The University of Edinburgh
School of Physics and Astronomy

Quantum Computing Project: Report

by
Michael Chiang
Gennaro di Pietro
William McNichols
Christoph Messmer

Edinburgh, 24th of March 2015

Contents

1 Introduction

This report is part of the result within the context of the course 'Quantum Computing Project' at the University of Edinburgh and is meant to be comprehensible to 3rd year undergraduate students of physical science (especially informatics and physics).

We start off describing the aims of the project and are going to provide a background for the topic of Quantum Computing. That is followed by a chapter about the theory needed for the course which is, however, purposely not intended to be exhaustive (see references for further information).

After that we will describe the implementation of the project including how we organised the programming, which development environment we used, how we structured the program, etc.

We finish with a chapter about the results followed by a discussion.

1.1 Aims

The aims of the project:

- **Comprehend quantum computing**
  One goal is to get familiar with quantum computing as a generalisation of conventional computing.
- **Programming**
  The main goal is to simulate a quantum computer on a conventional classical computer. This includes programming basic concepts like qubits, quantum registers and quantum gates. Finally it should be possible to run quantum algorithms like Grover's algorithm (optional: Shor's algorithm).
- **Presenting results**
  This includes not only this report and a verbal presentation but also proper documentation of the programming code to make it comprehensive to other programmers.
- **Teamwork and organisation**
  A project like this needs organisation and division of task but also successful communication between all group members. It is a further goal to encourage teamwork and organisation skills.

1.2 Background

The history of computers reaches back to the middle of the 19th century when a design for an Analytical Engine was proposed by Charles Babbage who is considered to be one of the early pioneers of computation. However, for almost 100 years this branch stayed an interesting but rather conceptional one until the invention of the transistor in 1925. The first working computers were built in the 1940s and up to today computers work principally the same way.

Quantum computation on the other hand is a quite recent research field which emerged from the physics of quantum mechanics (1920s). In 1982 Richard Feynman theorised that there seemed to be essential difficulties in simulating quantum mechanical system on classical computers and suggested that a quantum computer would solve these issues.[4]

Remarkable theoretical breakthroughs in the 1990s followed, when Peter **Shor** demonstrated that essential problems – like factorising integers – could be solved far more efficiently on quantum computers than on conventional, classical computers. Besides Shor's algorithm Lov **Grover** proposed another algorithm in 1995 (only one year later) showing that the problem of conducting a search through some unstructured search space is as well more efficient on quantum computers.

The practical challenges for building a quantum computer are high and therefore the realisation of real quantum computers is still in it's infancy. However, in 2001 the first real quantum computer was able to factorise 15 into its prime numbers (3 and 5) by using a 7-qubit system.[6] Since then experimental progress is booming but the state-of-the-art is still a fair way off from practical (and even less daily-life) usage.

2 Theory

In this chapter we introduce the basic concepts of quantum computation, starting off with definitions of **qubits**, **quantum registers** and the presentation of several **quantum gates**. Afterwards we will talk about two **quantum algorithms** that we implemented in our virtual quantum computer. The last chapter will briefly talk about the challenges of **building a real quantum computer**.

2.1 Qubits

2.1.1 Generalised bits

A qubit (from *qu*antum *bit*) is the smallest unit in a quantum computer and therefore the quantum mechanical **generalisation of a classical bit**, as it is used in computers nowadays. A classical bit has one and only one of the two possible states

$$|0\rangle \quad \text{or} \quad |1\rangle \tag{1}$$

at the same time, whereas a qubit is able to be in a state $|\Psi\rangle$ which is a superposition of these two classical states:

$$|\Psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \quad \text{where } |\alpha|^2 + |\beta|^2 = 1 \tag{2}$$

One can depict the states via matrices with basis $(|0\rangle\,, |1\rangle)$:

$$|0\rangle = \begin{pmatrix}1\\0\end{pmatrix}, \qquad |1\rangle = \begin{pmatrix}0\\1\end{pmatrix}, \qquad |\Psi\rangle = \begin{pmatrix}\alpha\\\beta\end{pmatrix} \tag{3}$$

2.1.2 Measurement

The superposition of states leads to a new understanding of measurement. There are two things to consider:

