Submitted in part fulfilment for the degree of MEng.

Quantum Algorithm Synthesis Workbench

Sam Ratcliff

15th March 2011

Supervisor: John A Clark

Number of words = 8832, as counted by wc -w.
This includes the body of the report only.

Abstract

TO BE DONE

Contents

List of Figures

List of Tables

1 Introduction

2 Literature Review

2.1 Introduction to Quantum Computation

In 1980, Richard Feynman noted 'it is impossible to represent the results of quantum mechanics with a classical universal device'[3]. This statement was a seed for interest in the field of Quantum Computation. The true power of quantum computation was not initially realised. -The discovery of a quantum algorithm by David Deutsch[4] in 1985 that performed better than a classical computer was the first glimpse of the potential power provided by harnessing quantum mechanics. However, with slow progress of research into both their implementation and algorithms, the energy behind the research started to decrease. It would take a discovery by Peter Shor[5] to reignite the excitement surrounding the subject.

In classical computers the computation is performed using the discrete values of 0 and 1. These values are indicated by +5V and 0V signals propagating round circuits. A signal can only be 0 or 1, there is no in between value. Each signal can indicate the value of a single 'bit' of data. A combination of n bits can be used to represent a number from 0 to $2^n - 1$, an n-bit number. Classical computation works through the manipulation of these n-bit numbers.

Quantum Computation uses the properties of quantum mechanics to perform computation. The power of quantum computers come from the use of particles in superpositions. Qubits are the quantum equivalent of the classical bit. It is these qubits which can be placed into superpositions. Just as classical computers manipulate bits to perform computation, quantum computers manipulate qubits and their superpositions to perform computation. The power of these superpositions is not obviously apparent.

It is not possible to observe the superposition of a particle. When observed the superposition 'collapses' to either logical 0 or 1, the basis states. The probability the the superposition collapses to 0 is determined by the superposition's properties.

To write the state of a superposition it is usual to use the 'Bra-Ket' notation, introduced by Dirac[6]. A 'Ket' is mathematical notation, $|a\rangle$, which represents a basis function of the respective Hilbert space, $\mathcal{H}$, as a column vector.

$$|a\rangle = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \end{pmatrix} \tag{2.1}$$

Hilbert spaces extend the simple Euclidean vector space into a potential infinite dimension function space. A quantum state space can also been visualised in terms of the Bloch sphere, shown in Figure 2.1. The shown Bloch sphere is for a single qubit system, it can be extended to an n-qubit system however the visualisation breaks down. All 'pure' quantum states can be described using the Bloch sphere and all exist on the surface produced by the unit sphere. In this report only pure quantum states will be used and all explanation of quantum states are more precisely explanations of 'pure' quantum states. This means that all superpositions of states can be expressed in terms of $|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle$ with $0 \leq \theta \leq \pi, \quad 0 \leq \phi \leq 2\pi$, ignoring global phase factors[7].

In quantum mechanics, Kets are used to indicate a state, for example $|0\rangle$ is the state of a logical 0 whereas $|1\rangle$ is the state of logical 1. Using this notation and the inclusion of probabilities, the state of the superposition can be expressed.

A dual to the Ket notation is the 'Bra' notation, $\langle a|$. This notation is used to denote the 'dual vector'
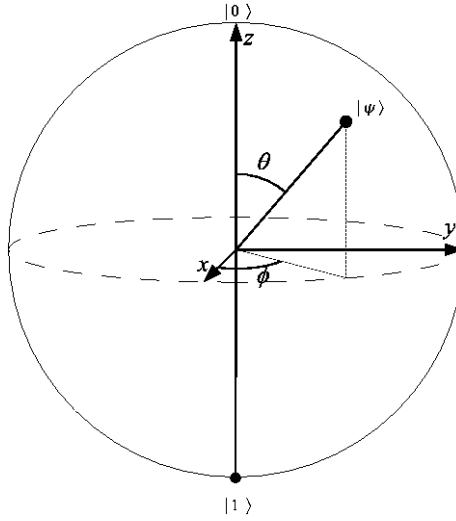
Figure 2.1: The 1-Qubit Bloch Sphere [1]

of the corresponding Ket. For a state vector represented by

$$|a\rangle = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \end{pmatrix} \tag{2.2}$$

there is a dual vector representing its Hermitian conjugate

$$\langle a| = \begin{pmatrix} a_1^* a_2^* a_3^* \cdots \end{pmatrix} \tag{2.3}$$

Combining the two vectors $\langle a|$ and $|b\rangle$, written $\langle a||b\rangle$, represents the inner product of the two vectors. If a and b are unit vectors and $a == b$, $\langle a||b\rangle == 1$. If a and b are orthogonal, $\langle a||b\rangle == 0$.

The outer product of two vectors, a and b, can be represented by $|a\rangle\langle b|$. This represents the transformation from a to b. It can also be represented in matrix form.

With $|0\rangle == \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $|1\rangle == \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ it is possible to represent 1-qubit operations in the bra-ket notation. For example, the NOT gate performs a simple negation of a qubit's value. This can be written as $|0\rangle\langle 1| + |1\rangle\langle 0|$. Substituting in the vector values we have

$$\begin{aligned} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \begin{pmatrix} 0 & 1 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \end{pmatrix} &= \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \end{aligned} \tag{2.4}$$

This matrix can be seen as a transformation matrix for the NOT operation. In quantum computation, the NOT gate is one of the 4 gates known as the Pauli gates, more specifically the Pauli-X gate. It is called the Pauli-X gate as it can be seen as a rotation of π radians about the X axis of the Bloch sphere, Figure 2.1.

The matrices representing the Pauli-X gate and all other quantum logic gates are unitary. A unitary matrix, U, is one which adheres to

$$U * \dagger U == UU * \dagger = I_N \tag{2.5}$$

where I_N is the identity matrix in N dimensions and $U * \dagger$ is the complex conjugate of U. The implication of all quantum logic operations being unitary is that they are reversible, this is a difference to classical computation. With many classical logic gates irreversible, there is not a set of quantum logic gates which is as computationally powerful as the set of classical logic gates. This seems

like a major issue, quite the contrary. The set of classical logic gates can be replaced by reversible equivalents and therefore it is possible to produce a set of quantum logic gates with the equivalent computational power as the classical logic gates.

As with all probabilities, the overall probability of a superposition collapsing to any of the states it contains must equal 1.

$$\alpha|0\rangle + \beta|1\rangle \tag{2.6}$$

$$\frac{1}{2^{\frac{1}{n}}} \sum_{i=0}^{N} |x_i\rangle \tag{2.7}$$

Equation (2.6) is how the combination of the logical 0 and 1 states in a superposition can be represented for a single particle. It can also be represented in the for of Equation (2.6). This is equivalent to the representation provided by the Bloch sphere, Figure 2.1. The probability of this state collapsing to the basis state 0 can be calculated by $|\alpha|^2$ where α is a complex number. α is known as the probability amplitude of $|0\rangle$. Similarly, the probability of this state collapsing to the basis state 1 can be calculated by $|\beta|^2$, β being the probability amplitude of $|1\rangle$. It follows that $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$ is an equal superposition where the collapse to 0 is just as likely as collapsing to 1. This provides the first glimpse of where a single qubit has the ability to perform a function not currently possible on a classical computer. With n-qubits in the equal superposition, we have n binary values which have an equal probability of taking the value 0 as the value 1. With an ordering decided of these qubits, collapsing the superposition of each qubit will result in a binary value of length n. With all probabilities being $\frac{1}{2}$ this binary value takes a truly random value between 0 and $2^n - 1$. It is not possible to produce a truly random number using a classical computer.

A second indication of the power held within the idea of superposition becomes clear if we look at the n-qubits in their equal superposition. In 1935, Erwin Schröginger[8] proposed a thought experiment to explain the idea of superposition. Imagine a cat in a fully opaque box with a vile of poison. The vile may break at any time, a truly random variable. After sealing the box the state of the cat is not known. The cat could be alive if the vile has not broken but could just as likely be dead. Only by looking inside the box will the state of the cat be known. Until this time the cat could be thought of as both alive and dead at the same time. If we assign 'dead' to the state $|0\rangle$ and 'alive' to the state $|1\rangle$ the situation looks very similar to the state we have previously seen. Therefore, just as the cat can be thought of as both dead and alive at the same time, a qubit in the superposition $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$ can be thought of as both 0 and 1 at the same time. This leads to a very powerful property of quantum computers. With n classical bits, a single number in the range 0 to $2^n - 1$ can be expressed at any one time. With n quantum qubits, every number in the range 0 to $2^n - 1$ can be expressed at any one time. This effectively allows computation over the whole range of 2^n inputs to be carried out in parallel.

This parallelism is very powerful and has been shown to enable the computation of problems classified as NP to be performed in polynomial time. This does however have a caveat. As mentioned previously the superposition cannot itself be observed or measured. When observed the superposition collapses to a basis state with respect to the superposition probability amplitudes. This means that even though 2^n calculations can be performed in parallel, only a single answer can be observed.

Along with the Pauli-X gate, there are an additional 3 Pauli gates. The Pauli-I gate is the simplest of all quantum gates. It is the identity gate, the output is identical to the input. In Dirac notation this is $|0\rangle\langle 0| + |1\rangle\langle 1|$, and in matrix form below

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$$
$$= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{2.8}$$

The Pauli-Z gate is similar to the Pauli-X gate, but differs in the axis about which it performs the rotation. The Pauli-Z gate rotates the quantum state by π radians about the Z axis. This represents

0	0	0
0	1	1
1	0	1
1	1	0

Table 2.1: Classical CNOT Truth Table

(a) Basic Circuit

(b) Full Circuit
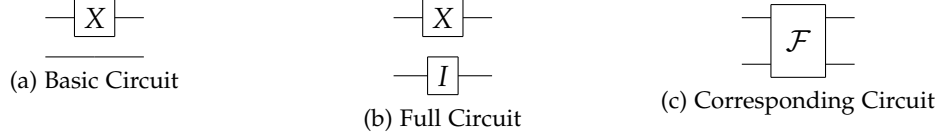
(c) Corresponding Circuit

Figure 2.2: Single Qubit Gate, 2 Qubit Circuit

a phase flip of the quantum state. The phase of a state is important when interference is used in computation. In Dirac notation this is $|0\rangle\langle0| + |1\rangle(-\langle1|)$, and in matrix form below

$$
\begin{aligned}
\begin{pmatrix} 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} \begin{pmatrix} 0 & -1 \end{pmatrix} &= \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & -1 \end{pmatrix} \\
&= \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}
\end{aligned}
\tag{2.9}
$$

The Pauli-Y gate is similar to both the Pauli-X and Pauli-Z gates, but differs in the axis about which it performs the rotation. The Pauli-Y gate rotates the quantum state by π radians about the Y axis. This represents a phase flip followed by a bit flip. In Dirac notation this is $|0\rangle(i\langle1|) + |1\rangle(-i\langle0|)$, and in matrix form below

$$
\begin{aligned}
\begin{pmatrix} 1 \\ 0 \end{pmatrix} \begin{pmatrix} 0 & -i \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} \begin{pmatrix} i & 0 \end{pmatrix} &= \begin{pmatrix} 0 & -i \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ i & 0 \end{pmatrix} \\
&= \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}
\end{aligned}
\tag{2.10}
$$

Along with the single qubit operations, like those above, there are operations which can act over n-qubits. A simple example of a 2 qubit operation is the controlled-NOT, CNOT, operator. This is a simple extension of the Pauli-X gate. The CNOT gate has a control input which it requires to be in the logical 1 state for the NOT operation on the second input to be carried out. In classical logic this would extend the truth table to be as shown in Table 2.1. The truth table of the CNOT gate is the same as that of the XOR gate. The Dirac notation of the CNOT gate is $|00\rangle\langle00| + |01\rangle\langle01| + |10\rangle\langle11| + |11\rangle\langle10|$.

For a 2 qubit circuit, the state has 4 elements in the state vector. This means that to apply a single qubit gate to this state, multiply the state by the respective $2 - by - 2$ unitary matrix, there will be a problem due to the matrix dimensions not matching. The matrix of the single qubit gate has to be expanded to a $4 - by - 4$ matrix so that it can be applied to the state vector. Taking the circuit shown in Figure 2.2a as an example we can see that the it is a simple Pauli-X gate, bit-flip operator, acting on the higher order qubit while nothing occurs to the lower order qubit. Nothing happening to the lower order qubit is the same as applying the Pauli-I gate, identity operator, to this qubit. The circuit in Figure 2.2b shows the resulting circuit with identical effect on all possible input states. This ciruit still doesn't solve the issue as there are now $2 - by - 2$ matrices for each of the two gates. These can't be applied to the state individually for that same reason as before.

To produce a $4 - by - 4$ matrix for the circuit, the two $2 - by - 2$ matrices have to be combined. The matrix operation that is used for this is the Tensor Product. The tensor produce is a simple matric operation that is denoted using the $\otimes$ symbol. The result of the tensor product operation is shown in Equation 2.11.

$$A \otimes B = \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} \otimes \begin{pmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{pmatrix} = \begin{pmatrix} a_{00}B & a_{01}B \\ a_{10}B & a_{11}B \end{pmatrix}$$

$$= \begin{pmatrix} a_{00}b_{00} & a_{00}b_{01} & a_{01}b_{00} & a_{01}b_{01} \\ a_{00}b_{10} & a_{00}b_{11} & a_{01}b_{10} & a_{01}b_{11} \\ a_{10}b_{00} & a_{10}b_{01} & a_{11}b_{00} & a_{11}b_{01} \\ a_{10}b_{10} & a_{10}b_{11} & a_{11}b_{10} & a_{11}b_{11} \end{pmatrix} \quad (2.11)$$

So when applying this to the matrices of Pauli-X as A and the Pauli-I as B we get the unitary matrix for the corresponding $\mathcal{F}$ gate. This can be seen in Equation 2.12.

$$\mathcal{F} = X \otimes I = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0B & 1B \\ 1B & 0B \end{pmatrix}$$

$$= \begin{pmatrix} 0x1 & 0x0 & 1x1 & 1x0 \\ 0x0 & 0x1 & 1x0 & 1x1 \\ 1x1 & 1x0 & 0x1 & 0x0 \\ 1x0 & 1x1 & 0x0 & 0x1 \end{pmatrix}$$

$$= \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \quad (2.12)$$

This use of the tensor product is scalable to any number of qubits and therefore can produce $2^n - by - 2^n$ matrices for systems involving x qubits. The tensor product cannot however be used to produce unitary matrices for controlled gates. It can only be used when the action on each qubit is independent.

2.2 An Introduction to Quantum Algorithms

Just as with classical computers, the computation to produce the required output given inputs is given in the form of an algorithm. Quantum algorithms can be constructed in several ways. These definitions are based on those provided by Massey[9].

- A quantum circuit can be used to represent an algorithm at the level of quantum logic gates. This is similar to a specific purpose circuit diagram for classical systems.

- A quantum program is a representation of the algorithm in some higher level quantum 'programming' language which would generate the required circuit. The circuit generated is not defined in this method, just it's behaviour. This could be seen as slightly more flexible that the quantum circuit model as the 'compiler' can be updated to reflect the findings of future research.

- A parameterisable quantum algorithm is a representation in pure Pseudo-code. It proves the flexibility of changing some value n, which is used to indicate the number of input qubits, to produce quantum circuits or programs with the desired behaviour on n qubits. This is the most flexible construction of quantum algorithms. It can cope with the changing in input size and can use findings of research just like in the 'compilation' of a quantum program.

Currently there are very few quantum algorithms known. Peter Shor has carried out, and published, a discussion on the progress made 'in discovering algorithms for computation on a quantum computer'[10]. Shor suggests two possible reasons for the lack of quantum algorithms. The first is 'that there might really be only a few problems for which quantum computers can offer a substantial speed-up over classical computers'[10]. This would indeed make the discovery of useful

$$|0\rangle \quad \boxed{H} \quad \boxed{\mathcal{F}} \quad \boxed{H} \quad \text{(meas)}$$
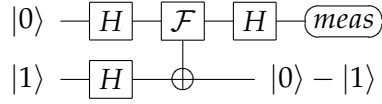$$|1\rangle \quad \boxed{H} \quad \oplus \quad\quad |0\rangle - |1\rangle$$

Figure 2.3: Deutsch Circuit

quantum algorithms difficult. However, I feel this is somewhat pessimistic. The main focus of the paper published by Feynman[3] was the problem of simulating the physics of quantum mechanics on a classical computer. This to suggests there would be the potential of many applications of quantum computers, even if they aren't analogous to the classical computational applications.

The second is 'that quantum computers operate in a manner so non-intuitive, and so different from classical computers'[10] that our current algorithm knowledge is close to useless. This, in my opinion, is a much more believable obstacle. Quantum mechanics is seen by many as a confusing and mystical subject. Even prize winning mathematician and physicist Roger Penrose is attributed to the remark 'Quantum mechanics makes absolutely no sense'. Statements like this and the atmosphere surrounding quantum mechanics makes the potential of its study more than some what daunting. As computer scientists, the exposure to and therefore our understanding of quantum mechanics is limited, in general. With this in mind it is, currently, unreasonable to expect the discovery of algorithms that exploit the finer details of this complex and subtle theory to become an everyday occurrence.

