent of flipping a coin. Initially, we have one qubit that is $|0\rangle$. Then, the Hadamard gate is applied. Using equation (1.20), we find that the final state is $|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. When $|\psi\rangle$ is measured, both $|0\rangle$ and $|1\rangle$ can be obtained with equal probability of $\frac{1}{2}$. This is a truly random coin flip, something that cannot be accomplished classically.

A very basic circuit can be used to entangle two qubits in a Bell state as described by equation (1.12). A schematic of the circuit is given in figure 1.2. We start out with two qubits in $|0\rangle$. Then, the Hadamard gate is applied to the first qubit followed by the CNOT on the second qubit controlled by the first qubit. The result of this small circuit can be found by sequentially looking at the quantum gates. The initial state is given by $|\psi\rangle = |0\rangle \otimes |0\rangle$, or $|\psi\rangle = |00\rangle$ for short. Using equation (1.20), the state after the Hadamard gate is given by $\frac{1}{\sqrt{2}}(|00\rangle + |10\rangle)$. Then, according to equation (1.26) the CNOT gate leaves $|00\rangle$ unchanged and changes $|10\rangle$ to $|11\rangle$, so the final state is $|\psi\rangle = \frac{1}{\sqrt{2}} |00\rangle + |11\rangle$.

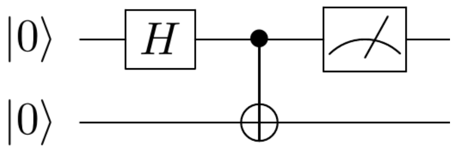


Figure 1.2. The quantum circuit to entangle two qubits.

Finally, a note on the description of quantum circuits is in place. Quantum circuits described as gates and schematics are merely some kind of model to describe quantum algorithms. In practice, chips are of course not 'wired' together in order to perform instructions. One way to execute a quantum circuit is translating these instructions into microwave pulses with which the physical state of some particle acting as a qubit is manipulated.

1.3.1 Perfect Qubits and Perfect Operations

Throughout most of the calculations in this thesis, the qubits are assumed to be perfect. In that case, once a quantum register is in a certain quantum state, it will stay in that state forever. In addition the operations that are performed on the qubit register yield the correct and predicted outcome consistently. It is important to note, however, that in reality this is not the case. There is a certain level of noise on the qubits and on the operations. The main source of errors is decoherence: interactions with an uncontrolled environment. This stems from the fact that no quantum system can be perfectly isolated from its surroundings. Similar to a cup of coffee that loses its heat to the environment if left alone for a while, a quantum system loses its relevant quantum behaviour due to interactions with the outside world.

In order to run quantum algorithms reliably, quantum error correction has to be applied. Since quantum error correction is not a concern throughout this thesis, the reader is refered to [11] for a guide on quantum error correction.

1.4 Quantum Algorithms

A larger number of quantum gates can be combined to build more complex quantum circuits. By combining these gates in a clever way, calculations can be performed. Such a clever combination of quantum gates is described by a quantum algorithm. A quantum algorithm modifies the probability amplitudes of the qubit states.

As described in section 1.2, due to the nature of qubits a wide variety of processes for which classical algorithms are used can be speeded up by exploiting some advantages that qubits have over classical bits. In many cases, exponential speedup can be accomplished. Exponential speedup is a term from computer architecture, which describes the replacement of an exponential time algorithm (e.g. $\mathcal{O}(2^n)$, in which n is a measure for the 'size' of the input) by a polynomial time algorithm (e.g. $\mathcal{O}(n^2)$). This is an important development, because exponential time algorithms are virtually useless for large inputs.

The main concern on the other hand, is also caused by the nature of quantum mechanics. Measurements performed on quantum registers are non-deterministic and can yield unwanted outputs. Good quantum algorithms have a way of dealing with this ambiguity, diminishing the impact of this possibility.

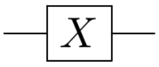
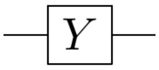
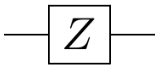
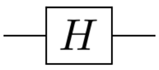
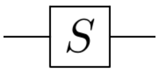
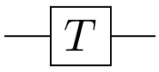
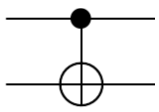
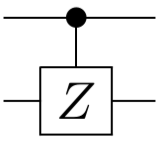
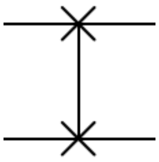
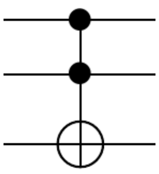
There are two algorithms that are key in this thesis. The first one is an algorithm devised by Lov Grover[18] in 1996. This is a very illustrative quantum algorithm. The algorithm starts with an equal superposition of all quantum states and attempts to find a certain qubit state. This state is found by some *oracle* that picks out the state that is searched for. Then, some operation is performed to increase the amplitude of this state. Therefore, the probability of finding this state in a measurement is increased. Grover's algorithm tends to optimize this procedure to maximize the probability of finding the state by bringing its amplitude as close as possible to 1.

This procedure is described in great detail in section 2.1. Grover's algorithm is so illustrative because the amplitude of the different qubit states can be kept track of throughtout the algorithm. This is done in figure 2.2 through figure 2.6.

The second quantum algorithm described in this thesis is a more abstract algorithm proposed by Peter Shor[31] in 1994. Prime factorization is very tedious if done by classical algorithms. The RSA cryptosystem[24] is based on the fact that it takes thousands of years to factor the multiple of two large (in general 1024 bits or larger) primes. However, Shor's algorithm can be used on a quantum computer to factorize numbers in polynomial time (see section 5). As a result, RSA encryption will no longer be safe. However, all is not lost because quantum mechanics also brings new, much better cryptography methods. For more on quantum cryptography, the reader is referred to Richard Hughes[21] and similar articles.

Shor's algorithm does not make use of a linear superposition of qubit states, but instead uses the quantum analagon of the Fourier transform (see section 3). The quantum Fourier transform is, together with the oracle function, widely used in quantum computation.

Table 1.1. An overview of the single- and multiple qubit gates described in this section. Along with the matrix representation, also the circuit representation of each gate is given.

Name	Circuit Representation	Matrix Representation
Pauli-X		$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Pauli-Y		$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
Pauli-Z		$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Hadamard		$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
S		$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$
T		$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$
CNOT		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$
C(Z)		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$
SWAP		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Toffoli		$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$

2 Grover's Algorithm

Searching through an unordered list with N entries in a classical way can take up to N evaluations, because in the worst case all elements of the list must be checked. The algorithm thus uses $\mathcal{O}(N)$ operations. On a quantum computer, this problem can be solved significantly more efficient. Grover's algorithm[18] requires only $O\left(\sqrt{N}\right)$ operations to search through an unordered list with high probability of success. Although the speedup is not exponential, it can reduce computation time significantly if N is very large. Moreover, it was proven by Bennett, Berstein, Brassard and Vazirani that Grover's algorithm is the optimal (quantum) algorithm for searching through unordered lists[3].

Grover's algorithm is not only an efficient algorithm for searching through unordered databases, it could also used to break certain cryptography methods. *Advanced Encryption Standard* (AES) is a cipher that is used by WinRaR and several other companies. This encryption method uses keys with a length of 128, 192 and 256 bits. With Grover's algorithm, a 256 bit key can be found in 'only' 2^{128} iterations. That is still an enormous number, however it is almost 10^{40} times faster compared to brute forcing! If large quantum computers become available, AES users would most likely be forced to use keys with significantly more than 256 bits[5].

In section 2.1, the steps of Grover's algorithm are described. Then, the circuit implementation for two qubits and for an arbitrary number of qubits is outlined in section 2.2. In section 2.3, the optimal number of iterations for Grover's algorithm and its consequences for the time complexity are analysed. Thereafter, in section 2.4, Grover's algorithm is extended to a multi-search algorithm and analysis on the optimal number of iterations is performed. Finally, in section 2.5 an application of this multi-search algorithm to the SAT-problem is described.

2.1 Steps in Grover's Algorithm

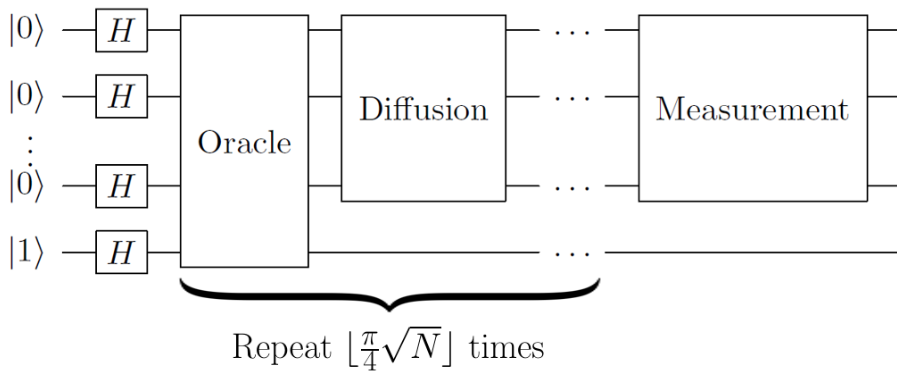


Figure 2.1. A schematic overview of the circuit for Grover's Algorithm.

In figure 2.1, a schematic overview of the circuit for Grover's algorithm is given. Grover's algorithm relies on an oracle. An oracle can be viewed as a black box that performs an operation on a quantum state that is not readily specified by universal quantum gates. In Grover's algorithm, an oracle is implemented such that it flips the sign of $|x\rangle$ iff x is a state we are looking for (the 'correct' quantum state). This can be expressed as

$$|x\rangle \overset{\mathcal{O}}{\longmapsto} (-1)^{f(x)}|x\rangle \tag{2.1}$$

with $f(x) = 1$ if x is the correct state and $f(x) = 0$ otherwise. For now, this oracle is treated as a black box. The implementation of the oracle is described in section 2.2.

Grover's algorithm for searching through n-qubit states starts with a register of $n+1$ qubits initialized to $|\psi\rangle = |0\rangle^{\otimes n} \otimes |1\rangle$. The $n+1$th qubit a so-called ancillary qubit, that is used in performing the oracle. The ancillary qubit is not of interested for now and will be ignored for now. The number of possible quantum states is thus $N = 2^n$. Then, the Hadamard gate is applied to all qubits, resulting in an equal superposition of all possible states,

$$|\psi\rangle = H^{\otimes n}|0\rangle^{\otimes n} = \frac{1}{\sqrt{N}} \sum_{x\in\{0,1\}^n} |x\rangle \tag{2.2}$$

Since the coefficients of the states are real throughout the whole algorithm, the state can also be visualized in a diagram representing the amplitudes of the states. Recall that the modulus squared of the amplitude is the measurement probability of that quantum state. After applying the Hadamard gate, the quantum state is visually given in figure 2.2.

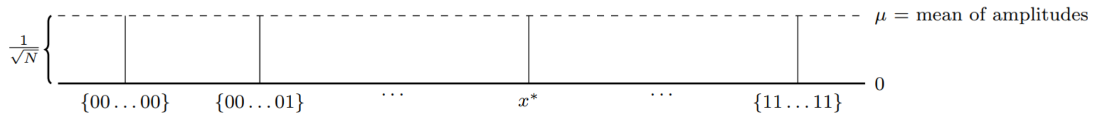


Figure 2.2. The amplitudes of the quantum states after applying the set of Hadamards. Reprinted from *Grover's Algorithm* (Wright & Tseng, 2015)

Now, the so-called Grover iteration is applied to the quantum state. Grover iteration starts with applying the oracle described in equation (2.1). This results in the correct state having its sign flipped while all other states are untouched. Let x^* denote the correct state. The resulting state is

$$|\psi\rangle = -\frac{1}{\sqrt{N}}|x^*\rangle + \frac{1}{\sqrt{N}} \sum_{\substack{x\in\{0,1\}^n \\ x\neq x^*}} |x\rangle \tag{2.3}$$

This is represented graphically in figure 2.3.

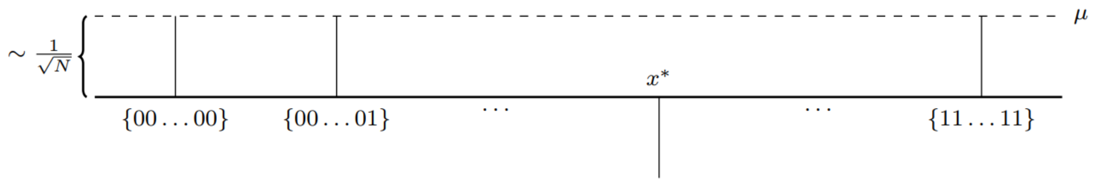


Figure 2.3. The amplitudes of the quantum states after applying the oracle. Reprinted from *Grover's Algorithm* (Wright & Tseng, 2015)

In order to find the correct state in a measurement with some probability of success, the probability amplitude of the correct state must be increased. This is done by applying the Grover diffusion operator. For now, this is defined as a black box. The implementation in a quantum circuit is given in section 2.2. Define μ as the average of the coefficients α_x in front of $|x\rangle$ (this is a real number, since all coefficients are real),

$$\mu = \frac{1}{N} \sum_{x \in \{0,1\}^n} \alpha_x = \frac{(N-1)\alpha_x - \alpha_{x^*}}{N} \tag{2.4}$$

The Grover diffusion operator then flips the coefficients around the average of the coefficients by the mapping

$$\alpha_x |x\rangle \mapsto (2\mu - \alpha_x) |x\rangle \tag{2.5}$$

If N is large (this is not a requirement, but merely an assumption for the sake of illustration), μ is very close to α_x. Therefore, using $\alpha_{x^*} = -\alpha_x$ we find that the mapping takes the correct state $\alpha_{x^*} |x^*\rangle$ to $(2\alpha_x - \alpha_{x^*}) |x^*\rangle = 3\alpha_x |x^*\rangle$ and takes the other states to $(2\alpha_x - \alpha_x) |x\rangle = \alpha_x |x\rangle$. The amplitude of the correct state is thus amplified to $\frac{3}{\sqrt{N}}$. Graphically, the quantum state is given in figure 2.4.

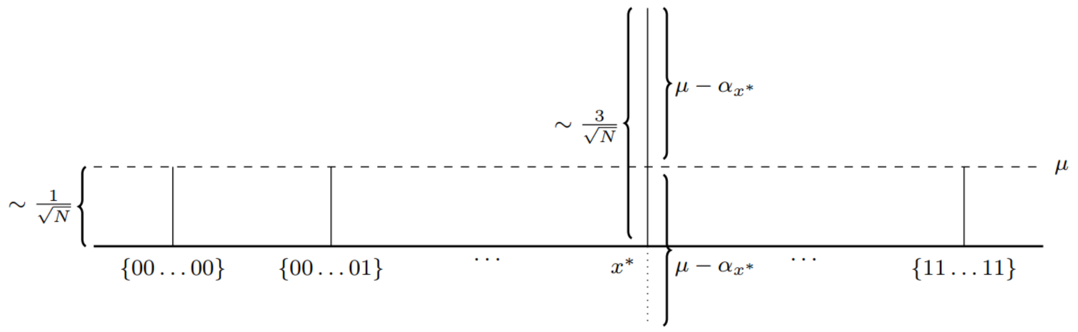


Figure 2.4. The amplitudes of the quantum states after applying the Grover diffusion gate. Reprinted from *Grover's Algorithm* (Wright & Tseng, 2015)

Since the mean amplitude was approximately equal to $\frac{1}{\sqrt{N}}$, the amplitude of all states but $x*$ stays roughly the same. The amplitude of $x*$ however is amplified to about $\frac{3}{\sqrt{N}}$[34].

This iteration can be applied multiple times to further increase the amplitude of $x*$. After applying the oracle once more, we have the quantum state in figure 2.5.

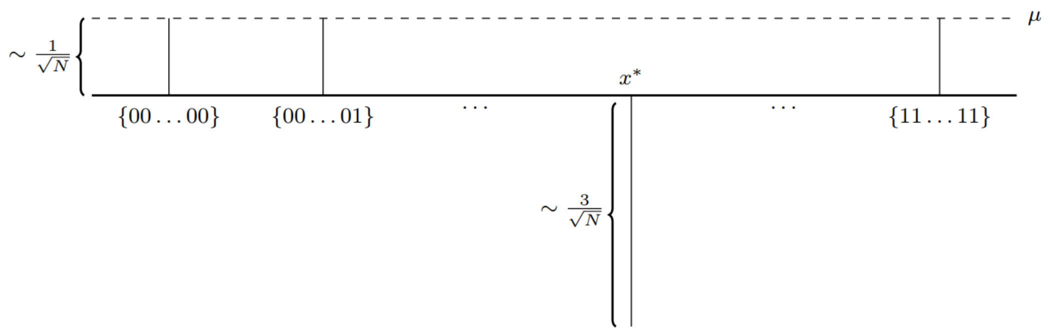


Figure 2.5. The amplitudes of the quantum states after applying the oracle once more. Reprinted from *Grover's Algorithm* (Wright & Tseng, 2015)

And after applying Grover's diffusion gate one more time the amplitude of the correct state increases to approximately $\frac{5}{\sqrt{N}}$, see figure 2.6.

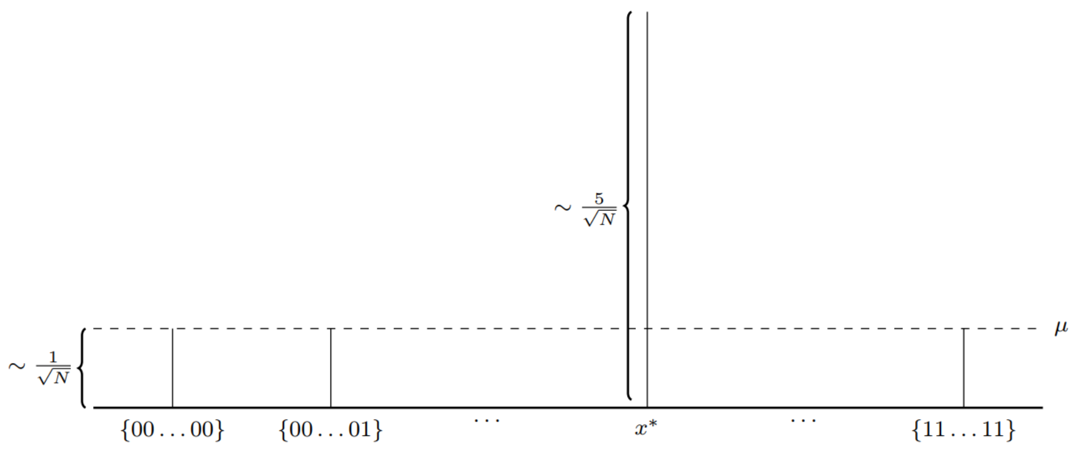


Figure 2.6. The amplitudes of the quantum states after applying the Grover diffusion gate once more. Reprinted from *Grover's Algorithm* (Wright & Tseng, 2015)

The amount by which the amplitude increases in each iteration is determined by the difference between the (negative) amplitude of $x*$ and the mean of all amplitudes. As the amplitude of $x*$ grows, the amplitudes of the other states decrease. The mean therefore also decreases resulting in diminishing growth of the amplitude of $x*$. The amplitude can at some point even decrease because the mean of the amplitudes after the sign flip becomes negative. This occurs when the (negative) amplitude of x^* is much larger than the amplitudes of the other states. In the illustration given above, however, the amplitude will only keep increasing if we assume $N \to \infty$[32]. This is in line with the general formula for the optimal number of iterations. In section2.3, it will be shown that the optimal number of iterations is given by $R^* = \left\lfloor \frac{\pi}{4}\sqrt{N} \right\rfloor$.

2.1.1 The Oracle Function

At first sight, you may think that Grover's algorithm does not seem too useful. You can find a certain quantum state, but only after you know some oracle function. But if you have the oracle function, is it not already clear what the outcome of the algorithm will be? In the case of searching through a database, you are absolutely right. However, there are some cases in which it is not readily clear from the oracle function what the outcome will be.

Consider for example the SHA256 algorithm[28], which is used for example as the proof of work algorithm in several cryptocurrencies. SHA256 is a one-way function that hashes the input that is given. As a result, the output looks completely random. The objective in proof of work is to find some input that results in a given output. Since it is a one-way function, the desired input cannot directly be deduced from the output. Using Grover's algorithm, the outcome of all possible input states can be checked simultaneously. The algorithm then increases the amplitude of the input states that give the desired output. That way, an appropriate input state can be found way quicker than by classical brute forcing.

Another application of Grover's algorithm to a mathematical problem is given in section 2.5

2.2 Implementation of Grover's Algorithm

In order to implement Grover's algorithm in the QX Quantum Computer Simulator[29], the algorithm (including black boxes) must be decomposed to quantum gates, as introduced in section 1.2.3. Also, one or more ancillary qubits are needed in the implementation of the oracle as stated before.

2.2.1 Implementation for 2-Qubit States

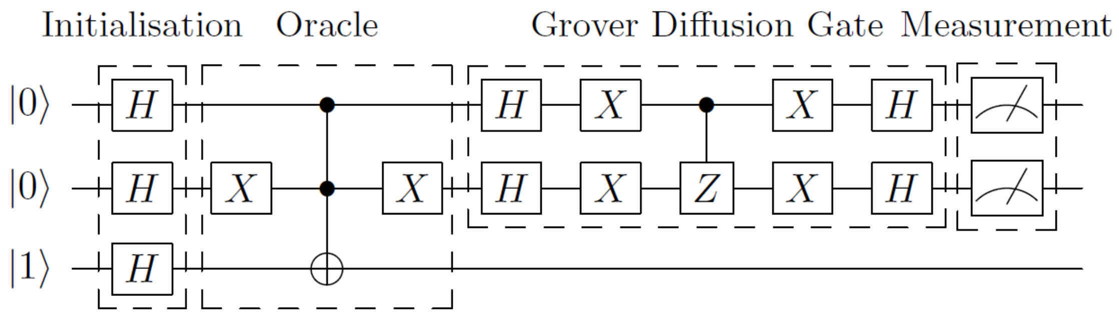


Figure 2.7. The full quantum circuit that is used in Grover's algorithm for searching through two-qubit states. In particular, this circuit searches for $|10\rangle$.

In figure 2.7, the full quantum circuit for searching through two qubit states is given. This circuit will be broken down into the parts outlined in the schematic overview from section 2.1. The implementation of every part is discussed in this section.

In order to search through two qubit states, one ancillary qubit is needed, initialized to $|1\rangle$. The two qubits that store the two-qubit quantum states are both initialized to $|0\rangle$. The initial quantum state is therefore given by $|\psi\rangle = |001\rangle$. Then, the Hadamard gate is applied to all three qubits (figure 2.8).

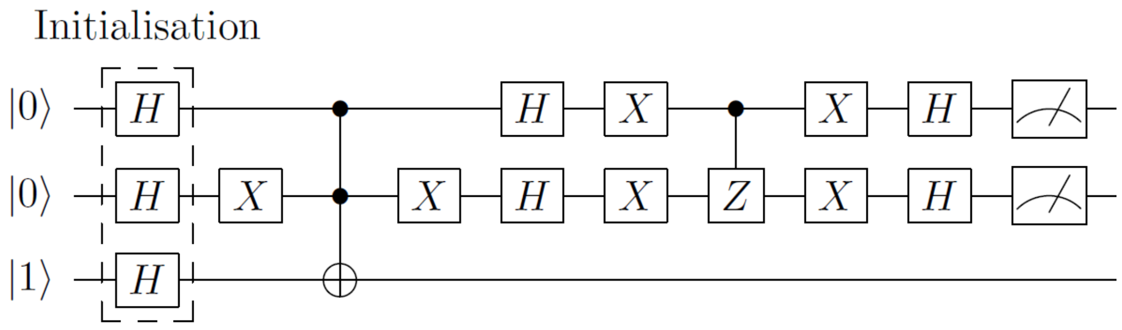


Figure 2.8. A series of Hadamard gates is applied to the initial quantum state.

We then obtain

$$|\psi\rangle = H^{\otimes 3}|001\rangle = \frac{1}{\sqrt{8}}(|00\rangle + |01\rangle + |10\rangle + |11\rangle) \otimes (|0\rangle - |1\rangle) \tag{2.6}$$

in which the ancillary qubit is separated from the other two qubits. This will make it easier to track the following calculation. $H^{\otimes 3}$ is common notation for applying the Hadamard gate to three qubits.

Afterwards, Grover iteration is applied. In the two qubit case $N = 2^n = 4$. Using the analysis that will be done in section 2.3, it is determined that the optimal number of iterations is $R^* = \left\lfloor \frac{\pi}{4}\sqrt{N} \right\rfloor = \left\lfloor \frac{\pi}{2} \right\rfloor = 1$.

In order to achieve the sign flip of the correct state, the oracle black box is used (see figure 2.9.

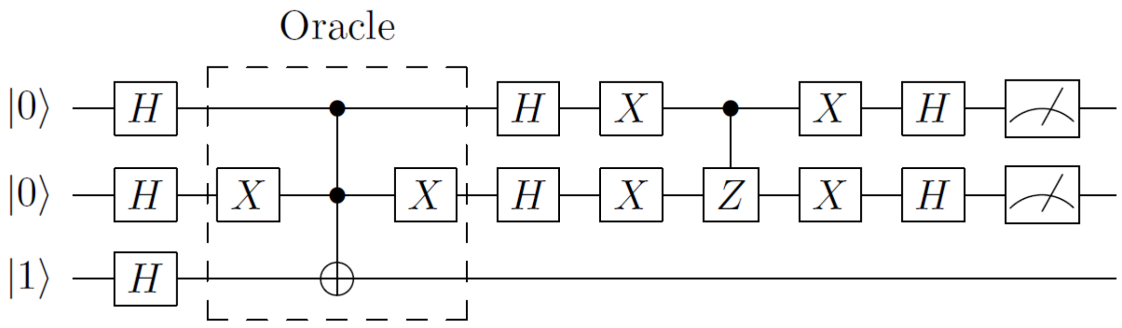


Figure 2.9. The gates inside the dashed rectangle constitute the oracle.

First, the X gate is applied to the qubits that are desired to be $|0\rangle$. We choose to search for $|10\rangle$. Thus, the X gate is applied to the second qubit only. Then, the Toffoli gate is applied to the ancillary qubit, controlled by the other two qubits. This results in the sign change of the states $|110\rangle$ and $|111\rangle$,

$$|\psi\rangle = \frac{1}{\sqrt{8}}\left(|11\rangle \otimes (|1\rangle - |0\rangle) + (|00\rangle + |01\rangle + |10\rangle) \otimes (|0\rangle - |1\rangle)\right) \tag{2.7}$$
$$= \frac{1}{\sqrt{8}}(|00\rangle + |01\rangle + |10\rangle - |11\rangle) \otimes (|0\rangle - |1\rangle) \tag{2.8}$$

Finally, the X gate is applied to the same qubits as before to rotate the qubits desired to be in $|0\rangle$ to that from $|1\rangle$. So, since we are looking for $|10\rangle$, the X gate is applied only to q_1, which results in

$$|\psi\rangle = \frac{1}{\sqrt{8}}(|00\rangle + |01\rangle - |10\rangle + |11\rangle) \otimes (|0\rangle - |1\rangle) \tag{2.9}$$

Now that the sign of $|01\rangle$ has been flipped, the ancillary qubit has done its job.

The next part of the circuit magnifies the amplitude of the correct state. This is conducted by the Grover diffusion gate (see figure 2.10.

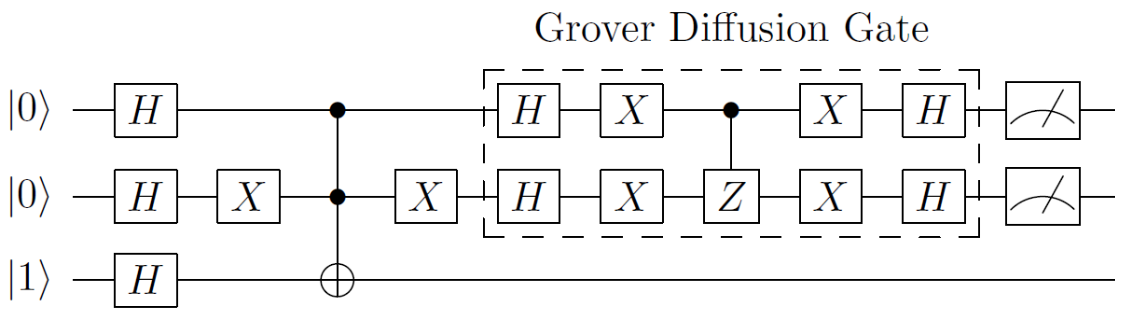


Figure 2.10. The gates inside the dashed rectangle constitute the Grover diffusion gate.

First, the Hadamard gate is applied to the first and second qubit, followed by the X gate on both qubits. Then, the Z gate, controlled by the first qubit, is applied to the second qubit. Afterwards, the X gate is applied to both qubits again, followed by the Hadamard gate.

This results in the mapping

$$|q_1 q_2\rangle \mapsto \frac{1}{2}(|q_1 q_2\rangle - |q_1 \oplus 1\ q_2\rangle - |q_1\ q_2 \oplus 1\rangle - |q_1 \oplus 1\ q_2 \oplus 1\rangle) \tag{2.10}$$

in which $\oplus$ denotes binary addition: $b_1 \oplus b_2 \equiv b_1 + b_2 \mod 2$. In this particular case, the mapping of the four different two qubit states is given by

$$\begin{array}{rll} |00\rangle & \mapsto \tfrac{1}{2}(& |00\rangle - |10\rangle - |01\rangle - |11\rangle) \\ |01\rangle & \mapsto \tfrac{1}{2}(& |01\rangle - |11\rangle - |00\rangle - |10\rangle) \\ \text{-}\,|10\rangle & \mapsto \tfrac{1}{2}(& \text{-}\,|10\rangle + |00\rangle + |11\rangle + |01\rangle) \\ |11\rangle & \mapsto \tfrac{1}{2}(& |11\rangle - |01\rangle - |10\rangle - |00\rangle) \quad + \\ \hline |00\rangle + |01\rangle - |10\rangle + |11\rangle & \mapsto \text{-}\,|10\rangle & \end{array}$$

Since the correct state has a minus sign, whereas the other states have a plus sign, this operation results in each non-correct state being present two times with a plus sign and two times with a minus sign and the correct state being present four times with a minus sign.

Including the ancillary qubit, the final quantum state is

$$|\psi\rangle = \frac{1}{\sqrt{2}}(\text{-}\,|10\rangle) \otimes (|0\rangle - |1\rangle) \tag{2.11}$$

Recall that $|10\cdot\rangle$ is the correct state. Since the state of the ancillary qubit is irrelevant, it is denoted with a dot. If we now measure the first and second qubit (figure 2.11), we obtain $|10\rangle$ with 100% certainty[17].

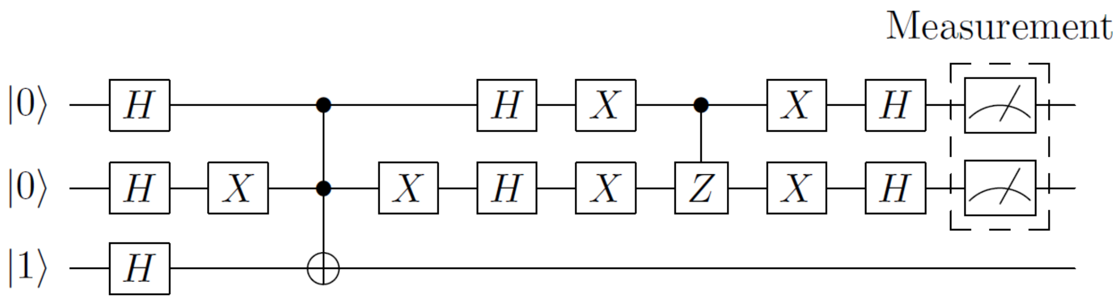


Figure 2.11. A measurement is performed on the first and second qubit.

To understand why this happens, we must go back to the 'flip around the mean'. In the two qubit case, the mean of the amplitudes is $\frac{1/2+1/2+1/2-1/2}{4} = \frac{1}{4}$, which is exactly half of the amplitude of the non-correct states. Therefore the Grover diffusion gate, which flips the amplitudes around the mean, maps the amplitude of the non-correct states to $2\mu - \alpha_x = 2 \cdot \frac{1}{4} - \frac{1}{2} = 0$ and the amplitude of the correct state to $2\mu - \alpha_{x^*} = 2 \cdot \frac{1}{4} - \text{-}\frac{1}{2} = 1$. It may seem like this is an exception, and it is indeed. However, it will be shown in section 2.4.2 that for each number of qubits, one such case exists if multiple correct states are considered.