1. **Probabilities** P
   Given a classical state ($|\Psi\rangle$ either $|0\rangle$ or $|1\rangle$) the result of a measurement is certain (and trivial). That's no longer true for the quantum state: Given the state $|\Psi\rangle$ in Eq. 2, it is solely possible to calculate the **probabilities** P_Ψ of the outcome:

$$P_\Psi(0) = |\langle 0|\Psi\rangle|^2 = \left|\alpha \underbrace{\langle 0|0\rangle}_{=1} + \beta \underbrace{\langle 0|1\rangle}_{=0}\right|^2 = |\alpha|^2 \tag{4}$$

$$P_\Psi(1) = |\langle 1|\Psi\rangle|^2 = \left|\alpha \underbrace{\langle 1|0\rangle}_{=0} + \beta \underbrace{\langle 1|1\rangle}_{=1}\right|^2 = |\beta|^2 \tag{5}$$

2. **Collapse of** $|\Psi\rangle$
   In classical measurements it is fair to say that the measurement itself has no (noticeable) influence on the result. This is no longer true in quantum mechanics: The wave function

$|\Psi\rangle_i$ **collapses after a measurement** to a projection onto the measured eigenstate and therefore becomes a different state $|\Psi\rangle_f$:

$$|\Psi\rangle_i = \alpha|0\rangle + \beta|1\rangle \xrightarrow{\text{Measurement: Value } m} |\Psi\rangle_f = \begin{cases} |0\rangle\,, \text{ if } m = 0 \\ |1\rangle\,, \text{ if } m = 1 \end{cases} \tag{6}$$

2.2 Quantum register

A quantum register of size n is a **collection of n qubits**. Therefore we get $N \equiv 2^n$ basic states:

$$|b_{n-1}\rangle \otimes |b_{n-2}\rangle \otimes ... \otimes |b_1\rangle \otimes |b_0\rangle \tag{7}$$

where $b_i \in \{0, 1\}$. One can interpret this chain of zeros and ones as binary code, able to store numbers in the range of $[0, 1, ..., N-1]$. For example for a 3-qubit system we get:

$$\begin{aligned} |0\rangle \otimes |0\rangle \otimes |0\rangle &\equiv |000\rangle \equiv |0\rangle & \quad |1\rangle \otimes |0\rangle \otimes |0\rangle &\equiv |100\rangle \equiv |4\rangle \\ |0\rangle \otimes |0\rangle \otimes |1\rangle &\equiv |001\rangle \equiv |1\rangle & \quad |1\rangle \otimes |0\rangle \otimes |1\rangle &\equiv |101\rangle \equiv |5\rangle \\ |0\rangle \otimes |1\rangle \otimes |0\rangle &\equiv |010\rangle \equiv |2\rangle & \quad |1\rangle \otimes |1\rangle \otimes |0\rangle &\equiv |110\rangle \equiv |6\rangle \\ |0\rangle \otimes |1\rangle \otimes |1\rangle &\equiv |011\rangle \equiv |3\rangle & \quad |1\rangle \otimes |1\rangle \otimes |1\rangle &\equiv |111\rangle \equiv |7\rangle \end{aligned}$$

We call this collection the **computational basis** of our register.

However, in contrast to a classical system, a quantum register is able to be in a **state of superposition** which turns out to be the fundamental advantage for quantum computation. If for example the second qubit is set to a superposition $|\Psi_{b_1}\rangle = \frac{1}{\sqrt{2}}\binom{+1}{-1}$ the total state of the register will be:

$$\begin{aligned} \left|\Psi^{\text{tot}}\right\rangle &= |\Psi_{b_2}\rangle \otimes |\Psi_{b_1}\rangle \otimes |\Psi_{b_0}\rangle & (8) \\ &= |0\rangle \otimes \left[\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\right] \otimes |1\rangle & (9) \\ &= \frac{1}{\sqrt{2}}[|001\rangle - |011\rangle] & (10) \\ &\equiv \frac{1}{\sqrt{2}}[|1\rangle - |3\rangle] & (11) \end{aligned}$$

Eq. 11 is always reducible to a (tensor) product of three single states, as Eq. 8 suggests. This is not always the case. Consider the 2-qubit system where

$$\left|\Psi^{\text{ent}}\right\rangle = \frac{1}{\sqrt{2}}[|00\rangle + |11\rangle]\,. \tag{12}$$

There is no way to separate this wave function into a (tensor) product of two states $\{|\Psi_{b_1}\rangle, |\Psi_{b_0}\rangle\}$. Thus, the state $|\Psi^{\text{ent}}\rangle$ is called **entangled**.

2.3 Quantum gates

After defining the quantum register, we now want to process it through a number of so-called quantum gates. These are (mathematically spoken) **unitary operations** applied to our register in order to change its total state $|\Psi^{\text{tot}}\rangle$. Since we work in the Hilbert space, all our operations are **linear**, therefore we can represent any gate working on a n-qubit register by a $N \times N$ matrix.