The following few sections outline a selection of the currently known quantum algorithms.

2.2.1 Deutsch Algorithms

The original Deutsch algorithm[4] was proposed to solve the following problem:

> Given a function $f : \{0,1\} \to \{0,1\}$, decide whether it is either a balanced, or constant function. The function $f(x)$ is guaranteed to be either constant or balanced.

The algorithm's major breakthrough was that it only required a single invocation of the function $f(x)$ to decide on which category it belonged. This is compared to the best classical approach requiring 2 invocations to the function. This was the first algorithm that it was possible to compute the solution to a problem, provably, more efficiently than a classical computer by exploiting quantum mechanics.

The circuit for this algorithm is presented in Figure 2.3. The maths for the way in which the state evolves is below

$$|01\rangle \xrightarrow{H \otimes H} \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \tag{2.13}$$

Using the identity $(|b\rangle - |a\rangle) \equiv -1(|a\rangle - |b\rangle)$, if $f(x)$ evalues to 0 the state transformation is

$$|x\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \xrightarrow{f} |x\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

Wheras if $f(x)$ evaluates to 1 the state transformation is

$$|x\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \xrightarrow{f} - |x\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

These two equations can be generalised to give

$$|x\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \xrightarrow{f} (-1)^{f(x)} |x\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

This is due to $(-1)^0 = 1$ and $(-1)^1 = -1$ which is the way the state evolves with the outputs of $f(x)$. Using this we can continue to analyse how the state evolves for the superposition $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$.

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \xrightarrow{f} \frac{1}{\sqrt{2}}((-1)^{f(0)} |0\rangle + (-1)^{f(1)} |1\rangle)\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$
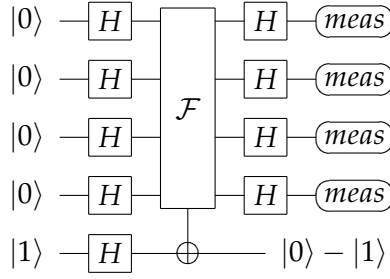
Figure 2.4: Deutsch-Jozsa Circuit

$$\frac{1}{\sqrt{2}}((-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle)\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \equiv (-1)^{f(0)}\frac{1}{\sqrt{2}}(|0\rangle + (-1)^{f(0)\oplus f(1)}|1\rangle)\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

This penultimate state, before the last hadamard, can be simplified. Global phase factors cannot be observed so the $(-1)^{f(0)}$ can be ignored. The second qubit can also be ignored as it is not entangled with the first, nor is it ever measured.

$$\frac{1}{\sqrt{2}}(|0\rangle + (-1)^{f(0)\oplus f(1)}|1\rangle) \tag{2.14}$$

This results in the penultimate state being able to be represented as 2.14. With this as the state we can reason about the output if the function $f(x)$ is balanced and if it is constant. If f were balanced, $f(0) \oplus f(1) = 1$ due to either $f(0)$ or $f(1)$ evaluating to 1 but not both. If f were constant, $f(0) \oplus f(1) = 0$ due to either $f(0)$ and $f(1)$ both evaluating to 0 or both evaluating to 1. Using these two observations we can see the penultimate state, 2.14, when f is constant and when f is balanced. When f is balanced $\frac{1}{\sqrt{2}}(|0\rangle + (-1)^{f(0)\oplus f(1)}|1\rangle) \equiv \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. When this is passed through the final hadamard $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \xrightarrow{H} |1\rangle$. When f is constant $\frac{1}{\sqrt{2}}(|0\rangle + (-1)^{f(0)\oplus f(1)}|1\rangle) \equiv \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. However, when this is passed through the final hadamard $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \xrightarrow{H} |0\rangle$. This means that using just a single query to the function $f(x)$, the algorithm can provide an answer to the problem.

2.2.2 Deutsch-Jozsa Algorithm

The Deutsch-Jozsa algorithm[11] is a generalisation and improvement over an earlier Deutsch algorithm[4]. The algorithm described here will be the algorithm including the improvements published in [12], the resulting algorithm is still referred to as the Deutsch-Jozsa algorithm.

In 1992, David Deutsch and Richard Jozsa[11] presented an extension to the Deutsch algorithm, Section 2.2.1, that allowed for functions $f : \{0,1\}^{2n} \to \{0,1\}$ to be categorised as constant or balanced. The algorithm was probabilistic but also required 2 invocations to the function $f(x)$. This means that in the worst case, with $f_1 : \{0,1\} \to \{0,1\}$, the algorithm will perform worst than both the original Deutsch algorithm[4] and the classical algorithm due to the probabilistic nature of its result. This is a very limited case and performance improves as the value of n increases. As with the original algorithm, the number of invocations of $f(x)$ is constant, truly independent of both n and $f(x)$. This is again an improvement over the classical algorithm which requires in the worst case $2^{n-1} + 1$ to be certain of the function's classification.

The algorithm requires n input qubits and a single control qubit. The n input qubits are initialised to $|0\rangle$. The control qubit is initialised to $|1\rangle$. The n-fold Hadamard gates are then used to produce the superposition of all $0..2^n - 1$ possible inputs, $x = (|0\rangle + |1\rangle)^n$. The Hadamard gate on the control qubit is used to produce the superposition $|0\rangle - |1\rangle$. The difference in superpositions between the input and control qubits is important to the way in which the algorithm classifies a function. This produces a state $(|0\rangle + |1\rangle)^n(|0\rangle - |1\rangle)$.

The result of $f(x)$ is used as the control for a CNOT gate acting on the control qubit. Remember at this point that the input x is in effect all possible inputs to $f(x)$, and as such the output is all the respective outputs. This means that each of the factors of the input are transformed, based on the

action of $f(x)$. This can be formalised as $U_f(|x\rangle, |y\rangle) = (|x\rangle, |f(x) \oplus y\rangle)$. Using the final Hadamard gates we can use the transformation to categorise $f(x)$.

If we assume n to be equal to 2, then after the initial Hadamard gates we have the superposition:

$$\frac{1}{2} |\Psi_{init}\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

$$|a\rangle (|c\rangle - |b\rangle) \equiv |a\rangle (-1(|b\rangle - |c\rangle)) \equiv -1 |a\rangle (|b\rangle - |c\rangle) \tag{2.15}$$

The CNOT on the target qubit is only activated if $f(x) = 1$ which, using the equivalent above, produces the superposition $-1(|0\rangle - |1\rangle)$. This can also be generalised as $(-1)^{f(x)}(|0\rangle - |1\rangle)$ where $-1^1 = -1$ and $-1^0 = 1$ by definition. By linearity the superposition after the CNOT controlled by f(x) on the target qubit becomes

$$\frac{1}{2}((-1)^{f(00)} |00\rangle + (-1)^{f(01)} |01\rangle + (-1)^{f(10)} |10\rangle + (-1)^{f(11)} |11\rangle)\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

After the application of $f(x)$, the target bit has served it purpose and is neither measured nor entangled with any of the qubits from the n qubit input. To simplify the equations I will now ignore the $(\frac{1}{\sqrt{2}} |0\rangle - |1\rangle)$ contributed by the target qubit, the remaining state will be refereed to as Ψ. If the function $f(x)$ is constant:

$$|\Psi_{const}\rangle = \pm\frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$$

Whereas if $f(x)$ is balanced then:

$$|\Psi_{bal}\rangle = (-1)^{f(00)} |00\rangle + (-1)^{f(01)} |01\rangle + (-1)^{f(10)} |10\rangle + (-1)^{f(11)} |11\rangle$$

However,as $|\Psi_{const}\rangle$ and $|\Psi_{bal}\rangle$ are orthogonal we can use this to detect if the function is balanced or constant.

$$\langle\Psi_{bal}| |\Psi_{const}\rangle = 0$$

Applying the n Hadamard gates to the state Ψ_{const} produces the state $\pm |0\rangle^n$. When measured this will return the result 0 as the global phase factor $\pm$ cannot be observed. As we have noted, Ψ_{const} is orthogonal to Ψ_{bal} so when the n Hadamard gates are applied, as they preserve orthogonality, the measured result will be anything orthogonal to 0. This gives us a clear way of distinguishing between whether f is constant of balanced with certainty and only a single invocation of f. If the measurement returns 0 then $f(x)$ is constant, if the measurement returns anything else $f(x)$ is balanced.

2.2.3 Grover's Search Algorithm

The Grover Search algorithm[13] is an unstructured search problem. The algorithm assumes no underlying structure in the search space. By this I mean the algorithm does not exploit, and therefore assume, any structure, such as sorting, in the data set being searched.

It has previously been proven[14] that the lower complexity limit for any algorithm identifying an element without knowledge of underlying structure in the data is $\Omega(\sqrt{N})$. For simplicities sake we assume $N = 2^n$. The complexity is measured by the number of elements which need to be queried in order to find the desired element. The Grover Search algorithm has the complexity $O(\sqrt{N})$ and so 'is within a constant factor of the fastest possible quantum mechanical algorithm'[13].

The mechanisms used within the algorithm to produce a solution to the problem is more subtle than those used but the Deutsch-Jozsa algorithm. The algorithm does not perform the computation in a single step. The algorithm requires $O(\sqrt{N})$ steps. The section of circuit which is repeated is that shown in Figure 2.5.

The algorithm is to initialise the state, Ψ_1, using n Hadamard gates.

$$\Psi_1 = \frac{1}{2^{\frac{1}{n}}} \sum_{i=0}^{N} |x_i\rangle (|0\rangle - |1\rangle)$$
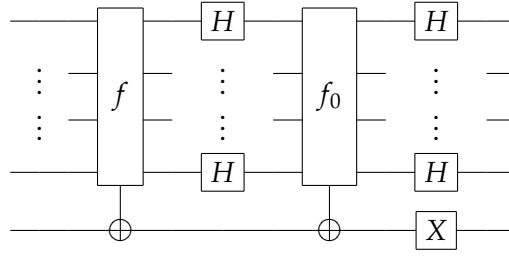
Figure 2.5: Grover's Search Circuit

The application of $f(x)$ is used in an analogous way to in the Deutsch-Jozsa algorithm. It can again be written as $U_f(|x\rangle, |y\rangle) = (|x\rangle, |f(x) \oplus y\rangle)$ and remember

$$\exists x_i \epsilon \{x_0, x_1, \ldots, x_{n-1}\} | f(x_i) = 1$$

$$\forall x_j : i \neq j : f(x_j) = 0$$

Just as in the Deutsch-Jozsa algorithm, the result of a 1 produces a bit flip on the target qubit. Using the identity in Equation 2.15 this flips the sign on the amplitude associated with x_i, where $f(x_i) = 1$. The state Ψ_1 is transformed by U_f to Ψ_2.

$$\Psi_2 = \frac{1}{2^{\frac{1}{n}}} \sum_{j=0 \wedge j \neq i}^{N} |x_j\rangle (|0\rangle - |1\rangle) - \frac{1}{2^{\frac{1}{n}}} |x_i\rangle (|0\rangle - |1\rangle)$$

The second function in the circuit, $f_0(x)$, is a fixed function. It is not dependant on $f(x)$ but is a function which evaluates to 1 only when the input is $|0\rangle$. This means that given a simple state $(\alpha |0\rangle + \beta |1\rangle + \ldots)$ the result is the state $(-\alpha |0\rangle + \beta |1\rangle + \ldots)$. The action of both these function can be written in the shorter and much simpler Dirac notation.

$$U_f = I - 2 |x_i\rangle \langle x_i|$$

$$U_0 = I - 2 |0\rangle \langle 0| \tag{2.16}$$

Functions $f(x)$ and $f_0(x)$ seem relatively unimpressive and don't appear to solve the problem. The importance of the two sets of n Hadamard gates, one each side of $f_0(x)$, is paramount. The effect they have on the action of $f_0(x)$ is shown below, the action of $f_0(x)$ and the Hadamard gates will be represented as V.

$$V = H^{\otimes n} U_0 H^{\otimes n}$$

$$V = H^{\otimes n} (I_n - 2 |0\rangle \langle 0|) H^{\otimes n}$$

$$V = H^{\otimes n} I_n H^{\otimes n} - H^{\otimes n} (2 |0\rangle \langle 0|) H^{\otimes n}$$

$$V = I_n - 2(H^{\otimes n} |0\rangle)(\langle 0| H^{\otimes n})$$

$$V = I_n - 2(H^{\otimes n} |0\rangle)(H^{\otimes n} |0\rangle)^\dagger$$

The application of Hadamard gates to the $|0\rangle$ state is the method of creating the superposition of all $2^n - 1$ possible states.

$$V = I_n - 2(\frac{1}{2^{\frac{1}{n}}} \sum_{i=0}^{N} |x_i\rangle)(\frac{1}{2^{\frac{1}{n}}} \sum_{i=0}^{N} |x_i\rangle)^\dagger$$

$$V = I_n - 2(\frac{1}{2^{\frac{1}{n}}} \sum_{i=0}^{N} |x_i\rangle)(\frac{1}{2^{\frac{1}{n}}} \sum_{i=0}^{N} \langle x_i|)$$

$$V = I_n - 2 |\Psi\rangle \langle \Psi|$$

Just as with U_f, V can be seen as a simple phase flip. However, the subtlety of this operator is that the flip is not in the computational basis, but in the basis described by $|\Psi\rangle$.

The last gate in Figure 2.5 is the Pauli-X operator. This is not functional but for convenience of mathematics as produces a global phase flip of $|\Psi\rangle$ which makes the mathematics simpler.

That is Grovers algorithm. Looking at the circuit and the mathematics of the operators it doesn't provide an obvious answer as to how it solves the search problem. This is partly due to the fact the circuit in Figure 2.5 has to be repeated roughly $\frac{\pi\sqrt{N}}{4}$ times and partly due to the effect of the circuit being hidden in implementation. The circuit is actually little more than a complex rotation gate. It is however more sophisticated than a standard rotation gate as it computes the direction to rotate the state so as to solve the problem. The power of this does not initially appear as immense as it possibly should. The Hilbert space in which this circuit is operating is of the order N, we as humans can only accurately imagine a maximum of a 3 dimensional space as it the most dimensions we can observe directly. Taking into account the vast Hilbert space when assessing this circuit produces a much better appreciation its power.

However, even though the power can now be appreciated, the way in which it solves the problem is still not clear.

2.2.4 Shor's Factorisation Algorithm

In 1994, Peter Shor astonished the computer science community with a quantum factorisation algorithm[5]. This allowed the factorisation of integers into their constituent primes in polynomial time. This algorithm is not a pure quantum algorithm, but a hybrid algorithm. It has both a quantum and classical portion.

The algorithm utilises the complex nature of the probability amplitudes. This means that the relative phase of the states is important. To explain Shors algorithm it is necessary to introduce the sum of complex roots of unity.

$$\sum_{j=0}^{N-1} z^j = \frac{1-z^N}{1-z} \tag{2.17}$$

Equation 2.17 provides a simplification of the sum of the geometric series $\sum_{j=0}^{N-1} z^j$. This simplification can be applied to both real and complex values of z, including the roots of unity. Taking $w = e^{\frac{2\pi i}{N}}$ to be the N-th root of unity, the geometric series $\sum_{i=0}^{N-1} w^{xy}$ can be simplified using Equation 2.17.

When $x = 0$

$$\sum_{y=0}^{N-1} w^{xy} = \sum_{y=0}^{N-1} w^0 = \sum_{y=0}^{N-1} 1 = N \tag{2.18}$$

However, when $x \neq 0$

$$\sum_{y=0}^{N-1} w^{xy} = \sum_{y=0}^{N-1} (w^x)^y = (\frac{1-w^N}{1-w})^y = 0^y = 0 \tag{2.19}$$

The result of this is generally described using a Kronecker delta

$$\sum_{y=0}^{N-1} w^{xy} = N \times \delta_{x,y} = N \times \begin{cases} 1, & \text{if } x = 0 \\ 0, & \text{if } x \neq 0 \end{cases} \tag{2.20}$$

This produces a very neat and convenient explanation of such a series. As the sum is of a root of unity raised to a power, it is a periodic function $w^0 = w^N = 1$. This holds true in Equation 2.21 when $x = 0, \pm N, \pm 2N, \ldots$. With this the Kronecker delta can be expressed in terms of modular arithmetic.

$$\sum_{y=0}^{N-1} e^{\frac{2\pi i xy}{N}} = N \times \delta_{x,y(modN)} = N \times \begin{cases} 1, & \text{if } x = 0 \\ 0, & \text{if } x \neq 0 \end{cases} \tag{2.21}$$
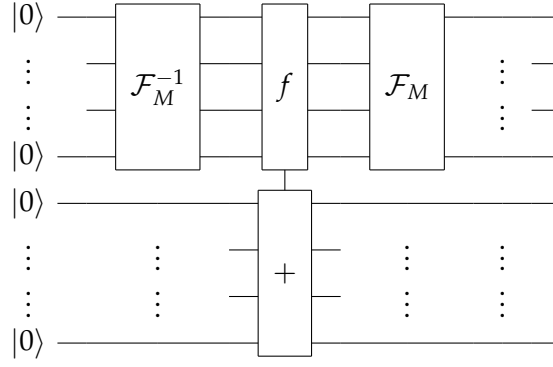
Figure 2.6: The circuit of Shor's algorithm

The quantum section of Shor's algorithm is a period finding function. The classical part then uses this period and number theory to deduce the factors. In the explaination below, the value that is trying to be factorised is N and the value M is a value which is larger than N. The use of the Kronecker delta, and understanding on the effect it has is vital to the explanation of the quantum section of Shor's algorithm.

