UPPSALA
UNIVERSITET

TVE-F 17 025 juni.
Examensarbete 15 hp
Juni 2017

# Quantum Compmutational Speedup For The Minesweeper Problem

Olof Terner
Villhelm Urpi Hedbjörk

UPPSALA
UNIVERSITET

**Teknisk- naturvetenskaplig fakultet**
**UTH-enheten**

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

# Abstract

## Quantum Computational Speedup For The Minesweeper Problem

*Olof Terner, Villhelm Urp Hedbjörk*

Quantum computing is a young but intriguing field of science. It combines quantum mechanics with information theory and computer science to potentially solve certain formerly computationally expensive tasks more efficiently. Classical computers are based on bits that can take on the value zero or one. The values are distinguished by voltage differences in transistors. Quantum computers are instead based on quantum bits, or qubits, that are represented physically by something that exhibits quantum properties, like for example electrons. Qubits also take on the value zero or one, which could correspond to spin up and spin down of an electron. However, qubits can also be in a superposition state between the quantum states corresponding to the value zero and one. This property is what causes quantum computers to be able to outperform classical computers at certain tasks. One of these tasks is searching through an unstructured database. Whereas a classical computer in the worst case has to search through the whole database in order to find the sought element, i.e. the computation time is proportional to the size of the problem, it can be shown that a quantum computer can find the solution in a time proportional to the square root of the size of the problem. This report aims to illustrate the advantages of quantum computing by explicitly solving the classical Windows game Minesweeper, which can be reduced to a problem resembling the unstructured database search problem. It is shown that solving Minesweeper with a quantum algorithm gives a quadratic speedup compared to solving it with a classical algorithm. The report also covers introductory material to quantum mechanics, quantum gates, the particular quantum algorithm Grover's algorithm and complexity classes, which is necessary to grasp in order to understand how Minesweeper can be solved on a quantum computer.

Handledare: Gregor Kälin
Ämnesgranskare: Natalia Ferraz
Examinator: Martin Sjödin
ISSN: 1401-5757, TVE-F 17 025 juni.

# 1 Populärvetenskaplig sammanfattning

Att lösa problem på datorer handlar om snabbhet, har man en tillräckligt snabb dator kan man lösa så gott som alla problem genom att helt enkelt testa alla lösningar för att se vilken som är rätt, ungefär som att testa alla möjliga kombinationer till pinkod för att komma åt någons kort. Rimligen så börjar man då med att testa 0000 och sedan 0001 och så vidare. Det verkar som ett väldigt långsamt sätt att lösa problemet på eftersom det i värsta fall betyder att om man har otur och 9999 är den rätta koden måste man testa 10000 olika kombinationer, betydligt snabbare verkar det då att testa alla på en gång. Detta kan tyckas omöjligt men är på en kvantdator är det faktiskt möjligt. Kvantdatorer har den mycket märkliga förmågan att de kan testa alla kombinationer på en gång, tyvärr kan man dock bara titta på svaret från en av dessa kombinationer åt gången och för att veta vilken kod som faktiskt låser upp pinkortet måste man upprepa sökningen med en kvantdator flera gånger. Man behöver i fallet med pinkoden dock bara testa ungefär 100 lösningar mot den vanliga datorns 10000, en mycket klar förbättring. Denna metod för att hitta lösningar till problem är tillämbar på de flesta typer av problem och har här använts till för att lösa det klassiska windows spelet röj genom att mycket effektivt testa olika lösningar. Vi har kommit fram till att genom att använda ovan nämnda metod för att få fram lösningen till minesweeper problem uppnås en kvadratiskt hastighetsökning, jämfört med en klassisk lösningsalgoritm.

# Contents

# 2 Introduction

Quantum computing have for some time been a purely theoretical problem, however in recent years the progress in making the actual hardware has been substantial. IBM, for example, reportedly already has a working 17-qubit processor [4]. Quantum computing is interesting because it allows for certain computationally expensive tasks to be solved much more efficiently. Perhaps the most well known and applicable quantum algorithm so far is Shor's algorithm [7]. The algorithm can solve the prime factoring problem, i.e. finding the prime factors of a (large) product, in polynomial time. The best known classical algorithm to this day runs in exponential time, which means it becomes inefficient as the size of the number increases. For large enough numbers, finding the prime factors with any known classical algorithm becomes impossible in reasonable time, even on the best classical supercomputer. In fact, nearly all of our cryptosystems are based on the fact that prime factorization cannot be solved efficiently with any of the currently known algorithms. In this case, the computational advantages of quantum computing could actually become a problem for internet security. However, quantum computing can also offer new ways of creating safe connection and communication. Creating algorithms that work and take advantage of the quantum properties of a quantum computer is one area of quantum computing. Making the actual hardware is another non-trivial aspect of quantum computing. Today, simple quantum computers with only a few quantum bits (or qubits) have been successfully implemented. IBM does even have a public 16-bit quantum computer available [4]. Quantum computing is an emerging field with many possible advantages. This project aims to illustrate its usefulness by applying quantum computing to the well known Microsoft game "Minesweeper". We do this by reducing the minesweeper problem to what is called a SAT-problem [5], which is a quite often encountered kind of problem and thus the results can be extrapolated to any similar problem.

The paper starts with a large theory chapter that contains a short section explaining the general SAT problem, an overview of complexity classes and an introduction to quantum computing that includes quantum gates, Grover's algorithm and quantum counting. Following the theory chapter are the results, where the procedure of reducing the minesweeper problem to an SAT is explained as well as the algorithm used to solve a general *Minesweeper* game. The report ends with an explicit example of the algorithm implemented on a small game of *Minesweeper* and a conclusion summarizing the effectiveness of the algorithm.

## 2.1 The Minesweeper Problem

The classical Microsoft game "Minesweeper" consists of a rectangular field filled with squares. In these squares are a set of hidden mines. The objective is to localize all the hidden mines by pressing the squares that are safe, i.e. the squares that do not contain any mines. The player gets hints from all squares that are adjacent to any of the mines. These squares each contain a number, indicating how many mines that they are adjacent to. This is the only lead that the player gets

to solve the problem. The minesweeper problem is defined as identifying, if possible, all the mines without having to make a guess. Formally that is: "Given a rectangular grid partially marked with numbers and/or mines, some squares being left blank, to determine if there is some pattern of mines in the blank squares that give rise to the numbers seen." [5] This problem is in fact proven to be in the NPC complexity class [5] (see section 3.2), meaning that it cannot be solved by any known algorithm on a classical computer in polynomial time. To illustrate that this problem indeed is not trivial, examine the following example. In figure 1 a game of minesweeper has been started by probing a square in the middle. It is not immediately apparent what the solution is at this point, i.e. where the mines are located. Assuming, for example, that the squares marked with a flag are mines and the squares marked with a check mark are safe in a manner shown in figure 2, we eventually end up in a state shown in figure 3. However, this cannot be the solution since it would leave the '2' to the right, second to the top with only one adjacent mine. The correct solution is shown in figure 4, again not immediately apparent upon first inspection of the set up.

| | | | | | |
|---|---|---|---|---|---|
| X | M | X | X | X | X |
| X | 2 | 2 | 2 | 2 | X |
| X | 2 | | | 2 | X |
| X | 2 | | | 2 | X |
| X | 2 | 2 | 2 | 2 | X |
| X | X | X | X | X | X |