In the following subsections we will first introduce the most important gates used in our project, and then speak about generalisations of these gates for bigger registers.

2.3.1 Not gate

The first example for a simple 1-qubit gate is the Not-gate. It simply maps the state $|0\rangle \to |1\rangle$ and vice versa and is therefore equivalent to a logic NOT. The representing matrix in the computational basis $\{|0\rangle, |1\rangle\}$ is:

$$G^{\text{not}} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{13}$$

$$G^{\text{not}} |0\rangle = |1\rangle \tag{14}$$

$$G^{\text{not}} |1\rangle = |0\rangle \tag{15}$$

$$\tag{16}$$

$$|b\rangle \; -\!\boxed{G^{\text{not}}}\!- \quad |1-b\rangle$$

However, this gate exists in exactly the same form for classical computation.

2.3.2 Hadamard gate

A common gate in quantum computation is the Hadamard gate. It performs the Hadamard transformation on a single qubit system in the following way:

$$G^{\text{H}} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{17}$$

$$G^{\text{H}} |0\rangle = \tfrac{|0\rangle + |1\rangle}{\sqrt{2}} \tag{18}$$

$$G^{\text{H}} |1\rangle = \tfrac{|0\rangle - |1\rangle}{\sqrt{2}} \tag{19}$$

$$|b\rangle \; -\!\boxed{G^{\text{H}}}\!- \quad \tfrac{1}{\sqrt{2}} \left[(-1)^b |b\rangle + |1-b\rangle\right]$$

The Hadamard gate is a 'real' quantum gate since it is able to set the state to a superposition of basic states.

For a n-qubit system it is necessary to define on which qubit a gate is acting. In the following we will use the subscript to depict this: The gate G_k is acting on the k-th qubit.

Note that a combination of Hadamard gates acting on every single qubit in a n-qubit register with initial state $|\Psi\rangle = |00...0\rangle$ will lead to an **uniform superposition** of all basic states:

$$\left(\prod_{k=0}^{n-1} G_k^{\text{H}}\right) |\Psi\rangle = G_{n-1}^{\text{H}} \underbrace{|b_{n-1}\rangle}_{=|0\rangle} \otimes ... \otimes G_0^{\text{H}} \underbrace{|b_0\rangle}_{=|0\rangle} \tag{20}$$

$$= \tfrac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes ... \otimes \tfrac{|0\rangle + |1\rangle}{\sqrt{2}} \tag{21}$$

$$= 2^{-\frac{n}{2}} \sum_{k=0}^{N-1} |k\rangle \tag{22}$$

2.3.3 Phase gate

The Hadamard gate already uses special properties of quantum computation, but all operations are part of the real subspace of the Hilbert space. In general the gates and quantum register can operate on a complex vector space. The phase gate is such a gate which is defined for a single qubit system as:

$$G^{\phi} = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix} \tag{23}$$

$$G^{\phi} |0\rangle = |0\rangle \tag{24}$$

$$G^{\phi} |1\rangle = e^{i\phi} |1\rangle \tag{25}$$

$$|b\rangle \; -\boxed{G^{\phi}}- \quad e^{ib\phi} |b\rangle$$

2.3.4 Extension to bigger registers

As roughly mentioned before, any single qubit gate can by applied to the k-th qubit of a n-qubit register. Consider for example an arbitrary gate G_1 in a 2-qubit system. The resulting matrix G_{tot} is:

$$G_{\text{tot}} = G_1 \otimes \underbrace{G_0}_{=\mathbb{I}_2} \tag{26}$$

$$= \begin{pmatrix} g_{00} & g_{01} \\ g_{10} & g_{11} \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{27}$$

$$= \begin{pmatrix} g_{00} \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right) & g_{01} \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right) \\ g_{10} \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right) & g_{11} \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right) \end{pmatrix} \tag{28}$$

$$= \begin{pmatrix} g_{00} & 0 & g_{01} & 0 \\ 0 & g_{00} & 0 & g_{01} \\ g_{10} & 0 & g_{11} & 0 \\ 0 & g_{10} & 0 & g_{11} \end{pmatrix} \tag{29}$$

The general expression for n-qubit systems is analogous:

$$G_{\text{tot}} = G_{n-1} \otimes ... \otimes G_k \otimes ... \otimes G_0 \tag{30}$$

$$= \mathbb{I}_2 \otimes ... \otimes \left(\begin{smallmatrix} g_{00} & g_{01} \\ g_{10} & g_{11} \end{smallmatrix}\right) \otimes ... \otimes \mathbb{I}_2 \tag{31}$$

2.3.5 Gate representations

The resulting matrix G_{tot} is in general a $N \times N$-matrix with two and only two non-zero entries in every row and column. That means that only $2 \cdot 2^n$ of 2^{n^2} entries are non-zero. The ratio is 2^{1-n} which rapidly goes towards zero for big n. Thus, most entries of the gate matrices will be zero.