The circuit in Figure **??** is the period finding section of Shor's Algorithm. The function f is a periodic function, whose period we want to find. $\mathcal{F}$ and $\mathcal{F}^{-1}$ are the Quantum Fourier Transform and the inverse, respectively.

$$\mathcal{F}_N \ket{x} = \frac{1}{\sqrt{N}} \sum_{y=0}^{N-1} e^{\frac{2\pi i x y}{N}} \ket{y} \tag{2.22}$$

$$\mathcal{F}_N^{-1} \ket{y} = \frac{1}{\sqrt{N}} \sum_{z=0}^{N-1} e^{\frac{-2\pi i x y}{N}} \ket{z} \tag{2.23}$$

The action of applying the Quantum Fourier Transform and the inverse can be seen in Equations 2.22 and 2.23 respectively.

The way in which this circuit computes the period is not initially apparent. The central section of the circuit, involving the periodic function, performs the operation $U_f \ket{x} \ket{y} = \ket{x} \ket{f(x) + y}$ in the computaional basis. However, it is preceded by the inverse Quantum Fourier Transform. The application of the inverse Quantum Fourier Transform on the initial state is as follows:

$$\ket{0}\ket{0} \rightarrow \frac{1}{\sqrt{M}} \sum_{x=0}^{M-1} e^{\frac{-2\pi i x.0}{M}} \ket{x}\ket{0}$$

$$\frac{1}{\sqrt{M}} \sum_{x=0}^{M-1} \ket{x}\ket{0}$$

This produces a uniform superposition, just as a series of Hadamard gates would have produced. The application of $f(x)$ to this superposition produces the state:

$$\frac{1}{\sqrt{M}} \sum_{x=0}^{M-1} \ket{x}\ket{f(x)}$$

Applying the final Quantum Fourier Transform manipulates this state quite substantially.

$$\frac{1}{\sqrt{M}} \sum_{x=0}^{M-1} \ket{x}\ket{f(x)} \rightarrow \frac{1}{M} \sum_{y=0}^{M-1} \sum_{x=0}^{M-1} e^{\frac{2\pi i x y}{M}} \ket{y}\ket{f(x)} \tag{2.24}$$

The final state is Equation 2.24. This does not initially appear particularly useful or interesting. However, remembering that $f(x)$ is periodic, we can separate and analyse a single branch of the

superposition. With $f(x) = f(x + r) = f(x + 2r) \ldots$ where r is the period of $f(x)$ the amplitude of the branch $|y\rangle \, |f(x)\rangle$, with $x = x_0 + mr \, for \, x = 0, 1, \ldots, \frac{M}{r} - 1$, can be written as:

$$\frac{1}{M} \sum_{m=0}^{\frac{M}{r}-1} e^{\frac{2\pi i(x_0 + mr)y}{M}} |y\rangle \, |f(x_0)\rangle$$

This initially still does not look particularly interesting or useful when we are trying to find the period of $f(x)$. However, with some rearragement this state can be expressed as:

$$\frac{1}{M} e^{\frac{2\pi i x_0 y}{M}} \sum_{m=0}^{\frac{M}{r}-1} e^{\frac{2\pi i m y}{\frac{M}{r}}} |y\rangle \, |f(x_0)\rangle \tag{2.25}$$

This contains a structure which was introduced with the sumation of roots of unity, 2.17. Using this observation, the $\sum_{m=0}^{\frac{M}{r}-1} e^{\frac{2\pi i m y}{\frac{M}{r}}}$ section of the amplitude for the state $|y\rangle \, |f(x_0)\rangle$ is simplified to:

$$\sum_{m=0}^{\frac{M}{r}-1} e^{\frac{2\pi i m y}{\frac{M}{r}}} = \frac{M}{r} \delta_{y,0(Mon\frac{M}{r})} = \frac{M}{r} \begin{cases} 1, & \text{if } y = 0(mod\frac{M}{r}) \\ 0, & \text{otherwise} \end{cases} \tag{2.26}$$

With this simplifcation in place it can be seen that the probability amplitude of any branch where $y \neq x(mod\frac{M}{r})$ will be zero simply due to the Kronecker delta. This results in the only branches with non-zero amplitudes being where $y = 0, \frac{M}{r}, \frac{2M}{r}, \ldots$ and so when the first input register is measured the state observed will be $\left| \frac{kM}{r} \right\rangle$. After several repeats of the algorithm the value of r can be deduced from the observed states.

To see how knowing the period can help factorise the number N we analyse the function $f(x) = a^x(modN)$. This function is obviously periodic as it is modulo N. With the period r, $a^r = 1(modN)$. The value of r will depend on the value of a. Only the even values of r are useful, if an odd value is found the value of a should be changed and the new r found.

When r is even, $a^r = 1(modN) \rightarrow a^r - 1 = 0(modN) \rightarrow (a^r + 1)(a^r - 1) = 0(modN)$. Once we have the equation in this form, we can use the values of $a^r + 1)$ and $(a^r - 1)$ separately as one of them, possibly both, with have a factor in common with N. This can simply be found using the Chinese remainder theorm.

The advantage of Shor's algorithm is that the value of r can be found much faster with all values of x up to the limit of $M - 1$ tested simultaneously.

2.2.5 Quantum Teleportation Protocol

The quantum teleportation protocol provides a solution to the problems

> Alice has a quantum state, $|\Psi\rangle$, which she wishes to send to Bob. There is only a classical communication channel by which Alica and Bob can communicate.

With a classical state this would not be an issue. Measurements on the different parts of that state Alice want to sent to Bob would be made, and these would then be communicated down the channel for Bob to replicate. Unfortunately this is not possible for quantum states. Quantum computation exploits the power of quantum mechanics, however quantum mechanics has its own caveats.

The "No Cloning Theorm" in the field of quantum computation is the results of quantum mechanic's "Uncertainty Principle". The "Uncertainty Principle" states that "that the values of a pair of canonically conjugate observables such as position and momentum cannot both be precisely determined in any quantum state"[15]. Essentially this means that an unknown quantum state cannot be reproduced.

The classical approach is also halted by Holevo's Theorem. This theorem states that from any n qubit state, at most n bits of classical information can be observed. With a one qubit state, $\alpha \, |0\rangle + \beta \, |1\rangle$, the value of α and β are required for the state to be reproduced. The each of these require 2 values as
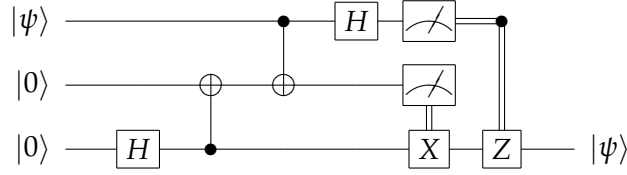
Figure 2.7: The Quantum Teleportation Circuit[2]

they are complex numbers, 4 values in total. The state can be rewriten to give $|\alpha|e^{i\theta_0}|0\rangle + |\beta|e^{i\theta_1}|1\rangle$. As global phase factors cannot be observed it can be simplified to $|\alpha||0\rangle + |\beta|e^{i\phi}|1\rangle$ This leaves the real values $|\alpha|, |\beta|$ and ψ, down to 3 values. We know that $|\alpha|^2 + |\beta|^2 = 1$ so the value of $|beta|$ is can be calculated from the value of $|\alpha|$. This leave just two values to specify. However, both of these two values can be specified to any arbitary precision, requiring an arbitrary number of bits of data. As Holevo's Theorem limits the observable information to just a single bit, for the one qubit state, the measure and recreate approach is impossible.

After stating these two theorems, the problem Alice and Bob face seems impossible. However, there are subtleties to the laws governing quantum mechanics which actually make it possible, with a few concessions.

For Alice to sent the state $|Phi\rangle$, $\alpha|0\rangle + \beta|1\rangle$, the protocol is as follows:

- Alice and Bob share the entangled state $\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$. This results in the state, the subscript letters are to show who is in possession of which qubits:

$$\frac{1}{\sqrt{2}}(\alpha|00\rangle_A|0\rangle_B + \beta|10\rangle_A|0\rangle_B + \alpha|01\rangle_A|1\rangle_B + beta|11\rangle_A|1\rangle_B) \tag{2.27}$$

- Alice takes a measurement in the Bell Basis. The Bell basis is a basis set which are not orthogonal to the computational basis. This means that the measurement does not collapse the state. The Bell basis vectors are $|00\rangle + |11\rangle$, $|00\rangle - |11\rangle$, $|01\rangle + |10\rangle$ and $|01\rangle - |10\rangle$.

Alice measures	Remaining state
$|00\rangle_A + |11\rangle_A$	$\alpha|0\rangle_B + \beta|1\rangle$
$|00\rangle_A - |11\rangle_A$	$\alpha|0\rangle_B - \beta|1\rangle_B$
$|01\rangle_A + |10\rangle_A$	$\alpha|1\rangle_B + \beta|0\rangle_B$
$|01\rangle_A - |10\rangle_A$	$\alpha|1\rangle_B - \beta|0\rangle_B$

- Based on the measurement Alice make she can instruct Bob, using the classical communication channel, to perform certain operations to change the remaining state into the original state $|\Psi\rangle$.

Remaining state	Bob applies to produce $|\Psi\rangle$
$\alpha|0\rangle_B + \beta|1\rangle_B$	Nothing
$\alpha|0\rangle_B - \beta|1\rangle_B$	Phase Flip
$\alpha|1\rangle_B + \beta|0\rangle_B$	Bit Flip
$\alpha|1\rangle_B - \beta|0\rangle_B$	Bit Flip followed by a Phase Flip

At the end of this protocol, the state $|\Psi\rangle$ has be teleported to Bob. However, this must not violate either the No Cloning Theorem or Holevo's Theorem. The No Cloning Theorem is not violated as to teleport the state to Bob, Alice must destroy her copy of $|\Psi\rangle$. This means that at no point are there two copies of $|\Psi\rangle$, therefore does not violate the No Cloning Theorm. The protocol does not violate Holevo's Theorem as at no point is any data observed from the state. The true identity and nature, the probability amplitues, of the state $|\Psi\rangle$ is never known.

The circuit to implement the quantum teleportation protocol can be seel in Figure **??**.

2.3 The Use of Evolutionary Computation in the Synthesis of Quantum Algorithms

Nature inspired computation is a highly active research area. Taking inspiration from nature and biological theories, search techniques such as Genetic Algorithms and Genetic Programming are being employed to a wide rage of industrial problems. What makes these approaches different is that they are based on a population, or 'generation', of individuals. The basic principle is to take an 'individual' of a defined representation, evaluate it's 'fitness' to perform the task required and to 'mutate' it randomly and add it to the next 'generation' of individuals. Along with mutation, a computational analogy to biology's reproduction, called crossover, can be used. Crossover takes two, or potentially more, individuals and combines them to produce other individuals which are then added to the next 'generation'. The each cycle of evaluate, selection and mutation and/or crossover produces a 'generation' of individuals. The process repeats until a required 'fitness' is found or a resource limit is reached, time or number of generations produced for example. As the process progresses, with the a reasonable representation and fitness function, the average 'fitness' of each generation should improve.

The use of evolutionary techniques to try synthesize quantum algorithms is not new. There are many examples of successes in producing solutions to problems already solved by a manual approach and some producing novel solutions to previously quantumly unsolved problems. The techniques used vary from Genetic Algorithms to Genetic Programming with varying success.

Not only is the technique varied, the desired solution is also varied. Some research focuses on the evolution of quantum circuits or programs, whereas some focus on more general quantum algorithms which take a parameter representing the number of input qubits. Due to the exponential increase in resources required for simulation with an increase in qubits the generality of the quantum algorithms is not usually tested on large systems.

2.3.1 Q-PACE I

Massey[9, 16] explores both Genetic Algorithms and Genetic Programming as search techniques. The software suites presented, Q-PACE I - IV, have varying success and increase in search power. Q-PACE I[16] is described as solving 'a number of basic proof of concept problems'[9] and 'proves the concept that evolutionary search techniques can be used to evolve quantum software'[9]. Q-PACE I uses a fixed length array of quantum gates and is based on a simple Genetic Algorithm found in [17].

Q-PACE II[9] is a suite based on Q-PACE I but uses Genetic Programming instead. In contrast to Q-PACE I, Q-PACE II is able to handle variable length solutions as individuals are represented as a list of quantum gates parameterized with the label, target and control bits and phase factor. It also includes the inclusion of vector manipulation rather than matrix manipulation to improve efficiency. Matrix manipulation is a simple concept, however it is very computationally expensive. Operators are just one to one functions acting on state vectors.

$$\begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{pmatrix} \rightarrow \begin{pmatrix} \alpha_2 \\ \alpha_3 \\ \alpha_0 \\ \alpha_1 \end{pmatrix} \tag{2.28}$$

The operation of the Pauli-X gate on the first qubit in a two qubit system can be represented as Equation 2.28. For more complex gates, such as the Hadamard gate, the matrix manipulation is much more expensive than the equivalent vector manipulation.

2.3.2 Q-PACE II

The representation used in Q-PACE II is not able to express a Toffoli, Controlled-Controlled-Not, gate as a single gate. This makes evolving a half-adder circuit more than a trivial test. When Q-PACE II is tested against producing a circuit with the specification $|x, y, z\rangle \rightarrow |x, x \oplus y, x \wedge y\rangle$ it is able
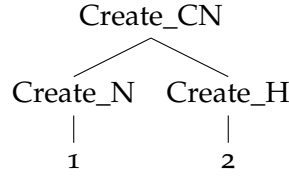
Figure 2.8: Q-PACE III Example Solution Tree
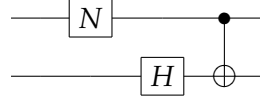
Figure 2.9: Q-PACE III Example Program Output

to produce several exact solutions. One of which was claimed, at the time, to be the 'best known solution to the problem'[9] with the restricted gate set. Q-PACE II was also challenged to produce a circuit to implement $|c, a, b, z\rangle \rightarrow |c, a, (a+b)_0, (a+b)_1\rangle$ and again produced 'the most efficient solution to this particular problem'[9].

$$
\begin{pmatrix}
a \\
b \\
0 \\
0 \\
0 \\
0 \\
0 \\
0
\end{pmatrix}
\tag{2.29}
$$

Both tests of Q-PACE outlined were carried out to produce a deterministic solution, would always present the correct answer after measurement. Further tests were performed on more complicated problems, however deterministic solutions were not found. Following on from the work carried out by Spector et al[18–20], Massey changed to a probabilistic approach. The definition, referring to 2.29, of a probabilistic solution used by Massey is:

> The probability of measuring $|000\rangle$ is at least $0.5 \times a\bar{a}$, and the probability of measuring $|001\rangle$ is at least $0.5 \times b\bar{b}$.[9]

With this new, relaxed, requirement of probabilistic correctness, the Q-PACE II software was used to 'evolve a quantum circuit to implement the specification

$$
|a_1, a_0, b_1, b_0, z_1, z_0\rangle \rightarrow |a_1, a_0, (a+b)_2, (a+b)_1, (a+b)_0, z_0\rangle'
$$

The result was, despite the requirement of only probabilistic correctness, a deterministic solution to the problem[9]. The evolved circuit was not as efficient by that presented in [21] but was still deterministic.

2.3.3 Spector *et al*, Deutsch's Problem

Lee Spector *et al*[18–20] published a paper which used a different evolutionary approach. They used Genetic Programming to produce a better than classical solution to the original Deutsch[4] problem. This was not an unsolved problem as explained in Section 2.2.1, however it was an example of where quantum computation was known to be more efficient than any possible classical approaches.

The genetic programming approach taken was the traditional tree based representation. The level at which the solution was represented was higher than that of both Q-PACE I and Q-PACE II. Both Q-PACE and Q-PACE II evolved quantum circuits whereas Spector represent solutions as "classical

programs which, when executed, construct quantum gate arrays". This is the quantum program representation as defined earlier.

The best solution produced a quantum gate array which did compute the answer to the Deutsch problem but was different to the Deutsch circuit2.3. However, it was still provably more efficient than any classical algorithm.

As well as fixed size solutions, the function set, non-terminal set for the Genetic Programming tree, included the control structures required to produce parameterisable quantum algorithms. These were not used to attempt a solution to the Deutsch Jozsa problem, which would have seemed the more obvious progression, but the *majority-on* problem. The majority-on problem it to decide whether the output of a function has a majority of outputs being 1.

The best solution was a parameterisable algorithm which, for functions with a large variation from 2^{n-1} of 1's the solution performed well. However, when tested on functions with 2^{n-1} inputs evaluating to 1 the solution ends up with an output error of 0.5.