Figure 1: A *Minesweeper* setup where M denotes possible mines, X unprobed mines, $\surd$ and numbers representing constraints.

| | | | | | |
|---|---|---|---|---|---|
| $\surd$ | M | $\surd$ | X | X | X |
| X | 2 | 2 | 2 | 2 | X |
| X | 2 | | | 2 | X |
| X | 2 | | | 2 | X |
| X | 2 | 2 | 2 | 2 | X |
| X | X | X | X | X | X |

Figure 2: A possible *Minesweeper* configuration.

| | | | | | |
|---|---|---|---|---|---|
| √ | M | √ | M | M | √ |
| X | 2 | 2 | 2 | 2 | X |
| X | 2 | | | 2 | X |
| X | 2 | | | 2 | X |
| X | 2 | 2 | 2 | 2 | X |
| X | X | X | X | X | X |

Figure 3: A setup showing that this placement of mines is wrong.

| | | | | | |
|---|---|---|---|---|---|
| √ | √ | M | M | √ | √ |
| √ | 2 | 2 | 2 | 2 | √ |
| M | 2 | | | 2 | M |
| M | 2 | | | 2 | M |
| √ | 2 | 2 | 2 | 2 | √ |
| √ | √ | M | M | √ | √ |

Figure 4: The correct placement of mines.

As evident, solving the minesweeper problem is not a trivial task, especially for large setups of the game. For a game of $m$ mines over $n$ squares the number of possible mine placements are given by

$$\frac{n!}{m!(n-m)!}. \tag{1}$$

For a standard game there are 99 mines over 480 squares giving a total of $5.6 \times 10^{104}$ different setups of mines, showing that this problem quickly grows out of hand.

# 3 Theory

To understand how to solve the minesweeper problem with quantum computing and why it could be advantageous, basic knowledge about quantum mechanics, complexity classes, quantum gates and particular quantum algorithm "Grover's algorithm" is needed . This section aims to make the reader acquainted with these subjects in order to understand how the minesweeper problem can be solved with quantum computing.

## 3.1 The SAT Problem

Before the minesweeper problem is solved, we will reduce it to what is called a SAT problem. SAT stands for *boolean satisfiability* These kinds of problems are of the type "can a particular boolean function be made to show TRUE?" A boolean function is a function that maps $B : \{0,1\}^n \to \{0,1\}$. For example the function

$$B(x) = x_1 \wedge x_2 \tag{2}$$

is a boolean function and is satisfied for the input vector $[x_1, x_2, ] = [TRUE, TRUE]$.

### 3.2 Complexity Classes

In computing, efficiency is an important aspect and since the main benefit of a quantum computer is that it can be faster than a classical computer, the following section is dedicated to efficiency and complexity classes.

There are several defined complexity classes which correspond to different rates of growth in solving time, depending on the size of the input. They do not give the actual time it would take to solve a given problem since this depends on the computer used to solve it and the exact implementation. They do however give bounds on how efficiently any problem in these classes can be solved. To describe this the term $\mathcal{O}$ is used. $\mathcal{O}$ means that the upper bound of the time it can take to solve a problem given some input, that is

$$f(x) = \mathcal{O}(g(x)) \quad x \to \infty \tag{3}$$

for some arbitrary functions $f(x)$ and $g(x)$. The efficiency of an algorithm can be realized by determining if the algorithm runs in polynomial time or exponential time. Polynomial time means that the upper bound for the time it takes is a polynomial expression proportional to time size of the algorithm input

$$\mathcal{O}(n^p), \tag{4}$$

where $n$ is the size of the input and $p$ a non-negative integer. Exponential time does in the same way mean that the upper bound for the time is a exponential expression with regard to the input size i.e

$$\mathcal{O}(q^n), \tag{5}$$

where $q$ is a non-negative real number. For practical purposes the difference between polynomial and exponential time is for larger problems the difference between doable and not doable. The problems that are used to define our complexity classes are called decision problems, problems with answers YES or NO, such as "does A divide B". Complexity classes are defined by what kind of machine it takes to define them, for us this will be slightly different kinds of the Turing machine. The Turing machine [6] is quite simple and can be made equivalent to any computational circuit, it consists of two parts, a head that tells the machine where in the computational algorithm we are, and a memory tape that is made up from discrete cells, in each a symbol from a finite alphabet, corresponding to some operation. When the machine starts the header reads the first symbol, depending on this symbol it can:

1. Write a new symbol from the alphabet in the active cell
2. Move one step to the left or right
3. Stop the header, thus quiting the process

There are several complexity classes defined from this, but two of particular interest with regards to the minesweeper problem are P and NP (for "polynomial time" and "non-deterministic polynomial time"). The problems in P are defined as the problems that can be solved by a deterministic Turing machine in polynomial time, and problems in NP as the problems that can be determined by a non-deterministic Turing machine in polynomial time. The difference between non-deterministic (NTM) and deterministic (DTM) Turing machines is that where as the DTM have that the state A always will go to state B, the NTM can for some probability P go from state A to either state B or C (or any other state). This mean that where a DTM can be said to span a line a NTM is said to span to tree, making it more complex. P and NP are related as $P \subset NP$. Problems in P can be solved in polynomial time. Most of the NP problems take exponential time to solve. Only for a limited number of problems that are in NP but not in P are there known algorithms that solve in polynomial time. The reason for this is that we usually try to solve them on a classical computer, which is a deterministic Turing machine. A quantum computer is however a non-deterministic Turing machine and could therefore possibly solve a NP-problem in polynomial time with a certain probability of success.

As mentioned above some NP-problems cannot be solved efficiently on a classical deterministic Turing machine, but any possible answers to them can be checked to be correct in polynomial time. An interesting important thing to note about the complexity class NP is that nobody has proven that NP does not equal P, although this is generally believed to be the case. In fact, a prize sum of one million dollars has been issued by Clay Mathematical Institute, Cambridge, Massachusetts to anyone who can prove that P equals, or does not equal NP [3]. If someone were to prove that $P = NP$, this would mean that all problems in NP could be solved efficiently on a classical Turing machine. There is a subset of the NP-problems called NP-complete problems (NPC). NP-complete problems are at least as hard as the hardest problem in NP, but is still said to belong to NP and any algorithm that solves a NPC problem in polynomial time can be used to solve any problem in NP in polynomial time . Thus, if someone finds a way to efficiently solve any of the NP-complete problems, they have solved all NP-problems efficiently and thus proven that $P = NP$.

Quantum computing offers a way to efficiently solve NP-problems. In fact, Shor's algorithm solves the NP-problem of factoring a number into prime numbers in polynomial time. This is partly what makes quantum computers so interesting. Since solving NP-problems on quantum computers involves probability, other complexity classes are used to describe problems solved using quantum computing. The quantum analogue for the complexity class P is BQP (Bounded-error quantum polynomial time) [1], and the quantum analogue for NP is called QMA (Quantum Merlin Arthur) [1]. QMA is defined in the same way as NP but since a quantum computer is a probabilistic computer, the probability for error must be bounded. That is, QMA is a class consisting of decision problems where for any answer there is a quantum state input that gives this answer with a bounded error probability.