This leads to the consideration of the following three possible representations:

1. **Dense matrix representation**
   A dense matrix representation is the standard representation, storing every single entry of a matrix. This leads to two major disadvantages in our special case:

   (a) *Memory*
   A classical computer reserves a certain amount of memory for every entry of a conventional matrix, no matter if the value is zero or non-zero. Therefore using dense matrices for big quantum registers might cause a serious lack of working memory.

   (b) *Calculation*
   We get a lot of trivial (and unnecessary) calculations using the standard matrix multiplication rule.

The quantum register, however, has mostly non-zero entries for most of the steps of the usual algorithms.

2. **Sparse matrix representation**
   A sparse matrix representation only stores non-zero elements of a matrix in a list. Therefore every non-zero element requires a positioning index and the number itself to store.

3. **Functional representation**
   Since there are only two non-zero elements in a row (in our special case), a functional representation would be reasonable.

2.4 Quantum algorithms

After defining quantum gates it is now simple to predict the next step: A so-called quantum algorithm defines in what manner a quantum register is applied to a sequence of quantum gates in order to achieve a certain computation result. Two very important algorithms are Grover's algorithm and Shor's algorithm which we are going to introduce next.

2.4.1 Grover's Algorithm

Grover's algorithm is a quantum search algorithm. To search through a non-ordered list with N entries on a classical computer, one would have to go through individual entries of the list, requiring $O(N)$ operations. This can be sped up to $O(\sqrt{N})$ using a quantum circuit implementing Grover's algorithm.

Suppose we are searching through a list of N elements and assume there are M solutions to the search. To make the problem simpler, we consider searching the index corresponding to the elements rather than the elements themselves. An index running from 0 to $N-1$ has been assigned to each element. The key object in Grover's algorithm is the quantum *oracle*, which, in the abstract sense, has the ability to *recognise* the solution states without *knowing* them. The oracle marks the solution states by flipping the sign of the state. Let $|x\rangle$ be one of the basis states and let $f(x) = 1$ if $|x\rangle$ is a solution state and $f(x) = 0$ if it is not a solution. The oracle performs the following linear operation (denoted as O):

$$|x\rangle \xrightarrow{O} (-1)^{f(x)} |x\rangle \tag{32}$$

What needs to be done now is to maximise the probability of observing one of the solution states. This cannot be achieved by re-applying the oracle right away, as this would undo the marking. Grover designed an algorithm to achieve this effect, and it is known as the Grover's diffusion operation. The procedure of the algorithm is as follows:

1. Apply the oracle O
2. Apply the Hadamard operation on all qubits $H^{\otimes n}$
3. Perform a phase shift of -1 on every state of the computational basis except $|0\rangle$:

$$|x\rangle \quad \longrightarrow -(-1)^{\delta_{x0}} |x\rangle \tag{33}$$

4. Apply the Hadamard operation on all qubits again $H^{\otimes n}$
5. Repeat step 1 to 4 for a certain number of times to maximise the probability of the solution states

Step 2 to step 4 is also known as the *inversion about mean* operation. Mathematically, this iteration can be written compactly as:

$$G = (2\,|\psi\rangle\,\langle\psi| - \mathbb{I})\,O. \tag{34}$$

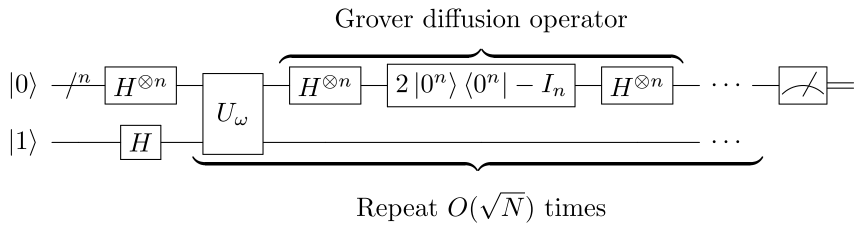


Figure 1: Circuit of Grover's Algorithm[1]

Fig. 1 shows a schematic circuit diagram for Grover's algorithm.