Both the solution for the Deutsch problem and the *majority-on* problem were searched for with a probabilistic approach, looking for solutions with an output error of lower than 0.48 rather than for deterministic solutions.

2.3.4 Q-PACE III

Massey's third generation, Q-PACE III, evolved quantum programs, inspired by the Spector[18–20] results and taking on one of the suggested "Future Work" topics. Just as with Q-PACE II, Q-PACE III was a Genetic Programming suite. The solutions evolved by Q-PACE III were executable programs, 'second order' solutions, which produce as an output a quantum circuit. An additional difference between the two suites is the representation. Q-PACE III represents programs as trees, rather than lists. The execution of the solutions is performed by a pre-order traversal of the solutions tree representation. The tree in 2.8 produces the circuit in 2.9.

Due to the change in representation, the evolutionary operations, mutation and crossover, occur at the second order level. As shown in 2.8, the different non-terminal nodes were of different arrity, allowing for more expressive trees. Whereas in Q-PACE I and II the fitness of an individual could be calculated directly from the individual, in Q-PACE III the individuals have to be 'executed' to produce the quantum circuit before the fitness can be evaluated. With this additional step, the fitness evaluation requires more computational resources.

Massey defines the PF Max problem as

> You are given a permutation function f(x) which operates over the integer range [0..3]. Using a suitable encoding, evolve a quantum program U which returns the value of x that gives the maximum value of f(x).[9]

Q-PACE III was used to try find a probabilistic solution to the PF Max problem. The experiment was successful. When tested against the 8 permutations used as fitness cases, the correct result was the probabilistic result in each case. When tested against the 24 possible permutations, the correct result was the probabilistic result in 20 of the 24 cases.

It was found that if the acceptance requirement was reduced to 0.4, from 0.5, a quantum program was evolved which returns the correct value for all 24 possible permutation exactly 50% of the time. This result was quite remarkable, this probability is twice that of the best classical approach, guessing.

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{\frac{2\pi ijk}{N}} \tag{2.30}$$

Q-PACE III was also used to evolve an solution which, when run, produced the circuit for the Quantum Fourier Transform on 3 qubits. The Quantum Fourier Transform is an operation defined by equation 2.30 where x, $(x_0, x_1, \ldots, x_{N-1})$ where $N = 2^n$, is the input state and y, $(y_0, y_1, \ldots, y_{N-1})$, is the resulting state. It is fundamental for Shor's factorisation algorithm. The problem was approached both deterministically and probabilistically, both were successful. The results of the probabilistic

experiments were unexpected. The definition of the acceptance requirement had to be generalised. Whereas for the PF Max problem the correct answer was a single value, the correct answer for the Quantum Fourier Transform is a state vector. An acceptance level of x% was redefined as the requirement that for all fitness cases, each state has a probability of being measured of at least x% of the probability for the respective state after running a 'perfect' Quantum Fourier Transform. It was found that for acceptance levels of 75%, 50% and even 25%, the evolved circuits often had an acceptance value in excess of 99%.

2.3.5 Q-PACE IV

With Q-PACE IV, Massey once again raised the level at which the solutions were represented. Q-PACE IV was a Genetic Programming suite to evole quantum algorithms, parameterisable with the system size. To reduce the complexity of the representation, all non-terminals were made to be the same arrity, 3. This was to remove the restrictions on the mutation operators while ensuring only syntactically correct algorithms were developed. As not all gates require 3 parameters, the excess parameters were ignored during evaluation.

The desire to produce quantum algorithms required the inclusion of an iteration construct, numerical arithmetic and a store of variables so loop variables can be used. Several issues were encountered. An issue with the numerical arithmetic inclusion was with the posibility to specify a qubit which does not exist. In a system of 3 qubits, there is no sixth qubit so the syntactically correct '*Create_H(MULTIPLY(3, 2, X), X, X)*'[9], where X is a don't care symbol, is syntactically correct depending on the system size. It was decided that any number above the system size would be interpreted as the system size. This is a solution but as is stated in [22], this means that the system size is over represented in the search space. Also due to the limitations of quantum simulation, a number larger than the upper limit on system size efficiently able to be simulated may need to be the system size. However, it may need to be that specific value but until simulation or production of adequately large quantum computers is possible the algorithm cannot be finalise. Due to this, if the value is assumed to be the system size it may make analysis of the algorithm, and therefore the resulting understanding, much harder and possibly misleading. The comment made in [22] was in reference to representing gates as the numbers between 0 and 7, but only needing to represent 5 gates. Using modulo 5 represents two gates with a single value but three with two values, potentially leading to favouring the over represented gates. Therefore the two potential solutions to the indexing of a non-existant qubit both have their potentially undesireable behaviours but the approach taken by Massey does appear to be the option which is unlikely to interfere with the evolution of solutions, only potentially with analysis.

It would seem that there is the posibility of using individuals containing such nodes to spawn two separate individuals, one with the numerical value and one with the variable holding the system size. Protection would have to be added into the operator carrying out this operation to ensure the individual with the numerical value is not duplicated each generation. If the evaluation of numerical nodes was altered to use the modulo of the system size the combination of the two individuals would cover both of the proposed solutions while compensating for the failings of both.

The first test for Q-PACE IV was to try and evolve an algorithm to produce an n-qubit Quantum Fourier Transform with 100% fidelity. There is a known algorithm to produce these circuits, provided as Figure 32 in [9], so the test was quantifiable. It was also shown that the gate set available to Q-PACE IV was indeed able to express the algorithm. Q-PACE IV was unsuccessful using the same fitness function as used by Q-PACE III in its evolution of the 3-qubit Quantum Fourier Transform. The fitness function was subsequently changed so that it used the polar representation of the complex numbers indicating the probability amplitude of each state rather than their Cartesian form. This was more successful and managed to produce an algorithm capable of producing a circuit with 100% fidelity for 1, 2 and 3 qubits.

However, this algorithm was not entirely system-size independent. The problem was due to the requirement of Quantum Fourier Transform to reverse the order of the qubits. This required the use of swap gates but the inclusion of these gates have a large effect on the fitness of individuals which include the phase rotation gates, another critical gate for the Quantum Fourier Transform. The

fitness was once again altered, however this alteration guided the search in the direction of using swap gates. A solution was found that was system size independent and produced 100% fidelity.

Both of these Quantum Fourier Transform examples show the importance of the fitness function. Even though the Cartesian and polar form of complex numbers are mathematically equivalent, they produced drastically different results. The Cartesian form restricted the search and no solution was found whereas the equivalent polar form had no such restriction. It appears to me that the reverse will be true in the search for other problems where the relative phases are not as fundamental as they are in the Quantum Fourier Transform. This gives an indication that the search for currently unknown quantum algorithms may require a series of parallel evolution streams using different representation within the fitness function. It may also prove helpful to use a series of fitness functions in a colaborative approach.

Evolutionary approaches are commonly used for multi-objective optimisation problems where the multiple objectives are in conflict. The use of different representations in fitness funcitons could be seen in a similar way to these approaches. However, different representations of the fitness function would not be a set of conflicting objectives but colaborating objectives. This would allow the search to be free from selecting the correct representation of the complex probability amplitudes. This would however require several fitness functions which give comparable values as well as a mechanism to chose the 'best' fitness value for the individual being evaluated. The representation of this also leads to numerous choices, using just a MAX function or an average function or to represent the fitness as an n-dimensional point to optimise for n fitness functions.

2.3.6 Williams and Gray

None of the approaches presented so far have been used to produce the circuit for the quantum teleportation protocol or any other distributed protocol. There is essentially no difference between the circuits required for the quantum teleportation protocol than to produce the quantum fourier transform. The main difference is the restriction on communication.

With the quantum teleportation protocol, only classical communication can be used as control when acting on the qubit held by Bob. In terms of automatically producing the circuits the restriction can be characterised as *only single qubit gates, controlled or not, can be applied to the qubit held by Bob*. The use of controlled gates can allow the search to loosely represent measurement and communication of the measured value.

Williams and Gray[23] introduce a GP approach used to tackle the quantum teleportation protocol. The result is successful and was infact required fewer gates than the smallest previously known circuit.

The Williams and Gray approach uses a comparison between the target unitary matrix and the unitary matrices that the produced circuits represent. The approach also tackled the quantum teleportation protocol in two parts, based on the two parts of the circuit introduced by Brassard[24].

The use of the target unitary matrix is the major disadvantage of this approach. The construction of the target unitary matrix is also usually a significantly difficult task. Therefore the approach can only really be used to try reduce the circuit size of problems that already have known solutions or for problems with target matricies that are simple to produce. If a solution is known, the target unitary matrix can be easily produced from the unitary operations of the gates in the solution. If the target matrix is easy to produce it is also likely that a correct circuit will be easier to produce.

The other major disadvantage of the use of matricies is that the search is for circuits rather than algorithms. A target matrix is specific for a certain number of qubits in a circuit, it is not scalable, requiring a new matrix for $1, 2, 3$ and so on qubits involved in the circuit. For problems where the matrix is hard to produce this results in a much less viable approach to circuit construction.

2.3.7 Yabuki

A genetic algorithm approach to produce the quantum teleportation protocol is introduced in [25]. The approach is again a circuit, rather than algorithm, generator.

The use of genetic algorithms rather than genetic programming also has a second dramatic impact on the scaling of the search. With a genetic algorithm the length of the chromosome is fixed. As a result the number of gates that can be encoded in the chromosome is limited. This means that not only is a new search required for each new system size, but also potentially if the chromosome is not long enough. However, it also requires there to be some indication as to how many gates shall be required. The success of a search is likely to be sensitive to the amount of "excess" chromsome space. By this I mean that for problems where the size of the solution circuit is unknown, setting an arbitaly large chromosome size is likely to hinder the search.

The approach is shown to be successful in respect to the search for a quantum teleportation circuit. The system presented produces a circuit that is "simpler than ever known"[25].

This has to be views with respect of the restrictions and that the problem being tackled had previously been solved, providing a good estimate of circuit size. Another crucial point is that the structure of the system is assumed. The assumed structure does not, like in Williams and Gray[23], allow separate searches for the structural sections. It does however change the interpretation of the genes. This can be viewed in two ways. Using this assumed structure and changing interpretation between sections does allow the system to acurately enforce the result of measurement. However, it does assume the structure of the solutions. For unsolved problems the assumption of the solution structure is potentially disasterous for the success of the system.

As a result the approach introduced and presented in [25] can only really be used when a solution is already known. It has been proven that it is able to produce circuits of a smaller size so is more of an optimisation approach. The use of the system without a previously found solution would appear to be much more difficult than the results suggest.

2.4 The Focus of this Project

When looking at all the papers that include the use an evolutionary approach there is one thing that they all have in common. Each time research is carried out in this area nearly everything is bespoke. Some research use a library to perform the evolutionary search but that seems the extent of reuse. QPace I - IV share properties and each iteration is developed with respect to the strengths and weaknesses of the previous version and indeed the strengths and weaknesses of the Spector research.

The other common feature is the lack of source code available. None of the QPace suites are easily available and neither is the Spector code. This is not to say that there are not tools available to help with Quantum Algorithm design. There are many available in many different languages, Java, C++, Matlab etc, but these are purely simulators. A user creates a circuit, provides an input state and the tool will provide a final state. What seems to be the largest gap in the tools available for such research is a tool or even a framework that allows researchers to concentrate on the research of quantum algorithms rather than all the peripheral, but necessary, tasks.

In this project I aim to produce a framework that will enable researchers to be abstracted away from the problem of simulation and representation and concentrate on searching for the desired Quantum Algorithm. The framework shall allow a researcher to come up with their own search mechanism, QPAce V for example, without having to reimplement the circuit simulation or representation. The framework shall not only allow researcher to provide new search techniques but also to research the most effective cost functions. As was seen in the work presented by Massey[9] the representation of complex numbers that is used within the cost funtion had a significant impact on the solutions found by the search.

The use of the framework shall also allow for the work of different researchers to easily be compared, contrasted and combined within the same framework.

Secondly in this project I will produce a fully working system using this framework. The search engine and cost functions will be based on those presented as QPace IV[9]. This fully working example will be an indication as to how the framework could be used by researchers.

Thirdly, in this project I will produce effectively a 3rd Party application that will use the fully implemented system to perform all Quantum Algorithm searching and evaluation. This 3rd Party application will simply be a client GUI. This could be seen not as a 3rd party application but as

part of the fully working system. This observation I accept. The use of 3rd Party in this instance is simply an indication of the separation of knowledge. The client GUI produced will use only the API available to the "traditional" 3rd party applications rather than any internal knowledge or interfaces not available through the API.

The creation of such a toolkit is, to the best of my knowledge, something that has not previously been produced. All previous work in the area has focussed purely on the discovery of Quantum Algorithms with each researcher working in isolation. Without there being a known "right way" to search for Quantum Algorithms it is essential for the framework produced not to limit or encourage any particular search method. Although in the literature review the focus has been on evolutionary approaches, the framework must not appear to push any potential researcher into using evolutionary approaches. This I feel is essential for the framework, and even the fully implemented system and client GUI, to be adopted by researchers in this area.

As a final stage of the project I will be using the fully implemented system to carry out a number of experiments searching for Quantum Algorithm. This will use the QPace IV based search engine implemented in stage two listed above.

3 Requirements

The requirements listed in this section are for the framework, fully implemented system and the client GUI. The requirements were maintained using an online tool called ReqMan[26] by RequirementOne. This section and the requirements have been formed using the guidelines provided in the IEEE standard 830[27].

3.1 Purpose

3.1.1 Framework

The framework is aimed to allow research into Quantum Algorithms to concentrate on producing Quantum Algorithms. The framework is aimed to make it much simpler for research by different researchers to be combined and contrasted.

3.1.2 Fully Implemented Tool

The full implementation of a Quantum Algorithm search tool using the framework is to provide a working toolkit for researchers interested in finding Quantum Algorithms rather than the search techniques to find Quantum Algorithms. As the toolkit will use the framework it will also provide a potential foundation for future research into the search techniques.

3.1.3 Client GUI

The client GUI will provide an interface that should make the toolkit more accessible for researchers. Without the GUI provided, researchers would have to either embed the toolkit in their own application or within their own specific GUI. This is likely to reduce the potential use of the toolkit in the academic community. One of the main focusses of the toolkit is to try and provide a standardised framework for research of Quantum Algorithms. Not providing a GUI, resulting in many bespoke GUIs, goes against this focus. This is not to say that inclusion of the framework in 3rd party systems or improvement to the GUI is not encouraged.

3.2 Definition, Acronyms, and Abbreviations

The definitions given here are consistent with those used in the rest of the document but are included as a matter of clarity.

System Size - The number of qubits in the system. For example the quantum teleportation protocol has a fixed system size of 3 whereas the Quantum Fourier Transform can scale to any system size.

Quantum State - A column vector of 2^n complex numbers representing the probability amplitudes and phase of the 2^n states $|0\rangle \rightarrow |2^n - 1\rangle$ for a system size n.

Quantum Gate - A complex unitary operation on a quantum state.

Quantum Circuit - An ordered list of quantum gates to be applied to the quantum state.

Quantum Algorithm - An ordered list of instructions used to construct a quantum circuit.

Suitability Measure - A function to provide a performance of a solution with 0 as the "Best" performance and performance decreasing as the function result increases.

3.3 Requirements Summaries

This section contains a summary of the requirements of each of the separate phases of the project. A full listing of specific requirements can be found in Appendix A.

3.3.1 Framework

Additional Search Engines

The framework shall allow researchers to provide search engines for the system to use. This is important as one of the intended uses of the framework is for research into the techniques used for searching for quantum algorithms. The way in which the framework provides this shall not imply the use of any search technique in favour to any other. It is important that the framework shall effectively be research direction independent.

Additional Suitability Measures

The framework shall allow researchers to provide suitability measures for the system to use. A suitability measure is effectively a fitness function. However, the term fitness function is associated with the use of evolutionary techniques. With the tool intended to be technique independent the term suitability measure shall be used.

It is well known that the suitability measure, performance metric, has a significant impact on the success of a search. However, it was also shown by Massey[9] that in the search for quantum algorithms the search can be sensitive even to the level of complex number representation. The ability for researchers to provide suitability measures is therefore paramount for the framework to be useful and improve research progress rather than hinder it.

Not only is this ability required by the researchers searching for quantum algorithms but also for those researchers concerned with finding successful suitability measures for use by the first group of researchers.

Quantum Algorithm Output

The solution of a search, a quantum algorithm, shall be presented to the user as a list of instructions. An algorithm is a list of instruction to follow in order to produce a circuit. The solution of a search using the framework is an algorithm. This solution shall be provided to the user as a list of instructions in a consistent format.

Visualising a Circuit

The system shall provide visualisation of the circuit produced by the solution of the search for a system of a user specified number of qubits. To ensure that the output of the search is helpful the framework shall provide a representation of the resulting circuit that can be rendered into a circuit diagram.

The circuit visualisations produced shall follow the widely recognised conventions of each gates appearance.