## 3.3 Introduction To Quantum Computing

A quantum computer utilizes quantum states for information processing. When measured, these states collapse to a classical state corresponding to either zero or one, like the possible values of a classical bit. In quantum computing, quantum bits (or qubits) are used instead of regular bits. The value of a regular bit is determined by a voltage difference in a transistor. The value of a quantum bit is determined by the state which the qubit is in when it is measured. For example, an electron could be a qubit, where spin up corresponds to the value one, and spin down to the value zero. The advantage of qubits and quantum computers is that each qubit can be in a superposition between a zero and a one, with an associated complex coefficient that relates to the probability of the qubit being in a particular state. When measured, a quantum computer will always yield a classical state, where any information about the superposition state is lost. The qubits can be manipulated before a measurement and this is done in order to reap the benefits of this "hidden" information about the superposition.

Since quantum computing is based on quantum mechanics, the Dirac notation used in quantum mechanics is adopted. The states 0 and 1 are thus represented as ket vectors $|0\rangle$ and $|1\rangle$ (using Dirac vector notation) in a two-dimensional Hilbert space [1]. The state of a qubit can be described by the superposition of these states with complex numbers $a$ and $b$,

$$|\psi\rangle = a\,|0\rangle + b\,|1\rangle\,. \tag{6}$$

The vectors are normalized so that $|a|^2 + |b|^2 = 1$, where $|a|^2$ is the probability of measuring the state $|0\rangle$ and $|b|^2$ is the probability of measuring the state $|1\rangle$. A qubit initially in the state $|0\rangle$ can through a Hadamard transform (as explained further down in the section) be put into a state

$$|\psi\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}, \tag{7}$$

where the qubit has an equal possibility to be in either of the states corresponding to the values zero and one. Grover's algorithm, which will be the focus of this article since it is used to solve the minesweeper problem, starts with all qubits in this totally mixed state.

Since the $|\psi\rangle$ is normalized we can more formally say that the qubit state is a unit vector in a 2-D complex vector space. Using this definition the qubit state can be defined with angles

$$|\psi\rangle = cos\frac{\theta}{2}\,|0\rangle + e^{i\rho}sin\frac{\theta}{2}\,|1\rangle\,. \tag{8}$$

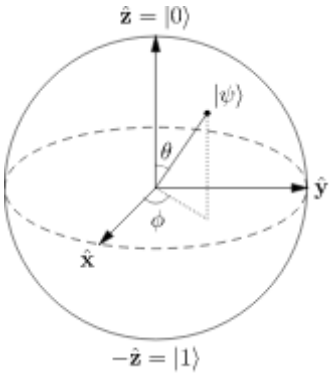


Figure 5: The bloch sphere describes the state of a single qubit. Each point on the surface of the sphere corresponds to a distinct state.

This way of describing the a qubit is called a Bloch sphere. Each point on the Bloch sphere corresponds to a state, and thus the Bloch sphere is useful for describing a single qubit state as it offers a graphical illustration of the phase of the qubit. Any transformations of the state of the qubit is illustrated by a rotation of the state vector on the sphere.

A fundamental operation in quantum mechanics is the tensor product. The tensor product forms larger vector spaces from smaller ones. For two vector spaces $V$ and $W$ of dimensions $m$ and $n$, the tensor product of $V$ and $W$, denoted $V \otimes W$ or often $|V\rangle |W\rangle$, is a $m \cdot n$ dimensional vector space. The elements of $V \otimes W$ is a linear combination of the tensor product $|v\rangle \otimes |w\rangle$ where $|v\rangle$ and $|w\rangle$ are elements of $V$ and $W$ respectively. The state $|x_1\rangle \oplus |x_2\rangle$ is abbreviated $|x_1 x_2\rangle$.

For a 2-qubit state there are four basis states $|00\rangle, |01\rangle, |10\rangle$ and $|11\rangle$. This gives the state vector

$$|\psi\rangle = a\,|00\rangle + b\,|01\rangle + c\,|10\rangle + d\,|11\rangle\,, \tag{9}$$

where $a$, $b$, $c$ and $d$ are normalized. An additional qubit will yield another four basis states. In general, $n$ qubits can take on $2^n$ basis states, i.e. all possible permutations of $n$ binary numbers. However, when a quantum mechanical system is measured the superposition properties of the system is lost, and only classical information can be directly derived. The advantage lies in manipulating the superpositioned states prior to measuring them to yield a desired outcome with a high probability.

Quantum parallelism is one of the properties of quantum computing that contributes to the speed up of quantum computers compared to classical computers. To illustrate this, a simple example of quantum parallelism is presented. Suppose that a register of two qubits is set up with the first qubit in the state $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and the second qubit in the state $|0\rangle$. The goal is to evaluate the function $f(x) : [0,1] \to [0,1]$. With an appropriate sequence of quantum gates it is possible to construct the operator $U_R$ so that $U_R$ applied on the state $|x, y\rangle$ yields $|x, y \oplus f(x)\rangle$, i.e.

$U_R : |x, y\rangle \rightarrow |x, y \oplus f(x)\rangle$. This final state then becomes

$$\frac{|0, f(0)\rangle + |1, f(1)\rangle}{\sqrt{2}}. \tag{10}$$

This state contains both f(1) and f(0), and thus both f(1) and f(0) have in some meaning been evaluated using only one call to the function f(x), something that is not possible on a classical computer. In general, a register of n qubits could be set up and thus evaluate a function $f(x) : [0, 1] \rightarrow [0, 1]$, with $x = 0, 1, 2, ..., 2^n - 1$. However, quantum parallelism alone is not enough to make use of the advantages of quantum computing. In these examples, all function values have been evaluated simultaneously, but when measured only one value can be acquired.

Going back to the example with only two qubits, preparing the $x$ state to $x = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$ but instead of initializing $y = |0\rangle$, $y$ is set to the superposition state $y = \frac{|0\rangle-|1\rangle}{\sqrt{2}}$ This can be done by applying the Hadamard transform to a qubit in the state $|1\rangle$. Applying the operator $U_R : |x, y\rangle \rightarrow |x, y + f(x)\rangle$ and finally a Hadamard transform on the first qubits results in the state [1, eq 1.43]

$$|\psi\rangle = \begin{cases} \pm |0\rangle \frac{|0\rangle-|1\rangle}{\sqrt{2}} \text{ if f(0)=f(1)} \\ \pm |1\rangle \frac{|0\rangle-|1\rangle}{\sqrt{2}} \text{ if f(0)} \neq \text{f(1).} \end{cases} \tag{11}$$

This can be then be rewritten as [1, eq 1.45]

$$|\psi\rangle = \pm(f(0) \oplus f(1)) \frac{|0\rangle - |1\rangle}{\sqrt{2}}, \tag{12}$$

where $\oplus$ denotes addition modulo 2. From this follows that even though $f(x)$ have only been evaluated once, a quantum computer can determine whether $f(0) = f(1)$, or $f(0) \neq$ f(1). This is impossible on a classical computer, which would require two calls to the function $f(x)$. This particular problem illustrates the possibilities of quantum computing.