To determine the number of times the diffusion operation should be applied, we need to first Geometrically, the iteration performs a rotation of the state vector $|\Psi\rangle$ towards the vector representing a uniform superposition of the solution states on the plane spanned by the

$$|\alpha\rangle \equiv \frac{1}{\sqrt{N-M}} \sum_{\text{solutions}} |x\rangle \tag{35}$$

$$|\beta\rangle \equiv \frac{1}{\sqrt{M}} \sum_{\text{non solutions}} |x\rangle \tag{36}$$

Substituting these to the state vector gives:

$$|\Psi\rangle = \sqrt{\frac{N-M}{N}}\,|\alpha\rangle + \sqrt{\frac{M}{N}}\,|\beta\rangle \tag{37}$$

2.4.2 Shor's Algorithm

Shor's algorithm was published 1994 by Peter Shor and proposes an procedure for factorising integers. It is able to find non-trivial divisors in polynomial time (that means, that there exists polynomial function which is the upper time limit for the calculation), whereas classical algorithms are significantly above polynomial time (though sub-exponential).

This has vast impact on today's cryptography: Many encryption systems (like RSA)[7] rely on the fact, that it is impossible to factorise integers in a reasonable time. Shor's algorithm showed that it is (theoretically) possible for quantum computers, however the practical difficulties of building a real quantum computer are still preventing Shor's algorithm to undermine modern cryptography.

Shor's algorithm belongs to the Monte-Carlo algorithms which means that it is based on probabilistic calculations. Therefore in some cases it might lead to undesired results.

Fig. 2 shows a schematic circuit diagram for Shor's algorithm.

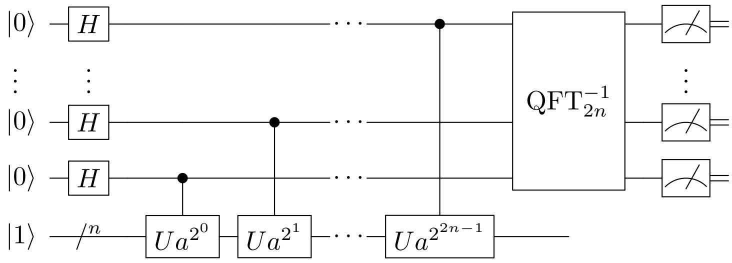


Figure 2: Circuit of Shor's Algorithm[2]

2.5 Building a quantum computer

3 Implementation

3.1 Project organisation

3.2 Development environment

3.3 Program structure

3.3.1 Overview

3.3.2 Matrix

3.3.3 The Computer

The design of the actual quantum computer simulator requires a careful consideration, as it is the backbone for running various quantum algorithms. An effective, well-structured design allows one to implement any algorithm at ease and compare the performance of the programme under different representations. Our group has identified some key specifications for the design of the computer:

1. It should be consistent with a circuit-model quantum computer.
2. It should allow different representations for each type of object within the computer (i.e. the circuit, gate and the register).
3. It should be extendible and compatible. One should be able to implement another representation for the objects within the computer that would work with the current code.
4. It should maximise code reusability

The first point is fundamental as the project brief specifically asks us to create a simulator of a circuit-model quantum computer. We therefore design the programme based on the physical objects that exist in such computer. This includes the register, which is an array storing n qubits, the quantum gates, and the circuits, which consist of a series of gates that perform more complex operations on the register, such as the quantum search algorithm. These form the three main groups of classes for the computer part of the programme.

To achieve the second to the fourth point of our specifications, we created an interface for each type of object, which are `QGate`, `QCircuit`, and `QRegister`. Each interface describes the fundamental behaviours of the corresponding object that are independent of the representation. This allows objects to interact without having to be concerned about each other's representation. It thus separates the task of representing an object and using it to build circuits for implementing the algorithms, which is key for maximising code reusability and making the programme extendible. The specific design for each of these objects and how they co-operate with each other is discussed in detail below:

Gates The QGate interface contains a single method called `applyGate(QRegister reg)`, as each gate must perform an operation on the register. As discussed in section 2.2, there are two general ways for representing a quantum gate: matrix representation and functional representation. It is therefore natural to create a class for each of these two representations, which we called MGate and FGate respectively.

A major challenge in creating the gates is how to construct the matrix or functional representation of a single qubit gate is when it is applied to a register with multiple qubits. Mathematically, this involves taking the tensor product of the linear operation of the gate on the desired target qubit with the identity operation on others, which is mentioned briefly in section 2.

Instead of carry out this tensor product explicitly, which is likely to be computationally expensive and inefficient, we notice the linear operation serves as an active transformation of the state vector. From linear algebra, we know the columns within the matrix that describes such transformation are the transformed basis vectors (the new basis) expressed in terms of the old basis. Therefore, we can construct the matrix for this transformation easily if we know how each basis state in the new basis is described by the old basis.