Third Party Software

The framework shall be able to be embedded in third party software. The framework is intended for use by the research community and it is not intended to limit the ways that it can be used. As a result it is not only important that the framework be able to use third party software, search engines and suitability measures, but is also important for the framework to be available for inclusion in third party software. To achieve this knowledge of the internal implementation detail shall not be required.
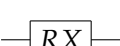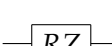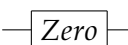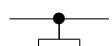
$-\boxed{X}-$	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	$-\boxed{Y}-$	$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$	$-\boxed{Z}-$	$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$
$-\boxed{H}-$	$\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix}$	$-\boxed{RX}-$	$\begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$	$-\boxed{RY}-$	$\begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$
$-\boxed{RZ}-$	$\begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix}$	$-\boxed{V}-$	$\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$	$-\boxed{W}-$	$\begin{pmatrix} 1 & 0 \\ 0 & -i \end{pmatrix}$
$-\boxed{Zero}-$	$\begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}$	$\overset{\times}{\underset{\times}{}}$	$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \rightarrow \begin{pmatrix} a \\ c \\ b \\ d \end{pmatrix}$	$\begin{smallmatrix}\bullet\\ \boxed{U}\end{smallmatrix}$	$\begin{pmatrix} I & 0 \\ 0 & U \end{pmatrix}$

Figure 3.1: Supported Gates and Definitions

Definition of Search Target

The framework shall provide a standardised definition format for users to specify the target of the search. All searches have a target, the shortest path or the minimum value for example. The searches that the framework are intended for are those to find a quantum algorithm to produce a circuit to solve a specific problem, to produce an equal superposition or the Quantum Fourier Transform circuit. The framework needs to provide a standard way of defining what the search target is. The standard shall be formalised so it is able to be used and produced by third party software.

Use of Configuration Files

The customisation of the framework shall be provided through a series of configuration files. All third party additions to the framework, search engines and suitability measures, shall be specified using a series of configuration file. These configuration files shall be well defined and able to be used and produced by third party software.

Provided Gates and Algorithm instructions

The framework shall provide implementations of all gates specified in Figure 3.1. The framework shall provide algorithm instructions for each of these gates and for the instantiation of the Controlled-U gate with all single qubit gates. Figure 3.1 defines all the most well known quantum gates and indicates the visual representation convention used in the project.

Algorithm Control Structures

The system shall provide the iterate control structure and support nested iterate instructions.

Producing Circuits from Algorithms

The framework shall be able to produce a circuit, for any given number of qubits, from a quantum algorithm.

Circuit Simulation

The framework shall provide the simulation of a circuit given an initial state. Using the gate definitions given in Figure 3.1, a circuit constructed of the supported gates shall be able to be accurately simulated. Given an initial state the framework shall be able to give the final state up to the accuracy of floating point arithmetic.

Step-by-Step State Evolution

The framework shall provide a way to perform step-by-step evaluation of a circuit given an initial state To aid researchers in understanding the algorithms and circuits produced as the result of a search a step-by-step evaluation shall be provided. Given an initial state and a circuit, the state after the application of each unitary operation, gate, shall be reported so the state evolution can be traced.

This shall also provide a debugging mechanism to ensure that all unitary operations are performing the expected opertation on state.

3.3.2 Fully Implemented Tool

Sample Search Engine

The tool shall provide at least one implemented search engine. The tool shall provide a basic search engine that will allow researchers interested in the quantum algorithms, rather than the search techniques, to use the tool "out of the box". The specific search engine is not specified.

Sample Suitability Measure

The tool shall provide at least one implemented suitability measure. The tool shall provide a basic suitability measure that will allow researchers interested in the quantum algorithms, rather than the suitability measure, to use the tool "out of the box". The specific suitability is not specified but shall be proven to allow basic circuit to the produced by search.

Sample Search Targets

The tool shall provide a number of search targets with known outputs. To allow search engine and suitability measure researchers to perform simple tests the tool shall provide a selection of basic search targets. The search targets included are not specified.

3.3.3 Client GUI

Search Engine Selection

The GUI shall provide a user with a selection of search engines to use in a search. The GUI shall provide a selection between all search engines registered in the framework. The most recently selected search engine shall be used by subsequent search.

Suitability Measure Selection

The GUI shall provide a user with a selection of suitability measures to use in a search. The GUI shall provide a selection between all suitability measures registered in the framework. The most recently selected suitability measure shall be used by subsequent search.

Search Target Selection

The GUI shall provide a user with a selection of search targets to be used as the search goal. The GUI shall provide a seleciton between all search targets registered in the framework. The most recently selected search target shall be set for subsequent searches.

Search Target Creation

The GUI shall provide a way for users to create a new search target without needing to explicitly write a configuration file. Writing configuration files is quite monotonous and highly error prone. The GUI shall provide a way to create these configuration files that reduces the error rate. The way in which the GUI provides this is not expected to dramatically decrease the monotony due to the nature of the amount of information that need be specified for problems when high number of Qubits are involved. The inclusion of such a feature is very important to improve the usability of the system and improve the potential level of use in the research community.

Search Target Editing

The GUI shall provide a way for users to edit the contents of a previously created search target without manual editing of the configuration file. The size of the configuration file required to specify a search target will increase linearly with respect to the amount of test data. The size of the data is likely to follow the same rate of expansion as the quantum search space as the number of qubits, n, increases, 2^n. With the size of configuration file increasing in such a dramatic way the risk of error when directly and manually editing the values in such files increases in a similar fashion. To improve the risk of errors the GUI shall provide a way to graphically edit the test data in a way that abstracts away from the configuration file structure.

Loading a Search Target From a Previously Defined Configuration File

The GUI shall provide a way to import a predefined search target from a configuration file. One of the intended uses of the GUI is for research into producing quantum algorithms. As part of this research it is likely that researchers will want to distribute the search target definitions they create. This distribution may be to collegues or simply to other computers for them to continue work. Either way once a search target is defined and distributed, the use of received search target configuration files should be supported by the GUI. The GUI shall provide a way for users to import search targets using the respective search target configuration file as long as the configuration file is of the correct format.

State Visualisation

The GUI shall provide a way to visualise any quantum state. A quantum state is defined as a vector of complex numbers. Depending on size, comparing two or more state can become monotonous. If the comparision of the two states does not need to be exact, a visual representation of the two states can provide a simpler, and quicker, method for comparision. To provide such comparison the GUI shall provide a way to visualise a quantum state.

Reporting the Search Result

The GUI shall provide a way to report the search result, a quantum algorithm, to the user. The GUI would be of no use to any quantum algorithm researcher if it did not provide the results of any searches. The GUI shall provide the quantum algorithm in the same way the framework reports the quantum algorithm result. This is to ensure that the format of the quantum algorithm reported does not change depending on whether the GUI is used or not.

Circuit Visualisation

Given a quantum algorithm and a system size, the GUI shall produce a visualisation of the resulting circuit. Some quantum algorithms produced using the search are likely to be hard to understand in pure algorithm form. Understanding a circuit is likely to be easier. To save researcher time in drawing the circuits by hand, the GUI shall provide a visualisation of the circuit for a specified system size.

Step-by-Step State Evolution

The GUI shall provide a way to perform, control and visualise the step-by-step state evolution for an intial state and circuit. The framework provides the ability to analyse the evolution of a satet with respect to an initial state and a circuit. The GUI shall provide a way of controlling and reporting this step by step evaluation to the user.

Tooltips

The GUI shall provide user help through the use of tooltips. All elements of the GUI shall be explained through the use of tooltips.

3.3.4 General Requirements

Portability

The framework, fully implemented tool and the GUI shall be able to be used on a range of Operating Systems. The produced software shall be able to be run on:

- Windows 7 32-Bit

- Windows 7 64-Bit

- Linux 32-Bit

- Linux 64-Bit

Usability

Using either the fully implemented tool or the GUI a user shall be able to start a search within 30 seconds. Using a predefined search target a user shall be able to initiate a search with a chosen search engine and suitability measure within 30 seconds of starting the software.

4 Design and Implementation

This section shall include both design decisions and implmentation decisions. Java was selected as the programming language for the implementation. The main reason for this choice is that Java is widely adopted by many researchers. The level of tool support is significantly higher than many other programming languages providing better development environments and support tools, debuggers and profilers for example. As a result of the requirement for portability, Java provides much simpler support for portability without numerous versions required for each operating system.

Also considered was the availability of libraries likely to be used by researchers. The research inspiring this framework are all evolutionary search techniques, as a result the availability of the ECJ[28] strengthed the decision to implement the framework in Java. As part of the development, a search engine based loosely on the QPACE IV search engine introduced by Massey[9]. This search engine implementation is done using ECJ.

4.1 Framework

In this section I will outline the design decisions that only directly effect the framework produced and how it was implemented.

4.1.1 Complex Numbers

Complex numbers are central to Quantum Computing. As such, any attempt to simulate the behaviour of a Quantum Circuit must handle complex numbers.

There are really only two ways to handle the existence of complex numbers. One can represent a complex number explicitly as a pair of floating point numbers, or to encapsulate the representation inside a "complex number" data structure.

The framework uses the second of these options and provides the "Complex" class. This was chosen for several reasons. The primary reason was to reduce the risk of programming errors effecting the simulation. If complex multiplication, addition and other operations had to be replicated throughout the frameworks codebase, and the codebase of any research work, the likelihood of implementation error is much higher, and the tests required to find the error become more specific. It is much better software engineering practice to encapsulate the properties, real and imaginary values, and the operations on those properties, arithmetic etc.

A season reason is that after brief research online, there are complex number libraries already available. This reuse of previously written software can also reduce the likelihood of errors in the code. This is not necessarily due to the software being written by people that are more intelligent or that are better programmers, or even that the software has been explicitly tested more thoroughly than if I were to write a complex number class. It is due to the size of the deployment footprint. The number of times the software has previously been deployed, and therefore the number of times it has been implicitly tested by users.

The third reason is that one of the principles behind producing the framework is the attempt to try standardise the research from different researchers. Without the provision of this "Complex" class one researcher could use Cartesian representation, two floating point values, while a second researcher could use the Polar representation, also two floating point values. If the documentation of the software produced by the two researchers did not mention the representation used, a third researcher could try combine, or compare, the two pieces of software using the framework. The third researcher is likely to receive very confusing and highly misleading results. The provision of a Complex class that is used throughout the framework where complex numbers need to be used will reduce the risk of such an event.

The implementation of the Complex class is based on an implementation provided as part of a "Complex Function Grapher"[29]. The Complex class provides both Cartesian form *real + imaginary*, through real() and imag() returning the real and imaginary components respectively, and Polar form $re^{i\theta}$, through mod() and arg() returning r and θ respectively. The implementation has been adapted to better suit this application. A calculation of the euclidean distance between two complex numbers is provided. A second addition is the ability to parseComplex in a similar way to parseInt or parseDouble provided by the standard Integer and Double classes respectively. This allows a string such as "$2 + 3i$" to be input by a user and for a Complex object to be created with a real component of 2 and imaginary component of 3.

4.1.2 Matrices

As seen in Equation 2.4 the application of a quantum gate is simply the application of a unitary operation, represented as a matrix, to a quantum state. This adds the requirement on the framework to provide a manner in which matrices will be represented.

In a similar way to the complex numbers discussed above, there are two distinct ways the framework could have been designed. The framework could either use an explicit representation, two-dimensional arrays, or could provide a Matrix data structure.

The framework has been designed to use the data structure encapsulation as the matrix representation. The justification is identical to that discussed above. Matrix operations are easy to get wrong in implementation and there are matrix libraries for many languages. The incomplete documentation argument also holds with matrices. If the framework were to just simply represent matrices as two dimensional arrays, two researchers could order the dimensions differently leading to similar problems to that of conflicting complex representations for the third researcher.

The implementation of the Matrix used in the application is based on the JAMA Matrix package[30]. The JAMA Matrix package provides matrices of double values. For use in the framework this needed to be updated so that it provides matrices of Complex objects and performs all the required operations as though they are matricies of complex numbers.

The JAMA Matrix package provides extra functionality for matricies of doubles that are not required for the framework. In the conversion from double matrices to Complex matrices this additional functionality was removed. This was due to the functionality not being required and therefore can be seen as dead code so should be removed.

One matrix operation that was not included in the JAMA Matrix package was the tensor product operation. This operation is used heavily when applying a single qubit gates to multi-qubit systems and therefore necessary for the framework.

4.1.3 State

With a representation of matrices defined, the definition of a quantum state naturally followed. Using the matrix representation, quantum states are defined simply as $2^n \times 1$ matrices, vectors. This representation makes unitary application much simpler as it automatically supported as matrix multiplication.

4.1.4 Test Suite Structures

With most problems there are a series of expected results that are used to measure the suitability of any suggested solution. The expected results are also usually coupled with the respective inputs.

For Quantum Algorithms the expected results are the state vectors produced by circuits constructed by the algorithm. As such it was chosen that a test case would be represented as a pair of state vectors, the stating state and the expected state. The application of a quantum gate is a simple mapping from a starting state to a resulting state. When a circuit can be defined as a single unitary operation, a custom quantum gate, this representation seems a natural choice.

Each circuit produced by the Quantum Algorithms has n qubits. This means that it can only be evaluated using test cases for n qubits. Test cases for any other number of qubits would not produce

```
<testsuite>
    <testset NumQubits="1">
        <testcase><!—0—>
            <starting_state>
                <matrix_element><!— 0—>
                    <Real>1.0</Real>
                    <Imag>0.0</Imag>
                </matrix_element>
                <matrix_element><!— 1—>
                    <Real>0.0</Real>
                    <Imag>0.0</Imag>
                </matrix_element>
            </starting_state>
            <final_state>
                <matrix_element><!— 0—>
                    <Real>0.0</Real>
                    <Imag>0.0</Imag>
                </matrix_element>
                <matrix_element><!— 1—>
                    <Real>1.0</Real>
                    <Imag>0.0</Imag>
                </matrix_element>
            </final_state>
        </testcase>
    </testset>
</testsuite>
```

Figure 4.1: Partial Test Set for Pauli X Gate

useful results. The notion of a test set was introduced to hold all test cases for a specific n. All test cases are held within a test set.

A test suite is used to hold all the test sets produced for the same problem. There is only one test set for each distinct value of n. The implemented structure can be seen in Figure 4.2.

The test suite is fully defined in a single XML file. The XML in Figure 4.1 is a sample of such a file. It is easy to see file structure reflects the internal structure of test suites just described.

The use of a seperate files to specify the test suite for each Search Problem rather than including it in the Search Problem Manager configuration file, discussed in Section 4.1.5, ensures that the files remain readable. It also ensure that researchers can distribute Test Suite easier than if all Search Problems were fully specified in a single file.

It can be seen in the class diagram, in Figure 4.2, that an additional class, SuperPositionalTestSet, is included as a subclass of the TestSet class. In the framework the test cases that have starting states in a superposition are held within SuperPositionalTestSets. A SuperPositionalTestSet for 2 qubits returns "-2" to indicate the superposition starting state. This initially doesn't seem that useful as both a superpositional and non-superpositional test case contain both a start and final state that are treated exactly the same during evaluation. This is however useful for GUI's and 3rd party software and also for the creation of test cases from the XML files. The labelling of non-superpositional test cases, to make most sense to users, are set to the binary value of the input state. For example, the label for the input state $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ is set as 0 and for the input state $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ the label is set as 1. However, for superpositional test cases the use of the expanded superpositional state, $\begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$ to $\frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |1\rangle$, is not sensible due to the exponential increase in size as the number of qubits increases, therefore
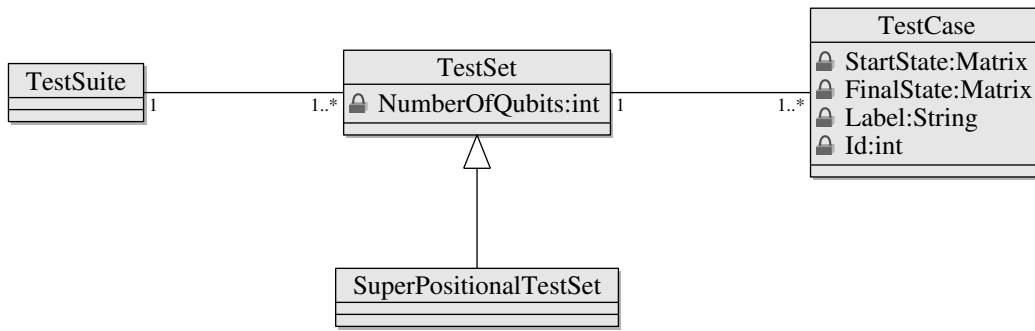
Figure 4.2: Test Suite Class Diagram

reducing readability. For the superpositional test cases the labels are therefore set as "Test Case 0" to "Test Case $n-1$".

The negation of the "NumQubits" attribute is used to specify in the XML file that the test set is of superpositional input states. This negation is required for the labels of the test cases to be set correctly.

Alongside the framework, an independant test suite graphical editor is provided. A user can use this to produce the XML file and therefore do not need to explicitly consider the XML file. The separation of test cases between superpositional and non-superpositional test cases improved the design and implementation of this editor. The design and implementation of the graphical editor can be found in Section 4.2.1.