## 3.4 Quantum Gates

Just like classical computers, quantum computers are built from gates. Since quantum computing uses qubits that span a finite dimensional vector space, each state can be represented by a column vector, where the first element is the complex coefficient corresponding to the state

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{13}$$

and the second element is the complex coefficient to the state

$$|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{14}$$

Each operator can be represented as a matrix. For the quantum NOT-gate the matrix is

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \tag{15}$$

Applying this operator on a general single qubit state

$$\begin{bmatrix} a \\ b \end{bmatrix} \tag{16}$$

where a and b represent the amplitudes for $|0\rangle$ and $|1\rangle$ respectively, gives

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} b \\ a \end{bmatrix}. \tag{17}$$

This operator takes the complex coefficient corresponding to the zero-state and switches it with the coefficient corresponding to the one-state, hence it is a NOT-gate. All classical as well as quantum gates have a matrix representation and they always have to be normalized, this condition means that the matrix operator acting on a state must not interfere with the normalization. From this condition it follows that the matrices have to be unitary. That is, $UU^\dagger = I$ where $U$ is the matrix operator, $I$ the identity matrix and $\dagger$ is the adjoint operator which transposes and takes the complex conjugate of the matrix. From this follows that all quantum gates have to be reversible. This reversibility means that for every output needs to corresponds to a unique input. For example, for a classical AND gate the output TRUE means that all inputs are TRUE, however for FALSE output the input can be [FALSE FALSE], [FALSE TRUE] or [TRUE FALSE], thus there is no definite conclusion to be drawn from the output TRUE regarding the input. The AND analogue in quantum computing is the Toffoli gate, which has the following matrix representation

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}. \tag{18}$$

The toffoli gate utilizes an extra *target* bit and works by flipping the state of the target bit if the *control* bits are TRUE, but leaving the two control bits unchanged.

Another useful quantum gate is the CNOT-gate (21). The CNOT-gate has one control bit (a) and one target bit (b). It performs the operation $|a, b\rangle \rightarrow |a, a \oplus b\rangle$, where $\oplus$ is addition modulo 2. If the control bit is TRUE the state of the target bit is flipped, otherwise the target bit is unchanged. There are also several single qubit gates. Examples are the gates that can be represented by the Pauli matrices [1], the phase gates that changes the phase of the qubit by different degrees and the Hadamard gate. The Hadamard gate is particularly interesting as it takes the basis states $|0\rangle$ and $|1\rangle$ and transforms them to the completly mixed states $\frac{|0+|1\rangle\rangle}{\sqrt{2}}$ and $\frac{|0-|1\rangle\rangle}{\sqrt{2}}$

respectively, meaning that it takes the basis states to an equally weighted superposition of these. The Hadamard gate has the matrix operator

$$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}. \tag{19}$$

Applying this to the $|0\rangle$ grants

$$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}1\\ 0\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\ 1\end{bmatrix} = \frac{|0\rangle + |1\rangle}{\sqrt{2}}. \tag{20}$$

From these gates a universal gate language is created, meaning that any quantum circuit can be realized with these gates. All of these gates have corresponding matrices and any circuit created from these gates can be represented by a matrix $U_R$ made from these matrices. Going from the specific gate matrices to $U_R$ takes a series of linear operations. For two gates in cascade the operation is quite simple. This circuit will always have the same input and output since the not gates negate each other. A CNOT-gate is represented by the matrix

$$\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{bmatrix}. \tag{21}$$

Cascading two of these we get

$$\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{bmatrix}\cdot\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{bmatrix} = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}. \tag{22}$$

It is clear that since two CNOT-gates applied on any input state is equivalent to applying the identity matrix on the input, the output will always be the same as the input. Gates that are applied to qubits in parallel, are combined using the Kronecker product. Given the Pauli matrices

$$X = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix} \tag{23}$$

and

$$Z = \begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix} \tag{24}$$

the Kronecker product is

$$X \otimes Z = \begin{bmatrix}0\cdot Z & 1\cdot Z\\ 1\cdot Z & 0\cdot Z\end{bmatrix} = \begin{bmatrix}0 & 0 & 1 & 0\\ 0 & 0 & 0 & -1\\ 1 & 0 & 0 & 0\\ 0 & -1 & 0 & 0\end{bmatrix}. \tag{25}$$

The above operation is analogous to an X and a Z gate in parallel. From using both matrix multiplication and the Kronecker product we can represent any circuit as one single unitary matrix that performs all the operations of the circuit.

## 3.5 Quantum Computing vs Classical Computing

So we do now wonder, what are the the advantages of a quantum computer over a classical one? The truth is that in most cases there are none. For almost all cases a classical computer can do the same tasks just as fast. For a quantum computer to be faster we need to use quantum parallelism and in most cases this does not give a increase in speed since there are no processes to be done in parallel, or so few that setting up the quantum mechanical system costs more computing power than just solving it regularly. However for the kind of problems that we can apply quantum algorithms to, such as our problem, the speed up is so big as to still make it an interesting topic. It should be noted that Grover's search algorithm can be used for a quadratic speedup of any NP problem. A quantum computer would of course excel at the simulation of quantum systems compared to a classical computer, since they are very hard to simulate on a classical one.

One problem with quantum algorithms is that even if there might be lot of problems that actually can get a significant speedup on a quantum computer, there is the intrinsic problem of coming up with the algorithm since it can be difficult to think in terms of quantum mechanical systems.

## 3.6 Grover's Algorithm

Grover's algorithm is a quantum search algorithm that performs a search over an unstructured set of $N = 2^n$ elements. It takes advantage of the superposition of states, quantum parallelism (see section 3.3) and amplitude amplification to sort out the correct solution to a search problem with a high probability after $\frac{\pi}{4}\sqrt{N}$ iterations. As evident, it does this in $\mathcal{O}(\sqrt{N})$ time, compared to the best classical algorithm which does this in $\mathcal{O}(N)$[1]. From this follows that Grover's algorithm does not provide a BQP algorithm (see section 3.2) that solves NPC problems, but the quadratic speedup is still significant, especially when $N$ is large. This section will introduce the general idea of the algorithm, then derive each step in depth including a worked example, continue by looking at the gate implementation of the algorithm and finally explore how Grover's algorithm can be used to solve the minesweeper problem.

### 3.6.1 The Algorithm

The steps of the algorithm are first listed and then explained in depth.

1. Prepare a set of n qubits in the states $|0\rangle^{\otimes n}$.

2. Perform the Hadamard transformation to all qubits:
   $H^{\otimes n}|0\rangle^{\otimes n} = \sum_{i=0}^{2^n-1} \frac{1}{\sqrt{2^n}}|x_i\rangle = |\psi\rangle$, where $x_i$ is a binary basis state with a value corresponding to $i = 0, 1, 2, ..., n-1$, e.g. for a two qubit system $x_0 = |00\rangle$, $x_1 = |01\rangle$ etc.

3. Apply the Grover iteration: $(2|\psi\rangle\langle\psi| - I)\mathcal{O}|\psi\rangle$, $\frac{\pi}{4}\sqrt{N}$ times.

[1] If there are several correct solutions, Grover's algorithm runs in $\mathcal{O}(N/k)$, where k is the number of solutions.