2.2.2 Implementation for n-Qubit States

The two qubits Grover's algorithm can be generalized to a search through a list with entries of length n. In addition, as will follow from the upcoming notes on the implementation, two ancillary qubits will be needed. In this section, a general single qubit operator A, controlled by n qubits, is denoted as $C^n(A)$.

The first series of Hadamard gates will be extended to all qubits. Also, the X gates enclosing the Toffoli gate are still applied to each qubit that is $|0\rangle$ in the correct state, just like in the two qubit case. The Toffoli gate is replaced by the $C^n(X)$ gate which is controlled by all other qubits and is applied to the ancillary qubit. The same goes for the controlled Z gate, which is changed into the $C^{n-1}(Z)$ gate that is controlled by the first through the $n-1$th qubit and is applied to the nth qubit. These two gates, however, cannot be implemented in QX Simulator. Luckily, it can be shown that both the multi-control Toffoli gate as well as the multi-control Z gate can be decomposed into a series of Toffoli- and Hadamard gates. For this, the other ancillary qubit that is unused so far is needed.

For the $C^3(X)$ gate and the $C^2(Z)$ gate, this decomposition is quite straightforward. The result is shown in figure 2.12, which shows Grover's algorithm that searches for $|010\rangle$. The optimal number of iterations is (see section 2.3) $R^* = \left\lfloor \frac{\pi}{4}\sqrt{2^3} \right\rfloor = 2$, so two Grover iterations are performed.

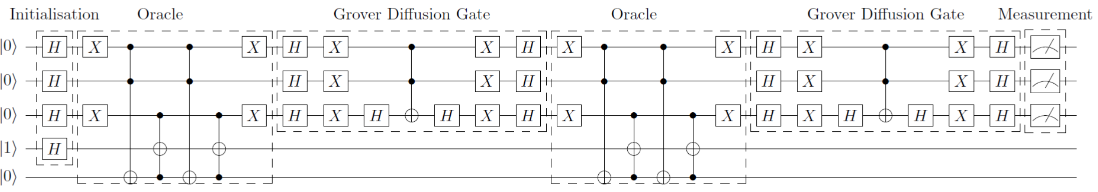


Figure 2.12. The quantum circuit used in Grover's Algorithm for searching through three-qubit states. Note that both the oracle black box and Grover diffusion gate are repeated twice. In particular, this circuit searches for $|010\rangle$.

As the number of qubits increases, however, the decomposition becomes more complicated. Nevertheless, it can be done for any multi-controlled single qubit gate.

2.2.3 Implementation of the $C^n(X)$ gate and the $C^n(Z)$ gate

A multi-control X gate can be decomposed into Toffoli gates using three equivalent circuit parts[9]. The proofs follow from matrix multiplication but will be omitted, since it is a tedious calculation.

Lemma 2.1. *There holds $X = HZH$. Therefore, the $C^n(X)$ gate is equivalent to a $C^n(Z)$ gate squeezed by two Hadamard gates.*

For example, the $C^3(X)$ gate can be decomposed into the $C^3(Z)$ gate and two Hadamard gates. This is shown in figure 2.13.

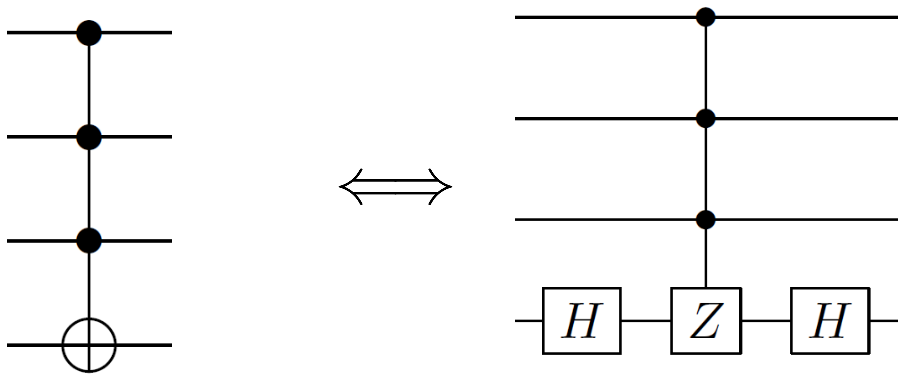


Figure 2.13. The $C^3(X)$ gate is equivalent to the $C^3(Z)$ gate enclosed by two Hadamard gates.

The Hadamard gate is, like all quantum gates, unitary. Consequently, there holds that $H^{-1} = H^{\dagger}$. Moreover, for the Hadamard gate, there even holds that $H^{\dagger} = H$. Therefore another conclusion can be drawn from lemma 2.1.

Corollary 2.1.1. *There holds $X = HZH$. Since H is unitary and $H^{\dagger} = H$, it follows that $Z = H^{-1}XH^{-1} = H^{\dagger}XH^{\dagger} = HXH$. Therefore, lemma 2.1 can also be applied in reverse to obtain that the $C^n(Z)$ gate is equivalent to a $C^n(X)$ gate squeezed by two Hadamard gates.*

The multi-controlled Z gate that is then obtained can be broken up into multi-controlled X gates and (multi)-controlled Z gates with fewer control qubits.

Lemma 2.2. *Let the total number of qubits $n^* \geq n+2$ and $0 \leq w < n$, $w \in \mathbb{N}$. Then, the $C^n(Z)$ gate can be decomposed into two $C^{n-w}(X)$ gates and two $C^{w+1}(Z)$ gates. The order of these gates is key to get the right decomposition. They have to be alternated (i.e. first the $C^{n-w}(X)$ gate, then the $C^{w+1}(Z)$ gate, then the second $C^{n-w}(X)$ gate and finally the second $C^{w+1}(Z)$ gate or vice versa). $n-w$ of the initial control qubits can be chosen randomly as control qubits for the $C^{n-w}(X)$ gate. The target qubit has to be an ancillary qubit in whichever qubit state. The w remaining qubits and the target of the $C^{n-w}(X)$ are the controls for the $C^{w+1}(Z)$ gate and its target qubit is the same as the original target qubit.*

Since two $C^k(X)$ gates are applied with the same control qubits and the same target qubit, it will always be reset to its original value. By also applying $C^m(Z)$ gate twice, the result is the same for whatever qubit state the ancillary qubit is in. Thus the ancillary qubit in this decomposition does not necessarily have to be a qubit that is not used at all throughout the circuit. In order to save qubits, a qubit that is not used in the $C^n(Z)$ gate can be chosen to serve as ancillary qubit.

An example to clarify the previous lemma may be in place. The $C^3(Z)$ gate can be decomposed into two $C^2(X)$ (Toffoli) gates and two $C^2(Z)$ gates. This corresponds to $n* = 5$, $n = 3$ and $w = 1$. The second-last qubit is not used in the $C^3(Z)$ gate, so it can serve as ancillary qubit. The result is given in figure 2.14.

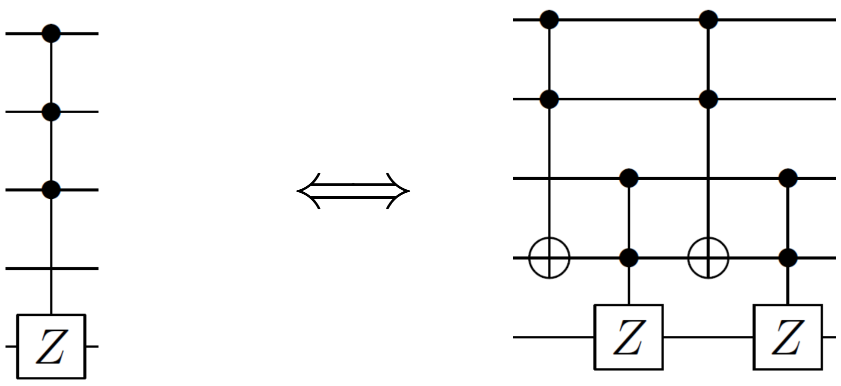


Figure 2.14. The $C^3(Z)$ can be rewritten as two Toffoli gates and two $C^2(Z)$ gates.

Using these two lemmas, each $C^n(X)$ and $C^n(Z)$ gate can be decomposed into Toffoli gates in two ways. The first way is straightforward and optimizes the number of gates, but the number of ancillary qubits required scales with n. To be exact, $n - 2$ ancillary qubits are required. The decomposition starts by applying the Toffoli gate controlled by the first and second qubit to the first ancillary qubit. Then, the Toffoli gate controlled by the third qubit and first ancillary qubit is applied to the second ancillary qubit, after which the Toffoli gate controlled by the fourth qubit and second ancillary qubit is applied to the third ancillary qubit. This is repeated up to the application of the Toffoli gate controlled by the $n - 1$th qubit and the $n - 3$th ancillary qubit to the $n - 2$nd qubit. Then, the $n - 2$nd qubit is $|1\rangle$ if and only if the first through $n - 1$th qubit are $|1\rangle$. Finally, by applying the Toffoli gate controlled by the nth qubit and the $n - 2$nd qubit to the target qubit the $C^n(X)$ gate is completed. In order to reset the ancillary qubits to $|0\rangle$, the Toffoli gates on the ancillary qubits are repeated in reverse order.

As an example, consider the $C^5(X)$ gate. The result of the decomposition is shown in figure 2.15. Here, the first through fifth qubit are the controls. The sixth qubit is the target qubit and the seventh through ninth qubit are the ancillary qubits for the decomposition.

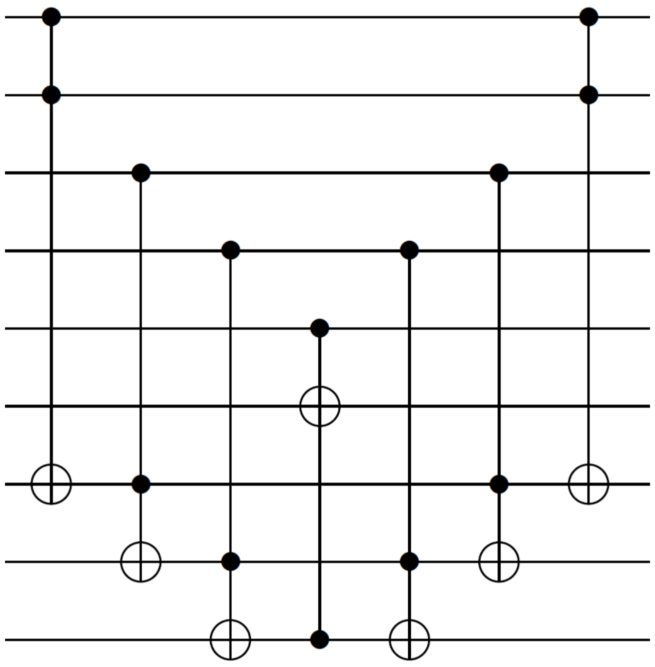


Figure 2.15. The decomposition of the $C^5(X)$ gate.

In order to decompose a $C^n(Z)$ gate, the same procedure can be applied. The one difference is that a Hadamard gate is added to the target qubit before and after the Toffoli gates, in accordance with corollary 2.1.1.

The second way of decomposing the $C^n(X)$ and $C^n(Z)$ gate into Toffoli gates uses both lemmas. It requires only one ancillary qubit, but the number of gates scales exponentially with n. Each $C^n(X)$ gate can be substituted by a $C^n(Z)$ gate enclosed by two Hadamard gates using lemma 2.1. Then, by choosing $w = n - 1$ in lemma 2.2 the $C^n(Z)$ gate can be replaced by two $C^2(X)$ gates and two $C^{n-1}(Z)$ gates. The former is the Toffoli gate. To the latter, lemma 2.2 can be applied over and over, until only $C^2(X)$ (Toffoli) gates and $C^2(Z)$ gates are left. These $C^2(Z)$ gates can be substituted by a Toffoli gate squeezed between two Hadamard gates according to corollary 2.1.1[9].

Furthermore, if we are dealing with a $C^n(Z)$ gate, the first application of lemma 2.1 can be dropped. The result will be the same, except for one Hadamard gate on the target qubit right at the beginning and one right at the end.

From this procedure, it can be deduced that both the decomposition of the $C^n(X)$ gate as well as the $C^n(Z)$ gate require $n - 2$ rounds of applying lemma 2.2 along with one application corollary 2.1.1. The decomposition of $C^n(X)$ requires an additional application of lemma 2.1.

The number of Toffoli gates resulting from the decomposition is, for $n \geq 2$ (for $n = 1$, obviously it is 1),

$$2^{n-2} + \sum_{j=1}^{n-2} 2^j = 2^{n-2} - 1 + \sum_{j=0}^{n-2} 2^j \tag{2.12}$$

$$= 2^{n-2} - 1 + \frac{1 - 2^{n-1}}{1 - 2} \tag{2.13}$$

$$= 2^{n-2} + 2^{n-1} - 2 \tag{2.14}$$

$$= 3 \cdot 2^{n-2} - 2 \tag{2.15}$$

By cancelling as much of the resulting Hadamard gates as possible using $HH = I$, the $C^n(X)$ gate only decomposes into Toffoli gates. In the decomposition of the $C^n(Z)$ gate, additionally two Hadamard gates are left resulting in a total of $3 \cdot 2^{n-2}$ gates.

2.2.4 Example: The Decomposition of the $C^4(X)$ Gate

In order to illustrate the iteration to decompose the $C^n(X)$ gate, this section features the decomposition of the $C^4(X)$ gate, see figure 2.16.

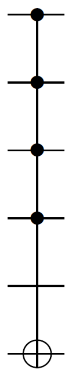


Figure 2.16. The $C^4(X)$ gate.

Lemma 2.1 is used to substitute the $C^4(X)$ gate by the $C^4(Z)$ gate enclosed by two Hadamard gates. The result is given in figure 2.17. If one is interested in the decomposition of the $C^4(Z)$ gate, the two outermost Hadamard gates (one at the left and one at the right) can be omitted from this point onwards (see figure 2.23 for the result of the decomposition of the $C^4(Z)$ gate).

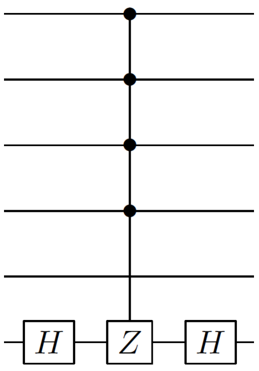


Figure 2.17. The $C^4(Z)$ gate, enclosed by two Hadamard gates.

The total number of available qubits is $n^* = 6 = 4 + 2$, so lemma 2.2 can be applied to reduce the number of control qubits. Since $n = 4$, we take $w = 4 - 2 = 2$. The $C^4(Z)$ gate is then decomposed into two $C^2(X)$ gates and two $C^3(Z)$ gates, see figure 2.18.

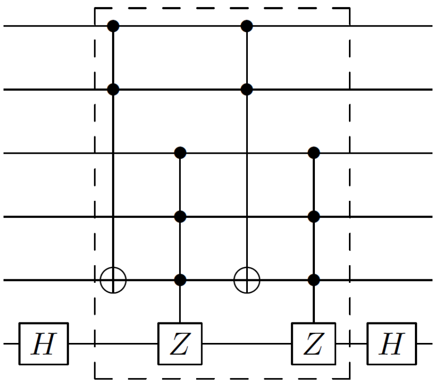


Figure 2.18. The part within the dashed rectangle is equivalent to the $C^4(Z)$ gate.

Once again, lemma 2.2 is applied. Because the ancillary qubit for the previous step is now a control qubit for the $C^3(Z)$ gate, it cannot be used as ancillary qubit in this step. However, the first and second qubit could be chosen as 'ancillary' qubit for this step, since these are not involved in the gate that still needs to be decomposed. Since $n = 3$, now we take $w = 3 - 2 = 1$. Then, the $C^3(Z)$ gates are decomposed into two $C^2(X)$ gates and two $C^2(Z)$ gates, see figure 2.19.

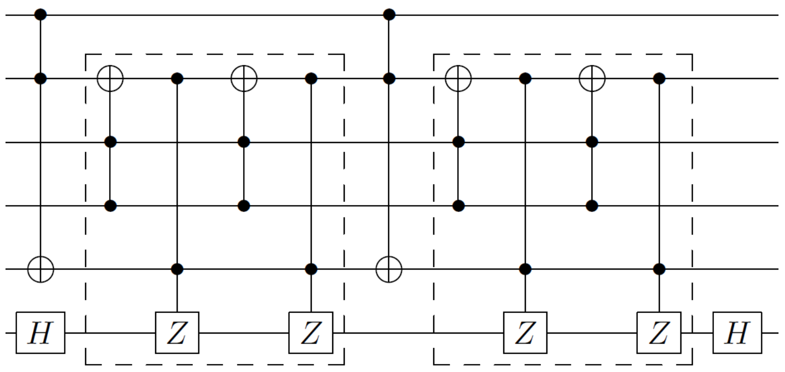


Figure 2.19. The parts within the dashed rectangles are equivalent to the $C^3(Z)$ gates from figure 2.18.

Now, the controlled X gates are all Toffoli gates. However, there are still some $C^2(Z)$ gates that have to be decomposed into Toffoli gates. This is where corollary 2.1.1 comes into play. According to this corollary, each of the $C^2(Z)$ gates can be replaced by a Toffoli gate enclosed by two Hadamard gates. The result is given in figure 2.20.

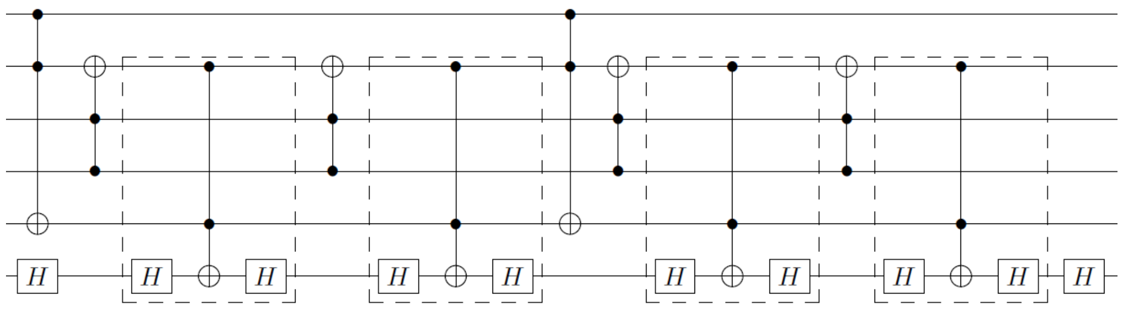


Figure 2.20. The parts within the dashed rectangles is equivalent to the $C^2(Z)$ gates from figure 2.19.

Finally, we can use the fact that $HH = I$ to cancel all Hadamard gates. In figure 2.21, it is shown which Hadamard gates cancel each other.

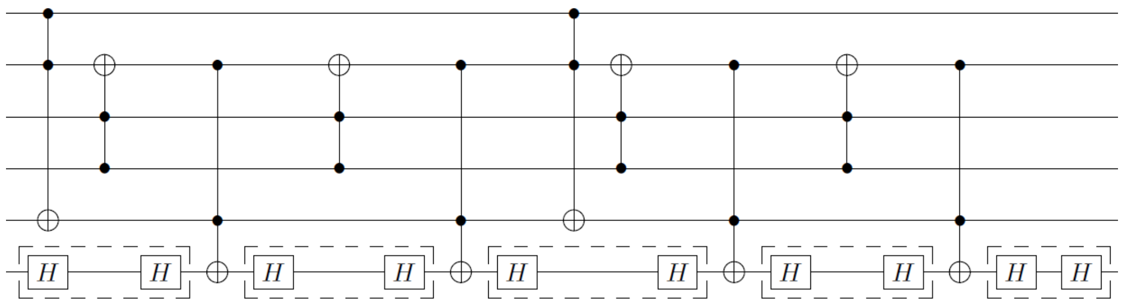


Figure 2.21. The Hadamard gates within the dashed rectangles cancel each other.

The final decomposition of the $C^4(X)$ gate into Toffoli gates is thus given in figure 2.22.

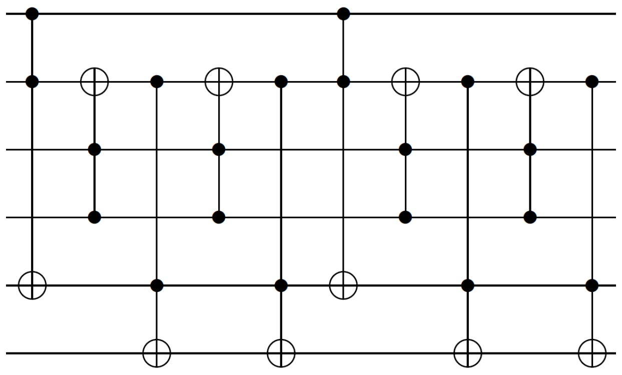


Figure 2.22. The decomposition of the $C^4(X)$ gate into Toffoli gates.

If one were to decompose the $C^4(Z)$ gate, two additional Hadamard gates have to be added to the result as stated before. These Hadamard gates have to be added to the original target qubit, which is the lowermost qubit in this case. One Hadamard gate has to be added to this qubit in front of the first Toffoli gate that targets the lowermost qubit and one Hadamard gate has to be added after the final Toffoli gate that targets the lowermost qubit. These two Hadamards do not cancel and are therefore still present in the decomposition. The result of the decomposition of the $C^4(Z)$ gate into Toffoli gates and Hadamard gates is given in figure 2.23.

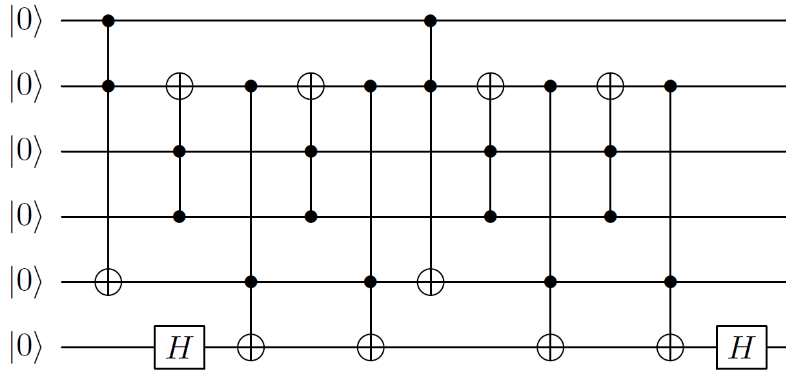


Figure 2.23. The decomposition of the $C^4(Z)$ gate into Toffoli gates and Hadamard gates.

2.3 Analysis of the Optimal Number of Iterations

It was stated earlier without proof that the optimal number of Grover iterations is given by

$$R = \frac{\pi}{4}\sqrt{N} \tag{2.16}$$

An elegant way to derive this formula is by considering the result of one Grover iteration as a rotation along the unit circle. We first write the initial superposition of all states in a slightly different manner, namely

$$|\psi\rangle = \alpha_n \left(\frac{1}{\sqrt{N-1}} \sum_{\substack{x\in\{0,1\}^n \\ x\neq x^*}} |x\rangle \right) + \beta_n |x^*\rangle \tag{2.17}$$

Where α_n and β_n are arbitrary 'normalization' constants. This is possible since the states $x \neq x^*$ all have the same amplitude. Initially, equation (2.2) must be satisfied. Therefore, $\alpha_0 = \frac{\sqrt{N-1}}{\sqrt{N}}$ and $\beta_0 = \frac{1}{\sqrt{N}}$ so that the amplitudes of all $|x\rangle$ and $|x^*\rangle$ are equal to $\frac{1}{\sqrt{N}}$.
Flipping the amplitude of the correct states results in

$$|\psi\rangle = \alpha_n \left(\frac{1}{\sqrt{N-1}} \sum_{\substack{x\in\{0,1\}^n \\ x\neq x^*}} |x\rangle \right) - \beta_n |x^*\rangle \tag{2.18}$$

Now we want to find out what happens if the amplitudes are flipped around the mean by the Grover diffusion gate. Since the sum over the non-correct states runs over $N-1$

elements, the average amplitude is given by

$$\mu = \frac{1}{N}\left((N-1)\left(\alpha_n \frac{1}{\sqrt{N-1}}\right) - \beta_n\right) = \frac{\alpha_n\sqrt{N-1}-\beta_n}{N} \tag{2.19}$$

Then, we apply the grover diffusion gate which maps $\alpha_x \mapsto 2\mu - \alpha_x$. We obtain for the amplitude of the states $x \neq x^*$

$$\frac{\alpha_n}{\sqrt{N-1}} \mapsto \frac{2\alpha_n\sqrt{N-1}-2\beta_n}{N} - \frac{\alpha_n}{\sqrt{N-1}} \tag{2.20}$$

$$= \frac{2\alpha_n(N-1)-2\beta_n\sqrt{N-1}-\alpha_n N}{N\sqrt{N-1}} \tag{2.21}$$

$$= \frac{(N-2)\alpha_n - 2\beta_n\sqrt{N-1}}{N\sqrt{N-1}} \tag{2.22}$$

Or, equivalently after multiplying by $\sqrt{N-1}$,

$$\alpha_n \mapsto \left(1-\frac{2}{N}\right)\alpha_n - \frac{2\sqrt{N-1}}{N}\beta_n \tag{2.23}$$

And for β_n we find

$$\beta_n \mapsto \frac{2\alpha_n\sqrt{N-1}-2\beta_n}{N} + \beta_n \tag{2.24}$$

$$= \frac{2\sqrt{N-1}}{N}\alpha_n + \left(1-\frac{2}{N}\right)\beta_n \tag{2.25}$$

We can write this mapping in matrix form in the following way

$$\begin{bmatrix}\alpha_{n+1}\\ \beta_{n+1}\end{bmatrix} = \begin{bmatrix}1-\frac{2}{N} & \frac{-2\sqrt{N-1}}{N}\\ \frac{2\sqrt{N-1}}{N} & 1-\frac{2}{N}\end{bmatrix}\begin{bmatrix}\alpha_n\\ \beta_n\end{bmatrix} \tag{2.26}$$

Since $\left(1-\frac{2}{N}\right)^2 + \left(\frac{2\sqrt{N-1}}{N}\right)^2 = 1$, this matrix can be interpreted as the rotation matrix, so we set

$$\begin{bmatrix}1-\frac{2}{N} & \frac{-2\sqrt{N-1}}{N}\\ \frac{2\sqrt{N-1}}{N} & 1-\frac{2}{N}\end{bmatrix} = \begin{bmatrix}\cos\varphi & -\sin\varphi\\ \sin\varphi & \cos\varphi\end{bmatrix} \tag{2.27}$$

Where $\cos\varphi = 1-\frac{2}{N}$ and $\sin\varphi = \frac{2\sqrt{N-1}}{N}$.

When applying Grover iteration, we are actually rotating a vector in the α-β plane. If N is large, initially (in an equal superposition between all states) $\alpha_0 = \frac{\sqrt{N-1}}{\sqrt{N}} \approx 1$ and $\beta_0 = \frac{1}{\sqrt{N}} \approx 0$. The correct state $|x^*\rangle$ is most likely to be found when β_R is as close to one as possible. This is the case when the rotation angle is $\frac{\pi}{2}$. Since it is assumed that $\beta_0 \approx 0$, the initial angle $\varphi_0 = 0$ and the number of iterations can be found by solving $R\varphi \approx \frac{\pi}{2}$. When the rotation angle φ is small (which is the case if N is large), $\sin\varphi \approx \varphi$. Therefore, $\varphi \approx \frac{2\sqrt{N-1}}{N}$. See figure 2.24 for a graphical representation. If we solve for R, we obtain for large N

$$R \approx \frac{\pi}{2} \frac{N}{2\sqrt{N-1}} \approx \frac{\pi}{4}\sqrt{N} = \frac{\pi}{4} 2^{n/2} \tag{2.28}$$

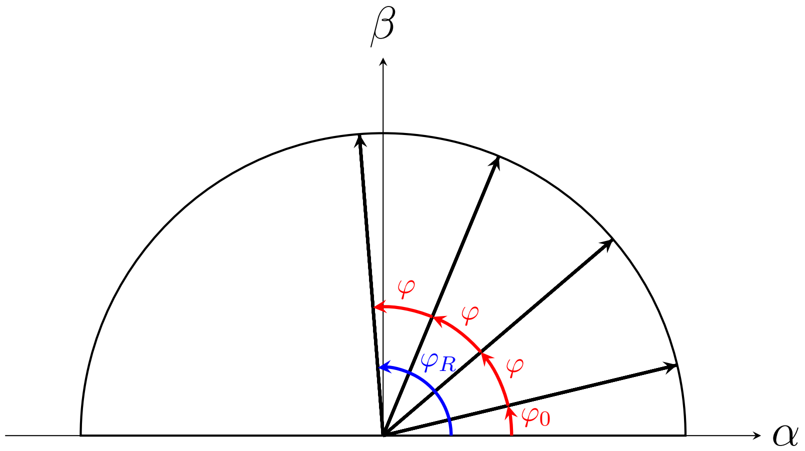


Figure 2.24. With each Grover iteration, the vector representing the probability is rotated over ϕ radians. The probability of finding the correct state ($|\beta|^2$) is highest if the total rotation angle ϕ_R is $\frac{\pi}{2}$.

This approximation becomes more accurate with larger N. However, most existing quantum computers have few bits. Therefore, it is important to look at the exact values. These have been calculated numerically using MATLAB.

We stick with the description using rotation around the unit circle. However, since the assumption that N is large and therefore φ is small has been dropped, the rotation angle per iteration φ is no longer approximated using $\sin\varphi = \varphi$. Instead, the rotation angle is given by

$$\varphi = \text{asin}\, \frac{2\sqrt{N-1}}{N} \tag{2.29}$$

Also, the initial angle φ_0 is no longer assumed to be zero. Instead, the initial angle is given by

$$\varphi_0 = \operatorname{atan} \frac{\beta_0}{\alpha_0} = \operatorname{atan} \frac{1}{\sqrt{N-1}} \tag{2.30}$$

The chance to find the correct state is obviously still the largest if $\beta_m = 1$ and $\alpha_m = 0$, or $\varphi_R = \frac{\pi}{2}$. The exact number of iterations can then be found by solving $\varphi_0 + R\varphi = \frac{\pi}{2}$.

In figure 2.25a, the optimal number of iterations obtained by exact (numerical) calculation is compared to the approximation of equation (2.28). Since the above approximation has been derived for large N, one would expect this approximation to converge to the real optimal number of iterations. Looking at figure 2.25a, this seems to be the case indeed. However, if we zoom in on the difference between the exact value and the approximate value in figure 2.25b, it turns out that the approximation actually converges to the optimal number of iterations plus one half.

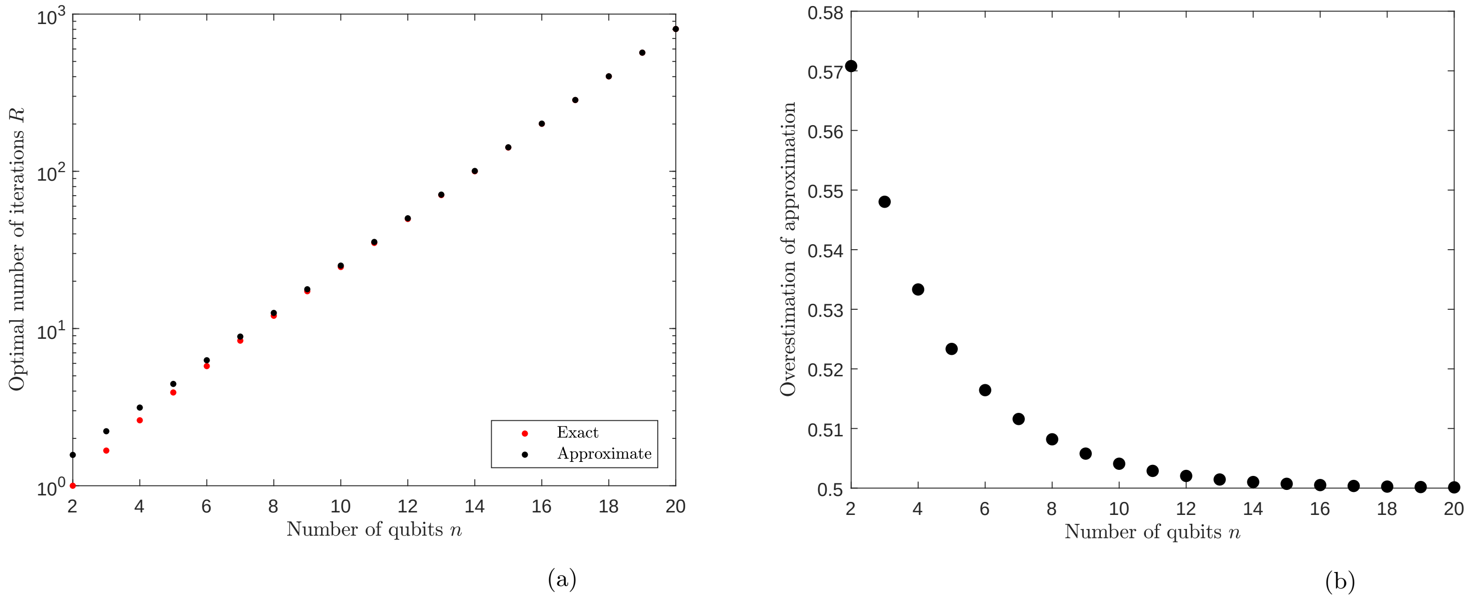


Figure 2.25. Left: The optimal number of iterations and the approximation as a function of the number of qubits.
Right: The overestimation of the approximation for optimal number of iterations needed.