The contents of a test suite are not fixed. If a user finds an error or wishes to add test cases, either a third party application or the provided test suite graphical editor can be used to update the respective test suite elements. These changes need to be reflected in the XML file to become persistant so to ensure that the updated XML file is well formed the framework provides a class to produce from a given test suite. This class is used in the independant test suite graphical editor and is recommended for use by any third party application.

4.1.5 Manager Classes

As can be seen in the architecture diagram of the framework, Appendix B, there are several classes with names suffixed with "Manager". These classes provide access to the extendible areas of the framework. There is a Manager class for the Fitness Functions, the Search Engines and the Search Problems. Each of these are a specific site of expansion.

Each Manager is configured using an XML file specfying all options for the specific Manager. The Fitness Function Manager will be configured for all the available Fitness Functions, the Search Engine Manager for all the available Search Engines, and the Problem manager for all the available Search Problems. The Search Problem Manager class is outlined in detail in Section 4.1.8.

This configuration is performed at runtime rather than at compile time. It was designed to allow researchers to add, for example, extra Fitness Functions without altering the code of the framework. This independence of the framework implementation and the results of research, specific Fitness Funcitons etc, has been identified as one of the key foundations of the framework concept. The inclusion of this knowledge separation encourages the use of the standardised interfaces specified for each expansion site.

The XML outline shown in Figure 4.3 is an outline of the XML file used to specify the available Fitness Functions. The XML files specifying the available Search Engines and the available Problems can be found in Appendix C.1 and C.2.

These XML files are used to regiser the available implementations with the respective Managers. The Manager classes use these registrations to provide the choice of available instantiations of Search Engines and Fitness Functions.

Both the Search Engine and Search Problem Manager classes are implemented in the same way. The class diagram in Figure 4.4 shows the Manager design. A mapping between the Name of the registered options and Tag objects is held in each Manager. A Tag contains the information within the

```
<FitnessFunc>
        <FitnessFunctionTag>
            <Name>FITNESS FUNCTION NAME</Name>
            <Class>IMPLEMENTING FULLY QUALIFIED CLASS NAME</Class>
            <Desc>FITNESS FUNCTION DESCRIPTION</Desc>
        </FitnessFunctionTag>
</FitnessFunc>
```
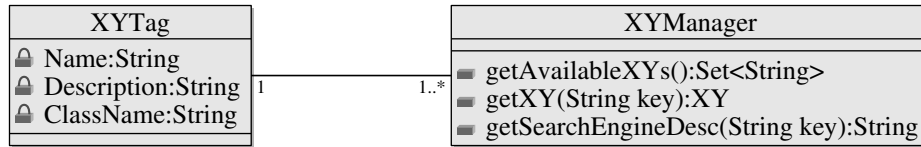
Figure 4.3: XML for Fitness Function Manager Configuration

XYTag	XYManager
🔒 Name:String 🔒 Description:String 🔒 ClassName:String	▬ getAvailableXYs():Set<String> ▬ getXY(String key):XY ▬ getSearchEngineDesc(String key):String

1   1..*

Figure 4.4: Manager - Tag Class Diagram

<XYTag></XYTag> element of the XML file. The getAvailableXYs() method returns a set of strings, the registered names, rather than the XY Tags to ensure the Manager classes have full control over the creation of the managed elements. To provide an XY object when requested using thegetXY(key) method, the Manager classes use reflection to instantiate an object of the respective class, the value of the ClassName variable of the Tag registered with *Name = key*.

4.1.6 Multiple Search Engines

The framework is aimed to be used universally by Quantum Algorithm researchers. The search techniques used by these researchers are also a matter of research effort. If the tool were to provide a search engine, with no option for change, the use of the tool is likely to be significantly impacted.

Providing a simple interface that allows each researcher to potentially use a different search technique is likely to increase the tools applicability. The simple interface allows a user to:

- retreive the names, used as the search engine identifier, of all registered Search Engines

- retreive the instantiation of the specified Search Engine instantiation

- retreive the description of the specified Search Engine

All registered Search Engines must implement the supplied interface. The Search Engines are not restricted to evolutionary approaches. The internal workings of the different search engines are unrestricted.

The alternative approach would have been to implement a series of search engines based on several different techniques and provide researchers this choice. This was not accepted as it moved the tool away from the framework intended. The provision of search engines without a simple manner to add additional engines would restrict research and not allow researchers to easily use techniques developed in the research community within the system.

4.1.7 Multiple Fitness Functions

As was noted by Massey[9], different Fitness Functions can have a dramatic impact on the success of a Quantum Algorithm search. The inclusion of a choice of Fitness Functions is to account for this. As with Search Engines, the choice of Fitness Functions is provided by the Manager class through a simple interface with methods synonomous to those provided for the Search Engine selection. A Fitness Function interface is provided to ensure that all Fitness Functions are able to be used universally within the tool and are not specific to any particular Search Engine for example.

```
<Problems>
  <prob>
    <Name>Final Pauli X</Name>
    <DefFile>config/finalpaulix.xml</DefFile>
    <Desc>A Pauli X gate on the final Qubit</Desc>
  </prob>
</Problems>
```

Figure 4.5: XML for Problem Manager Configuration

Similarly to the Search Engine, a series of Fitness Functions could have been implemented and provided without provision for extension. The justification for the approach taken is the same as listed for the multiple Search Engines. It was deemed detrimental and in contradiction of the frameworks purpose to limit the Fitness Functions to those provided by the tool.

4.1.8 Multiple Problems and Problem Specification

As has been mentioned on several occasions, one of the foundation principles of the framework is the ability to "Plug and Play" the work of other researchers without the proplem of integration. With Search Engines and Fitness Functions developed to adhere to the respective interfaces, a user should be able to work with the toolkit and treat it as a "Black Box".

In Section 4.1.4 how test suites and all their contained test cases are specified in XML was described. The use of XML files does however increase the effort required from the user. The user needs to specify, each time they use the framework, the location of the XML file containing the correct test suite. To reduce this effort the problem container is introduced alongside its manager.

The problem manager allows multiple problems to be defined within a single XML file so the user need not provide the test suite XML each time the framework is used. This single XML file contains the definition of mutiple problems. An example of these XML files can be seen in Figure 4.5. A problem has a name, description and file name for the respective test suite XML file. The name and description are used to provide a human readible explanation of the problem represented by the test suite XML file. The use of a separate XML file to collate all defined problems makes maintenance much simpler.

Providing a problem manager allows the framework to be used for different problems without having to restart the system and without any external software needing to provide different problems explicitly.

The Search Problem Manager works in a similar way to the Search Engine and Fitness Function Managers detailed in Section 4.1.5. However, the difference between the registered Search Problems is the test suite configuration XML file rather than the implementing class. A second difference is that the registered options are not static, a third party application can interact with the Manager to create new Search Problems and update the already registered Search Problems. Therefore, the Search Problem Manager has to be able to produce an updated Search Problem Manager configuration file so the changes made are persistent. The changes that the Seach Problem Manager is concerned with is only updates to the values in the Search Problem Manager configuration file. Any changes made to the set of Test Cases are not maintained by the Search Problem Manager. This ensures that the different concerns are separated. The update of Test Cases is detailed in Section 4.1.4.

4.1.9 Quantum Algorithms

The result of the search engines are quantum algorithms. To maintain the "Plug and Play" nature of the framework, the representation of these algorithms needed to be specifed and standardised. However, the representation also had to ensure that it was not limiting the search engines.

To provide a standardised and non-limiting represenation the framework provides an internal quantum algorithm structure that can be simply built by any search engine. This allows the search

Instruction	Gate1	Gate2	Phase	Sub-Algorithms
QuantumInstruction(Enumeration)	expnode	expnode	expnode	QuantumAlgorithm[]

Figure 4.6: Quantum Instruction Structure

$exp \rightarrow e + e$	$e \rightarrow exp$
$exp \rightarrow e - e$	$e \rightarrow SystemSize$
$exp \rightarrow e * e$	$e \rightarrow Value$
$exp \rightarrow \frac{e}{e}$	
$exp \rightarrow e^{e}$	
$exp \rightarrow \frac{\pi}{2^{e}}$	
$exp \rightarrow LoopVars[e]$	

Figure 4.7: expnode Context Free Grammar

engines to have a different internal representation that is then used to build the standardised algorithm. Using this there are no limitations on the structures used internal to the search engines.

The use of the standardised quantum algorithm also ensures that the reporting of an algorithm to the user is consistent.

An algorithm is a list of instructions. The instructions that are used in the standardised quantum algorithms take the form shown in Figure 4.6. The list of values that the Instruction element can take can be found in Appendix D.

*Gate*1, *Gate*2 and *Phase* are all listed as being of type *expnode*. The final type for *Gate*1 and *Gate*2 is integer and for *Phase* is double. They are not explicitly these types to increase the expressive power of the algorithms. The grammar that defines the *expnode* type can be seen in Figure 4.7. The *expnode* type is needed so that the algorithms can react to the parameterisation that provides the increased power when compared to quantum circuits. As the circuit size is a variable a constant cannot be used for *Gate*1, *Gate*2 or *Phase* as the value may depend on the circuit size and have to be calculated when a circuit is being built using the algorithm.

```
1  for(int i = 0; i < n ; i++){
2    // You can use i here
3    for(int j = 0; j < n ; j++){
4      // You can use i and j here
5    }
6  }
```

With the inclusion of an loop control construct in the form of the *iterate* and *reviterate* instructions additional parameters are introduced. In Java and many programming languages, as can be seen in the example of Java code above, when using nested loops access to all loop variables is provided. This means that at line 2 the loop variable *i* is able to be used but at line 4 both loop variable *i* and *j* can be used. This nesting is quite a powerful language feature. With the inclusion of the iterate instructions and therefore the posibility of nested iterations this loop variable access could be implemented in one of two ways. The most naïve implementation would be to just allow access to the "closest" loop variable. This would mean that the code at line 2 wouldn't be effected but the code at line 4 would no longer be able to use the *i* variable. The more sophisticated option, that is implemented, is to provide access to all loop variables. This is a second parameter that requires the use of the *expnode* type. All current iteration variables are provided in a *LoopVars* array. As can be seen in Figure 4.7 the array is indexed by the value of a sub-expression. To ensure that all indicies requested are valid the value of the sub-expression is calculated using modulus *LoopVars.length()*.

The difference between the *iterate* and *reviterate* instructions is that *iterate* counts from 1 to *Gate*1 and *reviterate* counts from *Gate*1 to 1. As is explained in Section 4.1.10, the qubits are numbered from 1 to *n* which is why the iteration instructions count to and from 1 rather than 0 as is normal

in Computer Science. The two different interate instructions are needed as they can express some looping constructs in a much simpler form that would be possible with just one of them.

For all *Create_** instructions, *Gate*1 is used to index the qubit the actual gate should be assigned to. *Gate*2 is only used by *Create_C** and *Create_SWAP* instructions to index the control qubit and second qubit respectively. For the *Iteration* instruction, Gate1 is used for the number of iterations, *n*. For the *Create_R**, *Create_CR**, *Create_P* and *Create_CP* instructions, the Phase element is used to parameterise those gate to specify the amount of rotation applied by the resulting gate.

4.1.10 Qubit Numbering

One of the major decisions made relating to the way in which the produced quantum algorithms are produced was that of which way the qubits should be numbered. The two options were obviously in accending or decending order.

The chosen approach was the decending order. This meant that for state $|ab\ldots st\rangle$ the qubit represented by *a* would always be given the identifier equal to the number of qubits in the system. For example, if there were three qubits in the system the identifier of the qubit represented by *a* would be 3. This was chosen to ensure that an identifier always represented the same qubit, irrespective of the number of qubits in the system.

The justification for this is to make the algorithm much more understandable. If the identifers were dependant on the number of qubits it would make the results of the system much less comprehensible.

The use of this numbering is also much more natural as the identifer, *x*, of a qubit, *a*, is related to the value of the qubit when read in binary. The value of the qubit *a* is 2^{x-1}. This makes the optimisation of gate application, see Section 4.1.12, much simpler.

4.1.11 Quantum Circuits

To perform the evaluation of an algorithm the circuits for the test sets need to be produced. Both the representation of the circuit and the mechanism to construct the circuit from the algorithm needed to be standardised to ensure the "Plug and Play" nature of the framework.

The framework provides a default circuit builder. The framework does allow a separate circuit builder to be provided as long as it conforms to the interface and the circuits it produces also adheres to the respective interface. There is no manager class provided for circuit builders. This was due to an assessment of the intended uses of the framework. It is intended that the framework would be used primarily to perform the following:

- Perform research into the effect of different fitness functions on the search for quantum algorithms

- Perform research into different search techniques that could be used to produce quantum algorithms

- Perform research to produce new quantum algorithms for a specific problem

It is not seen as a priority of the system to provide the same level of flexibility to the circuit building as the search engines and fitness functions.

The circuits that are produced by a circuit builder are hidden behind an interface. This is to allow third party circuit builders to use their own internal representation and also to allow any future optimisations made in future work on this framework to be made without impacting the work of researchers.

The circuits produced provide the represented quantum circuit as an ordered iterator of quantum gates. The use of an ordered iterator rather than a specific data structure is to ensure that any future optimisation or third party circuit representation is not limited. It also reduced the potential errors involving the interpretation of a more complex data structure.

The circuits also provide a Latex representation to allow the circuit to be visualised. The Latex representation uses the QCircuit package that can be freely obtained at [31].

$$\begin{pmatrix} a \\ b \end{pmatrix} \rightarrow \begin{pmatrix} b \\ a \end{pmatrix}$$

$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \rightarrow \begin{pmatrix} b \\ a \\ d \\ c \end{pmatrix}$$

Figure 4.8: Visual Representation of Bit Manipulation Equivelent of Pauli X Operation on Qubit 1

The circuit implementation provided in the framework

4.1.12 Quantum Gates

Any quantum circuit will be a series of quantum gates on specified qubits. The quantum gates provided by the system are hidden behind an interface. This is to ensure that any future optimisation of any gate's implementation cannot interfere with the implementation of any other component of the system.

Each quantum gate is required to provide a unitary matrix but it is not required that the matrix must be used in the application of the gate. For quantum circuits with a high number of qubits, the cost of simulation increases rapidly. This is mainly due to the increase in state vector and unitary matrix sizes. Matrix multiplication is used to apply a unitary operation to a state vector, yet it is a very expensive operation.

To improve the performance optimisations can be applied for several gate types. This is most obvious when analysing the operation of the Pauli X gate. Figure **??** shows, with the help of colour, that the application of a Pauli X gate on Qubit 1 is essentially a flip of neighbouring values. This is also true for a Pauli X gate on any other qubit, just the definition of a state's "neighbour" is modifed with respect to the identifier of the qubit on which the gate is applied.

The use of these tricks is not specified but the interface has been designed to ensure that the gate implementations can use such tricks or matrix multiplication interchangeably.

Each gate must also provide a QCircuit represenation for use by the circuit to produce the QCircuit representation of the complete circuit.

The implementation of gates effecting two qubits are hidden by an extended interface to provide access to the identifier of the second qubit but ensures that all standard gate operations are also available.

4.1.13 Search Engine Parameters

With many search techniques there are a number of parameters that can be configured and altered, sometimes having dramatic effects on the search results. To enable the configuration of these parameters through the framework a suitable, and flexible method had to be introduced into the design. Not all search techniques have the same number of parameters to configure and due to search engines being developed potentially by different researchers, even if the parameters are the same, their internal representation may be different. To ensure that all parameters that any particular search engine requires can be configured, the user interface for the configuration is provided by the search engine. The user interface is set to be an onscreen dialog box to ensure that the configuration is controlled by the search engine rather than any third party software or the framework.

This option was chosen as it is fair to assume that the person with best understanding of the parameters provided by a search engine is the same person who implemented the search engine using the parameters. It therefore follows that to reduce the risk of misunderstanding between GUI developers and search engine developers, who could be spacially and temporally separated, the search engine configuration should be tightly maintained within the search engine implementation.

A second justification for this based on the "Plug and Play" nature that is a foundataion of the framework. If the configuration were provided by third party software providing a GUI for the framework, or embedding it in a larger software suite, the third party software would have to be adapted for each search engine implementation. This would not be acceptable and is totally against the motivation behind the framework. With such a design it is likely that the research community would see no advantage to using the framwork than working as they are now, in complete isolation.

4.1.14 Design as a Black Box

The framework is design to be used as a black box. All external code is only allowed access through the use of interfaces and final or abstract class types. The implementation of the internal framework functionality is hidden and not required by third party software. This desgin ensures that third party software shall be loosely coupled with the framework.

4.1.15 Step-By-Step Evaluation

The framework is required to provide a step-by-step evaluation facility. There are two abstract ways this can be provided. The framework can provide an interactive process that only applies a gate when it receives the command to do so and reporting the "current state". The second option is to record the "current state" at each point in the circuit.

The main advantage of the first method is that it doesn't require processing of gate applications if the interactive evaluation does not reach them. A second advantage is that the memory requirement is very small as only the "current state" and the position in the circuit the evaluation is up to. A disadvantage is that for complex gates the application of the unitary operation could take longer than it is acceptable for a user to wait. A second disadvantage is that if the memory required is kept to the minimum, moving backward will also require the application of a gate. However, the gate that needs to be applied is not necessarily the same gate that is applied when working forward. This could make the step-by-step evaluation quite complex to implement. The time required for the reversed gate to be calculated and applied could also be an issue for responsiveness.