This turns out to be a simple task to do on the computer using bit-wise operations, provided that we know how the gate behaves when it is acted upon a single qubit. Consider a n qubit system and we wish to apply an arbitrary gate G on the kth qubit. Suppose G has the following effect on the states $|0\rangle$ and $|1\rangle$ of a single qubit:

$$G|0\rangle = \alpha_0|0\rangle + \beta_0|1\rangle \tag{38}$$

$$G|1\rangle = \alpha_1|0\rangle + \beta_1|1\rangle \tag{39}$$

We can write this more compactly as

$$G|y\rangle = \alpha(y)|0\rangle + \beta(y)|1\rangle, \tag{40}$$

where we let $\alpha(0) = \alpha_0$, $\alpha(1) = \alpha_1$ and similarly for β. We wish the find the relationship between the new basis $|x\rangle'$ and the old basis $|x\rangle$ such that $|x\rangle' = G|x\rangle$. For this discussion, it would be most convenient to write each basis state in terms of the computational basis, which is $|x\rangle \equiv |x_{n-1}x_{n-2}\cdots x_0\rangle$ where $x = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_0 2^0$. The new basis state $|x\rangle'$ is therefore given by:

$$|x\rangle' = G|x\rangle = G\left(|x_{n-1}\rangle|x_{n-2}\rangle\cdots|x_k\rangle\cdots|x_0\rangle\right) \tag{41}$$

$$= |x_{x-1}\rangle \otimes |x_{n-2}\rangle \otimes \cdots \otimes G|x_k\rangle \otimes \cdots \otimes |x_0\rangle \tag{42}$$

where we used the fact that G only operates on the kth qubit. Using equation 40, we can see that every new basis state can be described by two and only two basis vectors from the old basis:

$$|x\rangle' = |x_{x-1}\rangle \otimes |x_{n-2}\rangle \otimes \cdots \otimes (\alpha(x_k)|0\rangle + \beta(x_k)|1\rangle) \otimes \cdots \otimes |x_0\rangle \tag{43}$$

$$= \alpha(x_k)|x_{n-1}x_{n-2}\cdots 0\cdots x_0\rangle + \beta(x_k)|x_{n-1}x_{n-2}\cdots 1\cdots x_0\rangle \tag{44}$$

Let $G(i,j) \equiv G_{ij}$ and recall that $G_{ij} = \langle i|G|j\rangle$, we therefore know that the elements of the matrix representing G are:

$$G(x_{n-1}x_{n-2}\cdots 0\cdots x_0,\, x_{n-1}x_{n-2}\cdots x_k\cdots x_0) = \alpha(x_k) \tag{45}$$

$$G(x_{n-1}x_{n-2}\cdots 1\cdots x_0,\, x_{n-1}x_{n-2}\cdots x_k\cdots x_0) = \beta(x_k) \tag{46}$$