There is an obvious reason for this. Not only is $\sin x$ approximated by x, φ_0 is neglected as well. Moreover, we can find an interesting relation between φ_0 and φ.

$$\frac{\varphi}{\varphi_0} = \frac{\operatorname{asin} \frac{2\sqrt{N-1}}{N}}{\operatorname{atan} \frac{1}{\sqrt{N-1}}} \tag{2.31}$$

For $N = 2$, this results in $\frac{\varphi}{\varphi_0} = \frac{\operatorname{asin} 1}{\operatorname{atan} 1} = \frac{\pi/2}{\pi/4} = 2$. If we consider N to be a continuous variable, after a lengthy calculation involving the complex identities of the asin and atan that is omitted here, it can also be found that for $N \geq 1$

$$\frac{\frac{d\varphi}{dN}}{\frac{d\varphi_0}{dN}}\frac{\frac{d}{dN}\arcsin\frac{2\sqrt{N-1}}{N}}{\frac{d}{dN}\operatorname{atan}\frac{1}{\sqrt{N-1}}} = \frac{2\operatorname{sgn}(x-2)}{\operatorname{sgn}(x)} \tag{2.32}$$

where $\operatorname{sgn}(x) = \begin{cases} -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \\ 1 & \text{if } x > 0 \end{cases}.$

In particular, this fraction is equal to 2 for $N > 2$.

From the above two results, we find that for any number of qubits ($n \geq 2$) there holds

$$\frac{\varphi}{\varphi_0} = 2 \tag{2.33}$$

Thus, using the approximation to find the optimal number of iterations R, the final rotation angle is

$$\varphi_R = \varphi_0 + R\varphi = \frac{\varphi}{2} + R\varphi = \left(R + \frac{1}{2}\right)\varphi \tag{2.34}$$

which is 'half an iteration' too much.

Nevertheless, this is a problem that can be avoided. Since the approximation is an irrational number, it always has to be rounded to a natural number. A better performance of Grover's algorithm can be obtained by not just naively rounding to the closest natural number. Knowing that the approximation systematically overestimates the optimal number of iterations, it makes sense to take the floor of the approximation, i.e.

$$R^* = \lfloor R \rfloor = \left\lfloor \frac{\pi}{4} 2^{n/2} \right\rfloor \tag{2.35}$$

This is supported by the numerical calculations, as shown in figure 2.26. Using the approximation R^* yields the best result for any number of qubits, compared to rounding (which yields an equal or worse result) or taking the ceil of R (which always yields a worse result).

Alternatively, one can take an equivalent approach by using

$$R^{**} = R - \frac{1}{2} = \frac{\pi}{4} 2^{n/2} - \frac{1}{2}$$

and then round to the nearest integer.

In literature though, there is barely any note on how to round the approximate optimal number of iterations. Probably this is because the way it is rounded does not have a

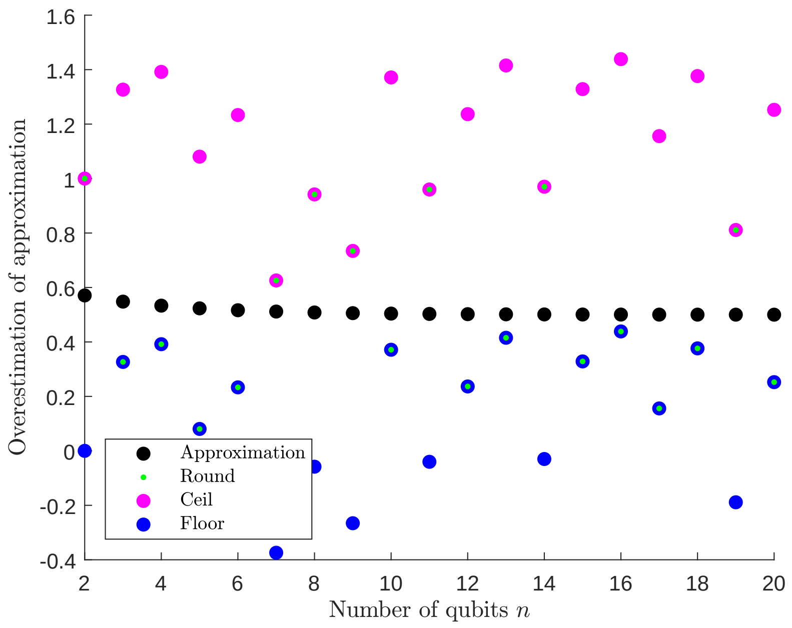


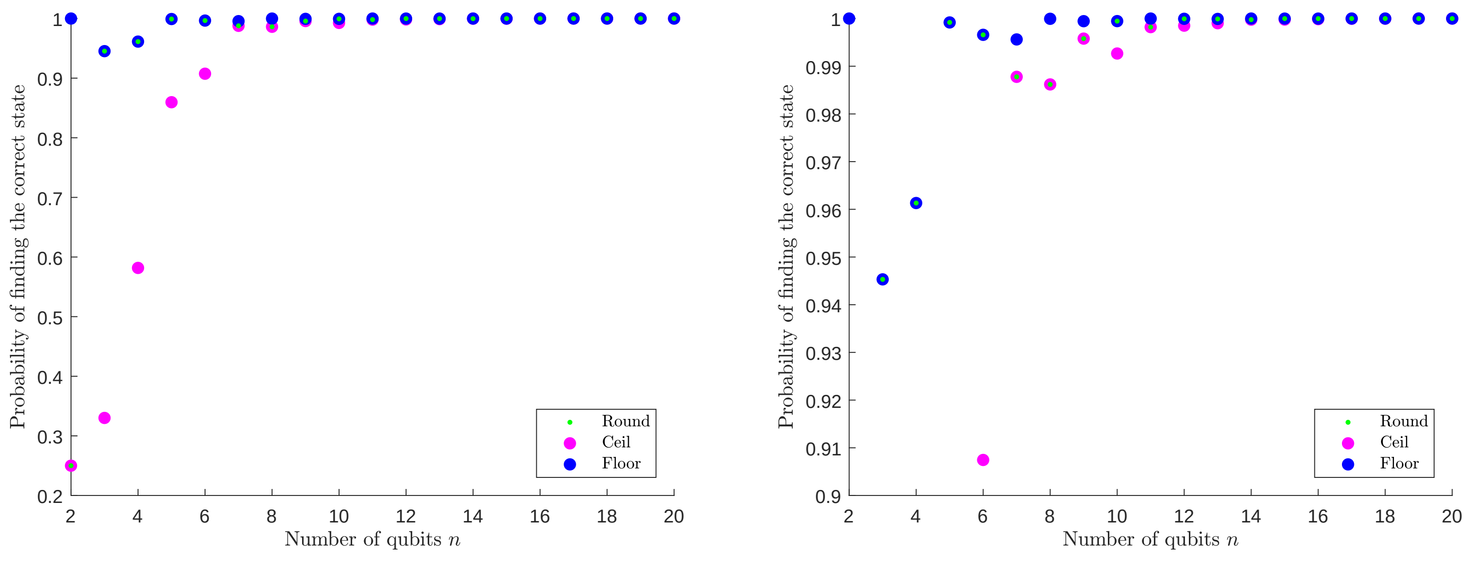
Figure 2.26. The overestimation of the approximation for the optimal number of iterations as a function of the number of qubits. Clearly, taking the floor yields the best result.

significant result on the success rate of Grover's algorithm. From figure 2.27a it can be seen that if the number of qubits is larger than six, the probability of success is over 98%. If we zoom in even further (see figure 2.27b), it is clear that for more than fifteen qubits, the probability of success is virtually 100%, independent of the choice of the rounding method.

Nevertheless, this analysis provides good insight in why one should take the floor of the generally accepted approximation $R = \frac{\pi}{4}2^{n/2}$, especially when a small number of qubits is involved.

2.3.1 Time Complexity

In the previous section, it is confirmed that Grover's algorithm is most probable to yield the correct output state if Grover iteration is applied $R^* = \left\lfloor \frac{\pi}{4}\sqrt{N} \right\rfloor$ times. Thus, the time complexity of Grover's algorithm is $\mathcal{O}(\sqrt{N})$. More commonly, the time complexity is expressed as a function of the input size, i.e. the number of (qu)bits to store the quantum state. Therefore the time complexity of Grover's algorithm is $\mathcal{O}(\sqrt{2^n}) = \mathcal{O}(2^{n/2})$, whereas the time complexity of classical algorithms is $\mathcal{O}(2^n)$. Both of them are exponential time algorithms. We see that Grover's algorithm does not yield exponential speedup, which a fair number of quantum algorithms do. Nevertheless, the speedup is very significant.



(a) (b)

Figure 2.27. The probability of finding the correct state as a function of the number of qubits. Clearly, taking the floor of the approximation yields the best result.

2.4 Searching for multiple states

Suppose that one does not want to search for one particular entry of an unordered list, but instead wants to find one out of multiple states. For example, given a database containing postal codes, find one postal code from a certain province. One could of course apply Grover's algorithm to one of these possibilities, but it can be done in a more efficient manner.

The procedure is much like the usual procedure described by Grover's algorithm, with one modification. The part that flips the sign of the amplitude of the correct state is now applied multiple times in a row.

2.4.1 Grover's Quantum Circuit for Searching Multiple States

Instead of flipping the sign of just one particular state, this part of the algorithm is applied each of the correct states. Recall that this is accomplished by applying the X gate to the qubits that are $|0\rangle$ in the correct state. So if $|010\rangle$ and $|011\rangle$ are the two correct three qubit states, the quantum circuit is given by

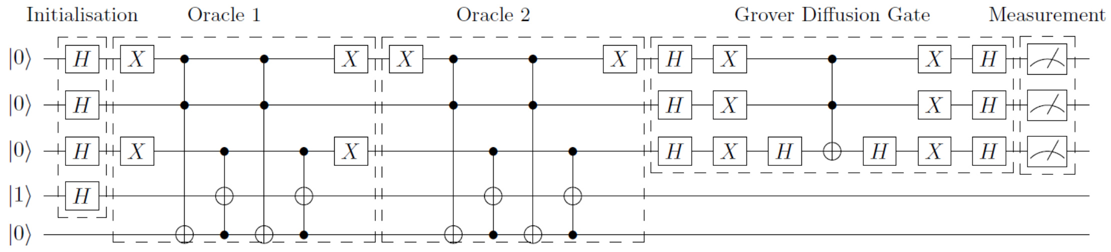


Figure 2.28. The quantum circuit that is used to find $|010\rangle$ and $|011\rangle$ using Grover's algorithm.

This looks familiar to figure 2.12. After the superposition is initiated, the usual procedure is applied to flip the sign of $|010\rangle$. But then, instead of applying the Grover diffusion gate, first the sign flip is repeated for $|110\rangle$. Afterwards, Grover's algorithm is continued in the normal way. Note that the $C^3(X)$ gate and the $C^2(Z)$ gate are decomposed according to the method outlined in section 2.2.3. Two ancillary qubits are added to the quantum register for this reason.

The attentive reader will notice another difference between figure 2.28 and the usual three qubit circuit from figure 2.12. We found that for $n = 3$, $R = \lfloor \frac{\pi}{4}\sqrt{2^3} \rfloor = 2$ so the Grover iteration was applied two times. However, in figure 2.28 it is only applied once. From further analysis (see section 2.4.2) it turns out that the optimal number of iterations does also depend on the number of correct states. In this case, one iteration happens to be optimal number.

Let's track the quantum state throughout the circuit to see what it results in. In this calculation, the second ancillary qubit (which is the bottommost qubit in the figure) is omitted, because it is reset after each circuit part. For a more detailed breakdown of the working of the different circuit parts, the reader is referred to section 2.2

After applying the set Hadamard gates, the quantum state is

$$|\psi\rangle = \frac{1}{4}(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle) \otimes (|0\rangle - |1\rangle) \quad (2.36)$$

Then, the sign of $|010\rangle$ and $|011\rangle$ is flipped. No problems occur with this back-to-back sign flip. Afterwards, the state is given by

$$|\psi\rangle = \frac{1}{4}(|000\rangle + |001\rangle - |010\rangle - |011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle) \otimes (|0\rangle - |1\rangle) \quad (2.37)$$

The average amplitude is $\mu = \frac{1}{8}\frac{1+1-1-1+1+1+1+1}{4} = \frac{1}{8}$. This is, as in the two qubit example, exactly half of the amplitude of the non-correct states. As a result, the amplitude of the non-correct state reduces to zero after applying the Grover diffusion gate. The amplitudes of the two correct states on the other hand grow to $\frac{1}{2}$. It will be shown in section 2.4.2 that this is again a special case. The final state is

$$\frac{1}{2}(|010\rangle + |011\rangle) \otimes (|0\rangle - |1\rangle) \tag{2.38}$$

Performing a measurement on the relevant qubits yields either $|011\rangle$ or $|011\rangle$, both with a probability of 50% (so a 100% probability of obtaining one of both).

Note that if both correct states need to be found instead of just one of them, this adapted algorithm is not efficient. Grover's algorithm approximately has a 0.9453 chance to find the correct state. If it is first used to find $|010\rangle$ and consecutively to find $|110\rangle$, the success rate is $0.9453^2 \approx 0.8936$. The multiple state algorithm on the other hand has a 0.5 probability to find one of these states. After two executions, the probability of finding both states (first $|0101\rangle$ and then $|110\rangle$ or vice versa) is only $0.5^2 + 0.5^2 = 0.5$.

2.4.2 Analysis of the Optimal Number of Iterations

The analysis in section 2.3 can also be extended to the multi-state search. It will be shown that there is a limit on the amount of correct states for which the algorithm functions well. If the number of correct states becomes too large, it becomes impossible to obtain one of them with reasonable chance.

The analysis of the multi-search Grover's algorithm can be done in a similar way to the calculation of the optimal number of iterations in the original Grover's algorithm.

As usual, the total number of states is denoted by N. Let M be the number of correct states. Since it is possible to have $M > 1$, we now write

$$|\psi\rangle = \alpha_n \left(\frac{1}{\sqrt{N-M}} \sum_{x \neq x^*} |x\rangle \right) + \beta_n \left(\frac{1}{\sqrt{M}} \sum_{x = x^*} |x\rangle \right) \tag{2.39}$$

Initially, $\alpha_0 = \frac{\sqrt{N-M}}{\sqrt{N}}$ and $\beta_0 = \frac{\sqrt{M}}{\sqrt{N}}$ so that initially the amplitude of all qubit states is equal to $\frac{1}{\sqrt{N}}$.

Then, we can repeat the calculation we did before. First, the average amplitude after n iterations is calculated.

$$\mu = \frac{1}{N} \left(\alpha_n \frac{N-M}{\sqrt{N-M}} - \beta_n \frac{M}{\sqrt{M}} \right) = \frac{\alpha_n \sqrt{N-M} - \beta_n \sqrt{M}}{N} \tag{2.40}$$

Then, we apply the Grover diffusion gate which maps $\alpha_n \mapsto 2\mu - \alpha_n$. We obtain for the amplitude of the states $x \neq x*$

$$\begin{aligned}\frac{\alpha_n}{\sqrt{N-M}} \mapsto & \frac{2\alpha_n\sqrt{N-M}-2\beta_n\sqrt{M}}{N} - \frac{\alpha_n}{\sqrt{N-M}} \\ &= \frac{2\alpha_n(N-M)-2\beta_n\sqrt{M(N-M)}-\alpha_n N}{N\sqrt{N-M}} \\ &= \frac{(N-2M)\alpha_n - 2\beta_n\sqrt{M(N-M)}}{N\sqrt{N-M}}\end{aligned} \tag{2.41}$$

Or, equivalently after multiplying by $\sqrt{N-M}$,

$$\alpha_n \mapsto \left(1-\frac{2M}{N}\right)\alpha_n - \frac{2\sqrt{M(N-M)}}{N}\beta_n \tag{2.42}$$

And for the amplitude of x^*

$$\begin{aligned}\frac{\beta_n}{\sqrt{M}} \mapsto & \frac{2\alpha_n\sqrt{N-M}-2\beta_n\sqrt{M}}{N} + \frac{\beta_n}{\sqrt{M}} \\ &= \frac{2\alpha_n\sqrt{M(N-M)}-2\beta_n M+\beta_n N}{N\sqrt{M}}\end{aligned} \tag{2.43}$$

Or, equivalently after multiplying by $\sqrt{M}$,

$$\beta_n \mapsto \frac{2\sqrt{M(N-M)}}{N}\alpha_n + \left(1-\frac{2M}{N}\right)\beta_n \tag{2.44}$$

Again, we can write this mapping as a matrix multiplication in the following way

$$\begin{bmatrix}\alpha_{n+1}\\ \beta_{n+1}\end{bmatrix} = \begin{bmatrix}1-\frac{2M}{N} & \frac{-2\sqrt{M(N-M)}}{N} \\ \frac{2\sqrt{M(N-M)}}{N} & 1-\frac{2M}{N}\end{bmatrix}\begin{bmatrix}\alpha_n\\ \beta_n\end{bmatrix} \tag{2.45}$$

Since $\left(1-\frac{2M}{N}\right)^2+\left(\frac{2\sqrt{M(N-M)}}{N}\right)^2=1$, this matrix can again be interpreted as a rotation matrix, so

$$\begin{bmatrix}1-\frac{2M}{N} & \frac{-2\sqrt{M(N-M)}}{N} \\ \frac{2\sqrt{M(N-M)}}{N} & 1-\frac{2M}{N}\end{bmatrix} = \begin{bmatrix}\cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi\end{bmatrix} \tag{2.46}$$

Where $\cos\varphi = 1-\frac{2M}{N}$ and $\sin\varphi = \frac{2\sqrt{M(N-M)}}{N}$.

Now, we cannot use the same approximation ($\sin\varphi \approx \varphi$) as before. Therefore, $R^* = \lfloor \frac{\pi}{4}\sqrt{N} \rfloor$ is no longer valid. Also note that Grover's algorithm does not work for $M \geq \frac{N}{2}$. In that case, $1 - \frac{2M}{N} \leq 0$. As a result, $\cos\varphi \leq 0$, or $\varphi \geq \frac{\pi}{2}$.

Using the initial values α_0 and β_0, the initial angle can be obtained.

$$\varphi_0 = \arctan\frac{\beta_0}{\alpha_0} = \operatorname{atan}\sqrt{\frac{M}{N}}\sqrt{\frac{N}{N-M}} = \operatorname{atan}\sqrt{\frac{M}{N-M}} \tag{2.47}$$

Because $\operatorname{atan} x$ is a strictly increasing function, for $M \geq \frac{N}{2}$ we also have $\varphi_0 \geq \operatorname{atan} 1 = \frac{\pi}{4}$. So already after one iteration, the rotation angle is $\frac{3\pi}{4}$ which overshoots the target of $\frac{\pi}{2}$ radians.

In order to obtain one of the correct states with maximum probability, we must apply grover iteration until $\beta_0 \approx 1$, which corresponds to an angle of $\frac{\pi}{2}$. The number of iterations R' can be found by solving

$$R'\varphi = \frac{\pi}{2} - \varphi_0 \tag{2.48}$$

Using $\varphi = \operatorname{asin}\frac{2\sqrt{M(N-M)}}{N}$ and $\varphi_0 = \arctan\sqrt{\frac{M}{N-M}}$, we find

$$R' = \left(\operatorname{asin}\frac{2\sqrt{M(N-M)}}{N}\right)^{-1}\left(\frac{\pi}{2} - \operatorname{atan}\sqrt{\frac{M}{N-M}}\right) \tag{2.49}$$

This formula yields some interesting results. When we obtained the correct state with certainty in the two qubit case, it seemed like it maybe was an exception. However, in general for $M = \frac{N}{4}$ we have

$$\operatorname{asin}\frac{2\sqrt{M(N-M)}}{N} = \operatorname{asin}\frac{2\sqrt{\frac{N}{4}\frac{3N}{4}}}{N} = \operatorname{asin}\frac{2\frac{N}{4}\sqrt{3}}{N} = \operatorname{asin}\frac{\sqrt{3}}{2} = \frac{\pi}{3} \tag{2.50}$$

and also

$$\frac{\pi}{2} - \operatorname{atan}\sqrt{\frac{M}{N-M}} = \frac{\pi}{2} - \operatorname{atan}\sqrt{\frac{N/4}{3N/4}} = \frac{\pi}{2} - \operatorname{atan}\frac{\sqrt{3}}{3} = \frac{\pi}{2} - \frac{\pi}{6} = \frac{\pi}{3} \tag{2.51}$$

So equation (2.49) is solved exactly by $R' = 1$ in this respect.

Another interesting fact is that the number of iterations needed decreases with the number of correct states. Both $\operatorname{asin} x$ and $\operatorname{atan} x$ are strictly increasing functions. The arguments of these functions in this context are also strictly increasing, since the initial chance of finding one of the correct states grows.

The rotation angle φ grows too . However, if the number of correct states exceeds $\frac{N}{4}$, the maximum probability of finding a correct state decreases significantly. The reason for this is that the rotation angle φ grows with the number of correct states. If M approaches $\frac{N}{2}$, φ approaches $\frac{\pi}{2}$. Since $\frac{\beta_0}{\alpha_0} \geq \frac{\sqrt{3}}{3}$ for $M \geq \frac{N}{4}$, $\varphi_0 \geq \frac{\pi}{6}$. Therefore, the rotation angle overshoots $\frac{\pi}{2}$ by a larger amount if M grows larger than $\frac{N}{4}$.

Finally, the properties described in this section can be found in the following example, in which $N = 2^3 = 8$. The optimal number of iterations as well as the maximum probability of finding one of the correct states is given in table 2.1. If $M = \frac{16}{4} = 4$, one of the correct states is found with certainty after one iteration. If M is bigger than 6, the probability of obtaining a correct state reduces significantly. Grover's algorithm becomes far less efficient in that case.

Table 2.1. $N = 16$, φ_0 is the initial angle, φ is the rotation angle, m is the optimal number of iterations and $P(x^*)$ is the measurement probability of the correct state after m iterations.

M	φ_0 (rad)	φ (rad)	m	$P(x^*)$
1	0.2527	0.5054	3	0.9613
2	0.3614	0.7227	2	0.9453
3	0.4478	0.8957	1	0.9492
4	0.5236	1.0472	1	1.0000
5	0.5932	1.1864	1	0.9570
6	0.6591	1.3181	1	0.8438
7	0.7227	1.4455	1	0.6836
8	0.7854	1.5708	1	0.5000

2.5 Application of Multi-Search to the SAT-Problem

A real application of Grover's algorithm is the SAT-problem, or satisfiability problem. Let $x_1, x_2, \ldots$ be variables that are either TRUE (1) or FALSE (0). First, some notation is introduced.

$$\begin{aligned}
\overline{x_1} \text{ is TRUE} &\Leftrightarrow & x_1 \text{ is FALSE} \\
x_1 \vee x_2 \text{ is TRUE} &\Leftrightarrow & \text{at least one of } x_1 \text{ and } x_2 \text{ is TRUE} \\
x_1 \wedge x_2 \text{ is TRUE} &\Leftrightarrow & \text{both } x_1 \text{ and } x_2 \text{ are TRUE}
\end{aligned}$$

A literal is a boolean variable (x_i) or the negation of a boolean variable ($\overline{x_i}$). A clause is a disjunction of literals, e.g. $(x_1 \vee \overline{x_2} \vee \overline{x_4} \vee x_9)$. A clause is TRUE if and only if all literals are TRUE (in this case, $(x_1 \vee \overline{x_2} \vee \overline{x_4} \vee x_9) = (1, 0, 0, 1)$, the other boolean assignments can be chosen randomly). A conjunction of clauses $C_1, C_2, \ldots, C_k$ is $C_1 \wedge C_2 \wedge \cdots \wedge C_k$ is TRUE if and only if all clauses are TRUE.

The SAT-problem is whether, given a set of boolean variables and a conjunction C, there exists a boolean assignment that the evaluates to TRUE. An example of a SAT-problem is $C = C_1 \wedge C_2 \wedge C_3 = (x_1 \vee x_2) \wedge (\overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_3})$. A small SAT-problem can easily solved by observation. Suppose x_1 is TRUE. Then in order to satisfy C_3, x_3 should be FALSE. Knowing this, in order to satisfy C_2, x_2 should be FALSE. So a solution of this problem is $(x_1, x_2, x_3) = (1, 0, 0)$. Similarly, it can be found that $C = (x_1) \wedge (\overline{x_1} \vee x_3) \wedge (\overline{x_2} \vee x_3) \wedge (x_2 \vee \overline{x_3})$ cannot be satisfied.

A well-known case of the SAT-problem is 3-SAT. Here, each clause contains at most three literals. 3-SAT is known to be an NP-hard problem. This means that no polynomial time algorithm for solving 3-SAT is known and 'probably' does not exist. Most algorithms for solving NP-hard problems are based on a smart, semi-random way to find a solution. The fastest known algorithms for 3-SAT are about $\mathcal{O}(1.3^n)$, for example an algorithm developed by Hertli, Moser & Schneder runs in $\mathcal{O}(1.321^n)$[20]. Grover's algorithm runs, depending on the number of solutions, in maximum $\mathcal{O}(2^{n/2}) \approx \mathcal{O}(1.414^n)$ which is close to the best known classical algorithms. If the number of solutions is larger, this runtime decreases and eventually, in the best-case scenario, when there are $\frac{N}{4}$ solutions, a solution can be obtained with certainty in only one run.

The multi-search Grover's algorithm fits this problem very well because all possible boolean assignments can be checked in parallel. In this case, the oracle function is used to flip the sign of the states that satisfy the given conjunction of clauses. To accomplish this, a new quantum gate is introduced: the OR-$C^n(X)$ gate that flips the sign of the target qubit if and only if at least one of the controls is $|1\rangle$. For example, the OR-$C^2(X)$ gate is given by

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.52}$$

The OR-$C^n(X)$ gate can be decomposed into $2^n - 1$ $C^n(X)$ gates. It is accomplished by applying $\binom{n}{k}$ $C^k(X)$ gates with all possible combinations of control qubits for each $k \in \{1, 2, \ldots, n\}$ to the target qubit. For example, the OR-$C^3(X)$ gate is equivalent to the circuit in figure 2.29.

The $C^n(X)$ gates on their end can be decomposed into Toffoli gates and Hadamard gates as outlined in section 2.2.3.

In this section, the OR control qubits will be indicated by an open circle: $\circ$. Note that this is commonly used to indicate 'anti-control' (execute a gate if and only if the control

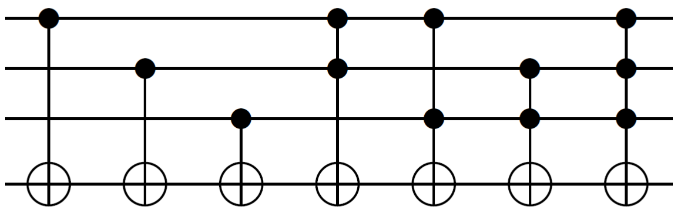


Figure 2.29. The decomposition of the OR-$C^3(X)$ gate.

is $|0\rangle$), but in this section that is not the case.

The qubit states that give a solution to the SAT-problem are the states which satisfy all clauses. Consider the example given earlier in this section:

$$C = C_1 \wedge C_2 \wedge C_3 = (x_1 \vee x_2) \wedge (\overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_3})$$

To find a solution to this problem using Grover's algorithm, the amplitude of any qubit state that satisfy this problem should be amplified. In particular, these qubit states satisfy ($|q_1\rangle = |1\rangle$ or $|q_2\rangle = |1\rangle$) and ($|q_2\rangle = |0\rangle$ or $|q_3\rangle = |1\rangle$) and ($|q_1\rangle = |0\rangle$ and $|q_3\rangle = |0\rangle$). This is exactly what happens in the circuit given in figure 2.30. First, the quantum register is initialized to $|\psi\rangle = |0000001\rangle$ and the Hadamard gate is applied to the qubits that are involved in Grover's algorithm. Then, a series of OR-$C^2(X)$ gates is applied so that the fourth through sixth qubit are all $|1\rangle$ if and only if the first through third qubit satisfy the values outlined above. If a literal is the negation of a boolean variable, the X gate is applied before and after the application of the OR-$C^2(X)$ gate. Consequently, the control is true if the qubit was originally $|0\rangle$. As a result, the sign of the qubit states that satisfy the conjunction of clauses is flipped (which are in this case $|100\rangle$ and $|011\rangle$). Thereafter, the first part is repeated in reverse order to reset the values to $|0\rangle$ which is necessary if more than one iteration is done. In this case, it turns out that there are two solutions so one iteration yields one of the possible boolean assignments after measurement. Either $|100\rangle$ or $|011\rangle$ is obtained, both with 50% probability.

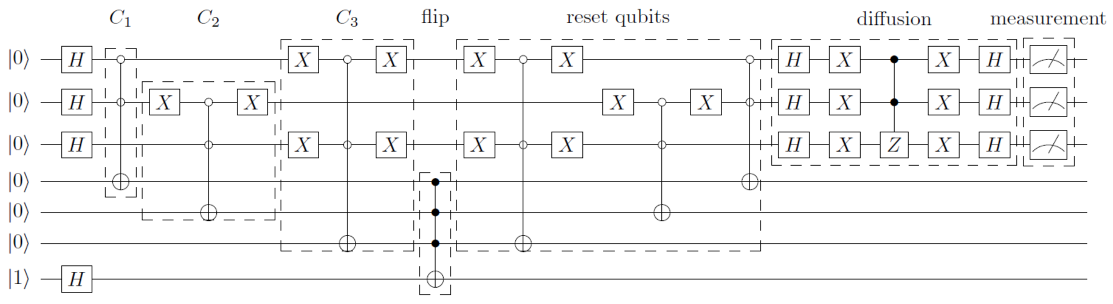


Figure 2.30. The quantum circuit implementation for this instance of SAT.

With this, the analysis on Grover's algorithm and the extension to the multi-search algorithm is finished. Before returning to these algorithms in the results of the simulations in section 6.3, the Quantum Fourier transform and its applications, including Shor's algorithm, will be presented in the upcoming chapters.

3 Quantum Fourier Transform

The Fourier transform is a widely used technique in solving mathematical and physical problems. However, computing the Fourier transform on a classical computer requires exponential time. As a result, complex algorithms in which the Fourier transform is applied can take very long to execute. On a quantum computer, however, the time complexity of the Fourier transform can be reduced to polynomial time. The quantum equivalent of the discrete Fourier transform is one of the greatest discoveries in the history of quantum computing. It is used in a wide variety of quantum algorithms, the most famous one probably being Shor's algorithm for factoring large numbers.

First, the discrete Fourier transform is described in section 3.1. In section 3.2, the quantum Fourier transform is introduced. The implementation of the quantum Fourier transform is described in section 3.3. Afterwards, in section 3.4 the implementation of the inverse quantum Fourier transform is described. Finally, the time complexity of the (inverse) Fourier transform is analysed in section 3.4.1.

3.1 Discrete Fourier Transform

The discrete Fourier transform that is used in classical computing and signal processing is defined as follows[33].

Definition 3.1. *The (classical) discrete Fourier transform (DFT) $(y_0, \ldots, y_{N-1})$ of a sequence $(x_0, \ldots, x_{N-1}) \in \mathbb{C}^N$ is given by*

$$y_j = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \omega_N^{jk} x_k \tag{3.1}$$

in which $\omega_N^{jk} = e^{2\pi i jk/N}$.