The main advantage of the second alternative is that to move forward, and also to move backward, is a simple loading of the respective "current state" from a data structure. This results in the requests from the user to move forward and backward in the circuit taking approximately constant time irrespective of the steped over gate's complexity. A second advantage is that taking a step backward in the evaluation never requires the inverse of a gate to be calculated. For gates such as the Pauli gates this is not an issue as they are their own inverse. However this is not the case for arbitrary gates requiring the inverse of aribitary matricies to be calculated. The main disadvantage of the trace method is that the memory requirements are not constant, they are linear with respect to the number of gates in the circuit but exponentially with respect to the number of qubits in the system. The second disadvantage is that although stepping through a circuit does not require the application of gates to the "current state", the full trace has to be produced before the step-by-step evaluation can be performed. This means that if the circuit produced is large and complex, the start up and initialisation time could be significant. The requirement for the full circuit to be evaluated irrespective of howmany steps are taken in the step-by-step evaluation could lead to trace elements never being reached.

The way the framework has been designed is to include the second of these two options. The additional memory requirements are not likely to be excessive when viewed in respect to the amount of RAM available in the average PC. The main reason for the choice was that the calculation of all "current state"s is done once and can be accessed as many times without extra computation required. This should make the reponsiveness if the framework much better.

There is also a second major design decision to be made with respect to the step-by-step evaluation, what the initial state should be to produce the trace. There are really two alternatives. The initial state could either be provided by the user explicitly or the test cases of the search problem could be used as the source of the initial states.

The decision was made to combine the two options. The framework accepts as an input to the step-by-step tracer a test suite data structure. The test suite can either be the same that was provided by the Search Problem and used by the suitability measure or can be a new test suite that has been created. This allows the client or third party application to provide either or both of the options.

4.1.16 Batch Processing

The framework produced is not expected to be used solely for Quantum Algorithm research. It is also expected to be used for research into the search techniques themselves. As a result the ability to carry out multiple iterations, a batch, and for the results to be collated to perform statistical analysis on them. To enable this the result of a search from Search Engine is returned in an array. This allows any number of results to be returned.

The framework has been designed so as not to provide any additional help for batch processing for a very simple reason. The way that batch processing is carried out is therefore not constrained. This allows the researcher to select how each iteration should be processed and what statistics need to be collected at which stage. If the framework were to manage the batch operation researchers would be restricted.

For example, the framework could be implemented to sequentially execute each iteration. For a researcher who has access to a cluster this could be highly frustrating. Equally if the framework were to provide a distributed computing mechanism, the researchers would be forced to use this. This would mean that even if the chosen framework would be highly inefficient for a specific search technique there would not be any option available to the researcher other than to modify the framework. This is obviously not desirable from a "Plug and Play" perspective.

The prescription of a batch processing manager may also restrict the statistics that are required by the researcher. As a result of the reporting of statistcs is also not provided by the framework but expected of the Seach Engine.

4.2 Provided Tools

4.2.1 Graphical Test Suite Editor

4.2.2 Graphical Matrix Editor

4.3 Provided GUI

To design the provided GUI I followed the following principles:

- **The structure principle:** Design should organize the user interface purposefully, in meaningful and useful ways based on clear, consistent models that are apparent and recognizable to users, putting related things together and separating unrelated things, differentiating dissimilar things and making similar things resemble one another. The structure principle is concerned with overall user interface architecture.

- **The simplicity principle:** The design should make simple, common tasks easy, communicating clearly and simply in the user's own language, and providing good shortcuts that are meaningfully related to longer procedures.

- **The visibility principle:** The design should make all needed options and materials for a given task visible without distracting the user with extraneous or redundant information. Good designs don't overwhelm users with alternatives or confuse with unneeded information.

- **The feedback principle:** The design should keep users informed of actions or interpretations, changes of state or condition, and errors or exceptions that are relevant and of interest to the user through clear, concise, and unambiguous language familiar to users.
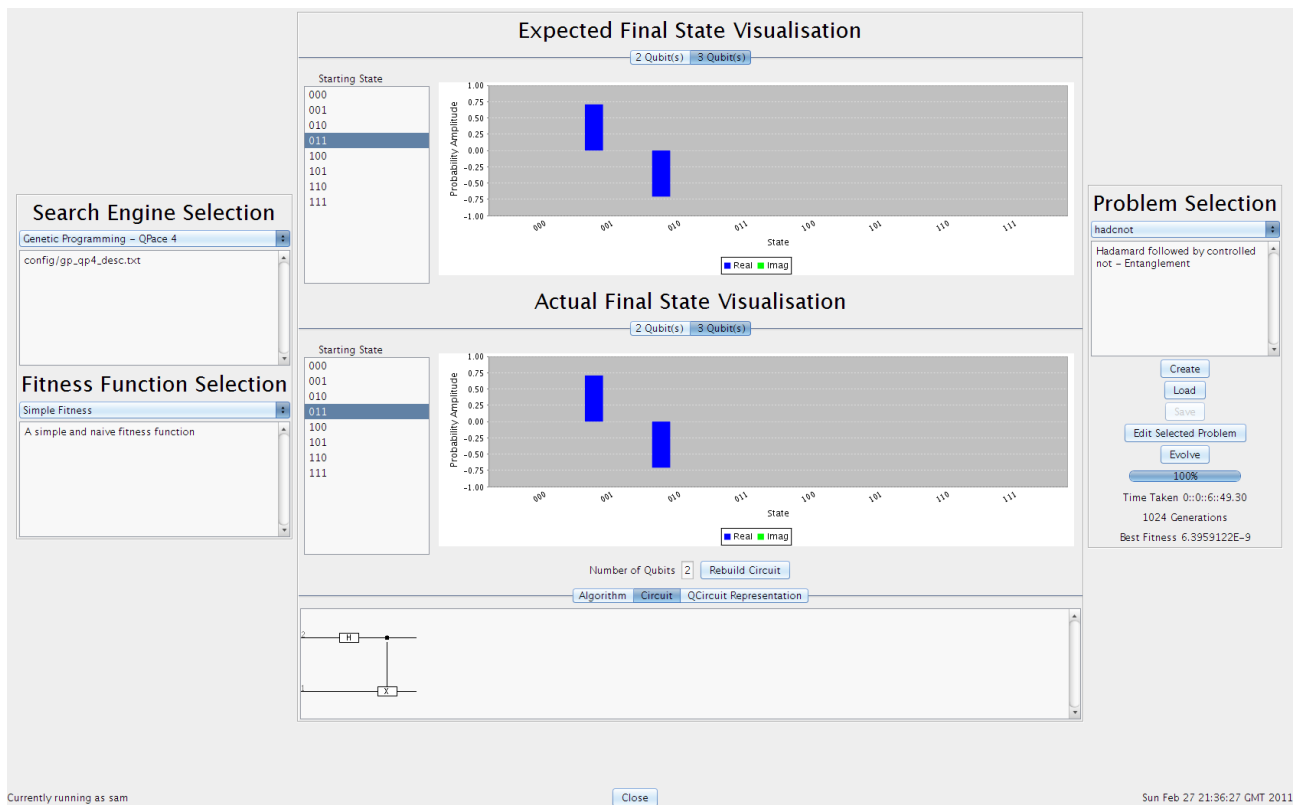
Figure 4.9: Main User Interface - After Search

- **The tolerance principle:** The design should be flexible and tolerant, reducing the cost of mistakes and misuse by allowing undoing and redoing, while also preventing errors wherever possible by tolerating varied inputs and sequences and by interpreting all reasonable actions.

- **The reuse principle:** The design should reuse internal and external components and behaviors, maintaining consistency with purpose rather than merely arbitrary consistency, thus reducing the need for users to rethink and remember.

Taken directly from REFERENCE. Where additional principles are used they are explained alongside the design element they refer to.

The design of the main screen of the GUI can be seen in Figure 4.9. The figure shows the main screen after a a search has been completed.

Each section of the GUI is explained separately with reference to the principles listed.

4.3.1 Main Window Layout

As can be seen in Figure 4.9 the layout of the main window separates the "dissimilar things" with the use of visible but subtle borders. This is a result of both the *structure* principle. The interface is structured so that the centre of the display contains the information of the highest importance, strafed by two control menus. This central panel collates all of the main results of the latest search.

The layout is also intended to take into account the recent move towards wide screen monitors. Wide screen monitors provide a new problem in GUI design. If a GUI fills the screen area and fills it fully with the display of information, it can appear stretched and distorted. If the GUI were to have a single menu along one side it also doesn't look "right", it looks excessively heavy on the non-menu side. This is a issue with standard monitors also but in my opinion is exacerbated by the wide screen ratios. Although it isn't directly related to any of the design principles it is in my opinion an important property of a GUI to appear well balanced across the available screen area.

With the two menu panels the separation of the configuration options allows for a simple layout of configuration. On the main window the only configuration that is available is the selection of the

Search Engine, Suitability Measure and Search Problem. The configuration of the Search Engine and the definition of the Search Problem is not handled by the main window. This is to ensure that the display does not become overly cluttered, it maintains a simple and clean appearance. This is a result of both the *simplicity* and the *visibility* principles.

4.3.2 Search Engine, Suitability Measure and Search Problem Selection

Before a search can be started, selections have to be made for the Search Engine, Suitability Measure and Search Problem. The selection of these are provided by three drop down lists. The available options for a user to select are limited to the Search Engines and Suitability Measures that are registered in the respective managers in the framework. For a user to add a new Search Engine or Suitability Measure the respective XML configuration file needs to be updated. This was done so as to follow the *simplicity* and *tolerance* principles. It ensures that any selection made by the user is a valid selection.

The selection of the Search Problem is again provided by a drop down list, following the *simplicity* and *tolerance* principles. The difference is that the creation of a new problem from scratch and using a predefined test suite held in an XML file can be performed within the GUI. Despite this difference, the use of the drop down list still ensures that any selection made by the user is a valid selection. The validity of each individual Search Problem is checked by the editor dialogs described next.

All three of these selections are performed in the same way and ensure that the GUI maintains a level of consistency. When a selection is made, the selection is shown and a description, provided in the configuration XML files, is shown. The combination of the description and the consistency were included for the *reuse* and *feedback* principles.

4.3.3 Search Problem Creator and Editor

As mentioned above, the creation of Search Problems is provided by an on-screen dialog. The creation and editing of Search Problems is also provided by a standalone application. The inclusion of a standalone application was thought to be useful as the framework is intended to be used within third party software so having to run the full GUI to create a Search Problem isn't desirable. This also allows a third party application to integrate the standalone editor.

The integrated editor and the standalone application use the same components and overall design. The same components are also used when creating a new Search Problem and when editing an existing Search Problem.

Not only are the individual components reused but each dialog follows the same design layout. This promotes the *reuse structure* principles.

The dialogs ensure that the user has entered values for the required fields and ensures that each entry is valid. This implicitly ensures that each selection available to the user in the Search Problem drop down list is a valid option. This follows the *Tolerance* principle.

4.3.4 Reporting Results

When a Search Problem is selected, in the central area a visual representation of the test suite is produced. This representation can be seen in Figure 4.10.

After a search has been completed, the final states produced by the realised algorithm are shown using the same representation. This can be seen in Figure 4.9. Using a simple visualisation like this makes the comparison between "desired" final states and the final state produced by the algorithm found by the search. It is true that the visualisation could be too simple for small differences to be noticed. To counter this problem the visualisation allows the user to hover the mouse over each column to get an actual value. This can be seen in Figure 4.11.

This provides users with both a quick, simple and visual way to compare final states as well as an accurate way to compare final states. The accurate comparison method provided was implemented rather than a value table as part of following the *simplicity* principle.
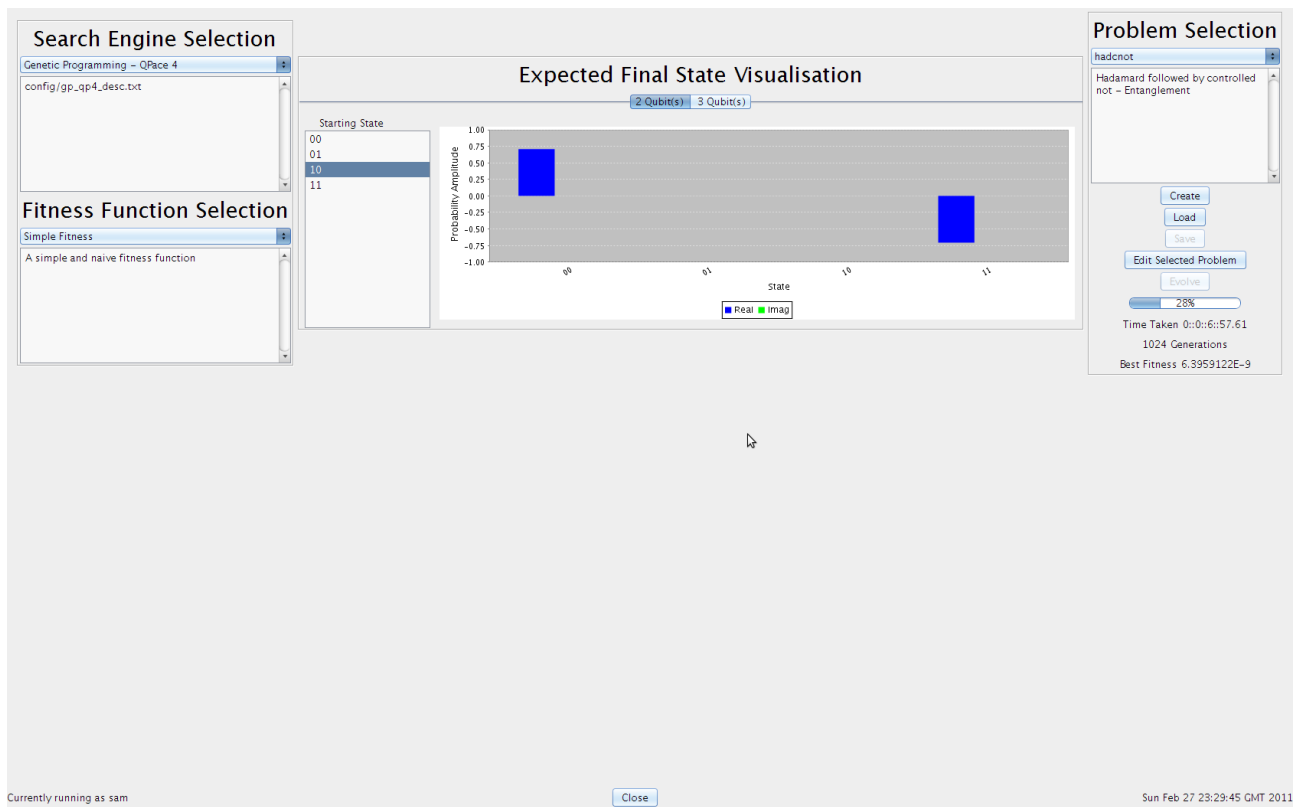
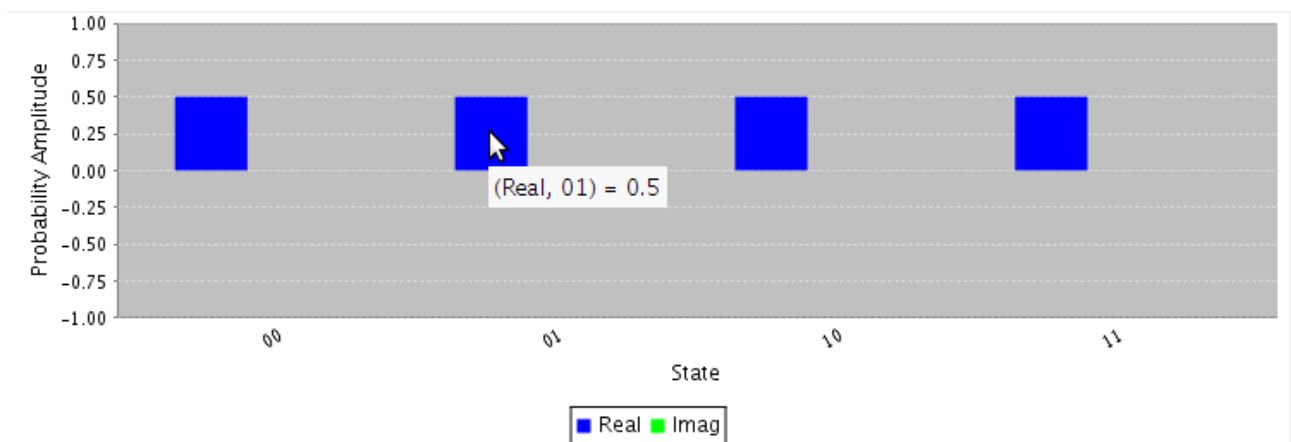Figure 4.10: Main User Interface - Before Search