We notice

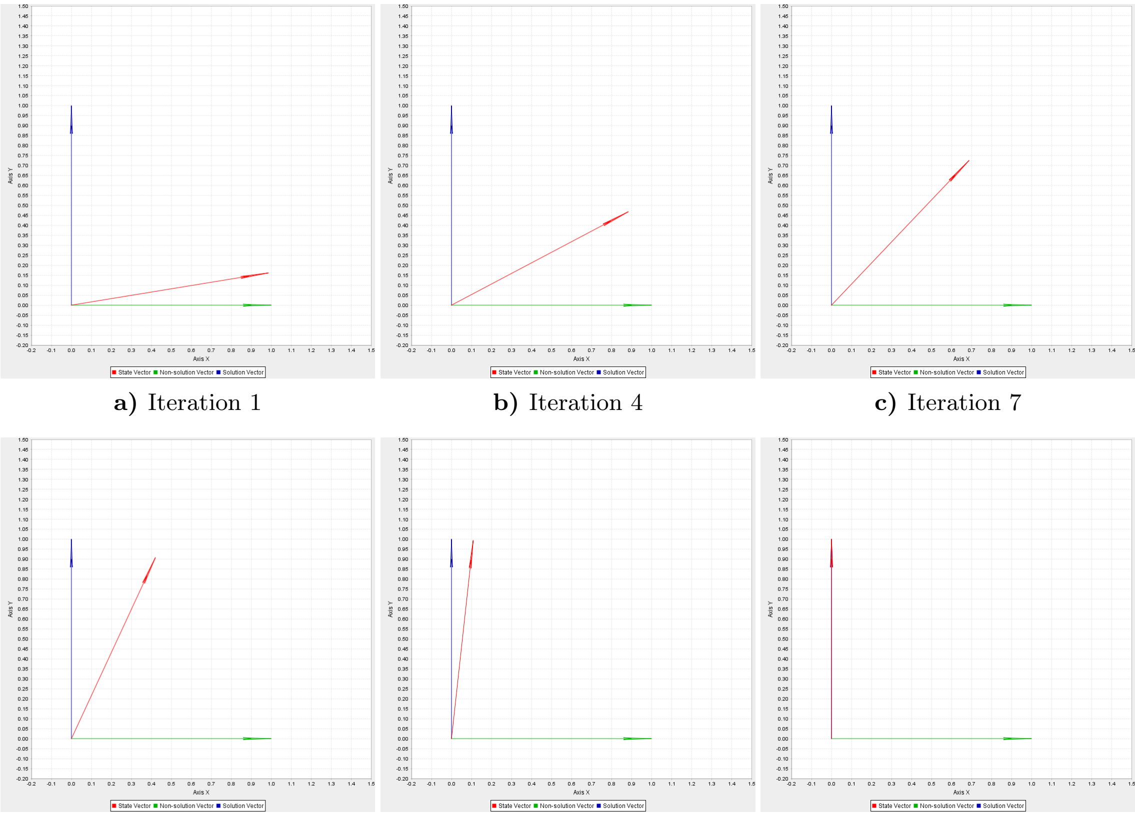
3.3.4 Graphics User Interface

3.4 Program execution

4 Results

4.1 Grover's Algorithm

4.1.1 Rotation of the state vector



a) Iteration 1 **b)** Iteration 4 **c)** Iteration 7

d) Iteration 10 **e)** Iteration 13 **f)** Iteration 14 (end)

Figure 3: Depiction of Grover's Algorithm: The green vector depicts the non-solution vector, the blue vector depicts the vector of the solution. The red vector is the projection of the current state. With every consecutive iteration the state vector rotates towards the solution state.

4.1.2 Computational time

Fig. 4 shows the computational time of Grover's algorithm. All the measurements were executed on the same computer by varying the number of qubits n for the three different representations.

We note the following observations:

- **Dense representation at a disadvantage**
  One can see that the dense representation is the slowest one since for already $n \geq 10$ first noticeable computation times occur whereas the algorithm using functional and spare representation is still immediately executed. However, when they start to show noticeable computational times, the dense representation already takes a unbearable amount of time.

- **Spare vs. functional representation**
  Considering computational time the sparse representation is superior to the functional representation. It appears to be roughly a factor 2, but since we don't have data for bigger registers, this quantitative observation is quite vague.

- **Exponential behaviour**
  All three representations show an exponential behaviour. To illustrate the point we plotted the same data in a log-plot (Fig. 5) and got a linear relation.

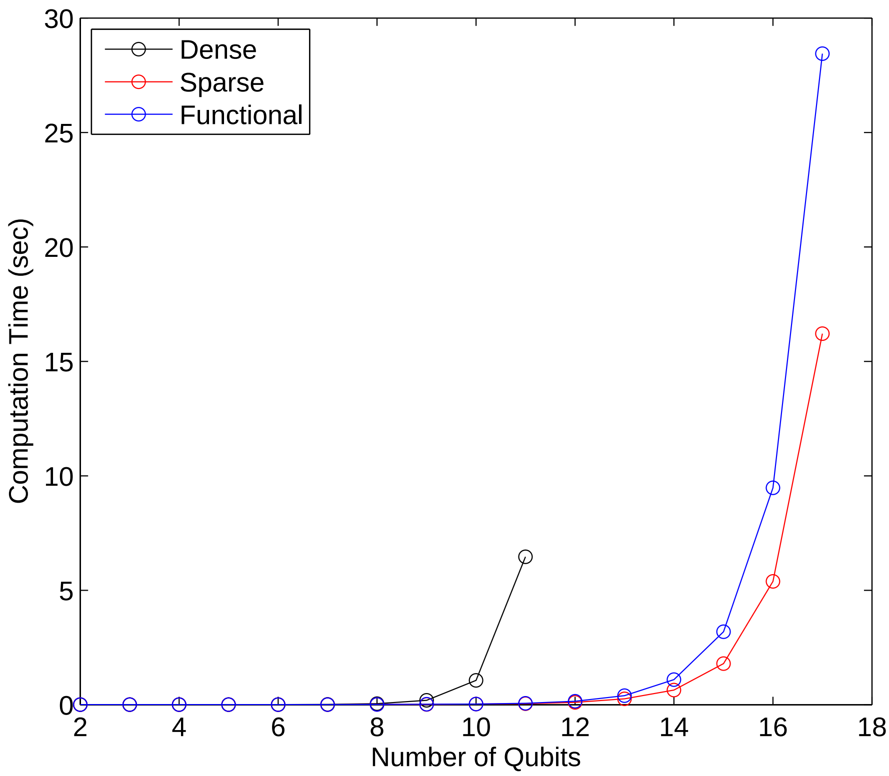


Figure 4: Computational time of Grover's Algorithm in nanoseconds as a function of the quantum register size for the three different forms of representation (dense, sparse, functional).