Computing one entry of the DFT of a vector with length $N = 2^n$ would require $\mathcal{O}(N) = \mathcal{O}(2^n)$ operations. Thus, to compute the whole vector $(y_0, \ldots, y_{N-1})$ it would take $\mathcal{O}(N^2) = \mathcal{O}(2^{2n})$ operations. The so-called Fast Fourier Transform (FFT) is a technique to speed-up the classical Fourier Transform. This technique relies on the fact that an N-dimensional DFT can be split up into two $N/2$-dimensional Fourier transforms (see [33] for further details on the FFT). By applying this trick recursively, the FFT calculates the DFT in $\mathcal{O}(N \log N) = \mathcal{O}(n2^n)$ steps, which is a significant speedup compared to the standard algorithm, but it is still exponential time.

3.2 The Quantum Fourier Transform

Not only is the Fourier transform very useful in solving classical problems, it is very useful in quantum mechanics as well. In section 4 the use of the quantum Fourier transform

(QFT) will be illustrated by means of quantum addition. But first, the quantum Fourier transform is introduced[26].

Definition 3.2. *Let $|x\rangle$ be some quantum register and let the QFT of $|x\rangle$ be denoted as the quantum register $|y\rangle$. The quantum Fourier transform (QFT) for $|x\rangle$ and $|y\rangle$ in the orthonormal basis $\{|0\rangle, |1\rangle, \ldots, |N-1\rangle\}$ is a linear transformation defined similarly to the DFT by the mapping*

$$|x\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} \omega_N^{yx} |y\rangle \tag{3.2}$$

in which $\omega_N^{yx} = e^{2\pi i yx/N}$ is the Nth root of unity.

Accordingly, the QFT of a superposition state is obtained by summing the QFTs of the separate states with their corresponding weights.

The QFT is a linear operator. Therefore, it can be expressed in a more convenient way by the matrix[26]

$$F_N \equiv \frac{1}{\sqrt{N}} \begin{bmatrix} \omega_N^{00} & \cdots & \omega_N^{0(N-1)} \\ \vdots & & \vdots \\ \omega_N^{(N-1)0} & \cdots & \omega_N^{(N-1)(N-1)} \end{bmatrix} \tag{3.3}$$

It can be checked that this is a unitary matrix by carrying out the matrix multiplication $F_N F_N^\dagger$[23] and thus it defines a valid quantum gate. As an example, consider $N = 4$. Then, F_4 is given by

$$F_4 \equiv \frac{1}{\sqrt{4}} \begin{bmatrix} \omega_4^{00} & \omega_4^{01} & \omega_4^{02} & \omega_4^{03} \\ \omega_4^{10} & \omega_4^{11} & \omega_4^{12} & \omega_4^{13} \\ \omega_4^{20} & \omega_4^{21} & \omega_4^{22} & \omega_4^{23} \\ \omega_4^{30} & \omega_4^{31} & \omega_4^{32} & \omega_4^{33} \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix} \tag{3.4}$$

If F_4 is applied to $|01\rangle$, we obtain

$$F_4 |01\rangle \equiv \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ i \\ -1 \\ -i \end{bmatrix} \equiv \frac{1}{2} (|00\rangle + i|01\rangle - |10\rangle - i|11\rangle) \tag{3.5}$$

In order to find the quantum circuit that defines the QFT on an n qubit quantum register, definition 3.2 must be rewritten in terms of the $\{|0\rangle, |1\rangle\}^n$ state.

3.2.1 Quantum Fourier Transform in the Computational Basis

Using binary notation, $|y\rangle$ can be rewritten in the computational basis as $|y_1 y_2 \dots y_n\rangle$ with $|y_i\rangle \in \{|0\rangle, |1\rangle\}$.

The QFT mapping can then be written as,

$$|x\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{2\pi ixy/2^n} |y\rangle \tag{3.6}$$

$$= \frac{1}{\sqrt{N}} \sum_{y_1=0}^{1} \dots \sum_{y_n=0}^{1} e^{2\pi ix\left(\sum_{k=1}^{n} y_k 2^{n-k}\right)/2^n} |y_1 \dots y_n\rangle \tag{3.7}$$

$$= \frac{1}{\sqrt{N}} \sum_{y_1=0}^{1} \dots \sum_{y_n=0}^{1} e^{2\pi ix\left(\sum_{k=1}^{n} y_k 2^{-k}\right)} |y_1 \dots y_n\rangle \tag{3.8}$$

$$= \frac{1}{\sqrt{N}} \sum_{y_1=0}^{1} \dots \sum_{y_n=0}^{1} \bigotimes_{k=1}^{n} e^{2\pi ixy_k 2^{-k}} |y_k\rangle \tag{3.9}$$

$$= \frac{1}{\sqrt{N}} \bigotimes_{k=1}^{n} \left(\sum_{y_k=0}^{1} e^{2\pi ixy_k 2^{-k}} |y_k\rangle \right) \tag{3.10}$$

$$= \frac{1}{\sqrt{N}} \bigotimes_{k=1}^{n} \left(|0\rangle + e^{2\pi ix 2^{-k}} |1\rangle \right) \tag{3.11}$$

$$\tag{3.12}$$

where $\bigotimes_{i=1}^{n} |A_i\rangle = |A_1\rangle \otimes |A_2\rangle \otimes \dots \otimes |A_n\rangle$[26].

But of course, $|x\rangle$ must be expressed in the computational basis, $|x_1 x_2 \dots x_\ell\rangle$ with $|x_i\rangle \in \{|0\rangle, |1\rangle\}^n$ as well. Using binary notation, it is found that

$$e^{2\pi ix2^{-k}} = e^{2\pi i\left(\sum_{\ell=1}^{n} x_\ell 2^{n-\ell}\right)2^{-k}} \tag{3.13}$$

$$= e^{2\pi i\left(\sum_{\ell=1}^{n} x_\ell 2^{(n-k)-\ell}\right)} \tag{3.14}$$

$$= e^{2\pi i\left(\sum_{\ell=1}^{n-k} x_\ell 2^{(n-k)-\ell}\right)} e^{2\pi i\left(\sum_{\ell=n-k+1}^{n} x_\ell 2^{(n-k)-\ell}\right)} \tag{3.15}$$

$$= e^{2\pi i\left(\sum_{\ell=n-k+1}^{n} x_\ell 2^{(n-k)-\ell}\right)} \tag{3.16}$$

$$= e^{2\pi i[0.x_{n-k+1} x_{n-k+2} \dots x_n]} \tag{3.17}$$

In the second-last step, the terms of the sum with $1 \leq \ell \leq n-k$ drop out since these result in integer multiples of $2\pi i$. The final result uses binary fractional notation,

$$[0.x_1 \dots x_m] = \sum_{m=1}^{n} x_m 2^{-m} \tag{3.18}$$

So finally, an expression for the QFT in the computational basis is obtained[26].

$$
\begin{aligned}
F_N\,|x_1\dots x_n\rangle &= \frac{1}{\sqrt{N}}\bigotimes_{k=1}^{n}\Big(|0\rangle + e^{2\pi i x 2^{-k}}\,|1\rangle\Big) && (3.19)\\
&= \frac{1}{\sqrt{N}}\bigotimes_{k=1}^{n}\Big(|0\rangle + e^{2\pi i[0.x_{n-k+1}\dots x_n]}\,|1\rangle\Big) && (3.20)\\
&= \frac{1}{\sqrt{N}}\Big(|0\rangle + e^{2\pi i[0.x_n]}\,|1\rangle\Big)\Big(|0\rangle + e^{2\pi i[0.x_{n-1}x_n]}\,|1\rangle\Big)\dots\Big(|0\rangle + e^{2\pi i[0.x_1\dots x_n]}\,|1\rangle\Big) && (3.21)
\end{aligned}
$$

Again, let us take $N = 4$ ($n = 2$) as an example. The result of applying the QFT to $|01\rangle$ is indeed the same as in equation (3.5).

$$
\begin{aligned}
F_4\,|01\rangle &= \frac{1}{2}\Big(|0\rangle + e^{2\pi i[0.x_2]}\,|1\rangle\Big)\Big(|0\rangle + e^{2\pi i[0.x_1x_2]}\,|1\rangle\Big) && (3.22)\\
&= \frac{1}{2}\Big(|0\rangle + e^{2\pi i[0.1]}\,|1\rangle\Big)\Big(|0\rangle + e^{2\pi i[0.01]}\,|1\rangle\Big) && (3.23)\\
&= \frac{1}{2}\Big(|0\rangle + e^{2\pi i/2}\,|1\rangle\Big)\Big(|0\rangle + e^{2\pi i/4}\,|1\rangle\Big) && (3.24)\\
&= \frac{1}{2}(|0\rangle - |1\rangle)(|0\rangle + i\,|1\rangle) && (3.25)\\
&= \frac{1}{2}(|00\rangle + i\,|01\rangle - |10\rangle - i\,|11\rangle) && (3.26)\\
& && (3.27)
\end{aligned}
$$

In order to find the QFT of a superposition state, e.g. $\frac{\sqrt{3}}{2}\,|10\rangle + \frac{1}{2}\,|11\rangle$ the QFT of $|01\rangle$ and $|10\rangle$ are summed with their corresponding weights.

$$
\begin{aligned}
\frac{\sqrt{3}}{2}F_4\,|01\rangle + \frac{1}{2}F_4\,|10\rangle &= \frac{\sqrt{3}}{4}\Big(|0\rangle + e^{2\pi i[0.1]}\,|1\rangle\Big)\Big(|0\rangle + e^{2\pi i[0.01]}\,|1\rangle\Big) && (3.28)\\
&+ \frac{1}{4}\Big(|0\rangle + e^{2\pi i[0.0]}\,|1\rangle\Big)\Big(|0\rangle + e^{2\pi i[0.10]}\,|1\rangle\Big) && (3.29)\\
&= \frac{\sqrt{3}}{4}(|0\rangle - |1\rangle)(|0\rangle + i\,|1\rangle) + \frac{1}{4}(|0\rangle + |1\rangle)(|0\rangle - |1\rangle) && (3.30)\\
&= \frac{1+\sqrt{3}}{4}\,|00\rangle - \frac{1-i\sqrt{3}}{4}\,|01\rangle + \frac{1-\sqrt{3}}{4}\,|10\rangle - \frac{i+\sqrt{3}}{4}\,|11\rangle && (3.31)
\end{aligned}
$$

There is one severe disadvantage that the QFT has compared to the classical Fourier transform: the transformed sequence is hidden in the coefficients of the quantum state.

There is no way to find these coefficients since only the amplitude squared influences a measurement outcome. This amplitude squared is equal to $\frac{1}{\sqrt{N}}$ for all coefficients. Therefore, it is not possible to determine the Fourier transform in explicit form. In spite of this, it is perfectly fine to use the QFT to make certain manipulations on qubits easier. First, the quantum register is Fourier transformed. Then, some gates are executed on the register. Afterwards, the state is back transformed using the inverse QFT. Finally, a measurement can be performed to find the desired result.

3.3 Implementation of the QFT

The key to a quantum circuit that performs the QFT is the controlled phase shift gate. The phase shift gate itself is given by

$$R_\varphi \equiv \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^\varphi} \end{bmatrix} \tag{3.32}$$

For $\varphi = 1, 2, 3$ you might recognize this gate as the Pauli-Z gate, the S gate and the T gate respectively.

$$R_1 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = Z \qquad R_2 = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} = S \qquad R_3 = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} = T \tag{3.33}$$

The generalized phase shift gate R_φ rotates the phase of $|1\rangle$ over an angle $2\pi i/2^\varphi$ around the unit circle and leaves $|0\rangle$ unchanged. The full quantum circuit for the QFT using Hadamard gates, controlled phase shift gates and SWAP gates is given in figure 3.1. The initial quantum register is the arbitrary state $|\psi\rangle = |q_1 \cdots q_n\rangle$.

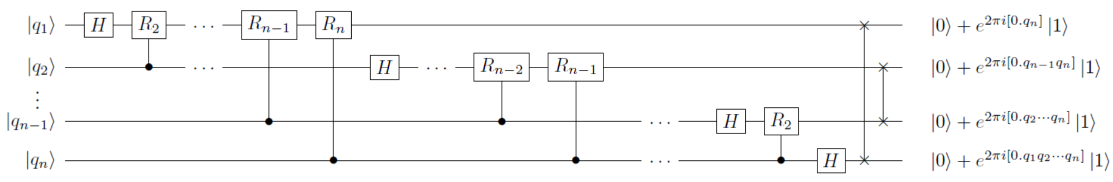

$|q_1\rangle$
$|q_2\rangle$
$|q_{n-1}\rangle$
$|q_n\rangle$
H
R_2
R_{n-1}
R_n
R_{n-2}
$|0\rangle + e^{2\pi i[0.q_n]}|1\rangle$
$|0\rangle + e^{2\pi i[0.q_{n-1}q_n]}|1\rangle$
$|0\rangle + e^{2\pi i[0.q_2\cdots q_n]}|1\rangle$
$|0\rangle + e^{2\pi i[0.q_1q_2\cdots q_n]}|1\rangle$

Figure 3.1. The quantum circuit that applies the quantum Fourier transform to an arbitrary n qubit register.

At the start, the first qubit is manipulated (see figure 3.2).

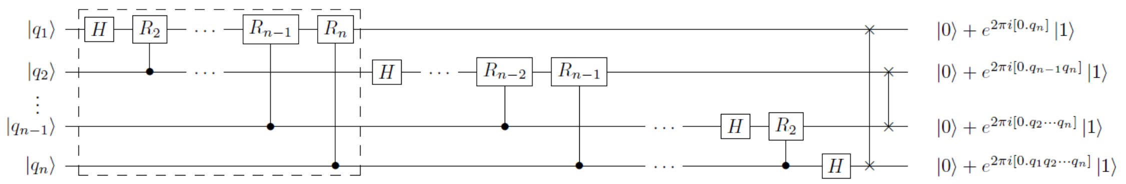

$|q_1\rangle$
$|q_2\rangle$
$|q_{n-1}\rangle$
$|q_n\rangle$
H
R_2
R_{n-1}
R_n
H
R_{n-2}
R_{n-1}
H
R_2
H
$|0\rangle + e^{2\pi i[0.q_n]}|1\rangle$
$|0\rangle + e^{2\pi i[0.q_{n-1}q_n]}|1\rangle$
$|0\rangle + e^{2\pi i[0.q_2\cdots q_n]}|1\rangle$
$|0\rangle + e^{2\pi i[0.q_1q_2\cdots q_n]}|1\rangle$

Figure 3.2. The gates inside the dashed rectangle cast the first qubit in the desired form.

First, the Hadamard gate is applied to the first qubit. This yields the state

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle)|q_1 \cdots q_n\rangle \tag{3.34}$$

with the plus sign if $|q_1\rangle = |0\rangle$ and the minus sign if $|q_1\rangle = |1\rangle$. The sign can be replaced by a factor $e^{2\pi i[0.q_1]}$ in front of $|1\rangle$, since $e^{2\pi i[0.0]} = e^0 = 1$ and $e^{2\pi i[0.1]} = e^{\pi i} = -1$.

$$|\psi\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i[0.q_1]}|1\rangle\right)|q_2 \cdots q_n\rangle \tag{3.35}$$

If now the controlled R_2 gate is applied to $|q_1\rangle$ with $|q_2\rangle$ as control qubit, this yields an extra factor $e^{2\pi i/4}$ in front of $|1\rangle$ if $|q_2\rangle = 1$ and a factor 1 if $|q_2\rangle = |0\rangle$. Using fractional binary notation, this can be written as $e^{2\pi i[0.0q_2]}$. Thus the state can be expressed as

$$|\psi\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i[0.q_1q_2]}|1\rangle\right)|q_2 \cdots q_n\rangle \tag{3.36}$$

After applying the R_3 gate to $|q_1\rangle$ controlled by $|q_3\rangle$, the R_4 gate controlled by $|q_4\rangle$, up to R_n controlled by $|q_n\rangle$, the first qubit is in the desired state,

$$|\psi\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{2\pi i[0.q_1q_2\cdots q_n]}|1\rangle\right)|q_2 \cdots q_n\rangle \tag{3.37}$$

In a similar way, $|q_2\rangle$ can be manipulated using the circuit part inside the dashed rectangle in figure 3.3.

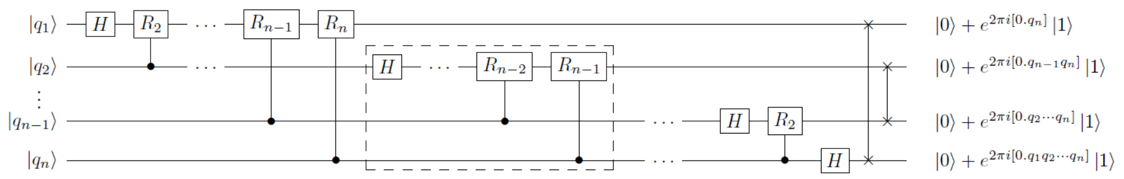

$|q_1\rangle$
$|q_2\rangle$
$|q_{n-1}\rangle$
$|q_n\rangle$
H
R_2
R_{n-1}
R_n
H
R_{n-2}
R_{n-1}
H
R_2
H
$|0\rangle + e^{2\pi i[0.q_n]}|1\rangle$
$|0\rangle + e^{2\pi i[0.q_{n-1}q_n]}|1\rangle$
$|0\rangle + e^{2\pi i[0.q_2\cdots q_n]}|1\rangle$
$|0\rangle + e^{2\pi i[0.q_1q_2\cdots q_n]}|1\rangle$

Figure 3.3. The gates inside the dashed rectangle cast the second qubit in the desired form.

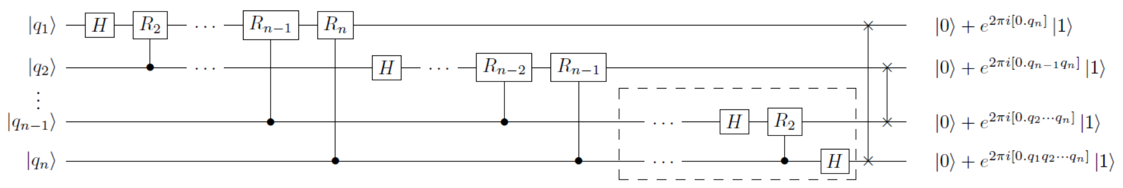
The Hadamard gate applied to $|q_2\rangle$ yields

$$|\psi\rangle = \frac{1}{4}\left(|0\rangle + e^{2\pi i[0.q_1q_2\cdots q_n]}|1\rangle\right)\left(|0\rangle + e^{2\pi i[0.q_2]}|1\rangle\right)|q_3\cdots q_n\rangle \tag{3.38}$$

After applying the R_2 gate to $|q_2\rangle$ controlled by $|q_3\rangle$ through the R_n gate controlled by $|q_n\rangle$, the second qubit is in the desired state as well,

$$|\psi\rangle = \frac{1}{4}\left(|0\rangle + e^{2\pi i[0.q_1q_2\cdots q_n]}|1\rangle\right)\left(|0\rangle + e^{2\pi i[0.q_2q_3\cdots q_n]}|1\rangle\right)|q_3\cdots q_n\rangle \tag{3.39}$$

The same procedure is used to manipulate $|q_3\rangle$ through $|q_{n-1}\rangle$, see figure 3.4.



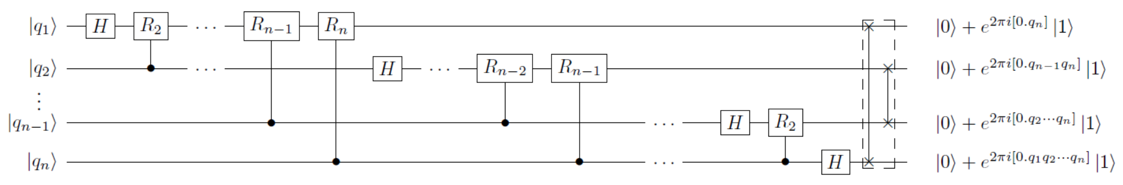
$|q_1\rangle$
$|q_2\rangle$
$|q_{n-1}\rangle$
$|q_n\rangle$
H
R_2
R_{n-1}
R_n
R_{n-2}
$|0\rangle + e^{2\pi i[0.q_n]}|1\rangle$
$|0\rangle + e^{2\pi i[0.q_{n-1}q_n]}|1\rangle$
$|0\rangle + e^{2\pi i[0.q_2\cdots q_n]}|1\rangle$
$|0\rangle + e^{2\pi i[0.q_1q_2\cdots q_n]}|1\rangle$

Figure 3.4. The gates inside the dashed rectangle cast the third through nth qubit in the desired form.

First, the Hadamard gate is applied to some qubit $|q_i\rangle$. Then the R_2 gate controlled by $|q_{i+1}\rangle$ through the R_{n+1-i} gate controlled by the $|q_n\rangle$ qubit are applied to the ith qubit. To $|q_n\rangle$ only the Hadamard gate is applied. The resulting quantum state is

$$|\psi\rangle = \frac{1}{\sqrt{N}}\left(|0\rangle + e^{2\pi i[0.q_1q_2\cdots q_n]}|1\rangle\right)\left(|0\rangle + e^{2\pi i[0.q_2q_3\cdots q_n]}|1\rangle\right)\cdots\left(|0\rangle + e^{2\pi i[0.q_n]}|1\rangle\right) \tag{3.40}$$

Note that compared to equation (3.21) this it is not quite the right output state. The qubits are in reverse order. In order to obtain the QFT of the quantum register $|q_1q_2\cdots q_n\rangle$, the SWAP gate must be applied to switch the position of $|q_1\rangle$ and $|q_n\rangle$, of $|q_2\rangle$ and $|q_{n-1}\rangle$ and so on (see figure 3.5).



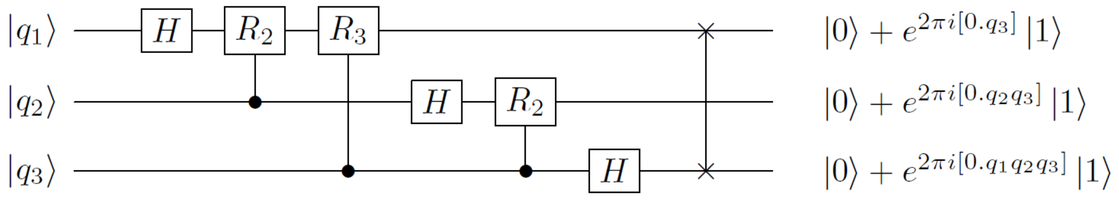
$|q_1\rangle$
$|q_2\rangle$
$|q_{n-1}\rangle$
$|q_n\rangle$
H
R_2
R_{n-1}
R_n
R_{n-2}
$|0\rangle + e^{2\pi i[0.q_n]}|1\rangle$
$|0\rangle + e^{2\pi i[0.q_{n-1}q_n]}|1\rangle$
$|0\rangle + e^{2\pi i[0.q_2\cdots q_n]}|1\rangle$
$|0\rangle + e^{2\pi i[0.q_1q_2\cdots q_n]}|1\rangle$

Figure 3.5. The SWAP gates inside the dashed rectangle reverse the order of the qubits.

Finally, equation (3.41) is the result of the QFT.

$$|\psi\rangle = \frac{1}{\sqrt{N}} \left(|0\rangle + e^{2\pi i[0.q_n]}|1\rangle\right)\left(|0\rangle + e^{2\pi i[0.q_{n-1}q_n]}|1\rangle\right)\cdots\left(|0\rangle + e^{2\pi i[0.q_1q_2\cdots q_n]}|1\rangle\right) \tag{3.41}$$

Using this general circuit, the circuit that applies the QFT to some quantum register can be obtained. For example, the QFT on three qubit is performed by the circuit in figure 3.6.



$|q_1\rangle$
$|q_2\rangle$
$|q_3\rangle$
H
R_2
R_3
H
R_2
H
$|0\rangle + e^{2\pi i[0.q_3]}|1\rangle$
$|0\rangle + e^{2\pi i[0.q_2q_3]}|1\rangle$
$|0\rangle + e^{2\pi i[0.q_1q_2q_3]}|1\rangle$

Figure 3.6. The quantum circuit that performs the QFT on three qubits.

If the QFT is used in a more extensive circuit, not all gates will be included in the schematic circuit. Else, the schematic will quickly become cluttered. Because the QFT is such a general routine, it will be represented in a circuit as in figure 3.7.

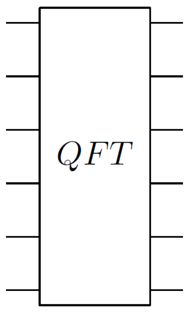


Figure 3.7. The representation for the QFT performed on all six qubits.

3.4 Implementation of the Inverse Quantum Fourier Transform

Similarly to the classical case, in order to transform a specific quantum register back from the Fourier domain an inverse QFT is needed. Because the QFT is (as all quantum gates) a unitary gate, the inverse QFT is obtained by reversing the order of all gates that constitute the QFT and taking their hermitian conjugate. It is easily seen that $H^\dagger = H$

and SWAP† = SWAP. On the other hand,

$$R_\varphi^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^\varphi} \end{bmatrix}^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & e^{-2\pi i/2^\varphi} \end{bmatrix} := R_{-\varphi} \tag{3.42}$$

The inverse QFT circuit is thus obtained by reversing the order of the gates and replacing R_φ by $R_{-\varphi}$. For example, the inverse circuit for the three qubit QFT from figure 3.6 is given in figure 3.8.

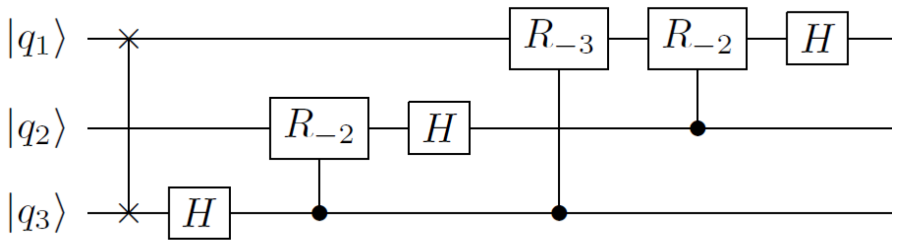


Figure 3.8. The quantum circuit that performs the inverse QFT on three qubits.

Just like the QFT, in a more extensive circuit the inverse QFT is represented as shown in figure 3.9.

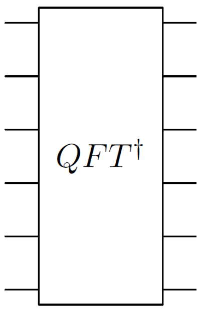


Figure 3.9. The representation for the inverse QFT on all six qubits.

3.4.1 Time Complexity

The quantum circuit to Fourier transform or inverse Fourier transform a register of n qubits uses a Hadamard gate and $n-1$ controlled phase shift gates on the first qubit, a Hadamard and $n-2$ controlled phase shift gates on the second qubit and so on, and only a Hadamard gate on the last qubit. The swaps in the end require a maximum of $\frac{n}{2}$ SWAP gates (if n is odd, the precise number is $\frac{n-1}{2}$). Therefore, the total number of gates used in the n qubit QFT is

$$n + (n-1) + \cdots + 1 + \frac{n}{2} = \frac{n(n+1)+n}{2} = \frac{n^2}{2} + n = \mathcal{O}(n^2) \tag{3.43}$$

It was shown that one of the most efficient classical algorithm, the Fast Fourier Transform, on the contrary, require $\mathcal{O}(n2^n)$ operations, which is exponentially more. The quantum Fourier transform thus accomplishes exponential speedup over the known classical algorithms, but with the shortcoming that it does not provide access to the Fourier coefficients as discussed before.

4 Quantum Addition

An application of the QFT is the addition of two numbers. The quantum circuit was first described by Draper (1998)[13]. In his paper, the quantum circuit as well as its action on the qubits are described briefly. In this section, the addition circuit and its action on the qubits are outlined in more detail. In addition, details on the allocation of the qubits are added supported by some examples. The addition circuit is based on the fact that the QFT of a quantum state can be described using complex exponentials (see equation (3.41)). In the quantum addition routine, the phase associated with these complex exponentials is increased depending on the number that is added. For the sake of clarity, only positive real numbers are considered to introduce the circuit. However, the representation of the numbers can be modified so that also negative numbers can be used and the inverse addition circuit (as one might expect) can be used to *subtract* one number from the other. This is a key part of Shor's algorithm (which is described in the next chapter).

First, the implementation of the quantum addition circuit is outlined in section 4.1. In section 4.2, the inverse of quantum addition, quantum subtraction, is described. These two quantum subroutines are combined to form a quantum adder/subtractor in section 4.3. Finally, the time complexity is broken down in section 4.4.

4.1 Implementation of Quantum Addition

Let $a, b \in \mathbb{N}$. The quantum register $|a\rangle = |a_1 a_2 \cdots a_n\rangle$ can be obtained by choosing the values of the qubits so that the register contains a in binary notation. Here, $|a_n\rangle$ and $|b_n\rangle$ are the least significant qubits of $|a\rangle$ and $|b\rangle$ respectively. If these would be the most significant qubits, the gates and controls in the circuit outlined below should be changed accordingly so that the circuit starts with applying controlled phase shifts to the least significant qubit. To accomplish this, the required number of qubits is $n = \lceil \log_2 a \rceil$. $|b\rangle = |b_1 b_2 \cdots b_m\rangle$ can be obtained in a similar way. This requires $m = \lceil \log_2 b \rceil$ qubits. For convenience, it is now assumed that $n = m$. Putting both quantum registers together, the total quantum register is

$$|\psi\rangle = |ba\rangle = |b_1 b_2 \cdots b_n a_1 a_2 \cdots a_n\rangle \tag{4.1}$$

The full quantum circuit for the addition circuit is given in figure 4.1[13].

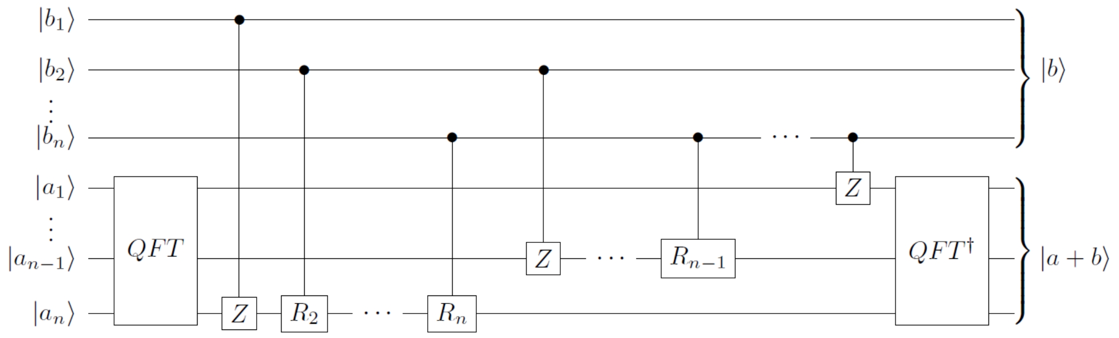


Figure 4.1. The quantum circuit that adds $|b_1b_2\cdots b_n\rangle$ to $|a_1a_2\cdots a_n\rangle$.

The addition circuit starts by applying the QFT to $|a\rangle$, see figure 4.2.

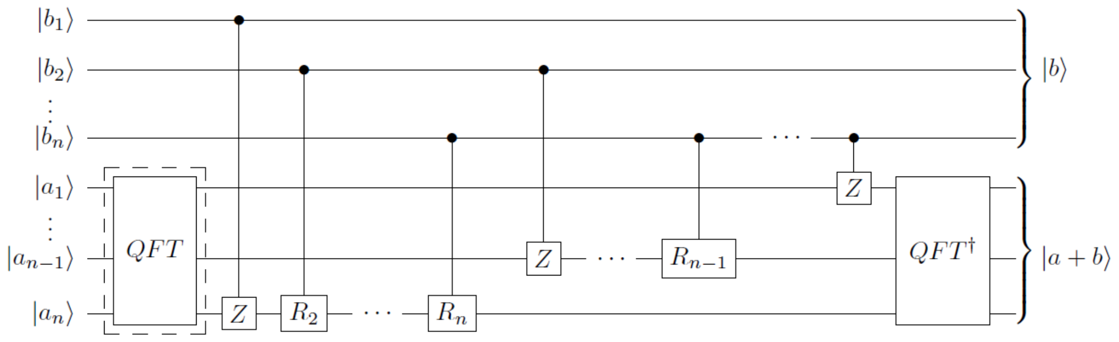


Figure 4.2. First, the QFT is applied to $|a\rangle$.