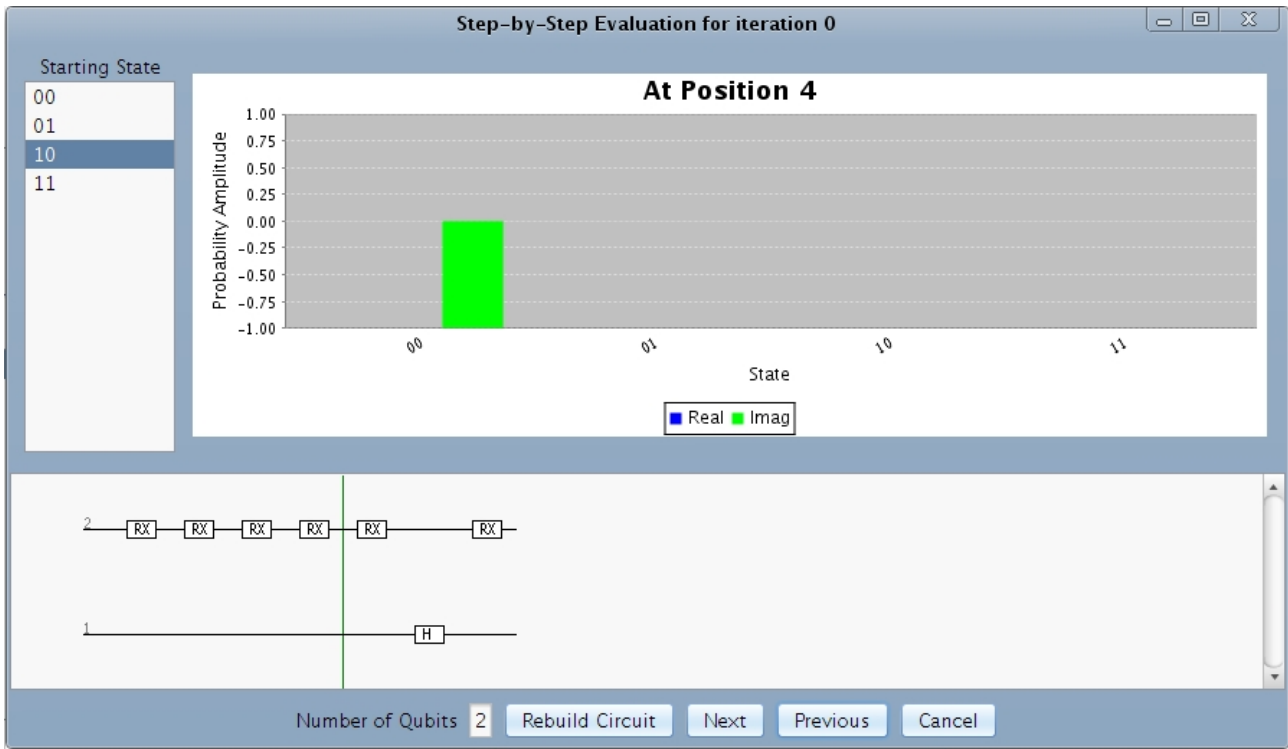

Figure 4.11: Accurate State Readout

Figure 4.12: Step-By-Step Evaluation Dialog

The result of a search is the quantum algorithm found by the search, rather than the final states for the test cases described so far. The GUI provides the user 3 different ways to see the search result. A simple textual listing of the algorithm is provided in the same format as the framework produces on its own. To help with the users understanding of what the algorithm means a circuit diagram is produced for a user controlled number of Qubits. The diagram is produced using the symbols that are shown in Figure 3.1. The circuits produced are not just provided as a circuit diagram but also in QCircuit representation so that the circuits can be placed into any publication produced in Latex simply with the use of the QCircuit package.

Providing the three representations meet the *simplicity*, *feedback* and *reuse* principles. The *simplicity* principle is met as the result of the search is simplified to a human readable algorithm and circuits can be created to help the user understand how the algorithm is working. The *feedback* principle is met as the circuit diagrams that are produced are done so using widely accepted symbols and conventions for quantum circuit drawing. The *reuse* principle is met with the use of QCircuit to produce circuits in a form that can be included in publications. An alternative would have been to output the circuit diagram that is drawn by the GUI as an image that could have been included in any, not just Latex, publication. I feel the use of QCircuit is a better choice as the user then has control and is able to carry out, if necessary, manual circuit optimisation.

4.3.5 Step-By-Step Evaluator

The way in which quantum algorithms and the circuits they produce work is usually subtle and hard to understand by simply looking at the circuit. The framework provides a step-by-step evaluation trace when provided with the input states. The input states are provided to the framework in a test suite structure. It was decided that the provided GUI would provide the test suite of the respective Search Problem. This means that an evaluation trace is produced for each of the test cases, in each of the test suites.

The step-by-step evaluator is provided in a dialog rather than integrated in the main frame, this dialog can be seen in Figure 4.12. This was done so as to focus the users attention and to ensure that the addition of the functionality did not result in a cluttered GUI. This follows the *structure* and *visibility* principles.

The step-by-step evaluation is performed with respect to a produced circuit. This requires the user to select the number of qubits the circuit should be produced for and the step-by-step evaluation is provided for all test cases of that number of qubits. Due to the design decision made for the framework, to provide a full trace rather than interactive evaluation, the test cases can be switched between at any step without returning to the start of the circuit.

The dialog provides a circuit diagram, an initial state selector and a visual representation of the state at the "current step" in the evaluation for the selected initial state. The circuit diagram is produced by reusing the circuit diagram drawn in the results pane of the main window. This ensure that the circuit is represented using the same standards as that shown in the results pane. The only difference is that the "current step" is indicated using a vertical line on the circuit diagram, this can be seen in Figure 4.12. This follows the *reuse*, *feedback*, *visibility* and *simplicity* principles.

The visual representation and initial state selector are the same as those used to report the final states produced for the test suite in the main window. The only difference is that only the test cases for the current number of qubits is shown. The use of the same visual representation ensure the user does not have to understand anything extra to use this functionality and follows the *reuse* principle.

5 Testing

5.1 Unit Tests

5.2 Integration Tests

5.3 System Testing

5.4 User Acceptance Tests

Bibliography

[1] P. Gawron, "File:bloch.png - quantiki | quantum information wiki and portal," http://www.quantiki.org/wiki/File:Bloch.png.

[2] Qcircuit tutorial. [Online]. Available: http://www.cquic.org/Qcircuit/Qtutorial.pdf

[3] R. Feynman and P. W. Shor, "Simulating physics with computers," *SIAM Journal on Computing*, vol. 26, pp. 1484 – 1509, 1982.

[4] D. Deutsch, "Quantum theory, the church-turing principle and the universal quantum computer," *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, vol. 400, no. 1818, pp. pp. 97–117, 1985. [Online]. Available: http://www.jstor.org/stable/2397601

[5] P. W. Shor, "Polynomial time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM J. Sci. Statist. Comput.*, vol. 26, p. 1484, 1997.

[6] P. Dirac, *The principles of quantum mechanics*. Oxford University Press, 1958.

[7] I. Glendinning, "The bloch sphere - talks and posters on quantum computing by ian glendinning," http://www.vcpc.univie.ac.at/~ian/hotlist/qc/talks/bloch-sphere.pdf.

[8] E. Schröginger, "[A translation by John D. Trimmer] 'The Present Situation in Quantum Mechanics'," http://www.tu-harburg.de/rzt/rzt/it/QM/cat.html.

[9] P. Massey, *Searching for Quantum Software*. University of York, 2006.

[10] P. W. Shor, "Progress in quantum algorithms," *Quantum Information Processing*, vol. 3, pp. 5–13, October 2004. [Online]. Available: http://portal.acm.org/citation.cfm?id=1032132.1032149

[11] D. Deutsch and R. Jozsa, "Rapid solution of problems by quantum computation," *Proc Roy Soc Lond A*, vol. 439, pp. 553–558, October 1992.

[12] C. E. Macchiavello, B. Y. R. Cleve, A. Ekert, and C. Macchiavello, "Quantum algorithms revisited," in *Proceedings of the Royal Society of London A*, 1997, pp. 339–354.

[13] L. K. Grover, "A fast quantum mechanical algorithm for database search," 1996.

[14] C. H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani, "Strengths and Weaknesses of Quantum Computing," 1996.

[15] G. Folland and A. Sitaram, "The uncertainty principle: A mathematical survey," *Journal of Fourier Analysis and Applications*, vol. 3, pp. 207–238, 1997, 10.1007/BF02649110. [Online]. Available: http://dx.doi.org/10.1007/BF02649110

[16] P. Massey, *Evolving Quantum Programs and Circuits*. University of York, 2000.

[17] D. E. Goldberg, *Genetic algorithms in search, optimization and machine learning*, Goldberg, D. E., Ed., 1989.

[18] L. Spector, H. Barnum, H. Bernstein, and NewAuthor4, "Genetic programming for quantum computing," in *Genetic Programming 1998 - Preceedings of the Third Annual Conference*, 1988, pp. 365–374.

[19] L. Spector, H. Barnum, H. Bernstein, and N. Swamy, "Finding a better-than-classical quantum and/or algorithm using genetic programming," in *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, 1999.

[20] L. Spector, H. Barnum, H. J. Bernstein, and N. Swamy, *Quantum computing applications of genetic programming*. Cambridge, MA, USA: MIT Press, 1999, pp. 135–160. [Online]. Available: http://portal.acm.org/citation.cfm?id=316573.317112

[21] V. Vedral, A. Barenco, and A. Ekert, "Quantum Networks for Elementary Arithmetic Operations," 1995.

[22] S. Stepney and J. A. Clark, "Searching for quantum programs and quantum protocols: a review," 2007.

[23] C. P. Williams and A. G. Gray, "Automated design of quantum circuits," in *Selected papers from the First NASA International Conference on Quantum Computing and Quantum Communications*, ser. QCQC '98. London, UK: Springer-Verlag, 1998, pp. 113–125. [Online]. Available: http://portal.acm.org/citation.cfm?id=645812.670824

[24] G. Brassard, S. L. Braunstein, and R. Cleve, "Teleportation as a quantum computation," *Physica D Nonlinear Phenomena*, vol. 120, pp. 43–47, Sep. 1998.

[25] T. Y. Yabuki, "Genetic algorithms for quantum circuit design –evolving a simpler teleportation circuit–," in *In Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference*. Morgan Kauffman Publishers, 2000, pp. 421–425.

[26] Requirementone requirements management tool. [Online]. Available: http://www.requirementone.com/Free_project_management_tool.aspx

[27] "Ieee recommended practice for software requirements specifications," *IEEE Std 830-1998*, 1998.

[28] Ecj. [Online]. Available: http://www.cs.gmu.edu/~eclab/projects/ecj/

[29] Complex number implementation. [Online]. Available: http://www.math.ksu.edu/~bennett/jomacg/

[30] Jama matrix library. [Online]. Available: http://math.nist.gov/javanumerics/jama/

[31] Qcircuit. [Online]. Available: http://www.cquic.org/Qcircuit/

A Full Requirements

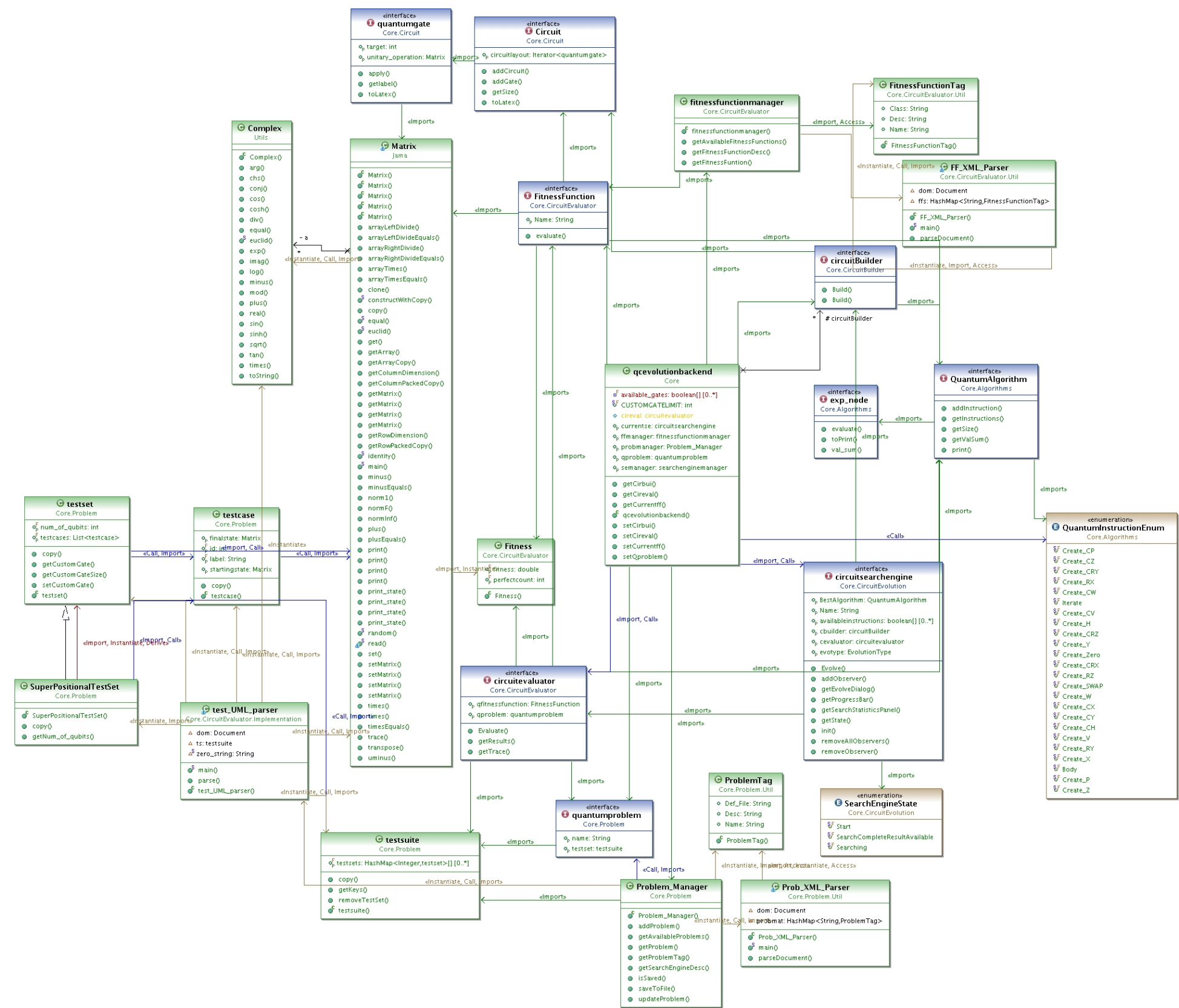## A.1 Framework

A.2 Fully Functional Tool

A.3 Client and GUI

B Architechture Diagram

C XML Outlines

C.1 Search Engine XML Outline

```
1   <searchengine>
2         <se>
3            <Name>SEARCH ENGINE NAMES</Name>
4            <Class>IMPLEMENTING FULLY QUALIFIED CLASS NAME</Class>
5            <Desc>SEARCH ENGINE DESCRIPTION</Desc>
6         </se>
7   </searchengine>
```

C.2 Problem Definition XML Outline

```
1   <Problems>
2         <prob>
3            <Name>PROBLEM NAME</Name>
4            <DefFile>PROBLEM DEFINITION FILE</DefFile>
5            <Desc>PROBLEM DESCRIPTION</Desc>
6         </prob>
7   </Problems>
```

D Available Algorithm Instructions

Instruction	Action		Instruction	Action
Create_H	Create a Hadamard Gate		**Create_CH**	Create a Controlled Hadamard Gate
Create_X	Create a Pauli-X Gate		**Create_CX**	Create a Controlled Pauli-X Gate
Create_Y	Create a Pauli-Y Gate		**Create_CY**	Create a Controlled Pauli-Y Gate
Create_Z	Create a Pauli-Z Gate		**Create_CZ**	Create a Controlled Pauli-Z Gate
Create_P	Create a Phase Gate		**Create_CP**	Create a Controlled Phase Gate
Create_V	Create a V Gate		**Create_CV**	Create a Controlled V Gate
Create_W	Create a W Gate		**Create_CW**	Create a Controlled W Gate
Create_RX	Create a Rotate-X Gate		**Create_CRX**	Create a Controlled Rotate-X Gate
Create_RY	Create a Rotate-Y Gate		**Create_CRY**	Create a Controlled Rotate-Y Gate
Create_RZ	Create a Rotate-Z Gate		**Create_CRZ**	Create a Controlled Rotate-Z Gate
Create_SWAP	Create a SWAP Gate			
Iterate	Run Sub-Algorithm$[0]$ for n iterations		**RevIterate**	Run Sub-Algorithm$[0]$ for n iterations in reverse order
Body	Perform each Sub-Algorithm in turn			
Create_Custom1	Create Custom Gate Number 1		**Create_CCustom1**	Create a Controlled Custom Gate Number 1
Create_Custom2	Create Custom Gate Number 2		**Create_CCustom2**	Create a Controlled Custom Gate Number 2
Create_Custom3	Create Custom Gate Number 3		**Create_CCustom3**	Create a Controlled Custom Gate Number 3