4. Measure the state.

The first step is simply to put all $n$ qubits in the state $|0\rangle$. Then after applying the Hadamard gate to each qubit, the total state of the system is an equally weighted superposition of all the basis states

$$\sum_{i=0}^{2^n-1} \frac{1}{\sqrt{2^n}} |x\rangle = |\psi_0\rangle \,. \tag{26}$$

Then the Grover iteration begins. First, the oracle $\mathcal{O}$ is applied. The oracle can be viewed as a black box that can recognize a solution, and causes a phase shift $\pi$ to the state corresponding to the solution. This is equivalent to multiplication of the state by $-1$. Mathematically, that is

$$\mathcal{O} |\psi\rangle = \sum_{i=0}^{2^n-1} (-1)^{B(x_i)} \langle\psi|x_i\rangle |x_i\rangle \tag{27}$$

where $B : \{0,1\}^n \to \{0,1\}$ equals 1 for the correct input configuration (see section 4.1). The exact workings of the oracle does not need to be known in order to understand Grover's algorithm, and depends on the specific problem being solved. Then the operator $(2 |\psi\rangle \langle\psi| - I)$ is applied, which causes an inversion about the mean. That the operator indeed causes an inversion about the mean can be seen by applying the operator to a general state

$$|\psi\rangle = \sum_{i=0}^{2^n-1} \alpha_x |x\rangle \,. \tag{28}$$

This will result in a state

$$(2 |\psi\rangle \langle\psi| - I) |\psi\rangle = \sum_x [-\alpha_x + 2\langle\alpha\rangle] |x\rangle \,, \tag{29}$$

where

$$\langle\alpha\rangle = \frac{1}{N} \sum_x \alpha_x \tag{30}$$

and

$$-\alpha_x + 2\langle\alpha\rangle = \langle\alpha\rangle + (\langle\alpha\rangle - \alpha_x). \tag{31}$$

The coefficient of each state will be increased or decreased by the amount that it differs from the mean, thus inverting about the mean. If the iteration is performed $\frac{\pi}{4}\sqrt{N}$ times, it will result in an amplification of the amplitude corresponding to the solution. The state is then measured and the correct configuration of mines can be derived.

### 3.6.2 Example

To obtain a better understanding of how Grover's algorithm works, it is helpful to work through an example. Consider a boolean satisfiability problem with three variables, $x_i$, $i = 1, 2, 3$. There are a total of $2^n = 2^3 = 8$ permutations. The problem is satisfied for $x_1 \wedge x_2 \wedge x_3$.

A system of $n(= 3)$ qubits are initialized to $|0\rangle$. Then the Hadamard transform is applied to the state, which puts it in an equal superposition of all possible basis states

$$|\psi_0\rangle = \sum_{i=0}^{7} \frac{1}{\sqrt{2^3}} |x_i\rangle \,. \tag{32}$$

Since all coefficients are real, they can be illustrated with a graph seen in figure 6.

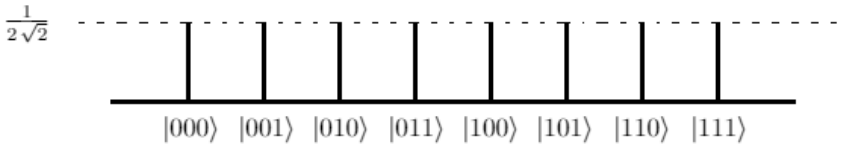


Figure 6: A visual interpretation of the initial, fully mixed state of a 3-qubit system.

The Grover iteration is then performed for a total of $\frac{\pi}{4}\sqrt{N} = \frac{\pi}{4}\sqrt{8} \approx 2$ iterations. Firstly, the oracle recognizes the correct solution (which in this case is $|111\rangle$) and multiples the corresponding coefficient by -1

$$|\psi_1\rangle = \mathcal{O}\,|\psi_0\rangle = \sum_{0}^{6} \frac{1}{2\sqrt{2}} |x_i\rangle - \frac{1}{2\sqrt{2}} |x_7\rangle \,. \tag{33}$$

The superposition of states after one oracle call can be seen in figure 7.

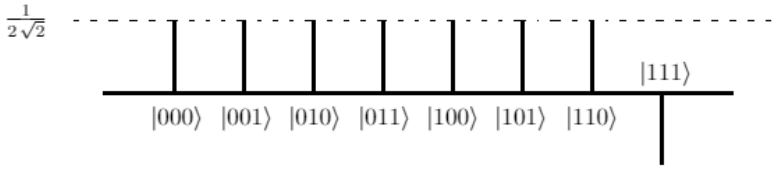


Figure 7: The state after the first oracle operation.

This is equal to $|\psi_0\rangle - \frac{2}{2\sqrt{2}} |111\rangle$ . Now, the operator that causes an inversion about the mean is applied to the state

$$|\psi_2\rangle = (2\,|\psi_0\rangle\,\langle\psi_0| - I)[|\psi_0\rangle - \frac{2}{2\sqrt{2}} |111\rangle] = \frac{1}{2}\,|\psi_0\rangle + \frac{1}{\sqrt{2}}\,|111\rangle \,. \tag{34}$$

Using equation (32) and substituting $|\psi_0\rangle$ yields

$$|\psi_2\rangle = \frac{1}{2}\sum_{i=0}^{7} \frac{1}{2\sqrt{2}} |x\rangle + \frac{1}{\sqrt{2}} |111\rangle = \frac{1}{2\sqrt{2}} |000\rangle + \frac{1}{2\sqrt{2}} |001\rangle + \frac{1}{2\sqrt{2}} |010\rangle + ... + \frac{1}{2\sqrt{2}} |110\rangle + \frac{5}{4\sqrt{2}} |111\rangle \,. \tag{35}$$

This leads to a new superposition state with real coefficients illustrated in figure 8.

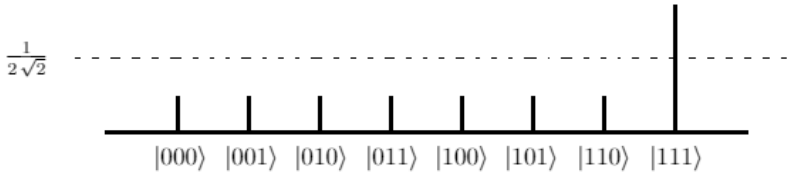


Figure 8: The resulting state after the first Grover iteration.

If a measurement was to be done after a single iteration the probability of measuring the correct state $|111\rangle = |x_7\rangle$ would be $|\alpha_7|^2 = |\frac{5}{4\sqrt{2}}|^2 = \frac{25}{32} \approx 78\%$. The oracle is then applied to the resulting state from iteration one. This results in the state

$$|\psi_3\rangle = \mathcal{O}\,|\psi_2\rangle = \frac{1}{4\sqrt{2}} \sum_{i=0}^{7} |x_i\rangle - \frac{3}{2\sqrt{2}}\,|111\rangle \tag{36}$$

illustrated in figure 9.

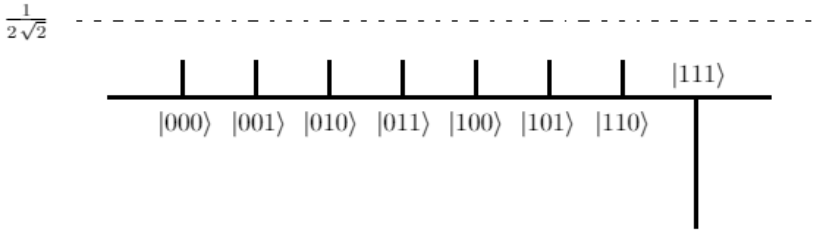


Figure 9: The resulting state after the oracle has been applied for the second time.