The QFT of $|a\rangle = |a_1a_2\cdots a_n\rangle$ is indicated by $|\varphi(a)\rangle = |\varphi_1(a)\varphi_2(a)\cdots\varphi_n(a)\rangle$. Thus, after the QFT the quantum state is

$$|\psi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle = |b_1b_2\cdots b_n\rangle \otimes |\varphi_1(a)\varphi_2(a)\cdots\varphi_n(a)\rangle \tag{4.2}$$

Because $|b\rangle$ is not affected at all throughout the quantum circuit, $|\psi_1\rangle$ will be omitted in the calculation. Using the notation from equation (3.41), the quantum state can then be denoted as

$$|\psi_2\rangle = \frac{1}{\sqrt{N}}\left(|0\rangle + e^{2\pi i[0.a_n]}|1\rangle\right)\cdots\left(|0\rangle + e^{2\pi i[0.a_2a_3\cdots a_n]}|1\rangle\right)\left(|0\rangle + e^{2\pi i[0.a_1a_2\cdots a_n]}|1\rangle\right) \tag{4.3}$$

In order to add a phase to the complex exponentials in $|\varphi(a)\rangle$ depending on $|b\rangle$, a quantum circuit similar to the QFT is used. But the circuit involves only a number of controlled phase shifts gates, and no Hadamard gates. The circuit begins by manipulating the last qubit of $|a\rangle$, see figure 4.3.

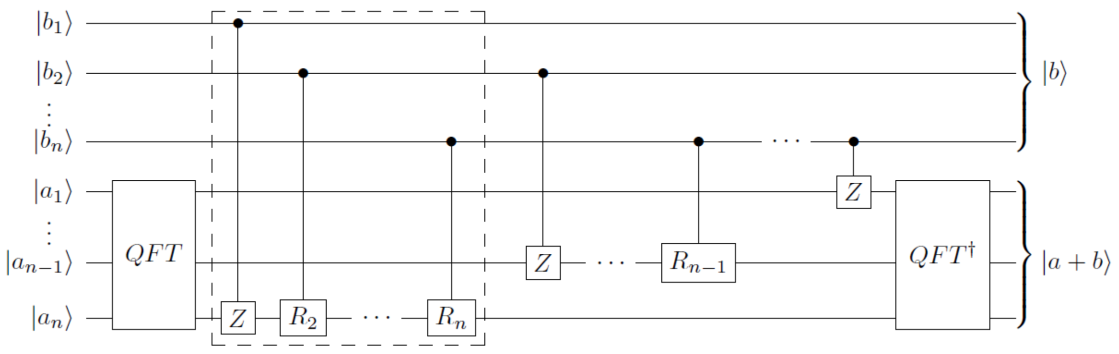


Figure 4.3. The gates inside the dashed rectangle add $|b\rangle$ to $|\varphi_n(a)\rangle$.

First, the R_1 gate controlled by $|b_1\rangle$ is applied to $|\varphi_n(a)\rangle$. So if $|b_1\rangle = |0\rangle$, nothing happens. If $|b_1\rangle = |1\rangle$, on the other hand, a rotation is added to $|\varphi_n(a)\rangle$. Continuing this way, the R_2 gate controlled by $|b_2\rangle$ up till the R_n gate controlled by $|b_n\rangle$ are applied to $|\varphi_n(a)\rangle$, resulting in

$$|\psi_2\rangle = \frac{1}{\sqrt{N}}\left(|0\rangle + e^{2\pi i[0.a_n]}|1\rangle\right)\cdots\left(|0\rangle + e^{2\pi i[0.a_2a_3\cdots a_n]}|1\rangle\right)\left(|0\rangle + e^{2\pi i[0.a_1a_2\ldots a_n+0.b_1b_2\ldots b_n]}|1\rangle\right) \tag{4.4}$$

So $|\varphi_n(a)\rangle$ is transformed into $|\varphi_n(a+b)\rangle$. Then, the R_1 gate (more commonly known as the Z gate) up to the R_{n-1} gate, controlled by $|b_2\rangle$ through $|b_n\rangle$, respectively) are applied to $|\varphi_{n-1}(a)\rangle$, see figure 4.4.

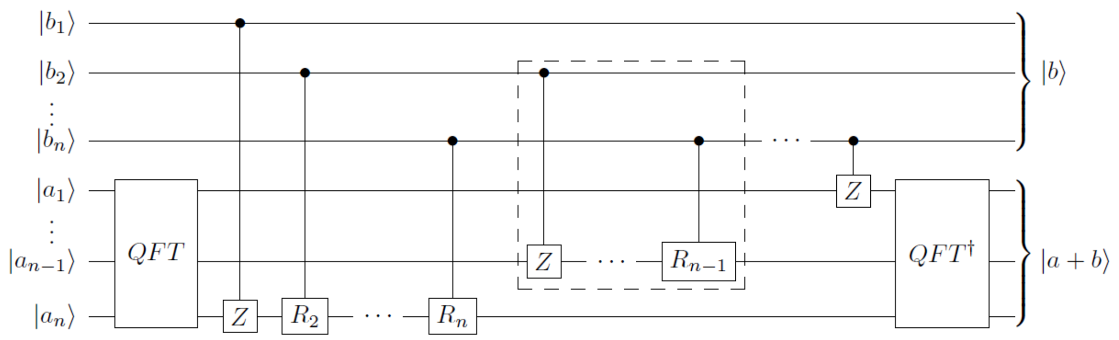


Figure 4.4. The gates inside the dashed rectangle add $|b\rangle$ to $|\varphi_{n-1}(a)\rangle$.

This results in

$$|\psi_2\rangle = \frac{1}{\sqrt{N}} \left(|0\rangle + e^{2\pi i[0.a_n]}|1\rangle\right) \cdots \left(|0\rangle + e^{2\pi i[0.a_2a_3\cdots a_n+0.b_2b_3\cdots b_n]}|1\rangle\right) \\ \left(|0\rangle + e^{2\pi i[0.a_1a_2\cdots a_n+0.b_1b_2\cdots b_n]}|1\rangle\right) \tag{4.5}$$

So $|\varphi_{n-1}(a)\rangle$ is transformed to $|\varphi_{n-1}(a+b)\rangle$ similarly to $|\varphi_n(a)\rangle$. Consecutively, to the ith qubit, the R_1 gate up to the R_i gate, controlled by $|b_{1-i}\rangle$ through $|b_n\rangle$ respectively, are applied to $|\varphi_i(a)\rangle$ transforming it to $|\varphi_i(a+b)\rangle$. This procedure is repeated up to $|\varphi_1(a)\rangle$, to which only the R_1 gate (Z gate) controlled by $|b_n\rangle$ is applied, see figure 4.5.

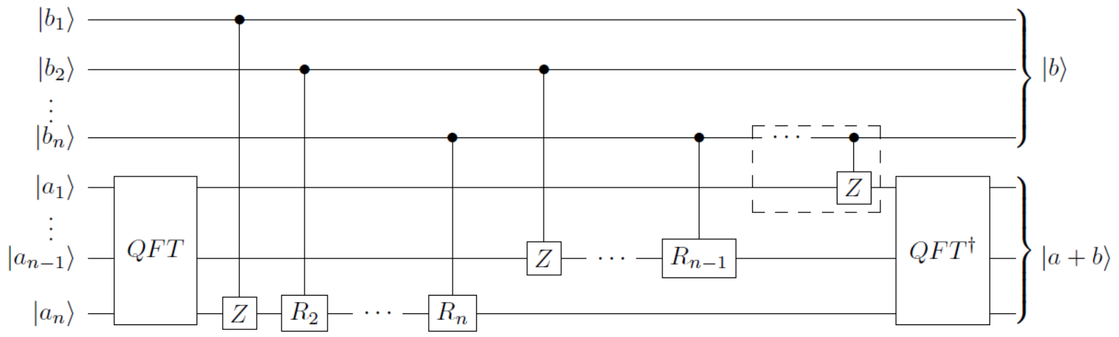


Figure 4.5. The gates inside the dashed rectangle add $|b\rangle$ to $|\varphi_{n-2}(a)\rangle$ through $|\varphi_1(a)\rangle$.

The result is

$$|\psi_2\rangle = \frac{1}{\sqrt{N}} \left(|0\rangle + e^{2\pi i[0.a_n+0.b_n]}|1\rangle\right) \cdots \left(|0\rangle + e^{2\pi i[0.a_2a_3\cdots a_n+0.b_2b_3\cdots b_n]}|1\rangle\right) \\ \left(|0\rangle + e^{2\pi i[0.a_1a_2\cdots a_n+0.b_1b_2\cdots b_n]}|1\rangle\right) \tag{4.6}$$

or, equivalently, $|\psi_2\rangle = |\varphi(a+b)\rangle$. Finally, the inverse QFT is applied to $|\varphi(a+b)\rangle$, see figure 4.6.

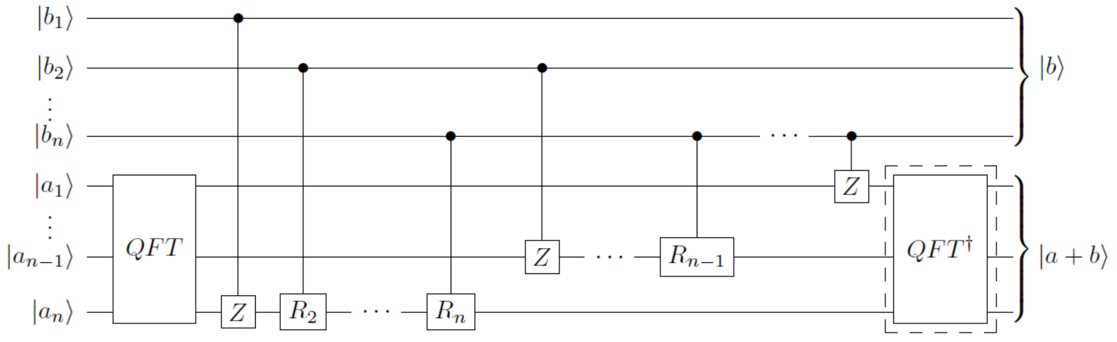


Figure 4.6. Finally, the inverse QFT is applied to $|\varphi(a+b)\rangle$.

By applying the inverse QFT to this register, the binary representation of $|a+b\rangle$ is obtained from which the outcome of the sum can be determined[13].

There is one thing that one should pay attention to when adding two numbers on a quantum computer. If the result of the addition is larger than $2^n - 1$, there are not enough bits to store the outcome. Consequently, the register will overflow and the result obtained is $a + b - 2^n$ because the 'actual' most significant qubit is forgotten. The action on the qubits that are in the register will thus not be influenced. For example, if $|b\rangle = |101\rangle$ is added to $|a\rangle = |110\rangle$, the actual result is $|1011\rangle$. But since only three qubits were used to store $|a\rangle$, the final quantum register is $|a\rangle = |011\rangle$. As a result, the outcome is 3 instead of $a + b = 6 + 5 = 11$. In order to prevent this from occurring, one should make sure that the most significant qubit of both registers $|a\rangle$ and $|b\rangle$ is always $|0\rangle$, so a and b must be smaller than 2^{n-1}.

If less qubits are required to store $|b\rangle$ than $|a\rangle$, the most significant qubit(s) of $|b\rangle$ are $|0\rangle$. Since $|b\rangle$ does nothing else than controlling the rotation gates that are applied to $|a\rangle$, nothing happens when $|b_i\rangle = |0\rangle$. Since qubits $|b_i\rangle$ with $i > \lceil \log_2 b \rceil$ are always $|0\rangle$, these qubits as well as the gates that are controlled by them can just as well be left out to minimize the number of qubits and to optimize the number of operations.

to Reduce # of qubits.

In addition, there is one more way to minimize the number of qubits that is necessary for quantum addition. Because the qubits in $|b\rangle$ are in the same deterministic state throughout the whole circuit and since they are only used to control rotation gates, the register to store b could just as well be a classical bit register.

Finally, some of the operations can be run in parallel[13]. In contrast to the QFT, the gates in quantum addition do not need to be in a particular order. The reason for this is that unlike the QFT, in which the Hadamard gate and rotation gates do not commute (see equation (4.7)), rotation gates do commute with each other (see equation (4.8)). Besides, the (qu)bits in $|b\rangle$ do not change in the process, so the order of the controls can as well be interchanged according to the changes in the order of the rotation gates.

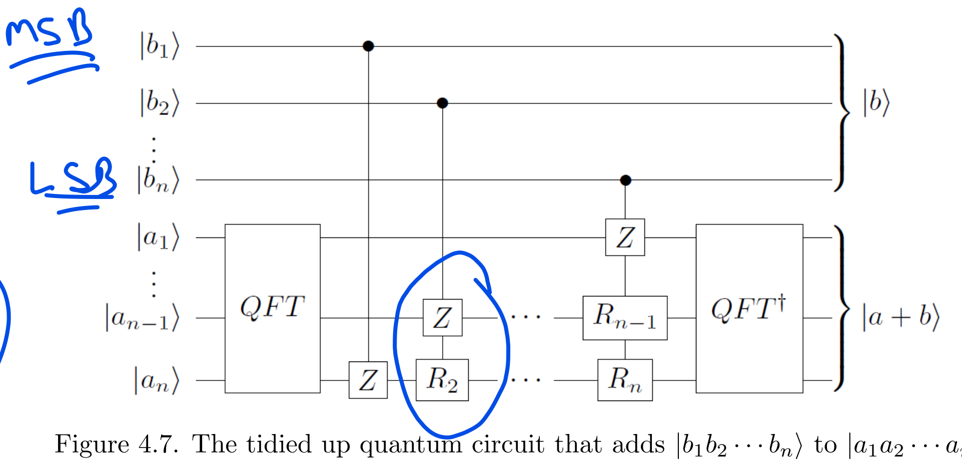
$$[H, R_x] = HR_x - R_xH\frac{1}{\sqrt{2}}\begin{pmatrix}1 & e^{2\pi i/2x}\\ 1 & -e^{2\pi i/2x}\end{pmatrix} - \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ e^{2\pi i/2x} & -e^{2\pi i/2x}\end{pmatrix} \neq \mathbf{0_{2,2}} \quad (4.7)$$

$|b_1\rangle \otimes \dots |b_n\rangle$

$$[R_x, R_y] = R_xR_y - R_yR_x\begin{pmatrix}1 & 0\\ 0 & e^{2\pi i/2(x+y)}\end{pmatrix} - \begin{pmatrix}1 & 0\\ 0 & e^{2\pi i/2(x+y)}\end{pmatrix} = \mathbf{0_{2,2}} \quad (4.8)$$

With this in mind, the quantum circuit from figure 4.1 could be tidied up. This results in the quantum circuit in figure 4.7 that adds two arbitrary n-qubit registers.

In this ckt



$(Z_{n-1} \otimes R_{n,n})$

Figure 4.7. The tidied up quantum circuit that adds $|b_1b_2\cdots b_n\rangle$ to $|a_1a_2\cdots a_n\rangle$.

The short-hand notation for quantum addition is given in figure 4.8.

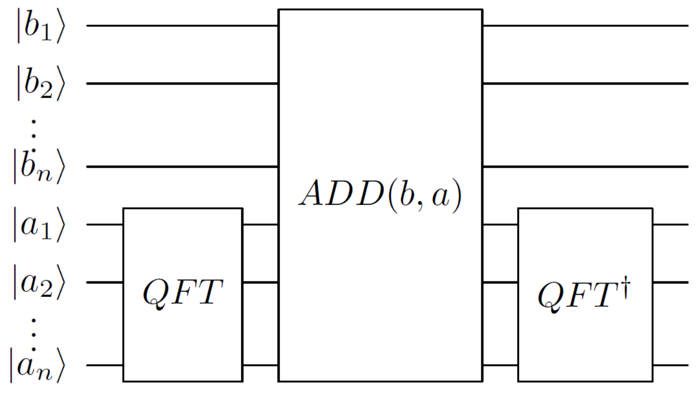
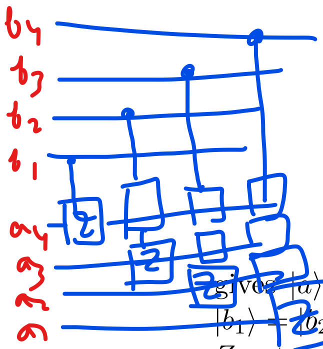


Figure 4.8. The addition gate that adds b to a.

To illustrate the quantum addition routine outlined in this section, consider $a = 6$ and $b = 3$. Then $n = 4$ in order to satisfy $a, b < 2^{n-1}$, so no overflow can occur. $a = 6$



∵ here Rot^n gates indexing starts from 1.

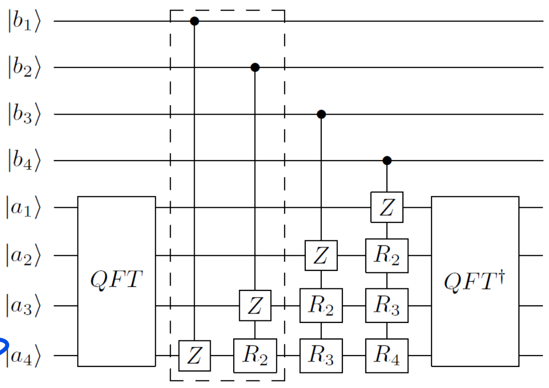
gives $|a\rangle = |a_1a_2a_3a_4\rangle = |0110\rangle$ and $b = 3$ gives $|b\rangle = |b_1b_2b_3b_4\rangle = |0011\rangle$. Because $|b_1\rangle = |b_2\rangle = 0$, $|b\rangle$ can just as well be represented using only two qubits. Therefore, the Z gate on $|a_3\rangle$ and $|a_4\rangle$ as well as the R_2 gate on $|a_4\rangle$ could be left out. These are the gates inside the dashed rectangle in the quantum circuit given in figure 4.9.

LSB MSB $\rightarrow$

MSB LSB $\rightarrow$

LSB MSB $\rightarrow$

MSB LSB $\rightarrow$



$$R_l = \begin{pmatrix} 1 & 0 \\ 0 & e^{j\frac{2\pi}{2^l}} \end{pmatrix}$$

$$\therefore Z = \begin{pmatrix} 1 & 0 \\ 0 & e^{j\frac{2\pi}{2}} \end{pmatrix}.$$

R_2

Figure 4.9. The quantum circuit that adds two 4-qubit registers. The gates inside the dashed rectangle could be left out since $|b\rangle$ requires only two qubits.

The most significant qubit in both registers is $|0\rangle$, so no overflow will occur. The circuit leaves $|b\rangle$ unchanged. On the other hand, $|b\rangle$ is added to $|a\rangle$ resulting in $|1001\rangle$. Indeed, this is the correct result: 9.

4.2 Quantum Subtraction

Furthermore, the quantum addition circuit can also be used to *subtract* two numbers. The implementation is actually quite straightforward. Namely, the circuit that does this is the inverse of the quantum addition circuit (also referred to as the subtraction circuit). This is mentioned briefly by Beauregard (2003)[2], without further elaboration. In the addition circuit, first the QFT is applied to $|a\rangle$. Then, the rotation gates are applied to $|\varphi(a)\rangle$ controlled by $|b\rangle$ and finally the inverse QFT is applied to $|\varphi(a+b)\rangle$. By reversing the order of the gates and taking their Hermitian conjugate, the subtraction circuit can be obtained. Hence, the subtraction circuit also starts with the application of the QFT to $|a\rangle$. From equation (3.42), it follows that the Hermitian conjugate of a rotation gate R_k is the R_{-k} gate. It was also shown in section 1.2 that $Z^\dagger = Z$. Because all rotation gates commute, the order of these gates is not of interest, so the order of the controlled rotation gate does not need to be changed. Finally, the inverse QFT is applied to $|\varphi(a-b)\rangle$. See figure 4.10 for a schematic of the subtraction quantum circuit.

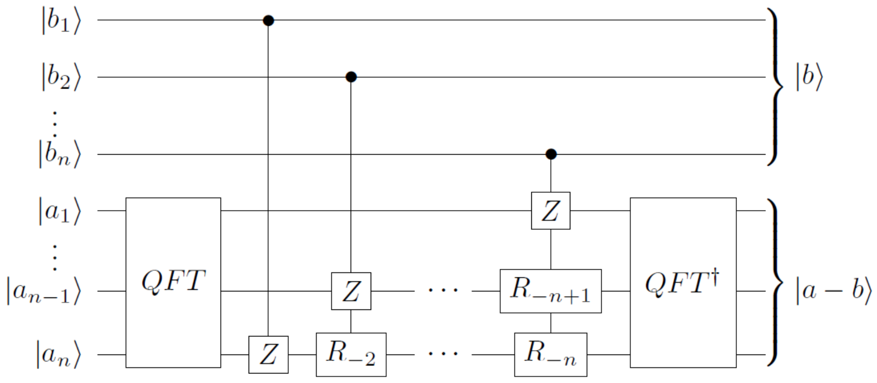


Figure 4.10. The quantum circuit that subtracts $|b\rangle$ from $|a\rangle$.

Similarly to the addition gate, the subtraction gate from figure 4.11 is used in more extensive circuits.

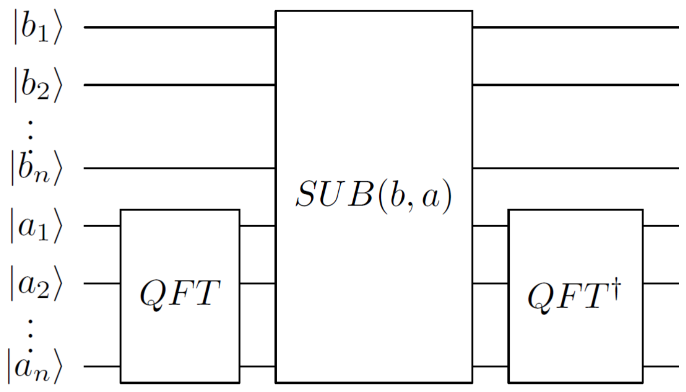


Figure 4.11. The subtraction gate that subtracts b from a.

Again, let $|a\rangle$ be the quantum register of interest and let $|b\rangle$ (which could also be a classical register) be the number we wish to subtract from $|a\rangle$. The result of the subtraction circuit can be derived from our previous calculations on the addition circuit.

Interchanging the controlled R_k gates by R_{-k} gates indeed yields the desired result, because the plus signs in equation (4.6) are replaced by minus signs. The quantum state just before the inverse QFT is given in the equation below.

$$|\psi_2\rangle = \frac{1}{\sqrt{N}}\left(|0\rangle + e^{2\pi i[0.a_n - 0.b_n]}|1\rangle\right)\cdots\left(|0\rangle + e^{2\pi i[0.a_2a_3\cdots a_n - b_2b_3\cdots b_n]}|1\rangle\right)$$
$$\left(|0\rangle + e^{2\pi i[0.a_1a_2\cdots a_n - 0.b_1b_2\cdots b_n]}|1\rangle\right) \tag{4.9}$$

This is equivalent to $|\psi_2\rangle = |\varphi(a-b)\rangle$. It is easily seen that everything works out as expected as long as $a \geq b$, because $a - b \geq 0$ in that case. However, if $a < b$ the result is not $a - b$. This case is very similar to the overflow that can occur in quantum addition. Because a 'negative' binary number does not exist in this context (later it will be shown that it is possible to work with this), the quantum register will cycle as if $|111\rangle$ is below $|000\rangle$. The result from quantum subtraction can thus be summarized as follows[2].

$$`a - b' = \begin{cases} a - b & \text{if } a \geq b \\ 2^n - (b - a) & \text{if } a < b \end{cases} \tag{4.10}$$

4.3 Quantum Adder/Subtractor

By adding a convention to the binary notation, both positive and negative numbers can be added and subtracted. If the most significant qubit of some binary state is $|0\rangle$, the number is positive. On the other hand, if the most significant qubit is $|1\rangle$, the number is negative. The conversion is given by the following equation.

$$x_1 x_2 \cdots x_n = \begin{cases} \sum_{k=2}^{n} x_k 2^{n-k} & \text{if } x_1 = 0 \\ -2^{n-1} + \sum_{k=2}^{n} x_k 2^{n-k} & \text{if } x_1 = 1 \end{cases} \tag{4.11}$$

So both $|a_1\rangle$ and $|b_1\rangle$ are used to store the sign of $|a\rangle$ and $|b\rangle$ respectively. In table 4.1, the meaning of all three bit states is outlined.

Table 4.1. The conversion of all three bit states.

000	0	100	-4
001	1	101	-3
010	2	110	-2
011	3	111	-1

As stated earlier, the qubit state 'cycles' through the binary states. Hence after $|111\rangle$ comes $|000\rangle$, so if $|001\rangle$ is added to $|111\rangle$, this results in $|000\rangle$ $(-1 + 1 = 0)$ as expected. Similarly, if $|001\rangle$ is subtracted from $|000\rangle$ using the subtraction circuit the result is $|111\rangle$ $(0 - 1 = -1)$ as expected. So the transition from positive numbers to negative numbers works out well. Also, when some number is added to a negative number the number always increases as it should.

When the circuit is implemented, similar to the addition circuit, the second (now most significant) qubit of both registers should be $|0\rangle$ to rule out the possibility of overflow. The number b that is added to some number a can be positive or negative. If $|b_1\rangle = |0\rangle$, b is positive, the addition circuit can be applied to obtain the result. $|b_1\rangle$ need not be included because it is $|0\rangle$ anyway, so if $|b_1\rangle = |0\rangle$ the addition circuit should be applied to add $|b_2 \cdots b_n\rangle$ to $|a_1 \cdots a_n\rangle$.

If $|b_1\rangle = |1\rangle$, b is negative. Adding a negative number is equivalent to subtracting the absolute value. Therefore, the subtraction circuit can be used to subtract $-b$ from a. Since the first qubit of $|-b\rangle$ is $|0\rangle$, it need not be included in the subtraction, so if $|b_1\rangle = |1\rangle$ the subtraction circuit should be applied to subtract $|b_2 \cdots b_n\rangle$ from $|a_1 \cdots a_n\rangle$.

In order to design a quantum circuit that can add any two integers smaller than 2^{n-2}, $|b_1\rangle$ (the qubit that represents the sign of $|b\rangle$) can be used as a control qubit for the subtraction circuit or as *anti*-control qubit for the addition circuit. Anti-control means that a gate is executed if and only if the control qubit is $|0\rangle$. In line with this analysis, the circuit of the quantum adder/subtractor is given by figure 4.12. This circuits adds $|b\rangle$ to $|a\rangle$ if $|b_1\rangle = |0\rangle$ and subtracts $|-b\rangle$ from $|a\rangle$ if $|b_1\rangle = |1\rangle$.

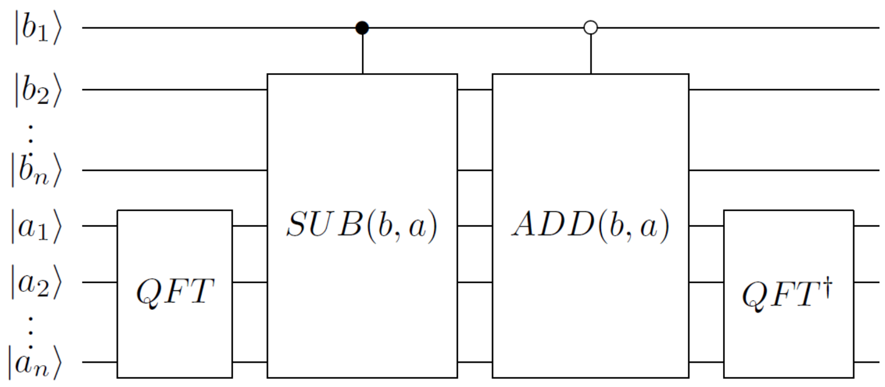


Figure 4.12. The quantum adder/subtractor

In a similar way, fractional numbers can be added or subtracted. By defining

$$x_1 x_2 \cdots x_n = \begin{cases} \sum_{k=2}^{n} x_k 2^{-k+1} & \text{if } x_1 = 0 \\ -1 + \sum_{k=2}^{n} x_k 2^{-k+1} & \text{if } x_1 = 1 \end{cases} \tag{4.12}$$

any fractional number x with $|x| < 1$ can be approximated. The maximum error in the approximation is 2^{-n+1}, which decreases with the number of qubits. Because the errors are additive, the error in the result can reach $2 \cdot 2^{-n+1} = 2^{-n+2}$. Thus, the error decreases exponentially with the number of qubits.

In order to add numbers that are larger than one but also have a fractional part, a certain number of qubits can be used to store the decimal part and the remaining qubits to store the fractional part. Say the decimal part of some number is stored by $i-1$ bits (the first qubit is used to store the sign). The remaining $n-i$ bits are used to store the fractional part. The conversion between the binary number and the real number is then given by

$$x_1 x_2 \cdots x_n = \begin{cases} \sum_{k=2}^{i} x_k 2^{i-k} + \sum_{k=i+1}^{n} x_k 2^{-k+i} & \text{if } x_1 = 0 \\ -2^{i-1} + \sum_{k=2}^{i} x_k 2^{i-k} - 1 + \sum_{k=i+1}^{n} x_k 2^{-k+i} & \text{if } x_1 = 1 \end{cases} \tag{4.13}$$

The maximum error in the result is then given by $2 \cdot 2^{i-n+1} = 2^{i-n+2}$. However, the error relative to the maximum number that can be stored is only $\frac{2^{i-n+2}}{2^i} = 2^{-n+2}$, which equals the maximum error for adding only fractional parts using n qubits. Using this approximation, two real numbers can be added or subtracted on a quantum computer with reasonable accuracy using the quantum circuit outlined in figure 4.12.

4.4 Time Complexity

Both the quantum addition and the quantum subtraction circuit use the QFT and the inverse QFT as well as $1+2+\cdots+n = \frac{n(n+1)}{2}$ controlled phase shift gates. In section 3.4.1, it was shown that the QFT (and thus the inverse QFT as well) requires $\mathcal{O}(n^2)$ operations. The controlled phase shift gates also constitute $\mathcal{O}(n^2)$ operations. However, the latter can be parallelized, hence their time complexity reduces to only $\mathcal{O}(n)$. Consequently, the overall time complexity of the addition circuit is $\mathcal{O}(n^2)$.

5 Shor's Algorithm

More than 30,000 customers worldwide use RSA encryption to protect their most valuable assets from cyber threats. Among these customers are many financial institutions and listed multinationals[24]. RSA encryption is public key cryptosystem based on multiplying large prime numbers. Classical computers are really inefficient reversing this process, namely in factoring large prime multiples: breaking a 2048-bit key would take a classical computer longer than the age of the universe.

However, the rise of quantum computing may threaten the safety of RSA encryption. Using the quantum algorithm designed by Peter Shor in 1996, breaking RSA encryption comes within reach. Shor reduced the problem of factoring large numbers to period finding by applying some smart tricks. Period finding can be done in an efficient way with a quantum computer, resulting in an exponential speedup compared to the best classical algorithm for factoring large numbers.

In short, Shor's algorithm goes through the following steps to factorize a (large) integer N[27].

1. If N is even, the number 2 is a factor of N. Done.
2. First, a classical algorithm is used to check if N is a prime multiple;
3. If $N = p^k$ with p prime, p is a factor of N. Done.
4. Choose $a \in \{2, \dots, N-1\}$ randomly;
5. If the greatest common divisor of a and N, $\gcd(a, N) \neq 1$, a is a factor of N. Done.
6. The quantum routine for period finding is used to find the period r of $a^x \mod N$;
7. If r is odd or $a^{r/2} + 1 = 0 \mod N$, go to 5;
8. Compute $\gcd(a^{r/2} + 1, N)$ and $\gcd(a^{r/2} - 1, N)$. These are two factors of N. Done.

These steps are described in more detail in section 5.1 and an example for factoring the number 35 is shown in section 5.3. The quantum routine for period finding described by Shor (1996)[31] is explained in section 5.2. Finally, the time complexity of Shor's algorithm is analysed in section 5.4.

5.1 Integer Factorization

Period finding is based on modular arithmetic.

Definition 5.1. *Let $x, y \in \mathbb{N}$. Let $n \in \mathbb{N}$ be the largest multiple of x such that $nx \leq y$ and let $z = y - nx$. Then $y \equiv z \mod x$ and $0 \leq z < x$.*

The set of all remainders from division by N is called a *multiplicative ring* in algebra. The problem of period finding is to find the smallest integer r with $a^r = 1 \mod N$.

This r is called the period of $a^x \mod N$, because the outcomes of $a^x \mod N$ repeat themselves with this period.