4.2 Shor's Algorithm

5 Discussion

5.1 Matrix or functional representation

5.2 Improvements and further steps

6 Conclusion

7 Appendix

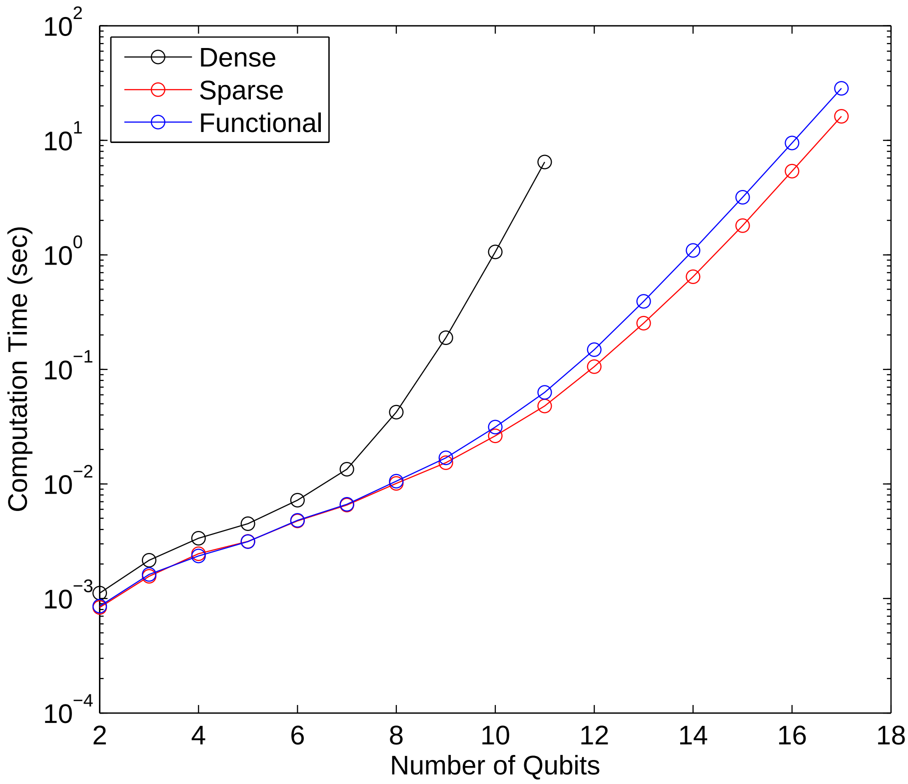


Figure 5: Computational time (log-scale) of Grover's Algorithm in nanoseconds as a function of the quantum register size for the three different forms of representation (dense, sparse, functional). Note that only the behaviour for bigger registers is meaningful (linear).

References

[1] *Grover's algorithm.* http://commons.wikimedia.org/wiki/File%3AGrovers_algorithm.svg, . – [Online; last accessed on 06.03.2015]

[2] *Shor's algorithm.* http://commons.wikimedia.org/wiki/File:Shor%27s_algorithm.svg, . – [Online; last accessed on 06.03.2015]

[3] Ekert, Artur ; Hayden, Patrick ; Inamori, Hitoshi: *Basic concepts in quantum computation*, 2008

[4] Nielsen, Michael A. ; Chuang, Isaac L.: *Quantum Computation and Quantum Information - 10th Anniversary Edition*. Cambridge : Cambridge University Press, 2010. – ISBN 978–1–139–49548–6

[5] Perry, Riley T.: *Quantum Computing from the Ground Up*. Singapur : World Scientific, 2012. – ISBN 978–9–814–41211–7

[6] Vandersypen, Lieven M. K.: *Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance*. – [Online; last accessed on 13.03.2015]

[7] Wikipedia: *RSA cryptosystem.* http://en.wikipedia.org/wiki/RSA_%28cryptosystem%29#Attacks_against_plain_RSA, . – [Online; last accessed on 17.03.2015]