The inversion about the mean operator is applied a second time

$$|\psi_4\rangle = (2\,|\psi\rangle\,\langle\psi| - I)\,|\psi_3\rangle = -\frac{1}{8\sqrt{2}} \sum_{i=0}^{6} |x\rangle + \frac{11}{8\sqrt{2}}\,|111\rangle\,. \tag{37}$$

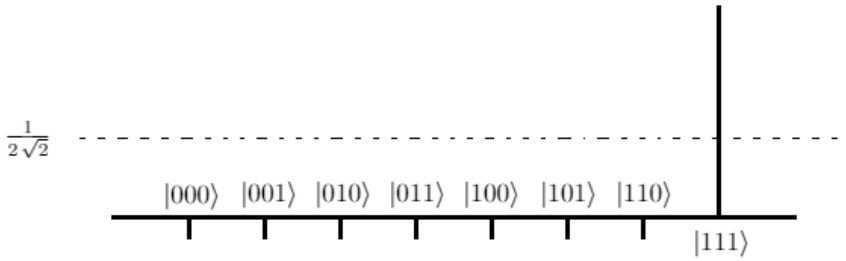


Figure 10: The final state after two Grover iterations.

Simply looking at the geometric representation of the coefficients shows the success of the algorithm. The probability of measuring the correct state after the second iteration is: $|\frac{11}{8\sqrt{2}}|^2 \approx 94.5\%$. The gate representation of the circuit solving the above example is given in figure 11.

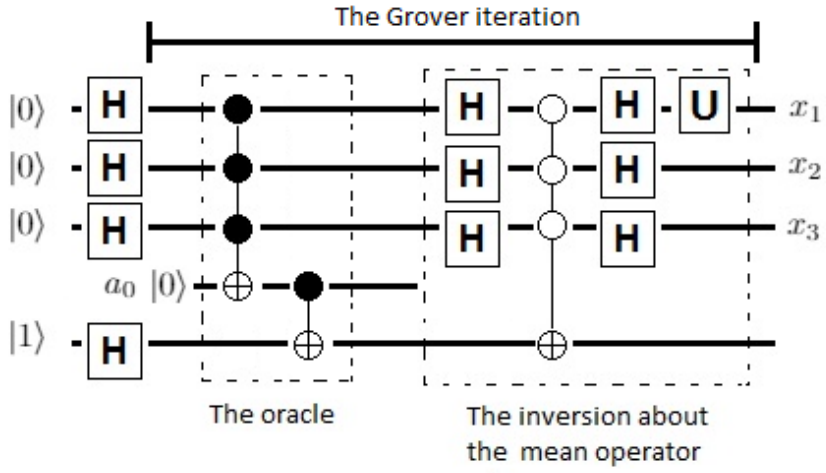


Figure 11: A visualisation of a quantum circuit that performs the Grover iteration.

The three qubits that will eventually be measured are prepared in the state $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$, by applying the Hadamard gate to each qubit initially in the state $|0\rangle$. A fourth bit is prepared in the state $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$ by initiating it in the $|1\rangle$ state and applying the Hadamard transform. Thus if this bit is the target bit of a C-NOT gate, it will flip the state if the control bit is $|1\rangle$. To make sure only the state corresponding to the correct solution is flipped, the oracle makes use of an auxiliary qubit $a_0$, prepared in the state $|0\rangle$. The auxiliary qubit is the target bit of a 4-input Toffoli-gate (illustrated by the three black dots connected to the auxiliary bit $a_0$), which means that if all the three control qubits are in the state $|0\rangle$[2], the auxiliary bit will be flipped to a $|1\rangle$. Since the auxiliary bit is the control bit to the C-NOT gate with the last qubit as target bit, the oracle operates in the desired manner, i.e. flipping the state representing the correct solution ($|111\rangle$).

[2]Note that the control bits never actually are in any of the states $|0\rangle$ or $|1\rangle$, but rather in a superposition of the states.

The next step is the inversion about the mean operator. It works by first applying a Hadamard transform to the workspace, then flipping the state if it is not equal to $|0\rangle^{\oplus n}$, and then applying the Hadamard transform again. This is equivalent to the inversion about the mean operator presented in section 3.6.1 since

$$H^{\oplus n}(2\,|0\rangle\,\langle 0| - I)H^{\oplus n} = 2\,|\psi\rangle\,\langle\psi| - I. \tag{38}$$

It is easily checked that $(2\,|0\rangle\,\langle 0| - I)$ indeed maps $|0\rangle \to |0\rangle$ and $|x\rangle \to -|x\rangle$, where $x \neq 0$. The quantum circuit in figure 11 accomplishes this whole operation by first applying the Hadamard gate to each qubit, then inverting the state if it is equal to $|0\rangle^{\oplus n}$, applying the Hadamard gate to each qubit again, but also applying the single input qubit inverting matrix U to any of the qubits, where

$$U = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}. \tag{39}$$

This results in the desired outcome, i.e. inverting the state if and only if it was not equal to $|0\rangle^{\oplus n}$ after the first Hadamard transform. The operator that inverts the state if it is $|0\rangle^{\oplus n}$ is unitary and thus an allowed operation. [1] The matrix representation of the operator is

$$A = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{40}$$

## 3.7 Quantum Counting

Since we do not want to make guesses if it is not necessary, we want our solutions to be unique. Why this is can be illustrated by a one constraint adjacent to three variables with no other constraints as seen in figure 12.

| | |
|---|---|
| 1 | X |
| X | X |

Figure 12: A example of a setup with non-unique solutions

Either of these could be a solution and since we do not want to guess we have to know if any such case arises. To find this out we can not use Grover's algorithm, but will use something called *quantum counting* [1]. Suppose we have two eigenvectors $|v\rangle$ and $|w\rangle$, we get these from looking at the Grover iteration as a rotation around the Bloch sphere that these have eigenvalues $e^{i\theta}$ and

$e^{i(2\pi-\theta)}$, to be sure that the algorithm converges we set the search space to $2N$ and we get that

$$\sin^2(\theta/2) \cdot 2N = M \tag{41}$$

where $M$ is the number of solutions. To find out $\theta$ something called *phase estimation* [1] is used. To do this we start by finding the eigenstate $|u\rangle$ with eigenvalue $e^{2\pi i\theta}$ corresponding to the oracle matrix that is used in Grover's algorithm. Suppose we want an estimation with accuracy of $2^{-n}$ with probability $1-\epsilon$, then preparing a qubit register $|t\rangle$ of size $n + log(2 + \frac{1}{2\epsilon})$ in the state $|0\rangle$.