For example, consider $a = 3$ and $N = 5$. The values of 3^x and $3^x \mod 5$ are shown in table 5.1.

Table 5.1. Values of a^x and $a^x \mod N$ with $a = 3$ and $N = 5$.

x	3^x	$3^x \mod 5$
0	1	1
1	3	3
2	9	4
3	27	2
4	81	1
5	243	3
6	729	4
7	2187	2
8	6561	1

In order to calculate the last column, it is not necessary to calculate 3^x. Because the set of remainders from division by 5 is multiplicative ring, if $3^3 \equiv 2 \mod 5$ is known, $3^4 \equiv 3 \cdot 3^3 \mod 5$ can be found as $3 \cdot 2 \equiv 6 \equiv 1 \mod 5$. The smallest integer r for which $3^x = 1 \mod 5$ is 4, so $3^x \mod 5$ has period 4.

Let a and x be natural numbers, and let $N = pq$ in which p and q are prime numbers. Because the remainder of division by N is a positive number smaller than N, $a^x \mod N$ only has N different outcomes. Therefore, there exists a natural number $y > x$, such that $a^y \equiv a^x \mod N$.

If $\gcd(a, N) = 1$, then also $\gcd(a^x, N) = 1$. Therefore the inverse of a^x, denoted a^{-x}, is a member of the multiplicative ring. Consequently, both sides can be multiplied by a^{-x} resulting in $a^{y-x} \equiv 1 \mod N$. But then, $a^{y-x} - 1 \equiv 0 \mod N$, so $a^{y-x} - 1 := a^r - 1$ is a multiple of N.

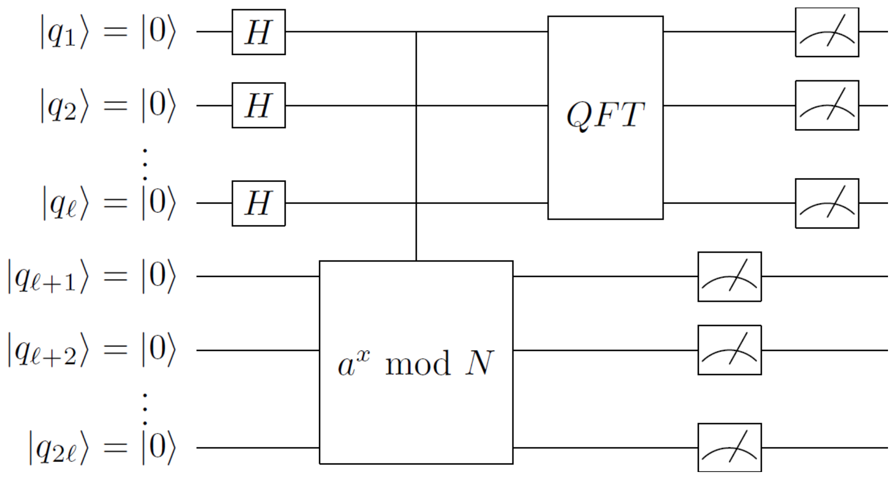
Assume r is even and $a^{r/2}+1 \not\equiv 0 \mod N$. $a^r - 1$ can be factored as $\left(a^{r/2} - 1\right)\left(a^{r/2} + 1\right)$. Since this is a multiple of N, there exists a natural number k such that $\left(a^{r/2} - 1\right)\left(a^{r/2} + 1\right) = kN$. Besides, $a^{r/2} - 1 \not\equiv 0 \mod N$, otherwise the period would not be r but $\frac{r}{2}$. Therefore, both $a^{r/2} + 1$ and $a^{r/2} - 1$ are not a multiple of N, so both $a^{r/2} + 1$ and $a^{r/2} - 1$ must have non-trivial common divisors. The factors of N can thus be found by calculating $\gcd(a^{r/2} + 1, N)$ and $\gcd(a^{r/2} - 1, N)$[27].

5.2 Quantum Routine for Period Finding

Peter Shor (1997)[31] has described the following quantum routine for period finding. Let a and N be natural numbers. The goal is to find the smallest integer r such that $a^r = 1$

mod N. Let $w = 2^\ell$ such that $N^2 \leq w < 2N^2$. Two registers of ℓ qubits, $|q_1 \cdots q_\ell\rangle$ and $|q_{\ell+1} \cdots q_{2\ell}\rangle$, are initialized to $|0\rangle^{\otimes \ell}$.

The full quantum circuit for period finding is shown in figure 5.1. Note that this is a high-level circuit. In particular, the $a^x \mod N$ gate can be implemented in a quantum circuit using modular exponentiation. The addition circuit described in section 4 is at the core of modular exponentiation (see [2]). The vertical wire connected to the $a^x \mod N$ gate is used to indicate the dependence of this gate on the first register $|q_1 \cdots q_\ell\rangle$.



$|q_1\rangle = |0\rangle$
$|q_2\rangle = |0\rangle$
$|q_\ell\rangle = |0\rangle$
$|q_{\ell+1}\rangle = |0\rangle$
$|q_{\ell+2}\rangle = |0\rangle$
$|q_{2\ell}\rangle = |0\rangle$
H
QFT
$a^x \mod N$

Figure 5.1. The quantum circuit for period finding.

Then, the Hadamard gate is applied to the first register of qubits, resulting in

$$|\psi\rangle = \left(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right)^{\otimes \ell} \otimes |0\rangle^{\otimes \ell} \tag{5.1}$$

The first register now is in a superposition of all ℓ-qubit states. This can be interpreted as all integer numbers x between 0 and $w-1$ in binary notation,

$$|\psi\rangle = \frac{1}{\sqrt{w}} \sum_{x=0}^{w-1} |x\rangle\, |0\rangle^{\otimes \ell} \tag{5.2}$$

The next step is to apply some unitary operator that implements the function we want to find the period of. Thus, in this case the unitary operator that manipulates the qubits in the second register to $a^x \mod N$. This can be done using quantum modular exponentiation, for example as described in Van Meter (2005)[25]. After applying this unitary circuit, the quantum state is

$$|\psi\rangle = \frac{1}{\sqrt{w}} \sum_{x=0}^{w-1} |x\rangle \, |a^x \mod N\rangle \tag{5.3}$$

Afterwards, the QFT (see section 3) is applied to the first register, resulting in

$$F_w \, |x\rangle = \frac{1}{\sqrt{w}} \sum_{y=0}^{w-1} e^{2\pi ixy/w} \, |y\rangle \tag{5.4}$$

The resulting quantum state is

$$|\psi\rangle = \frac{1}{w} \sum_{x=0}^{w-1} \sum_{y=0}^{w-1} e^{2\pi ixy/w} \, |y\rangle \, |a^x \mod N\rangle \tag{5.5}$$

in which y is, like x, an integer number that is represented in binary notation.

A measurement is performed on the qubits in the second quantum register. Consequently the state collapses to the states in which $a^x \mod N = z$ holds for some fixed value of z.

$$|\psi\rangle = \frac{1}{w} \sum_{x=0}^{w-1} \sum_{y=0}^{w-1} |y\rangle \, |z\rangle \sum_{x:f(x)=z} e^{2\pi ixy/w} \tag{5.6}$$

Let x_0 be the smallest x such that $f(x) = z$ and let k be the index of the x such that $a^x = z \mod N$ (ordered from small to large). Then the possible values for x are (since $a^x \mod N$ is periodic with period r) $x_0 + rk$, $k = 0, \ldots, \lfloor \frac{w-x_0-1}{r} \rfloor$. Therefore,

$$\sum_{x:f(x)=z} e^{2\pi ixy/w} = \sum_{k} e^{2\pi i(x_0+rk)y/w} \tag{5.7}$$

Using this notation, the final quantum state can be written as

$$|\psi\rangle = \frac{1}{w} \sum_{x=0}^{w-1} \sum_{y=0}^{w-1} |y\rangle \, |z\rangle \, e^{2\pi i(x_0+rk)y/w} \tag{5.8}$$

It remains to perform a measurement on the first register. Since $a^x \mod N$ is a periodic function, the probability of obtaining some outcome $|y\rangle \, |z\rangle$ is given by

$$P(y|z) = \left| \frac{1}{w} \sum_{k} e^{2\pi i(x_0+rk)y/w} \right|^2 \tag{5.9}$$

This probability is highest if $\frac{yr}{w}$ is close to an integer number. Intuitively, this is because the values almost have the same direction in the complex plane. If this is not the case, some factors will cancel against others resulting in a lower probability amplitude. Moreover, if r is not a power of two, $\frac{yr}{w}$ is not a factor of w. Suppose $\frac{yr}{w}$ is close to an integer c. Then, $\frac{y}{w}$ is close to $\frac{c}{r}$. Continued fraction expansion (using a classical algorithm) on $\frac{y}{q}$ leads to an approximation for $\frac{c}{r}$, denoted as $\frac{d}{s}$. If $s < N$ and $\left|\frac{y}{w} - \frac{d}{s}\right| < \frac{1}{2w}$, then s is a very good approximation for the period r[31].

5.3 Factoring Numbers: worked example

Let's say we have some natural number N that we want to factor. If N is even, we have found a factor of N (namely 2) and we are done. Also, there are good classical algorithms to check whether N is a prime or prime power. So in that case, a factor of N can be easily obtained.

However, if N is not even nor a prime power, classical algorithms are very inefficient in finding a factor. In that case, the quantum algorithm for factorization offers a solution. Choose some $a \in \{2, \ldots, N-1\}$ and determine $\gcd(a, N)$. This can also be done using a classical algorithm. If $\gcd(a, N) \neq 1$, we have found a factor of N and we are done. The chance of obtaining a factor in this way is astronomically low if N is some large prime multiple, so in general $\gcd(a, N) = 1$.

The procedure described in the previous section is applied to obtain the period r of $a^x \mod N$. If a period r is found that is even and such that $a^{r/2} + 1 \neq 0 \mod N$, the factor of N is given by $\gcd(a^{r/2} + 1, N)$ as prove there. If r does not satisfy the requirements, some other a should be chosen. This is repeated until some r is found that does satisfy the requirements. Since it is known that r is not a prime number (this was checked in the beginning of the algorithm), it is guaranteed that there exists some a such that the period satisfies the requirements and a factor of N will always be found.

As an example, consider $N = 35 = 7 \cdot 5$. Choosing $a = 9$. $\gcd 9, 35 = 1$, so the period algorithm can be applied to find a factor of N. Then, $w = 2^\ell$ is chosen such that $1225 \leq q < 2450$, or $w = 2^{11} = 2048$.

The values of $9^x \mod N$ have been calculated for the purpose of the algorithm. In table 5.2, the values are set out for $0 \leq x \leq 8$.

Table 5.2. The values of $9^x \mod N$ for $0 \leq x \leq 8$.

x	0	1	2	3	4	5	6	7	8	...
$9^x \mod N$	1	9	11	29	16	4	1	9	11	...

Therefore, the quantum state after applying the Hadamards to the first part of the register and manipulating the second register to $9^x \mod N$ is

$$|\psi\rangle = \frac{1}{\sqrt{2048}} \sum_{x=0}^{2047} |x\rangle |9^x \mod 35\rangle \tag{5.10}$$

$$= \frac{1}{\sqrt{2048}}(|0\rangle |1\rangle + |1\rangle |9\rangle + |2\rangle |11\rangle + |3\rangle |29\rangle + |4\rangle |16\rangle + |5\rangle |4\rangle + |6\rangle |1\rangle + |7\rangle |9\rangle + |8\rangle |11\rangle + \ldots + |2047\rangle |9\rangle) \tag{5.11}$$

And after applying the QFT,

$$|\psi\rangle = \frac{1}{2048} \sum_{x=0}^{2047} \sum_{y=0}^{2047} e^{2\pi ixy/2048} |y\rangle |9^x \mod 35\rangle \tag{5.12}$$

After measuring the second quantum register, some state $|y\rangle |z\rangle$ is obtained, say $z \equiv 9^3 \mod 35 = 29$. In order to find out if the algorithm worked well, the values for x_0 and r are needed. From table 5.2 we find $x_0 = 6$ and $r = 6$ (which is obviously not known in practice, since it is hidden in the quantum state). The probability of finding some state $|y\rangle |29\rangle$ is

$$P(y|z = 29) = \left| \frac{1}{2048} \sum_{k} e^{2\pi i(3+6ky)/2048} \right|^2 \tag{5.13}$$

in which k runs from 0 up to $\left\lfloor \frac{2048-6-1}{6} \right\rfloor = 340$.

This probability distribution is plotted in figure 5.2.

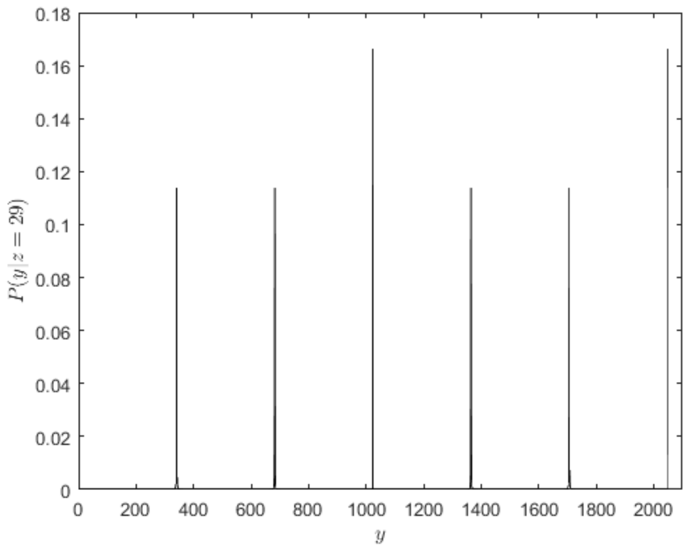


Figure 5.2

It is clearly visible that the probability peaks at multiples of $\frac{2048}{6} = 433\frac{1}{3}$ and is almost zero anywhere else. Say the measurement outcome is $y = 1707$. Using continued fraction expansion on $\frac{1707}{2048}$ until $\left|\frac{y}{w} - \frac{d}{s}\right| < \frac{1}{4096}$ yields $\frac{d}{s} = 0 + \dfrac{1}{1+1/5} = \frac{5}{6}$. So indeed, the period of $9^x \mod 35$ is found to be $r = 6$. r is even and $9^{6/2} + 1 = 30 \mod 35$, so r satisfies the requirements. Then, $\gcd(30, 35) = 5$ is a factor of 35 which is indeed right. $9^{6/2} - 1 = 28 \mod 35$, so the second factor of 35 is given by $\gcd(28, 35) = 7$. $35 = 7 \cdot 5$, so all factors of N have been found.

Let $p_1 p_2 \dots p_n$ be the prime factorization of N_1. All prime factors can be found using Shor's algorithm. However, only two factors can be found at a time. Suppose prime factors p_i and p_j are found in one run of Shor's algorithm. After dividing through by these numbers, a new number N_2 is obtained with prime factorization $p_1 \dots p_{i-1} p_{i+1} \dots p_{j-1} p_{j+1} \dots p_n$. Shor's algorithm can be applied once more and M can be divided by the two new factors obtained. Continuing this until $N_k = 1$ thus yields all prime factors of N_1.

5.4 Time Complexity

The most efficient classical algorithm for factoring large integers is the general number field sieve (see Briggs (1998)[6] for an extensive explanation). This algorithm has superpolynomial scaling,

$$\mathcal{O}\left(\exp\left\{\left[c(\log n)^{\frac{1}{3}}(\log\log n)^{\frac{2}{3}}\right]\right\}\right) \tag{5.14}$$

The classical parts of Shor's algorithm can all be performed in polynomial time. Checking whether N is prime or not can be performed in polynomial time by the Agrawal-Kayal-Saxena primality test[1] and even so it can be checked if N is a prime power using Bernstein's algorithm (1997)[4]. Also, the greatest common divisor can be found in polynomial time using Xi (2000)[35]. Performing the continued fractional expansion can be done in polynomial time using the technique developed by Hardy and Write (1979)[19].

The quantum part of Shor's algorithm, including modular exponentiation and the QFT run in polynomial time too. Therefore, it can be concluded that Shor's algorithm can factor integers in polynomial time. To be precise, the scaling is[31]

$$\mathcal{O}((\log n)^2(\log\log N)(\log\log\log N) \tag{5.15}$$

6 Algorithms Implementation and Discussion

The algorithms described in this thesis have been implemented and tested in two different quantum computer simulators. The first one is QX Quantum Computer Simulator[29], developed by QuTech. Static quantum circuits can be designed directly in QX Simulator. In order to add complexities such as loops or variables, C++ and Python can be used to generate and populate quantum gates. The second one is LIQ$Ui\,|\rangle$[8], a quantum programming framework integrated in F#, which was developed by the Quantum Architectures and Computation Group at Microsoft Research. The subroutine-algorithms that have been implemented in QX Simulator and Liquid can be used as quantum libraries.

In section 6.1, the features of both QX Simulator and Liquid are outlined. The error model that is available in QX Simulator and Liquid is explained in more detail in section 6.1.1. The limitations of quantum computer simulators are discussed in section 6.2. In section 6.3, the implementation of both Grover's algorithm and the multi-search is described. In addition, the results from simulations as well as the error analysis is described in this section. In section 6.4, the implementation of the quantum Fourier transform is discussed. Because the QFT is merely used as a building block of other circuits, no error analysis is performed on this algorithm. The implementation and error analysis of the quantum adder/subtractor are described in section 6.5.

The error analysis is done only with the QX Simulator. The error model in Liquid is not used, because in Liquid the error model can only be used in combination with a quantum error correction circuit that brings along a lot of complications. Finally, the implementation of Shor's algorithm is left for future work.

6.1 Features of QX Simulator and Liquid

In this section, the main features of QX Simulator and Liquid are discussed. In table 6.1, these features are summarized.

Table 6.1. The main features of QX Simulator and Liquid.

Feature	QX Simulator	Liquid
Developed by	QuTech	Microsoft
Open / Closed source	Open	Closed
Programming language	C++, Python, QASM	F#
Number of qubits available Non-entangled Maximally entangled	 29 28	 23 23
Custom gates available?	No	Yes
Error model	Depolarization channel	Depolarization channel

First of all, QX Simulator is open source. It is possible to modify anything in QX Simulator, because all files are stored locally and are editable. The platform can be pulled from GitHub. Microsoft Liquid on the other hand is closed source.

Besides, QX Simulator accepts files in which circuits are described using QASM (quantum assembly language). This is a low-level quantum language that is useful for simulating circuits with just a few qubits. However, more advanced circuits can be described by using a high-level programming language such as C++ or Python in order to generate the quantum circuit. The library that is used to implement the quantum circuit consists of not too many different commands, making it easy to understand. Liquid is programmed in F#, which is less common. Consequently, there is not a lot of documentation, especially not on Q&A communities. Moreover, the quantum part of F# has even less documentation available on the web making it sometimes hard to find the right information.

As outlined in the next section, a lot of memory is required to simulate quantum computation on a classical computer. As a result, the QX Simulator can handle up to 29 non-entangled qubits. A maximally entangled state can consist of up to 28 qubits. In Liquid, on the other hand, the number of qubits is capped at 23 for both entangled and non-entangled states.

An advantage of Microsoft's simulator is that custom gates can easily be implemented. This makes building quantum circuits very convenient. Additionally, it is very easy to add control qubits to a gate. For Grover's algorithm, it is possible to add as many control qubits as necessary for the $C^n(X)$ gate involved in the sign flip as well as the $C^n(Z)$ gate in the diffusion part. In QX Simulator, it is possible to add a binary control to an arbitrary gate. Consequently, the control qubit must be measured at first. This measurement destroys the superposition the qubit was in, so in some cases this is not useful.

Besides, custom gates cannot be designed in QX Simulator. However, there is a good reason why custom gates are not supported by QX Simulator. The QX simulator can be used as a back-end of the OpenQL compiler developed by QuTech[15]. This compiler takes as an input a quantum algorithm described in a high-level quantum programming (C++, Python) and compiles it into QASM instructions, which are technology independent and can be executed on the QX Simulator. The compiler will take care of the decomposition of the gates. In other words, it is responsible, among other things, for decomposing any arbitrary gate into the gates that the QX simulator supports. Note that this decomposition is not available yet.

Additionally, the OpenQL compiler allows for further compilation of this QASM to eQASM (executable QASM). This eQASM is suited for execution on a real quantum processor using superconducting qubits or spin qubits.

The error models in both QX Simulator and Liquid are very similar. Both do the same: they inject bit flips and phase flips. Of course, this does not cover the complete story of errors in physical quantum computers. The implementation is very binary since an error just will or will not occur. There is no in-between in the model, whereas in a physical quantum computer small errors will occur all the time. For QX Simulator, an alternative error model is being developed: the Pauli Twirling Approximation (see [16] for more information).

In Liquid, however, the error model can only be implemented in combination with a quantum error correction circuit. Because of this, the gate set that can be used is limited to only some basic gates.

6.1.1 Error Model

An error model that is commonly used in quantum computer simulators is the depolarizing channel. This error model inserts errors into the quantum circuit between each gate that is executed with a certain probability. This error can be a bit flip (i.e. $\alpha|0\rangle + \beta|1\rangle \rightarrow \alpha|1\rangle + \beta|0\rangle$), a phase flip (i.e. $\alpha|0\rangle + \beta|1\rangle \rightarrow -\alpha|0\rangle - \beta|1\rangle$) or both (i.e. $\alpha|0\rangle + \beta|1\rangle \rightarrow -\alpha|1\rangle - \beta|0\rangle$). These errors can be simulated by applying an X gate, Y gate or Z gate to the targeted qubit respectively. The effect of the error injection is shown in figure 6.1.

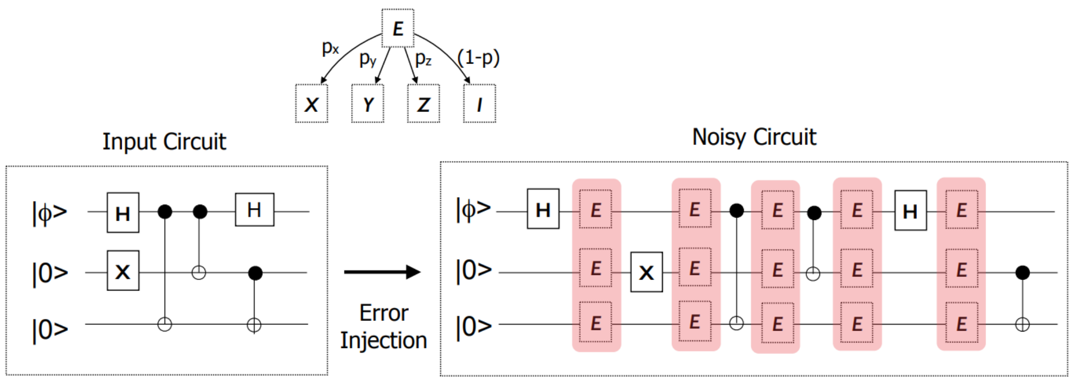


Figure 6.1. The result of error injection using the depolarisation channel. Reprinted from QX Quantum Code User Manual[29].

This error model is included in both QX Simulator and Liquid. Here, the error probability p is a user input variable and there holds $p_x = p_y = p_z$.

However, it is only possible to use the error model in Liquid in an quantum error correction environment. In this environment, each logical qubits are used instead of physical qubits if quantum error correction is used. Each logical qubit consists of 7 physical qubits (Steane code, see [10]). In addition, it is not possible to use customized quantum gates in the quantum error correction environment. Therefore, it is not possible to use the error model in Liquid if there are custom gates used in the quantum circuit.

6.2 Limitations of Quantum Computer Simulators

Quantum computer simulators like QX Simulator and Liquid are very useful for the time being, since the development of the first large-scale physical quantum computers still has a long way to go. However, simulating quantum computers on a classical computer also has its limitations.

An n qubit register has 2^n possible different qubit states. Therefore, 2^n complex coefficients must be stored. As a result, Liquid can only handle up to 23 qubits. With this, over 8 million states can be constituted. In QX Simulator on the other hand, a quantum state that is not entangled can consist of up to 29 qubits. The maximum number of states therefore is significantly higher: over 500 million. A maximally entangled state can be handled up to 28 qubits. One can imagine that it would take some serious supercomputers to simulate only a hundred qubits (which can constitute over 10^{30} different states!).

Besides, quantum acceleration can obviously not be achieved on a classical computer. As a result, running quantum algorithms can take a lot of time, especially when the number of qubits gets quite large. The predicted speedup of quantum algorithms compared to classical algorithms cannot be simulated and can only be tested on real quantum computers.

6.3 Grover's Algorithm

Grover's algorithm is implemented in QX Simulator for searching for single values as well as multiple values (the multi-search algorithm). The $C^n(X)$ gates and the $C^n(Z)$ gates have been decomposed using $n-2$ ancillary qubits as described in the first part of section 2.2.3. The code has been included in appendix A.1. The implementation of Grover's algorithm using only one ancillary qubit, using the decomposition described in the second part of that section is left for future work. In section 6.3.1, both the single-search and multi-search algorithm have been subjected to the error model in QX Simulator.

In Liquid, it is possible to add an arbitrary number of control qubits to a gate. Therefore, it is possible to write one piece of code to search for any states with different numbers of qubits without using the decomposition of the $C^n(X)$ gate and $C^n(Z)$ gate. The $C^n(X)$ gate implemented in Liquid is given by the following $2^{n+1} \times 2^{n+1}$ matrix.

$$C^n(X) \equiv \begin{bmatrix} 1 & & & & \\ & 1 & & \text{\Large 0} & \\ & & \ddots & & \\ & & & 1 & \\ & \text{\Large 0} & & & \begin{matrix} 0 & 1 \\ 1 & 0 \end{matrix} \end{bmatrix} \tag{6.1}$$

In Liquid, Grover's algorithm has been run for two to nine qubit states one thousand times. The code, including the description of the gate defined above is included in appendix B.1. The success rate is plotted against the probability of success that was determined numerically in section 2.3 in figure 6.2. The experimental results are very similar to the values that were calculated numerically.

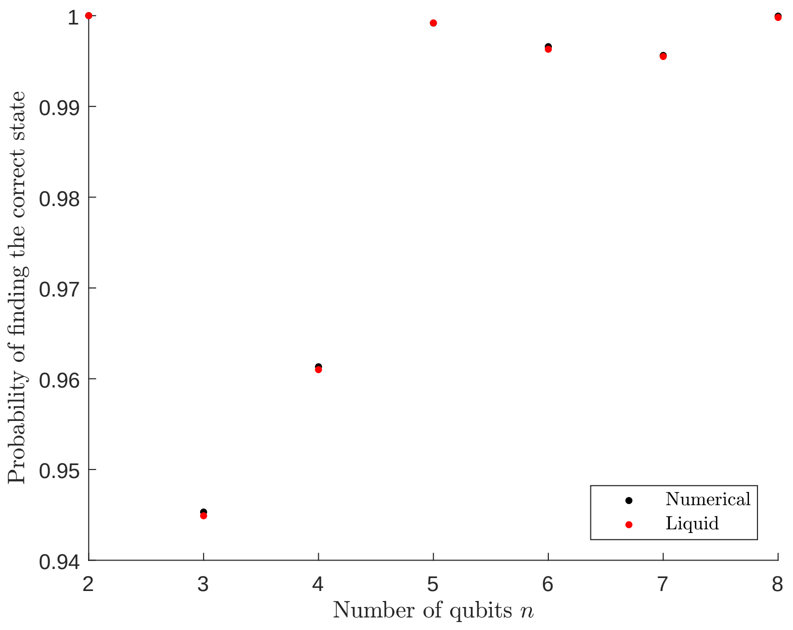


Figure 6.2. The probability of finding the correct state with Grover's algorithm in Liquid compared to the numerical calculations done earlier.

6.3.1 Single-Search Error Analysis

Grover's algorithm (single-search) has been subjected to the error model and the results have been analysed. The algorithm has been run with four different error probabilities ($p = 0.0005$, $p = 0.0010$, $p = 0.0020$ and $p = 0.0050$). For each of these probabilities, the algorithm has been run to search for the $|11 \cdots 1\rangle$ and $|00 \cdots 0\rangle$ qubit state consisting of two through six qubits. For each of these situations, Grover's algorithm has been run one thousand times and the statistics have been collected.

In figure 6.3, the success rate is plotted. The circle markers are used for the success rate searching the $|11 \cdots 1\rangle$ state and the square markers for the $|00 \cdots 0\rangle$ state. Also, the numerical values calculated in section 2.3 are plotted. Without errors, the results confirm this numerical analysis. The first thing that stands out in the error analysis is that the success rate is lower for the $|00 \cdots 0\rangle$ state. The cause of this phenomenon lies in the implementation of the sign flip in Grover's algorithm. Here, an X gate is applied to the qubits that are $|0\rangle$ in the correct state. If a qubit is $|1\rangle$ in the correct state, no gate needs to be applied. As a result, the more qubits are $|0\rangle$, the more X gates need to be applied. Naturally, a larger number of gates results in a larger amount of errors.

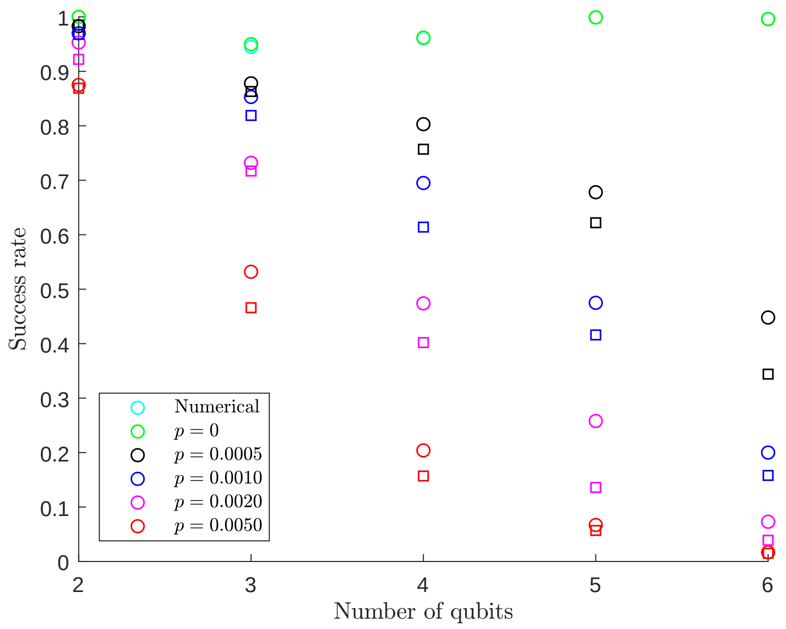


Figure 6.3. The success rate of Grover's algorithm as a function of the number of qubits for different error probabilities. The circle markers are used for success rate searching $|11\cdots1\rangle$, the square markers for $|00\cdots0\rangle$.

The success rate decreases quickly as both the error probability and the number of qubits is increased. With an error probability of 0.0050, the success rate of finding a state with more than three qubits drops below 21%. However, if the right state is not obtained, this does not mean that the measurement outcome of all qubits is affected by errors. By looking at the average measurement outcome of each qubit, conclusions can still be drawn about the correct state. Consider the four, five and six qubit searches with an error probability of 0.0050. In table 6.2, the measurement rate of $|0\rangle$ for each individual qubit can be found for these cases. The correct states are $|1010\rangle$, $|10101\rangle$ and $|101010\rangle$ respectively. If a qubit is $|0\rangle$, the measurement rate of $|0\rangle$ is expected to be significantly larger than 0.5. If a qubit is $|1\rangle$, on the other hand, it is expected that the measurement rate of $|0\rangle$ is significantly lower than 0.5. Clearly, the 4 qubit state can be found from these statistics. For the 5 qubit search, 4 qubit values can be obtained: $|10\cdot01\rangle$. The third qubit is measured as $|0\rangle$ and $|1\rangle$ almost an equal amount of times, so no conclusions can be drawn for this qubit. Finding the value of four out of five is a reasonable result, considering that the correct state is found in only 6% of the time. For the 6 qubit search, this method starts to fail as well. Because of the large amount of errors, all measurement rates are close to 50:50. No real conclusions can be drawn about the values of specific qubits, except for the fifth qubit.

Table 6.2. Measurement rate of $|0\rangle$ in the measurement statistics of individual qubits while searching for $|1010\cdots10\rangle$.

Qubit	4 qubits	5 qubits	6 qubits
q_1	0.406	0.450	0.519
q_2	0.576	0.544	0.511
q_3	0.394	0.506	0.476
q_4	0.608	0.534	0.499
q_5		0.467	0.471
q_6			0.519

6.3.2 Multi-Search Error Analysis

The multi-search algorithm, an extension of Grover's algorithm, has been subjected to the error model as well. The algorithm has been run with four different error probabilities ($p = 0.0005$, $p = 0.0010$, $p = 0.0020$ and $p = 0.0050$). For each of these probabilities, the algorithm has been run to search for one through eight correct 4 qubit states (so the total number of states is 16). For each of these situations, the multi-search algorithm has been run one thousand times and the statistics have been collected.

The results are shown in figure 6.4. In this figure, the success rate as well as the results of the numerical calculation done in section 2.4.2 have been plotted. The numerical calculation is clearly confirmed by the error-free simulations. As expected, the success rate quickly decreases if the number of correct states exceeds 4, a quarter of the total number of states. Besides, if the number of correct states is equal to a quarter of the total number of states, the success rate is indeed 100%.

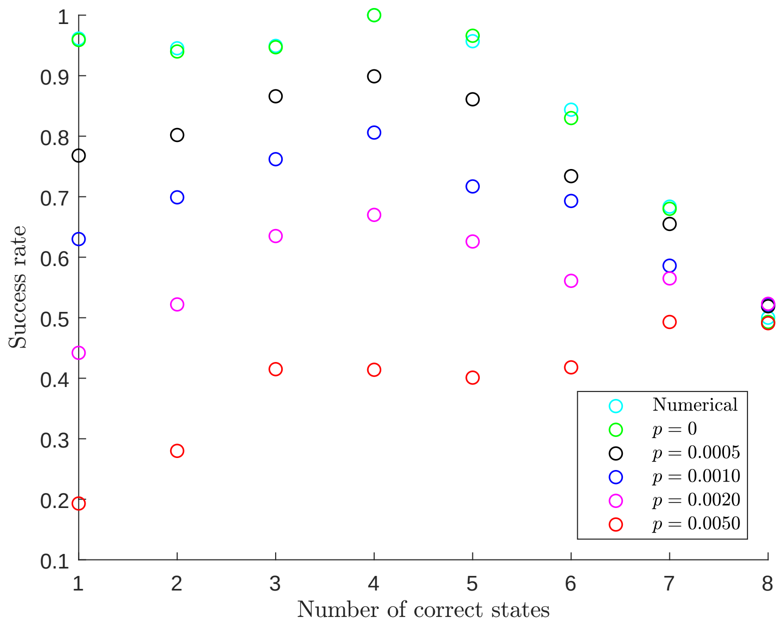


Figure 6.4. The success rate of the multi-search algorithm in QX Simulator for different error probabilities as well as the result from the numerical calculations done earlier..

As the error probability increases, the success rate levels off more and more. In particular, when the error probability is 0.0050 the success rate the success rate stabilises and even increases as the number of correct states is increased. But actually, the performance is still getting worse. If the number of correct states is larger, the probability to 'accidentally' find a correct state is larger as well. If an error occurs, there is a reasonable probability that still one of the correct states is found 'by accident'. Eventually, the success rate for any error approaches 0.5, just because half of the states is a correct state. In that case, even a measurement that returns a random state has a 50% chance of returning a correct state.

This gives a distorted view of the actual performance of the algorithm. Therefore, a better insight into the performance is given by subtracting the probability of 'accidentally' finding a correct state from the success rate. This way, the performance of the algorithm can be analysed more objectively. The difference between the success rate and the probability of 'accidentally' finding a correct state is shown in figure 6.5. From this figure, it is much more clear that the performance of the multi-search algorithm quickly declines when the number of correct states exceeds 4.Eventually, the performance of the quantum algorithm is no better than classical search.

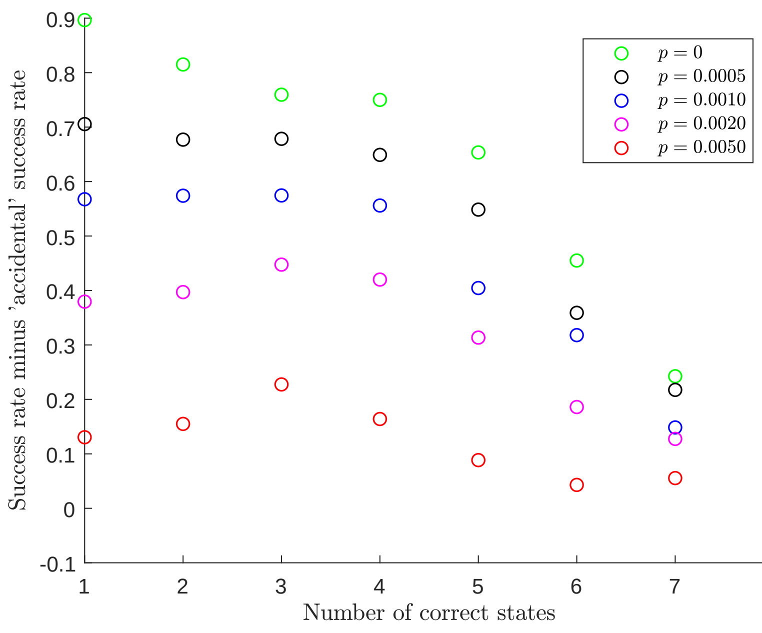


Figure 6.5. The success rate minus the probability of 'accidentally' finding a correct state for the multi-search algorithm in QX Simulator for different error probabilities.

Besides, it is no use looking at the measurement statistics of individual qubits. In each run, only one of the correct states is found, and each of the correct states has the same probability of being found. The different correct states will interfere the measurement statistics of individual qubits. For example, consider the case in which there are three correct states: $|1011\rangle$, $|1110\rangle$ and $|0101\rangle$. Each qubit is $|1\rangle$ in two thirds of the correct states. Therefore, looking at the measurement statistics of individual qubits with errors are injected into the circuit, all qubits will most likely be found in $|1\rangle$. From this, the only conclusion that could be drawn is that $|1111\rangle$ is a correct state. But $|1111\rangle$ is not one of the correct states. Consequently, it is impossible to find out which of the qubit(s) is $|0\rangle$ by measuring the individual qubits.

6.4 Quantum Fourier Transform

In QX Simulator, the quantum Fourier transform as well as the inverse Fourier transform have been implemented for an arbitrary number of qubits, see appendix A.2. The same has been done using Liquid, see appendix B.2 for the code. No error analysis is performed on the QFT, because it is merely used as a building block of the quantum adder/subtractor

In QX Simulator, there are very convenient gates available to build the QFT, because the phase shift applied by the controlled phase shift gate is automatically set to the right value based on the index of the control qubit and the index of the target qubit. Therefore, it is not necessary to think about which controlled phase shift gate should be applied. However, this also has some drawbacks if controlled phase shift gates are used in some circuit other than the QFT. In order to apply a controlled phase shift gate that does not match the indices of the control and target qubit, swaps have to be performed. As an example, consider the R_2 gate controlled by the third qubit that is applied to the first qubit. This gate is shown in figure 6.6a. If the CR gate from QX Simulator is applied to the first qubit controlled by the third qubit, however, this automatically generates the R_3 gate controlled by the third qubit based on the index of both qubits. The implementation of the desired gate is shown in figure 6.6b. In order to force an R_2 gate, the control qubit has to be swapped with the second qubit in order to change the index of the control qubit. After applying the CR gate, that now generates the R_2 gate controlled by the second qubit, the qubits must be swapped back.

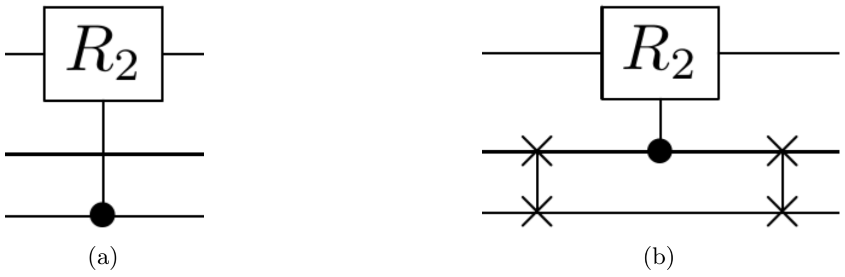


Figure 6.6. Left: the controlled R_2 gate applied to the first qubit controlled by the third qubit. Right: the implementation of this gate in QX Simulator.

In Liquid, on the other hand, the phase shift can be given as a parameter in the (R k) gate. A control can be added by calling Cgate (R k). This is more convenient when the qubits are not in the perfect order.

The QFT has been applied to $|01\rangle$ in both QX Simulator and Liquid to compare the outcome with equation (3.5). Indeed, both simulators give the same output as the theory prescribes:

$$QFT(|01\rangle) = \frac{1}{2}\left(|00\rangle + i\,|01\rangle - |10\rangle - i\,|11\rangle\right) \tag{6.2}$$

The output of QX Simulator is shown in figure 6.7. At the top, the executed gates are shown. This is optional and can also be disabled. Then, the output quantum state is given. The format of the coefficients is (*real*,*imaginary*). Finally, three measurement statistics are presented. First, the measurement averaging gives the average outcome of all measurements that have been performed on that qubit. The measurement prediction gives the result of a measurement when a qubit is not in a superposition. Finally, the most recent measurement outcome is shown.

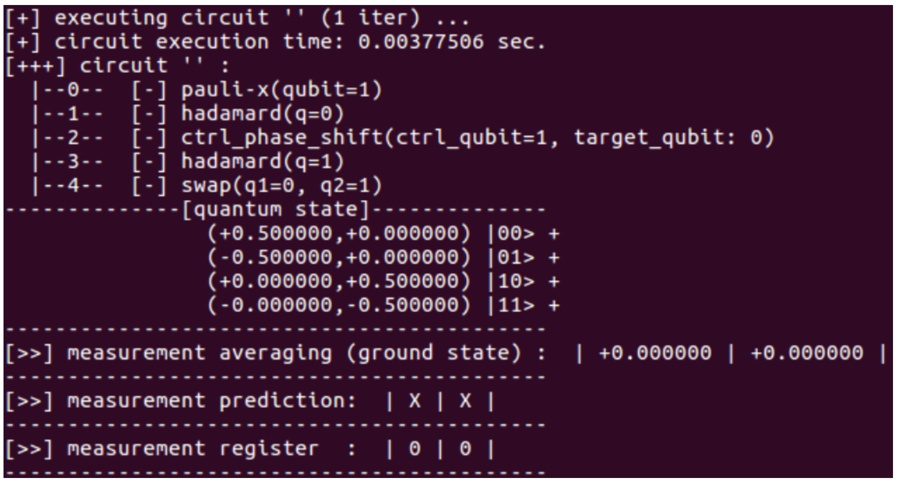

```
[+] executing circuit '' (1 iter) ...
[+] circuit execution time: 0.00377506 sec.
[+++] circuit '' :
  |--0--  [-] pauli-x(qubit=1)
  |--1--  [-] hadamard(q=0)
  |--2--  [-] ctrl_phase_shift(ctrl_qubit=1, target_qubit: 0)
  |--3--  [-] hadamard(q=1)
  |--4--  [-] swap(q1=0, q2=1)
-------------[quantum state]-------------
                (+0.500000,+0.000000) |00> +
                (-0.500000,+0.000000) |01> +
                (+0.000000,+0.500000) |10> +
                (-0.000000,-0.500000) |11> +
------------------------------------------
[>>] measurement averaging (ground state) :  | +0.000000 | +0.000000 |
------------------------------------------
[>>] measurement prediction:  | X | X |
------------------------------------------
[>>] measurement register  :  | 0 | 0 |
------------------------------------------
```

Figure 6.7. The output of the QFT in QX Simulator.

In Liquid, the output is briefer as shown in figure 6.8. In front of each line, the expired time is shown. The only relevant part is the display of the final quantum register. The different kets are shown as hexadecimal numbers, which is quite inconvenient compared to QX Simulator. For example, 0x00000003 $\equiv 4 \equiv |11\rangle$ and 0x0000001f $\equiv 31 \equiv |11111\rangle$.

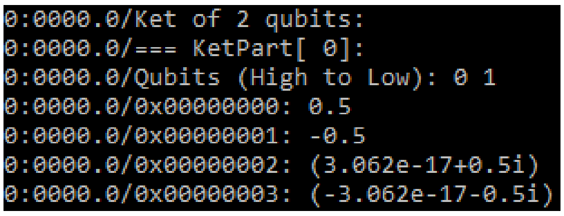

```
0:0000.0/Ket of 2 qubits:
0:0000.0/=== KetPart[ 0]:
0:0000.0/Qubits (High to Low): 0 1
0:0000.0/0x00000000: 0.5
0:0000.0/0x00000001: -0.5
0:0000.0/0x00000002: (3.062e-17+0.5i)
0:0000.0/0x00000003: (-3.062e-17-0.5i)
```

Figure 6.8. The output of the QFT in Microsoft Liquid

6.5 Quantum Adder/Subtractor

The final circuit that has been implemented is the quantum adder/subtractor. Here, the circuit from figure 4.12 has been implemented. This circuit is used to add or subtract two integer numbers. The maximum size of the numbers depends on the number of qubits. Because both $|a\rangle$ and $|b\rangle$ require one qubit to represent the sign and one qubit to prevent overflow, the numbers should satisfy $|a|, |b| < 2^{\frac{1}{2}n-2}$ where n is the total number of qubits. For example, a twelve-qubit circuit can handle all integers that have absolute value smaller than 16. The code implemented in QX Simulator and Liquid is included in appendix A.3 and appendix B.3 respectively.

6.5.1 Error Analysis

The circuit has been subjected to the error model and the results have been analysed. The algorithm has been run with four different error probabilities ($p = 0.0005$, $p = 0.0010$, $p = 0.0020$ and $p = 0.0050$). For each of these probabilities, the algorithm has been run with 6, 8 and 10 qubits, or $-1 \leq a, b \leq 1$, $-3 \leq a, b \leq 3$ and $-7 \leq a, b \leq 7$ respectively. For each combination of a and b, the algorithm has been run a thousand times.

The average success rate is set out in figure 6.9. In contrast to Grover's algorithm, this quantum subroutine is not probabilistic. Consequently, if the error probability is set to zero the right outcome is always obtained. With an error probability of 0.0005, the algorithm yields a quite reliable result with each number of qubits that has been tested. In particular, because the wrong outcomes are most often not the same, just a few runs are necessary to confirm the outcome by looking at which result is obtained the most. If the error probability is increased to 0.0010, the reliability decreases for ten qubits but is still reasonable for 6 and 8 qubits. After doubling that probability to 0.0020, the algorithm is still quite reliable for 6 and 8 qubits. However, the reliability of the 10-qubit algorithm quickly decreases. Finally, an error probability of 0.0050 yields the correct outcome on average only once every six runs for 8 qubits and once every twenty runs for 10 qubits. Considering that there is are 29 possible outcomes ($-14 \ldots 14$), the algorithm has to be executed many times in order to obtain the outcome with confidence.

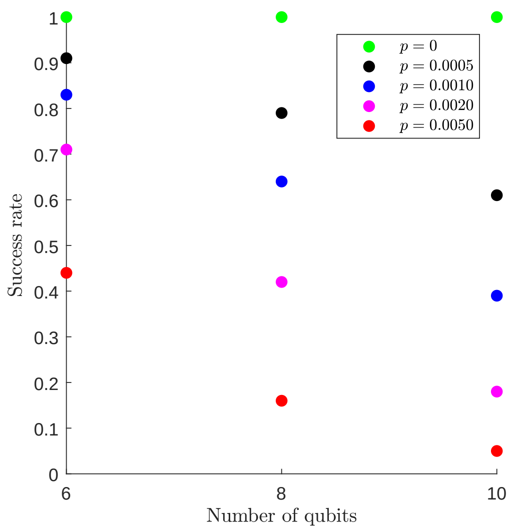
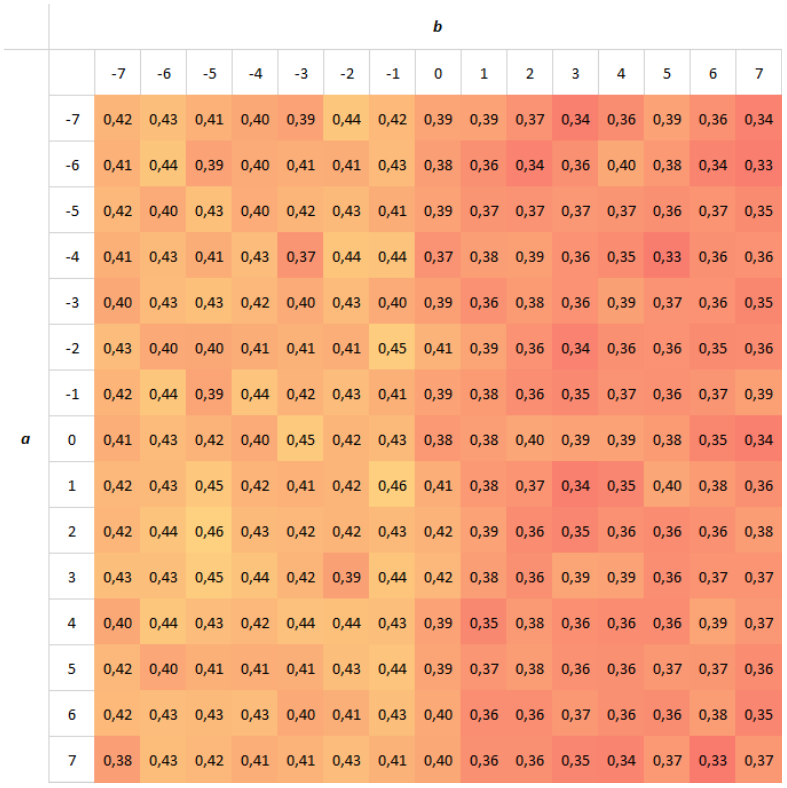


Figure 6.9. Success rate of the quantum adder/subtractor for different error probabilities.

The drop in reliability as the number of qubits increases has two obvious causes. First of all, the number of steps increases with the number of qubits, because the QFT as well as the quantum adder/subtractor and the inverse QFT require more controlled rotation gates. As a result, there are more points in the circuit where an error can occur. More specific, the number of circuit steps in which an error can occur is 68 for 6 qubits, 132 for 8 qubits and 225 for 10 qubits. Consequently, in the 10 qubit algorithm occur on average more than six times as many errors than in the 6 qubit circuit.

Looking at the success rate as a function of the input numbers a and b, an interesting trend occurs. In figure 6.10, the success rate of the 10 qubit algorithm as a function of a and b with an error probability of 0.0010 is shown. The error seems quite uniform as a function of a. However, in the b direction it does not. For negative b, the overall success rate is 0.42, whereas it is 0.37 for $b > 0$. A possible explanation for this can be found in the circuit schematic shown in figure 6.11. In this figure, we see that the subtraction subroutine is implemented in front of the addition subroutine. After the subtraction subroutine, an error on $|b_2\rangle$ through $|b_n\rangle$ does not affect the outcome any more. However, the number of steps before the addition subroutine is larger and therefore the probability of an error occurring in $|b\rangle$ that affects the outcome is much larger.



									b							
		-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
	-7	0,42	0,43	0,41	0,40	0,39	0,44	0,42	0,39	0,39	0,37	0,34	0,36	0,39	0,36	0,34
	-6	0,41	0,44	0,39	0,40	0,41	0,41	0,43	0,38	0,36	0,34	0,36	0,40	0,38	0,34	0,33
	-5	0,42	0,40	0,43	0,40	0,42	0,43	0,41	0,39	0,37	0,37	0,37	0,37	0,36	0,37	0,35
	-4	0,41	0,43	0,41	0,43	0,37	0,44	0,44	0,37	0,38	0,39	0,36	0,35	0,33	0,36	0,36
	-3	0,40	0,43	0,43	0,42	0,40	0,43	0,40	0,39	0,36	0,38	0,36	0,39	0,37	0,36	0,35
	-2	0,43	0,40	0,40	0,41	0,41	0,41	0,45	0,41	0,39	0,36	0,34	0,36	0,36	0,35	0,36
	-1	0,42	0,44	0,39	0,44	0,42	0,43	0,41	0,39	0,38	0,36	0,35	0,37	0,36	0,37	0,39
a	0	0,41	0,43	0,42	0,40	0,45	0,42	0,43	0,38	0,38	0,40	0,39	0,39	0,38	0,35	0,34
	1	0,42	0,43	0,45	0,42	0,41	0,42	0,46	0,41	0,38	0,37	0,34	0,35	0,40	0,38	0,36
	2	0,42	0,44	0,46	0,43	0,42	0,42	0,43	0,42	0,39	0,36	0,35	0,36	0,36	0,36	0,38
	3	0,43	0,43	0,45	0,44	0,42	0,39	0,44	0,42	0,38	0,36	0,39	0,39	0,36	0,37	0,37
	4	0,40	0,44	0,43	0,42	0,44	0,44	0,43	0,39	0,35	0,38	0,36	0,36	0,36	0,39	0,37
	5	0,42	0,40	0,41	0,41	0,41	0,43	0,44	0,39	0,37	0,38	0,36	0,36	0,37	0,37	0,36
	6	0,42	0,43	0,43	0,43	0,40	0,41	0,43	0,40	0,36	0,36	0,37	0,36	0,36	0,38	0,35
	7	0,38	0,43	0,42	0,41	0,41	0,43	0,41	0,40	0,36	0,36	0,35	0,34	0,37	0,33	0,37

Figure 6.10. Success rate of the 10 qubit algorithm with an error probability of 0.0010 as a function of a and b.

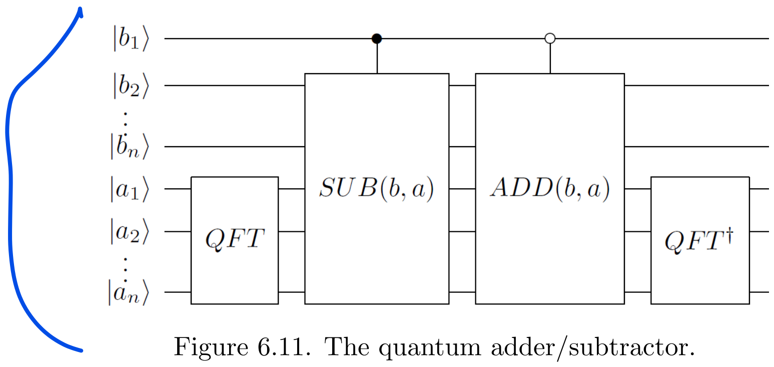
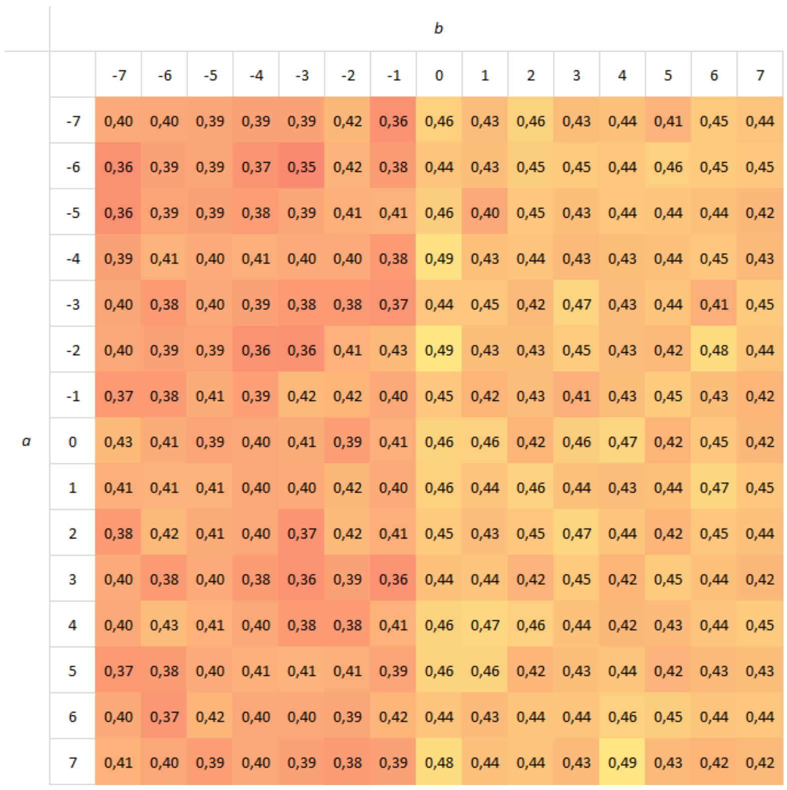


Figure 6.11. The quantum adder/subtractor.

In order to confirm this presumption, the subtraction subroutine and addition subroutine in the above figure have been interchanged, the addition subroutine has been implemented in front of the subtraction subroutine. This circuit has been run with 10 qubits and an error probability of 0.0010 as well. In figure 6.12, the success rate is shown as a function of a and b. Now, success rate is larger for negative b which indeed supports the presumption.



		b														
		-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
	-7	0,40	0,40	0,39	0,39	0,39	0,42	0,36	0,46	0,43	0,46	0,43	0,44	0,41	0,45	0,44
	-6	0,36	0,39	0,39	0,37	0,35	0,42	0,38	0,44	0,43	0,45	0,45	0,44	0,46	0,45	0,45
	-5	0,36	0,39	0,39	0,38	0,39	0,41	0,41	0,46	0,40	0,45	0,43	0,44	0,44	0,44	0,42
	-4	0,39	0,41	0,40	0,41	0,40	0,40	0,38	0,49	0,43	0,44	0,43	0,43	0,44	0,45	0,43
	-3	0,40	0,38	0,40	0,39	0,38	0,38	0,37	0,44	0,45	0,42	0,47	0,43	0,44	0,41	0,45
	-2	0,40	0,39	0,39	0,36	0,36	0,41	0,43	0,49	0,43	0,43	0,45	0,43	0,42	0,48	0,44
	-1	0,37	0,38	0,41	0,39	0,42	0,42	0,40	0,45	0,42	0,43	0,41	0,43	0,45	0,43	0,42
a	0	0,43	0,41	0,39	0,40	0,41	0,39	0,41	0,46	0,46	0,42	0,46	0,47	0,42	0,45	0,42
	1	0,41	0,41	0,41	0,40	0,40	0,42	0,40	0,46	0,44	0,46	0,44	0,43	0,44	0,47	0,45
	2	0,38	0,42	0,41	0,40	0,37	0,42	0,41	0,45	0,43	0,45	0,47	0,44	0,42	0,45	0,44
	3	0,40	0,38	0,40	0,38	0,36	0,39	0,36	0,44	0,44	0,42	0,45	0,42	0,45	0,44	0,42
	4	0,40	0,43	0,41	0,40	0,38	0,38	0,41	0,46	0,47	0,46	0,44	0,42	0,43	0,44	0,45
	5	0,37	0,38	0,40	0,41	0,41	0,41	0,39	0,46	0,46	0,42	0,43	0,44	0,42	0,43	0,43
	6	0,40	0,37	0,42	0,40	0,40	0,39	0,42	0,44	0,43	0,44	0,44	0,46	0,45	0,44	0,44
	7	0,41	0,40	0,39	0,40	0,39	0,38	0,39	0,48	0,44	0,44	0,43	0,49	0,43	0,42	0,42

Figure 6.12. Success rate of the 10 qubit algorithm with an error probability of 0.0010 as a function of a and b when the addition subroutine is implemented in front of the subtraction subroutine.

What stands out as well is that the overall success rate is higher than when the subtraction subroutine is implemented in front of the addition subroutine. There is a reasonable explanation for this as well. The subtraction circuit consists of more quantum gates than the addition circuit. The reason for this is that only counter-clockwise rotations of the probability amplitude phase over $\frac{2\pi}{2^k}$ radians ($k = 0, 1, 2, \ldots$) are possible in QX Simulator. This is sufficient for the implementation of the addition circuit. However, the subtraction circuit requires clockwise rotations. As an example, the implementation of the R_{-3} gate, which rotates the probability amplitude phase over $-\frac{\pi}{4}$ radians in QX simulator is shown in figure 6.13. The Z gate, the R_2 gate and R_3 gate have to be implemented successively. This combination of gates rotates the probability amplitude phase over an angle of $\pi + \frac{\pi}{2} + \frac{\pi}{4} = \frac{7\pi}{4}$ radians, which is equivalent to a rotation over $-\frac{\pi}{4}$ radians. Because this implementation is necessary for all clockwise rotations, the number of gates involved in the subtraction circuit is significantly larger than the number of gates involved in the addition circuit.

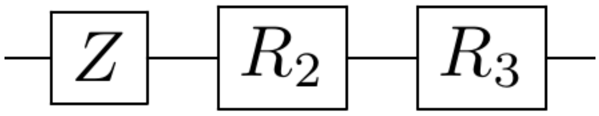
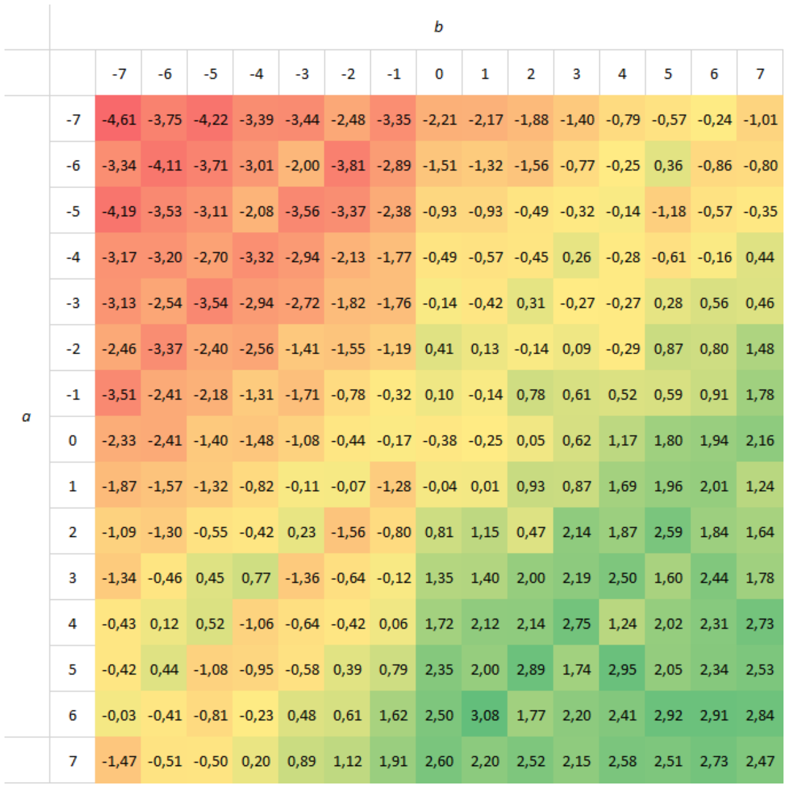


Figure 6.13. The implementation of the R_{-3} phase shift gate in QX Simulator.

For Grover's algorithm, the right outcome could be obtained with a higher success rate by looking at the statistics of measurements on individual qubits. However, because the quantum adder/subtractor is merely used as a building block of other algorithms, this does not offer a solution because the input may be in a superposition state.