1. We have the initial state $|0\rangle |u\rangle$
2. The superposition
$$\frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle |u\rangle \tag{42}$$
is created.
3. The oracle is then applied
$$\frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} |j\rangle U^j |u\rangle = \frac{1}{\sqrt{2^t}} \sum_{j=0}^{2^t-1} e^{2\pi i j\theta_u} |j\rangle |u\rangle \tag{43}$$
The inverse fourier transform is then applied to this giving:
$$|\theta'_u\rangle |u\rangle \tag{44}$$
4. The register that started as $|0\rangle$ is then measured granting us the state:
$$\theta'_u \tag{45}$$

From this $\theta'_u$ we can then calculate M as $\sin^2(\theta'_u/2) \cdot 2N = M$ with the error

$$|\Delta M| < (\sqrt{2MN} + \frac{N}{2^{n+1}})2^{-n} \tag{46}$$

All this can be made in $\mathcal{O}(\sqrt{N})$ oracle calls, as opposed to $N$ oracle calls on a classical computer.

# 4 Results

This section explains how the theory is applied to the minesweeper problem. The first subsection shows how to reduce the minesweeper problem to a general boolean satisfiability problem. It is followed by a step by step explanation of the algorithm used to solve a general *Minesweeper* game. Lastly, an explicit example of a game of *Minesweeper* is done, showing how the SAT-function is set up and solved with the algorithm presented earlier.

## 4.1 Reducing Minesweeper to SAT

All setups of minesweeper are reducible to an SAT problem [8]. It follows that all minesweeper setups can be reduced to a boolean expression. For a constraint with number $a$, meaning that it

should have $a$ mines adjacent to it to be satisfied, adjacent to $b$ variables, being empty squares. For the SAT to be satisfied all constraints have to be satisfied, this is analogue to combining them with AND gates, granting the expression

$$C_1 \wedge C_2 \wedge \ldots \wedge C_k = B(x) \tag{47}$$

where $C_i$ corresponds to a singular constraint and $k$ is the total number of constraints. For a constraint to be satisfied it has to be adjacent to $a$ mines. For one constraint surrounded by three variables, we this means that one and only one of these is a mine, the boolean expression for this is

$$C(x) = x_1 \oplus x_2 \oplus x_3. \tag{48}$$

In the same way all the different ways of placing the mines has to be combined by XOR gates. For a two constraint adjacent to three variables this gives

$$C(x) = (x_1 \wedge x_2) \oplus (x_1 \wedge x_3) \oplus (x_2 \wedge x_3). \tag{49}$$

Generally for $k$ constraints we get

$$B(x) = C_1 \wedge C_2 \wedge \ldots \wedge C_k \tag{50}$$

where $C_i$ has a constraint number $a$ adjacent to $b$ variables

$$C_i = x_1 \wedge x_2 \wedge \ldots \wedge x_{a-1} \wedge x_a \oplus x_1 \wedge x_2 \wedge \ldots \wedge x_{a-1} \wedge x_{a+1} \oplus \ldots \oplus x_{b-a+1} \wedge x_{b-a+2} \wedge \ldots \wedge x_{b-1} \wedge x_b. \tag{51}$$

The boolean function $B(x)$ is then what the oracle uses to identify the correct solution in Grover's algorithm.

## 4.2 The Minesweeper Algorithm

1. Start by probing a corner as this is the best starting position [8]
2. If the first square is a 0 all adjacent square to it are probed
3. When all squares adjacent to 0 constraints have been probed all constrains $> 0$ are numbered as well as all the unprobed squares adjacent to them and a SAT is constructed as described above.
4. Quantum counting is then used to check the number of solutions, if the answer is one we continue to step 6 and if we have more than one we proceed to step 5.
5. (a) Check if there are independent variables, as show in fig 13 and fig 14. If we have independent variables we can check them for solutions individually and they might have unique solutions.

| | | | | | |
|---|---|---|---|---|---|
| 1 | 1 | 1 | | | |
| 1 | X | 1 | | | |
| 1 | 1 | 1 | | | |
| | | | 1 | 1 | 1 |
| | | | 1 | X | 1 |
| | | | 1 | 1 | 1 |

Figure 13: A setup with two variables that are independent

| | | | |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | X | X | 1 |
| 1 | X | X | 1 |
| 1 | 1 | 1 | 1 |

Figure 14: A setup with four variables that depend on each other

(b) Should there be independent solutions we apply Grover's algorithm to them individually.

(c) The solutions are then again checked for multiple solutions and if that is the case a guess needs to be made, otherwise we go to step 6.

6. Grover's algorithm is used to find the proper solution to the SAT, in the SAT's solution variables that are FALSE corresponds to squares that are not mines and these are probed.

7. This is then repeated on the new setup that arises until the entire board is cleared or there is no unique solution to be found.

## 4.3 Solving a particular setup of minesweeper

This section will show an actual implementation of Grover's algorithm on the minesweeper problem formulated as a boolean satisfiability problem. A game of minesweeper has been started by probing the bottom left corner as seen in figure 15. All the constraints (probed squares with values $> 0$) are then numbered from the top left to the bottom right as shown in figure 16. All unprobed squares adjacent to the constraints are interpreted as boolean variables and also numbered following the same scheme.

| | | | | |
|---|---|---|---|---|
| 1 | 1 | X | X | X |
| X | 2 | X | X | X |
| 1 | 3 | X | X | X |
| | 2 | X | X | X |
| | 1 | 3 | X | X |

Figure 15: A setup of minesweeper.

| | | | | |
|---|---|---|---|---|
| $1_1$ | $1_2$ | $X_1$ | $X$ | $X$ |
| $X_2$ | $2_3$ | $X_3$ | $X$ | $X$ |
| $1_4$ | $3_5$ | $X_4$ | $X$ | $X$ |
| | $2_6$ | $X_5$ | $X_6$ | $X$ |
| | $1_7$ | $3_8$ | $X_7$ | $X$ |

Figure 16: The same figure as above but with numbered constraints and variables.

This then gives rise to the to vectors $C = [C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8]$ corresponding to the constraints $[1, 1, 2, 1, 3, 2, 1, 3]$ and $X = [X_1, X_2, X_3, X_4, X_5, X_6, X_7]$ corresponding to the variables respectively. From this the SAT-function $B(x)$ can be derived,

$$B(x) = C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5 \wedge C_6 \wedge C_7 \wedge C_8. \tag{52}$$

$B(x) = 1$ if and only if all the constraints are fulfilled, i.e. all the numbered squares have the exact number of mines adjacent to them as indicated by their value. This gives that all constraints correspond to clauses that in turn contain $\frac{b!}{(b-a)!a!}$ clauses, where b is the number of adjacent variables and a is the value of the constraint. These subclauses contain all possible permutations of mines, and are combined with XOR-operators. For example, since $C_2$ is a constraint with the value 1 adjacent to variables $X_1$, $X_2$ and $X_3$, $C_2 = X_1 \oplus X_2 \oplus X_3$. All the clauses that define $B(x)$ are

$$\begin{aligned}
C_1 &= (X_2) \\
C_2 &= (X_1 \oplus X_2 \oplus X_3) \\
C_3 &= (X_1 \wedge X_2) \oplus (X_1 \wedge X_3) \oplus (X_1 \wedge X_4) \oplus (X_2 \wedge X_3) \oplus (X_2 \wedge X_4) \oplus (X_3 \wedge X_4) \\
C_4 &= (X_2) \\
C_5 &= (X_2 \wedge X_3 \wedge X_4) \oplus (X_2 \wedge X_3 \wedge X_5) \oplus (X_2 \wedge X_4 \wedge X_5) \oplus (X_3 \wedge X_4 \wedge X_5) \\
C_6 &= (X_4 \wedge X_5) \\
C_7 &= (X_5) \\
C_8 &= (X_5 \wedge X_6 \wedge X_7)
\end{aligned} \tag{53}$$

This is the oracle function that is part of the Grover iteration. It recognizes solutions to this particular configuration. The corresponding matrix operator, with one added gate to swap the phase $U_B$ to this function will be an identity matrix, but with a $-1$ on the row corresponding to the correct solution. Since there are seven variables, seven qubits will be needed in order to cover all the possible configurations of mines. An additional qubit will be needed to perform the oracle operation of flipping the correct state. The oracle itself may need a set of auxiliary qubits, but these will be ignored for the purpose of illustrating the example and the oracle can be thought of as a black box that can recognize the correct solution if given it as input. The qubit register is initialized to

$$|\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{6} |x_i\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right], \tag{54}$$

where $n$ is the number of variables and $x_i$ is the binary basis state corresponding to the value i. The iteration begins by sending the state through the oracle, which performs a controlled NOT operation on the state associated with the correct state, resulting in the state

$$|\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{6} (-1)^{B(x)} |x_i\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]. \tag{55}$$

Then the inversion about the mean operator is applied (see section 3.6.1). The Grover iteration is repeated $\frac{\pi}{4}\sqrt{2^n} \approx 8$ times. After every iteration, the correct state is amplified and all the other states are attenuated, increasing the probability of measuring the correct state illustrated in figure 17.

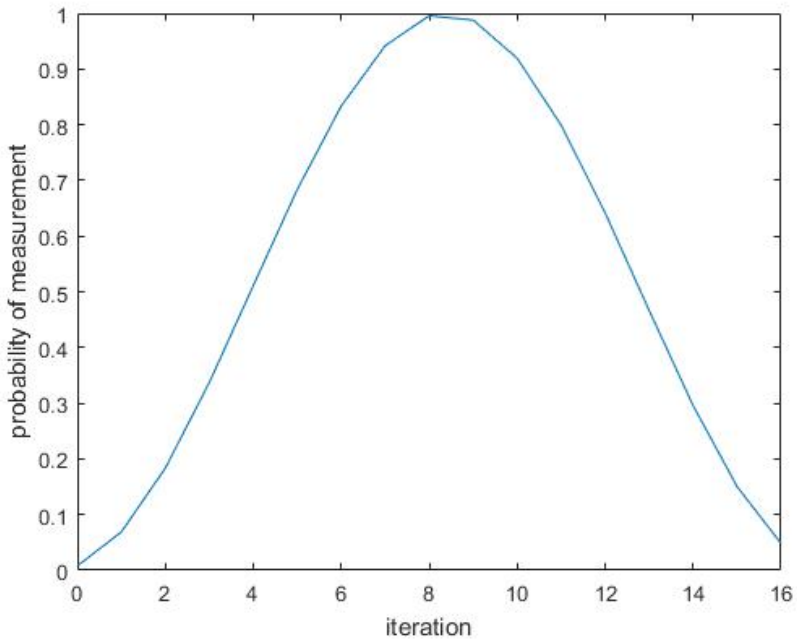


Figure 17: The probability of measuring the correct state as a function of the amount of Grover iterations performed.

Measuring the state after exactly eight iterations will yield the correct state with a probability of 99.6 %. If the iterations were to be continued, the probability of measuring the correct state would actually decrease as shown in figure 17. The correct solution to this particular configuration of constraints and variables is $X = [0, 1, 0, 1, 1, 1, 1]$. Therefore there has to be mines on the squares associated with variables $X_2, X_4, X_5, X_6$ and $X_7$ in order for the SAT-function B(x) to be true. Thus the squares corresponding to variables $X_1$ and $X_3$ are not mines and can safely be probed. The rest are mines and should be marked as mines, reducing any adjacent constraints so the next SAT-function B(x) can be set up properly. The new minesweeper board after the first

application of the quantum minesweeper solver in this particular case is shown in figure 18.

| | | | | |
|---|---|---|---|---|
| 1 | 1 | | 1 | X |
| M | 2 | 1 | 2 | X |
| 1 | 3 | M | X | X |
| | 2 | M | M | X |
| | 1 | 3 | M | X |

Figure 18: The setup after all mines have been found

This process is repeated until all mines have been located, and thus all safe squares probed, or until a guess has to be made.

# 5 Discussion

We have found that for a setup of $n$ variables there are a total of $2^n$ possible unique placement of mines. A classical algorithm would have to test all of these, making a total of $2^n$ oracle calls. Our algorithm does however just have to apply the quantum counting and Grover's search algorithm, with a total of $\frac{\pi}{4}\sqrt{2^n}$ oracle calls each, making a total of $\frac{2\cdot\pi}{4}\sqrt{2^n} = \mathcal{O}(\sqrt{2^n})$ oracle calls. This is a significant increase in efficiency, especially as trying all these setups is what takes the most time in a classical algorithm [8]. Since the speedup is quadratic the speedup becomes more significant as the setup increases.

The brute force method for finding solutions to the minesweeper problem that we have used is not optimal in terms of efficiency since brute forcing often is not necessary to find a new square. There are several trivial cases such as a constraint adjacent to the same amount of variables that the constraint show. These cases could easily be identified by a classical algorithm instead of setting up an oracle and using Grover's algorithm. However should a setup arise that is complex enough that no trivial solution can be found and that trying every possible setup becomes necessary, this can be done faster with our algorithm. For this reason our algorithm should be used as a part of a classical algorithm, used when it is necessary to brute force a solution.

# 6 Conclusions

Solving Microsoft's *Minesweeper* with a quantum algorithm gives a quadratic speedup compared to solving it with a classical algorithm. This is accomplished by first reducing the minesweeper problem to a boolean satisfiability problem. The SAT-problem is then solved using Grover's algorithm. A classical algorithm would require a number of iterations proportional to the size of the problem in order to solve it, while Grover's algorithm only needs a number of iterations

proportional to the square root of the size of the problem. To maximize speed, our quantum algorithm should be implemented in a classical minesweeper algorithm to be used when no trivial solutions can be found.

## References


[1] Michael A. Nielsen, Isaac L. Chuang, Quantum Computation and Quantum Information, 3rd Edition, University Press, Cambridge, 2000

[2] Tom Carter, Can we compute faster in a multiverse?, https://csustan.csustan.edu/~tom/MISC/qc-article/qc-article-htm.html, 2017-05-17

[3] Clay Mathematical Institute Cambridge Massachusetts, Millenium Problems, http://www.claymath.org/millennium-problems, 2017-05-21

[4] IBM, Quantum Computing, http://research.ibm.com/ibm-q/ 2017-05-21

[5] Richard W. Kayes, Minesweeper is NP-Complete, Mathematical Intelligencer, volume 22, nr 2, page 9-16, 2000

[6] A. Yu. Kitaev, A. H. Shen, M. N Vylayi, Classical and Quantum Computation, 1st Edidtion, American Mathematical Society, 2002

[7] Peter W. Shor, Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer, Society for Industrial and Applied Mathematics, Vol 26, 1997, page 1484-1509

[8] Chris Studholme, Minesweeper as a Constraint Satisfaction Problem, University of Toronto, 2001