In spite of this, the circuit can still be useful even with a low success rate. Take the 10 qubit circuit with an error probability of 0.0020. The average outcomes as a function of a and b over a thousand runs are shown in figure 6.14. Most of the outcomes are not even close to the right outcome, but the outcomes are still ordered quite well. A clear distinction can be made between the average outcome of $-1 + 2$ (0.78) and the average outcome of $4 + 2$ (2.14). So as long as only the exact outcome is not of importance, the quantum adder/subtractor is well suited for distinguishing between higher and lower outcomes, even under the influence of errors. If an exact outcome is required, however, the only solution is decreasing the amount and influence of errors in the circuit.



a \ *b*	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
-7	-4,61	-3,75	-4,22	-3,39	-3,44	-2,48	-3,35	-2,21	-2,17	-1,88	-1,40	-0,79	-0,57	-0,24	-1,01
-6	-3,34	-4,11	-3,71	-3,01	-2,00	-3,81	-2,89	-1,51	-1,32	-1,56	-0,77	-0,25	0,36	-0,86	-0,80
-5	-4,19	-3,53	-3,11	-2,08	-3,56	-3,37	-2,38	-0,93	-0,93	-0,49	-0,32	-0,14	-1,18	-0,57	-0,35
-4	-3,17	-3,20	-2,70	-3,32	-2,94	-2,13	-1,77	-0,49	-0,57	-0,45	0,26	-0,28	-0,61	-0,16	0,44
-3	-3,13	-2,54	-3,54	-2,94	-2,72	-1,82	-1,76	-0,14	-0,42	0,31	-0,27	-0,27	0,28	0,56	0,46
-2	-2,46	-3,37	-2,40	-2,56	-1,41	-1,55	-1,19	0,41	0,13	-0,14	0,09	-0,29	0,87	0,80	1,48
-1	-3,51	-2,41	-2,18	-1,31	-1,71	-0,78	-0,32	0,10	-0,14	0,78	0,61	0,52	0,59	0,91	1,78
0	-2,33	-2,41	-1,40	-1,48	-1,08	-0,44	-0,17	-0,38	-0,25	0,05	0,62	1,17	1,80	1,94	2,16
1	-1,87	-1,57	-1,32	-0,82	-0,11	-0,07	-1,28	-0,04	0,01	0,93	0,87	1,69	1,96	2,01	1,24
2	-1,09	-1,30	-0,55	-0,42	0,23	-1,56	-0,80	0,81	1,15	0,47	2,14	1,87	2,59	1,84	1,64
3	-1,34	-0,46	0,45	0,77	-1,36	-0,64	-0,12	1,35	1,40	2,00	2,19	2,50	1,60	2,44	1,78
4	-0,43	0,12	0,52	-1,06	-0,64	-0,42	0,06	1,72	2,12	2,14	2,75	1,24	2,02	2,31	2,73
5	-0,42	0,44	-1,08	-0,95	-0,58	0,39	0,79	2,35	2,00	2,89	1,74	2,95	2,05	2,34	2,53
6	-0,03	-0,41	-0,81	-0,23	0,48	0,61	1,62	2,50	3,08	1,77	2,20	2,41	2,92	2,91	2,84
7	-1,47	-0,51	-0,50	0,20	0,89	1,12	1,91	2,60	2,20	2,52	2,15	2,58	2,51	2,73	2,47

Figure 6.14. Average outcome of the 10 qubit algorithm with an error probability of 0.0020 as a function of a and b.

7 Conclusions

A multitude of quantum algorithms has been analysed and implemented in QX Simulator and Microsoft Liquid, including Grover's algorithm and the multi-search algorithm based thereon as well as the QFT and the quantum adder/subtractor. The last two form the first steps towards the implementation of the period finding quantum algorithm on which Shor's algorithm is based. In this section, the main conclusions from both the numerical calculations and the simulations are presented.

The formula for obtaining the optimal number of Grover iterations, to reach the maximum probability of finding the correct state, has been found to consistently overestimate the real value by half an iteration. Therefore, the result from the original formula should always be rounded down towards the greatest integer smaller than the result from the original formula. The resulting time complexity of Grover's algorithm is $\mathcal{O}(2^{n/2})$, which is better than the classical $\mathcal{O}(2^n)$, but is still exponential time. Additionally, the numerical calculations of the success probability are supported by the simulations of Grover's algorithm for two through nine qubit states.

If errors are injected in Grover's circuit, the probability of finding the correct state quickly decreases – especially for a larger number of qubits, because more iterations have to be performed. However, information on the correct state can still be obtained by looking at the measurement statistics of individual qubits even if the probability of finding the correct state is only a few percent.

Grover's algorithm has been extended to a multi-search algorithm. This algorithm searches for multiple states, instead of just one. The number of iterations decreases as the number of correct states increases. This reaches its optimum if the number of correct states is equal to a quarter of the total number of states. Then, only one iteration is required and a correct state is found with probability one. However, if the number of correct states is increased even more, the probability of finding a correct state rapidly decreases and approaches only 50% if the number of correct states is half of the total number of states. In that case, Grover's multi-search algorithm performs no better than classical search. These facts from the numerical calculations have been confirmed by the simulation of the multi-search algorithm.

Besides, the quantum Fourier transform has been analysed. This is a key part in many quantum algorithms. Based on the circuit, the time complexity of the quantum Fourier transform is $\mathcal{O}(n^2)$. This is an exponential speedup over the Fast Fourier Transform which requires $\mathcal{O}(n2^n)$ operations. One application of the quantum Fourier transform is found in the the quantum addition circuit. Because operations can be done in parallel, the addition subroutine only requires $\mathcal{O}(n)$ operations. But since the quantum Fourier transform is part of quantum addition as well, the overall time complexity is yet $\mathcal{O}(n^2)$.

Based on quantum addition and its inverse, quantum subtraction, a quantum adder/subtractor has been implemented in the quantum computer simulators. The error-free circuit always yields the correct outcome, because the routine is not probabilistic. Error analysis shows that the implementation of the circuit influences the success rate of different numbers. If the addition subroutine is implemented in front of the subtraction subroutine, more errors likely occur if a number is subtracted rather than if a number is added. If the addition subroutine and subtraction subroutine are interchanged, more errors likely occur if a number is added. Therefore, this can be taken into consideration for the exact implementation of the circuit.

If the error probability is increased, the exact outcome of the circuit becomes less reliable. However, even with a low success rate the average outcomes are still ordered quite well. Therefore, if one correct outcome is larger than another the average outcome of the first is very likely to be larger than the second. That way, the outcomes can be compared.

Further research can be done into the multi-search Grover's algorithm. A general formula for the optimal number of iterations depending on the number of correct states and the total number of states would provide useful insight. Additionally, more applications of this algorithm to mathematical problems could be found. Besides, the quantum adder/subtractor can be extended to the full period finding quantum algorithm on which Shor's algorithm is based. Numerical analysis can be performed on the period finding algorithm and its behaviour if errors are injected can be examined. Furthermore, the error analysis performed in this thesis could be elaborated, especially when more realistic error models become available. With that, quantum error correction could be applied to make the algorithms more reliable and ready for application on physical quantum computers.

References

[1] M. Agrawal, N. Kayal, and N. Saxena. *PRIMES is in P*. Aug. 2002.

[2] S. Beauregard. “Circuit for Shor’s Algorithm using 2n+3 Qubits”. In: *arXiv:quant-ph/0205095v3* (Feb. 2003).

[3] C.H. Bennett et al. “Strengths and Weaknesses of Quantum Computing”. In: *arXiv:quant-ph/9701001v1* (Jan. 1997).

[4] D.J. Bernstein. *Detecting Perfect Powers in Essentially Linear Time*. July 1998.

[5] D.J. Bernstein. *Grover vs. McEliece*. 2010.

[6] M.E. Briggs. *An Introduction to the General Number Field Sieve*. Apr. 1998.

[7] J.B. Conway. *A Course in Functional Analysis*. Springer Verlag, 1990.

[8] Microsoft Corporation. *Microsoft Liquid*. URL: `http://http://stationq.github.io/Liquid/`.

[9] S. Daraeizadeh. *Efficient Implementation of Multi-Control Toffoli Gates in Linear Nearest Neighbor Arrays*. May 2014.

[10] S.J. Devitt, W.J. Munro, and K. Nemoto. “Quantum Error Correction for Beginners”. In: *arXiv:0905.2794v4* (June 2013).

[11] S.J. Devitt, K. Nemoto, and W.J. Munro. “Decoherence, Control, and Symmetry in Quantum Computers”. In: *arXiv:0905.2794v4* (May 2009).

[12] P.A.M. Dirac. “A New Notation for Quantum Mechanics”. In: *Mathematical Proceedings of the Cambridge Philosophical Society* 35.3 (June 1932).

[13] T.G. Draper. “Addition on a Quantum Computer”. In: *arXiv:quant-ph/0008033v1* (Sept. 1998).

[14] E. Einstein, B. Podolsky, and N. Rosen. “Can Quantum-Mechanical Description of Physical Reality Be Considered Complete?” In: *Physical Review* 47 (May 1935).

[15] X. Fu et al. “An experimental microarchitecture for a superconducting quantum processor”. In: *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture* (2017), pp. 813–825.

[16] M.R. Geller and Z. Zhou. “Efficient error models for fault-tolerant architectures and the Pauli twirling approximation”. In: *arXiv:1305.2021v1* (Mar. 2018).

[17] J.L. Gomez-Munoz. *Quantum Circuit Implementing Grover’s Algorithm*. Tecnológico de Monterrey. May 2011. URL: `http://homepage.cem.itesm.mx/lgomez/quantum/v7grovercircuit.pdf`.

[18] L.K. Grover. “A Fast Quantum Mechanical Algorithm for Database Search”. In: *arXiv:quant-ph/9605043* (May 1996).

[19] G.H. Hardy and E.M. Wright. *An Introduction to the Theory of Numbers*. Oxford University Press, 1938.

[20] T. Hertli, R.A. Moser, and D. Schneder. “Improving PPSZ for 3-SAT using Critical Variables”. In: *arXiv:1009.4830v3* (May 2011).

[21] R.J. Hughes et al. “Quantum Cryptography”. In: *arXiv:quant-ph/9504002v1* (Apr. 1995).

[22] P.C. Humphreys et al. “Deterministic delivery of remote entanglement on a quantum network”. In: *arXiv:1712.07567* (Dec. 2017).

[23] F. Xi Lin. “Shor’s Algorithm and the Quantum Fourier Transform”. In: ().

[24] RSA Security LLC. *RSA Customers.* URL: `https://www.rsa.com/en-us/customers`.

[25] R. van Meter and K.M. Itoh. “Fast Quantum Modular Exponentiation”. In: *arXiv:quant-ph/0408006v2* (Mar. 2005).

[26] M.A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information.* New York, New York, USA: Cambridge University Press, 2000.

[27] G. Paul. *Quantum Period Finding and Shor’s Factoring.* Dec. 2015.

[28] W. Penard and T. van Werkhoven. “On the Secure Hash Algorithm Family”. In: (Mar. 2016).

[29] QuTech. *QX Quantum Computer Simulator.* URL: `http://quantum-studio.net/`.

[30] M. Schmassmann. *Universality of Quantum Gates.* ETH Zürich. Oct. 2007. URL: `https://qudev.phys.ethz.ch/content/courses/QSIT07/presentations/Schmassmann.pdf`.

[31] P. Shor. “Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer”. In: *arXiv:quant-ph/9508027v2* (Jan. 1996).

[32] E. Strubell. *An Introduction to Quantum Algorithms.* 2011.

[33] R. de Wolf. *The Classical and Quantum Fourier Transform.* Feb. 2011.

[34] J. Wright and T. Tseng. *Grover’s Algorithm.* Carnegie Mellon School of Computer Science. Sept. 2015. URL: `https://www.cs.cmu.edu/~odonnell/quantum15/lecture04.pdf`.

[35] H. Xi. *GCD is a polynomial time algorithm.* Oct. 2000.

A QX Quantum Computer Simulator Code

A.1 Grover's algorithm

Grover's algorithm for searching an arbitrary state.

```cpp
#include <iostream>

#include <xpu.h>
#include <xpu/runtime>

#include <core/circuit.h>
#include <qcode/quantum_code_loader.h>
#include <core/error_model.h>

#include <math.h>

#include <string>

using namespace std;

int main(int argc, char ** argv)
{
   string state;
   cout << "Fill in state\n";
   getline(cin, state);
   cout << "Searching for " << state << "\n\n";
   int num_state = state.length(); // n
   int num_ancillas = state.length()-2+1; // ancilla and n-2 for decomposition
   int num_qubits = num_state + num_ancillas;
   qx::qu_register reg(num_qubits); // register of n qubits
   qx::circuit c(num_qubits);

   c.add(new qx::pauli_x(num_state)); // X on ancillary
   for(int i=0; i<=num_state; i=i+1) {
      c.add(new qx::hadamard(i)); // Hadamard to all qubits
   }



   int R = floor(M_PI/4*pow(2,num_state/2));
   cout << "Performing " << R << " Grover iterations\n";
   // end of preprocessing

   int iter = 0;
   while(iter < R) {
      int i=1;
      for(char& q : state) {
         if (q == '0') {
            c.add(new qx::pauli_x(num_state-i)); // X on qubits in |0>
         }
         ++i;
      }
      if(num_state == 2) {
         c.add(new qx::toffoli(0,1,2)); // Toffoli on ancillary
      }
      else {
         c.add(new qx::toffoli(0,1,num_state+1));
         for(int i=num_state+2; i<num_qubits; ++i) {
            c.add(new qx::toffoli(i-num_state,i-1,i));
         }
         c.add(new qx::toffoli(num_qubits-1,num_state-1,num_state));
         for(int i=num_qubits-1; i>=num_state+2; --i) {
            c.add(new qx::toffoli(i-num_state,i-1,i));
         }
         c.add(new qx::toffoli(0,1,num_state+1));
      }
      i=1;
      for(char& q : state) {
         if (q == '0') {
            c.add(new qx::pauli_x(num_state-i)); // X on qubits in |0>
         }
         ++i;
      }

      for(int i=0; i<num_state; ++i) {
         c.add(new qx::hadamard(i)); // Hadamard to register
      }

      for(int i=0; i<num_state; ++i) {
         c.add(new qx::pauli_x(i)); // X to register
      }

      if(num_state == 2) {
         c.add(new qx::cphase(0,1)); // Controlled Z
      }
      else {
         c.add(new qx::hadamard(num_state-1));
         if (num_state == 3) {
            c.add(new qx::toffoli(0,1,2));
         }
         else {
            c.add(new qx::toffoli(0,1,num_state+1));
            for(int i=num_state+2; i<num_qubits-1; ++i) {
               c.add(new qx::toffoli(i-num_state,i-1,i));
            }
            c.add(new qx::toffoli(num_qubits-2,num_state-2,num_state-1));
            for(int i=num_qubits-2; i>=num_state+2; --i) {
               c.add(new qx::toffoli(i-num_state,i-1,i));
            }
            c.add(new qx::toffoli(0,1,num_state+1));
         }
         c.add(new qx::hadamard(num_state-1));
      }

      for(int i=0; i<num_state; ++i) {
         c.add(new qx::pauli_x(i)); // X to register
      }

      for(int i=0; i<num_state; ++i) {
         c.add(new qx::hadamard(i)); // Hadamard to register
      }
   ++iter
   }


   for(int i=0; i<num_qubits; ++i) {
      c.add(new qx::measure(i));
   }

   c.execute(reg);
   reg.dump();
   c.dump();

}
```

Multi-search algorithm for an arbitrary number of states with an arbitrary number of qubits

```
#include <iostream>
#include <iterator>

#include <xpu.h>
#include <xpu/runtime>

#include <core/circuit.h>
#include <qcode/quantum_code_loader.h>
#include <core/error_model.h>

#include <math.h>

#include <string>

#include <vector>

using namespace std;

int main(int argc, char ** argv)
{
   std::string states;
   cout << "Fill in state(s), separated by spaces\n";
   getline(cin, states);
   std::stringstream ss(states);
   std::istream_iterator<std::string> begin(ss);
   std::istream_iterator<std::string> end;
   std::vector<std::string> state_vec(begin, end);
   cout << "Searching for \n";
   for(int i=0; i<state_vec.size(); ++i) {
      cout << state_vec.at(i) << "\n";
   }

   int num_state = state_vec.at(0).length(); // n
   std::vector<int> results_vec(num_state,0);
   int R;
   cout << "How many iterations?\n";
   cin >> R;

   cout << "\nPerforming " << R << " Grover iterations\n";
   int num_ancillas = state_vec.at(0).length()-2+1; // ancilla and n-2 for
       decomposition
   int num_qubits = num_state + num_ancillas;
   qx::qu_register reg(num_qubits); // register of n qubits
   qx::circuit c(num_qubits);
   // end of preprocessing

```

```
c.add(new qx::pauli_x(num_state)); // X on ancillary
for(int i=0; i<=num_state; i=i+1) {
   c.add(new qx::hadamard(i)); // Hadamard to all qubits
}

int iter = 0;
while(iter < R) {
   for(int s=0; s<state_vec.size(); ++s) {
      int i=1;
      for(char& q : state_vec.at(s)) {
         if (q == '0') {
         cout << "X to " << i << "\n";
         c.add(new qx::pauli_x(num_state-i)); // X on qubits in |0>
      }
      ++i;
   }
   if(num_state == 2) {
      c.add(new qx::toffoli(0,1,2)); // Toffoli on ancillary
   }
   else {
      c.add(new qx::toffoli(0,1,num_state+1));
      for(int i=num_state+2; i<num_qubits; ++i) {
         c.add(new qx::toffoli(i-num_state,i-1,i));
      }
      c.add(new qx::toffoli(num_qubits-1,num_state-1,num_state));
      for(int i=num_qubits-1; i>=num_state+2; --i) {
         c.add(new qx::toffoli(i-num_state,i-1,i));
      }
      c.add(new qx::toffoli(0,1,num_state+1));
   }
   i=1;
   for(char& q : state_vec.at(s)) {
      if (q == '0') {
         c.add(new qx::pauli_x(num_state-i)); // X on qubits in |0>
      }
      ++i;
      }
   }

   for(int i=0; i<num_state; ++i) {
      c.add(new qx::hadamard(i)); // Hadamard to register
   }


   for(int i=0; i<num_state; ++i) {
      c.add(new qx::pauli_x(i)); // X to register
   }

   if(num_state == 2) {
         c.add(new qx::cphase(0,1)); // Controlled Z
      }
      else {
         c.add(new qx::hadamard(num_state-1));
         if (num_state == 3) {
            c.add(new qx::toffoli(0,1,2));
         }
         else {
            c.add(new qx::toffoli(0,1,num_state+1));
            for(int i=num_state+2; i<num_qubits-1; ++i) {
               c.add(new qx::toffoli(i-num_state,i-1,i));
            }
            c.add(new qx::toffoli(num_qubits-2,num_state-2,num_state-1));
            for(int i=num_qubits-2; i>=num_state+2; --i) {
               c.add(new qx::toffoli(i-num_state,i-1,i));
            }
            c.add(new qx::toffoli(0,1,num_state+1));
         }
         c.add(new qx::hadamard(num_state-1));
      }

      for(int i=0; i<num_state; ++i) {
         c.add(new qx::pauli_x(i)); // Hadamard to register
      }

      for(int i=0; i<num_state; ++i) {
         c.add(new qx::hadamard(i)); // Hadamard to register
      }
      ++iter;
   }

   for(int i=0; i<num_qubits; ++i) {
   c.add(new qx::measure(i));
   }

   c.execute(reg);
   reg.dump();
   c.dump();

}
```

A.2 QFT and inverse QFT

The QFT for an arbitrary quantum register.

```
#include <iostream>

#include <xpu.h>
#include <xpu/runtime>

#include <core/circuit.h>
#include <qcode/quantum_code_loader.h>
#include <core/error_model.h>

#include <string>

using namespace std;

int main(int argc, char ** argv)
{
   string qstate;
   cout << "Fill in the initial quantum state\n";
   getline(cin, qstate);
   int num_qubits = qstate.length();
   qx::qu_register reg(num_qubits);
   qx::circuit c(num_qubits);

   // initialize quantum state
   int i=1;
   for(char& q: qstate) {
      if (q == '1') {
         c.add(new qx::pauli_x(num_qubits-i));
      }
      ++i;
   }

   // QFT
   for(int i=0; i<num_qubits; ++i) {
      c.add(new qx::hadamard(i)); // Hadamard first
      for(int j=i+1; j<num_qubits; ++j) {
         c.add(new qx::ctrl_phase_shift(j,i)); // Controlled phase shift gates
      }
   }

   for(int i=0; i<floor(num_qubits/2); ++i) {
      c.add(new qx::swap(i,num_qubits-1-i)); // Swap
   }

   // execute
   c.execute(reg);
```

```
    c.dump();
    reg.dump();

}
```

The inverse QFT for an arbitrary quantum register.

```
#include <iostream>

#include <xpu.h>
#include <xpu/runtime>

#include <core/circuit.h>
#include <qcode/quantum_code_loader.h>
#include <core/error_model.h>

#include <string>

using namespace std;

int main(int argc, char ** argv)
{
   string qstate;
   cout << "Fill in the initial quantum state\n";
   getline(cin, qstate);
   int num_qubits = qstate.length();
   qx::qu_register reg(num_qubits);
   qx::circuit c(num_qubits);

   // initialize quantum state
   int i=1;
   for(char& q: qstate) {
      if (q == '1') {
         c.add(new qx::pauli_x(num_qubits-i));
      }
      ++i;
   }

   // inverse QFT
   for(int i=0; i<floor(num_qubits/2); ++i) {
      c.add(new qx::swap(i,num_qubits-1-i)); // Swap
   }

   for(int i=num_qubits-1; i>=0; --i) {
      for(int j=num_qubits-1; j>i; --j) {
         c.add(new qx::cphase(j,i));
         for(int k=1; k<=j-i; ++k) {
            c.add(new qx::swap(j,i+k)); // Swap qubit to position
            c.add(new qx::ctrl_phase_shift(i+k,i)); // Controlled phase shift
                gates
            c.add(new qx::swap(j,i+k)); // Swap back
         }
      }
         c.add(new qx::hadamard(i)); // Hadamard
```

```
    }


    // execute
    c.execute(reg);
    c.dump();
    reg.dump();

}
```

A.3 Addition

The addition / subtraction algorithm that can add and subtract integer numbers.

```cpp
#include <iostream>

#include <xpu.h>
#include <xpu/runtime>

#include <core/circuit.h>
#include <qcode/quantum_code_loader.h>
#include <core/error_model.h>

#include <string>

using namespace std;

int main(int argc, char ** argv)
{
   string operation;
   string inputtype = "int"; // int or real

   int num_bits = 5; // one bit is used to store the sign. one bit is added to
       prevent overflow
   int binar = 0; // 0: integer input, 1: binary input
   int signbit = num_bits+1; // index of the bit that indicates the sign of b

   string a;
   string b;

   if(inputtype == "int") { // integer input
      cout << "a + b\n";
      cout << num_bits << " bits (signed): " << -pow(2,num_bits-1) << " < a,b <
          " << pow(2,num_bits-1) << "\n";
      if(binar == 0) {
         int a_int;
         int b_int;
         cout << "\na = ";
         cin >> a_int;
         cout << "b = ";
         cin >> b_int;

         if(a_int<0) {
            a += "1"; // to represent the minus sign
            a_int = pow(2,num_bits)+a_int; // a is made positive for
                calculation of binary represetation
         } else {
            a += "0"; // to represent plus sign
         }
      for(int i=num_bits-1; i>=0; --i) { // convert decimal to binary
         if(pow(2,i)<=a_int) {
            a += "1";
            a_int -= pow(2,i);
         } else {
            a += "0";
         }
      }
      cout << "\na=" << a << "\n";
      if(b_int<0) {
         b += "1"; // to represent minus sign
         b_int = -b_int; // b is made positive for calculation of binary representation
      } else {
         b += "0"; // to represent plus sign
      }
      for(int i=num_bits-1; i>=0; --i) { // convert decimal to binary
         if(pow(2,i)<=b_int) {
         b += "1";
         b_int -= pow(2,i);
      } else {
         b += "0";
      }
   }
   cout << "b=" << b << "\n";
   }
}

if(inputtype == "frac") { // for adding positive fractions, approximation
   cout << num_bits << " bits: max error = " << 2*pow(2,-num_bits+1) << "\n";
   if(binar == 0) {
      float a_float;
      float b_float;
      cout << "Fill in state a\n"; // a
      cin >> a_float;
      cout << "Fill in state b\n"; // b
      cin >> b_float;

      for(int i=0; i<num_bits; ++i) {
         if(pow(2,-i)<=a_float) {
            a += "1";
            a_float -= pow(2,-i);
         } else {
            a += "0";
         }
      }
      cout << a << "\n";

      for(int i=0; i<num_bits; ++i) {
```

```
      if(pow(2,-i)<=b_float) {
         b += "1";
         b_float -= pow(2,-i);
      } else {
         b += "0";
      }
      }
      cout << b;
   }
}

int num_qubits_a = a.length();
int num_qubits_b = b.length();
int num_qubits = num_qubits_a+num_qubits_b;
qx::qu_register reg(num_qubits); // initialize qubit register
qx::circuit c(num_qubits); // initialize circuit

// Initialize qubit state a
int i=0;
for(char& q: a) {
   if (q == '1') {
      c.add(new qx::pauli_x(i));
   }
   ++i;
}
// Initialze qubit state b
i=num_qubits_a;
for(char& q: b) {
   if (q == '1') {
      c.add(new qx::pauli_x(i));
   }
   ++i;
}
// end of preprocessing

// apply QFT to a
for(int i=0; i<num_qubits_a; ++i) {
   c.add(new qx::hadamard(i)); // Hadamard first
   for(int j=i+1; j<num_qubits_a; ++j) {
      c.add(new qx::ctrl_phase_shift(j,i)); // series of CR
   }
}

for(int i=0; i<floor(num_qubits_a/2); ++i) {
   c.add(new qx::swap(i,num_qubits_a-1-i));
}
// end of QFT


```

```
c.add(new qx::measure(signbit)); // measure to obtain sign of b (0: +, 1: -)

// Inverse Addition: gates applied if sign of b is -
for(int i=0; i<num_qubits_a; ++i) { //
   if(num_qubits-i-1 != signbit) { // sign bit is not included in calculation
      c.add(new qx::bin_ctrl(signbit,new qx::cphase(num_qubits-i-1,i))); // CZ first
   }
   for(int j=1; j<=i; ++j) {
      c.add(new qx::bin_ctrl(signbit,new qx::cphase(num_qubits-i-1+j,i))); // rotation of pi
      for(int k=1; k<=j; ++k) {
         c.add(new qx::bin_ctrl(signbit,new qx::swap(i+k,num_qubits-i-1+j))); // swap to prepare for CR
         c.add(new qx::bin_ctrl(signbit,new qx::ctrl_phase_shift(i+k,i))); // stack rotations to get to -phi
         c.add(new qx::bin_ctrl(signbit,new qx::swap(i+k,num_qubits-i-1+j))); // swap qubits back to their original position
      }
   }
}
// end of Inverse Addition

c.add(new qx::pauli_x(signbit)); // apply X gate to the sign of b
c.add(new qx::measure(signbit)); // measure to obtain sign of b (now because of X gate 0: -, 1: +)

// Addition: gates applied if sign of b is +
for(int i=0; i<num_qubits_a; ++i) {
   if(num_qubits-i-1 != signbit) { // sign bit is not included in calculation
      c.add(new qx::bin_ctrl(signbit,new qx::cphase(num_qubits-i-1,i))); // CZ first
   }
   for(int j=1; j<=i; ++j) {
      c.add(new qx::bin_ctrl(signbit,new qx::swap(i+j,num_qubits-i-1+j))); // swap to prepare for CR
      c.add(new qx::bin_ctrl(signbit,new qx::ctrl_phase_shift(i+j,i))); // CR for addition
      c.add(new qx::bin_ctrl(signbit,new qx::swap(i+j,num_qubits-i-1+j))); // swap qubits back to their original position
   }
}
// end of Addition

// apply Inverse QFT to a
for(int i=0; i<floor(num_qubits_a/2); ++i) {
   c.add(new qx::swap(i,num_qubits_a-1-i)); // swap qubits to prepare for CR
}
```

```
  // Start of inverse QFT
  for(int i=num_qubits_a-1; i>=0; --i) {
     for(int j=num_qubits_a-1; j>i; --j) {
        c.add(new qx::cphase(j,i));
        for(int k=1; k<= j-i; ++k) {
           c.add(new qx::swap(j,i+k)); // Swap qubit to position
           c.add(new qx::ctrl_phase_shift(i+k,i)); // Conrolled phase shift
               gate
           c.add(new qx::swap(j,i+k)); // Swap back
        }
     }
     c.add(new qx::hadamard(i)); // Hadamard
  }

  // Postprocessing
  for(int i=0; i<floor(num_qubits/2); ++i) {
     c.add(new qx::swap(i,num_qubits-1-i));
  }

  for(int i=0; i<(num_bits+1)*2; ++i) {
     c.add(new qx::measure(i)); // measure all qubits
  }
  c.execute(reg); // execute QX

  // calculate decimal value from binary representation
  int result = 0;
  for(int i=num_qubits-2; i>=num_qubits_b; --i) {
     int qvalue = reg.get_measurement(i);
     result += qvalue*pow(2,i-num_qubits_b);
  }
  if(reg.get_measurement(num_qubits-1) == 1) { // sign of a
     result -= pow(2,num_qubits_a-1); // determine sign
  }

  cout << "\na + b = " << result << "\n";


  c.dump();
  reg.dump();
}
```

B Microsoft LIQ$Ui\,|\rangle$ Code

B.1 Grover's Algorithm

Grover's algorithm for searching through an arbitrary quantum register.

```
let grover (qs:Qubits,correct_state:string,k:Ket) =
   let n = qs.Length // Length of qubit string (this is not the same n as in grovern)
   let R = floor(Math.PI/4.*sqrt(2.**(float(n)-1.))) // Optimal number of runs of Grover's iteration
   // Initialize q{n-1} (ancillary qubit) in |1>
   X [qs.[n-1]]
   // Hadamard gate on all qubits
   H >< qs
   // Grover's iteration (R times)
   for j in 1..int(R) do
      // X gates to qubits that are |0> in correct state
      let mutable i = 0
      for q in correct_state do
      let k = q |> int
      if k = 48 then X [qs.[i]] // q{i} |> int yields 48 if q{i} = |0> and 49 if q{i} = |1>
      i <- i+1
      Cngate(X,qs,n-1) // Multi-controlled Toffoli on ancillary qubit
      // X gates to qubits that are |0> in correct state
      let mutable i = 0
      for q in correct_state do
         let k = q |> int
         if k = 48 then X [qs.[i]] // q{i} |> int yields 48 if q{i} = |0> and 49 if q{i} = |1>
         i <- i+1
      H >< List.rev((List.rev(qs)).Tail) // Hadamard on all qubits except ancillary qubit
      X >< List.rev((List.rev(qs)).Tail) // X on all qubits except ancillary qubit
      Cngate(Z,List.rev((List.rev(qs)).Tail),n-2) // Multi-controlled Z gate on last state qubit q{n-2}
      X >< List.rev((List.rev(qs)).Tail) // Hadamard on all qubits except ancillary qubit
      H >< List.rev((List.rev(qs)).Tail) // X on all qubits except ancillary qubit
   M >< qs
```

B.2 QFT

The QFT for an arbitrary quantum register.

```
let QFT(qs:Qubits) =
   let n = qs.Length
   for i in 1..n-1 do
      let q = n-i
      H [qs.[q]]
      for k in 2..(q+1) do
      Cgate (R k) [qs.[q+1-k];qs.[q]]
   H [qs.[0]]
   let nswaps = int(floor (float(n-2)/float(2)))
   for q in 0..nswaps do
      SWAP !!(qs,q,n-1-q)
```

The inverse QFT for an arbitrary quantum register.

```
let invQFT(qs:Qubits) = // This is not really optimal
   let n = qs.Length
   for i in 1..n-1 do
      let q = n-i
      H [qs.[q]]
      for k in 2..(q+1) do
         for j in 1..k do
            Cgate (R j) [qs.[q+1-k];qs.[q]] // This is not efficient, it should
                be a clockwise rotation
   H [qs.[0]]
   let nswaps = int(floor (float(n-2)/float(2)))
   for q in 0..nswaps do
      SWAP !!(qs,q,n-1-q)
```

B.3 Addition

```
let ADDSUB(qs:Qubits,n:int,k:Ket) =
   let n_a = int(float(n)/float(2))
   let mutable i = 0
   let mutable j = i+1
   // QFT
   for i in 0..n_a-1 do
      H [qs.[i]]
      for j in i+1..n_a-1 do
         Cgate (R (j-i+1)) [qs.[j];qs.[i]]
   let nswaps = int(floor(float(n_a)/float(2)))
   for i in 0..nswaps-1 do
      SWAP !!(qs,i,n_a-1-i)
   M [qs.[n_a]]
   if qs.[n_a].Bit.v = 1 then // subtraction
      for i in 0..n_a-1 do
         if n-i-1 <> n_a then
         CZ [qs.[n-i-1];qs.[i]]
      for j in 1..i do
         CZ [qs.[n-i-1+j];qs.[i]]
         for k in 1..j do
            Cgate (R (k+1)) [qs.[n-i-1+j];qs.[i]]
   else // addition
      for i in 0..n_a-1 do
         if n-i-1 <> n_a then
            CZ [qs.[n-i-1];qs.[i]]
   for j in 1..i do
      Cgate (R (j+1)) [qs.[n-i-1+j];qs.[i]]
   // Inverse QFT
   for i in 0..nswaps-1 do
      SWAP !!(qs,i,n_a-1-i)
   for i in n_a-1..0 do
      for j in n_a-1..i+1 do
         CZ [qs.[j];qs.[i]]
         for k in 1..j-i do
            Cgate (R (k+1)) [qs.[j];qs.[i]]
      H [qs.[i]]
```