GYMNÁZIUM EVOLUTION JIŽNÍ MĚSTO

JAKUB ZÁLOHA

ACHIEVING A QUADRATIC SPEEDUP WITH QUANTUM SEARCH ALGORITHMS

Thesis

Prague 2022

Statement of accuracy

I hereby declare that I have written this work alone with the use of appropriate citation standards.

Declared by Jakub Záloha in Prague on 22.4.2022.

Abstract

This paper is concerned with the process of designing a boolean satisfiability problem program. The designed algorithm took advantage of the novice technology quantum computation. The code was programmed in qiskit and was executed on a quantum computer owned by IBM. Grovers algorithm core principles were used and theoretically provide quadratic speedup over an equivalent method used in classical computing.

Este artículo se ocupa del proceso de diseño de un programa de problemas de satisfacción booleana. El algoritmo diseñado aprovechó la computación cuántica de la tecnología novata. El algoritmo fue programado en qiskit y fue ejecutado en una computadora cuántica propiedad de IBM. Se utilizaron los principios básicos del algoritmo Grovers y, teóricamente, proporcionan aceleración cuadrática sobre un método equivalente utilizado en la informática clásica.

Tato ročníková práce se zaobíra procesem navrhování algoritmu řešící problem splnitelnosti booleanovské formule. Tento algoritmus využívá nové technologie kvantových počítač. Kód byl naprogramován v SDK qiskit a exekuován na kvantovém počítači ve vlastnictví IBM. Groverův algorithm a jeho principy byli využity za účelem teoretického kvadratického zrychlení oproti ekvivalentním metodám používaným v klasické informatice.

Acknowledgments

The author would like to acknowledge the usage of provided tools from IBM. The author would also like to acknowledge and thank all authors of literature that was used to complete this paper.

The author would like to thank Mgr. Pavel Petrášek for guidance.

Table of content

Table of Figures

1 Introduction

Quantum computation is a novice technology that utilises quantum physics. In our paper, we will go over the core concepts of quantum computation and construct a search algorithm designed for quantum computers that should provide a theoretical speedup over equivalent classical computational algorithms. Our program will be designed as a boolean satisfiability problem algorithm. The motivation for writing a paper on this topic came from a coding competition, where we had to design an algorithm that would process a large data entry with many variables to consider. It caught us by surprise that even with modern computers, this algorithm is still computationally difficult and slow. When we had read a paper on potential use cases for quantum computing, we realised that this algorithm would theoretically run with a quadratic speed up on such technology. We had decided to construct a similar algorithm with qiskit and execute it on IBM quantum computer. We will use a modified Grover's algorithm for this purpose. Every referenced file in this paper has been uploaded on GitHub and can be accessed here.

2 Linear algebra for quantum computation

The purpose of this chapter is to acquaint the reader with linear algebra, notations and definition used in this paper. Familiarity with concepts of linear algebra will equip the reader with the necessary tools to further understand quantum algorithms, operations with qubits and logic gates.

2.1 Vector space

"In mathematics, a Hilbert space is an inner product space that is complete with respect to the norm defined by the inner product." (*Hilbert spaces | Quantiki*, 2015) Hilbert space is a specific vector space with the natural inner product or a dot product that equips it with a distance function. In quantum computation, we refer to Hilbert spaces over a field of complex numbers. (Portugal, 2013)

In quantum mechanics, we use infinite vector spaces more frequently than finite spaces. But in the context of programming for quantum computers specifically, infinite spaces are rare, thus we will be talking about vector spaces $\mathbb{C}^n$. These spaces consist of n vectors satisfying linear independence and have all the properties of a Hilbert space. Given a basis of $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ for $\mathbb{C}^n$, generic vector **v** equates to "$\mathbf{v}= \sum_{i=1}^{n} a_i v_i$ where coefficients a_i are complex numbers. The dimension of a vector space is the number of basis vectors." (Portugal, 2013)

2.2 Dirac notation

Dirac notation offers a more efficient system of writing down vectors and their inner products. Dirac notation consists of bra-ket pairs. Bra is denoted as $\langle\ |$ and ket as $|\ \rangle$.Previously we used bold letters to denote vector. In Dirac notation we would denote a vector

$$\mathbf{v} \equiv |v\rangle.$$

In the example of vector space, we used a form $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ to describe an indexed vector basis. With Dirac notation we would denote this basis

$$\{|v_0\rangle, \ldots, |v_n\rangle\}.$$

In these examples, we used the ket notation to describe vectors. Let's thus imagine a quantum system vector $|0\rangle$. This vector is one of two basis vector states, or qubits in quantum systems. We will cover more information on these basis states later. For now, note that $|0\rangle$ qubit

state has a representative matrix $\begin{bmatrix}1\\0\end{bmatrix}$. We commonly use the linearly transformed version of qubits and vector spaces. Dirac notation allows us to effectively distinguish one another. Ket vectors are commonly known as column vectors and when linearly transformed, we use the bra notation style, to describe a row vector. Given a vector $|0\rangle$ is an element of a vector space V, $\langle 0|$ would be an element of a dual space V^*. $\langle 0|$ would have a representative matrix $[1 \quad 0]$. Using these bra-ket pairs we would denote an inner product of $|\Psi\rangle$ and $\langle\Psi|$ as $\langle\Psi|\Psi\rangle$.(Shafi, 2020; Bradben, 2022)

2.3 Tensor product

As we will cover later, many advantages quantum systems have over traditional systems are due to interactions between multiple qubits. Thus, a topic of great importance is how we measure, think about and construct these interactions.

Tensor products are a way to describe a system consisting of multiple subsystems. We understand subsystems to be vectors in a Hilbert space. Let's consider a system consisting of two subsystems, each described by the vectors $|\Psi_i\rangle$ and $|\Psi_{ii}\rangle$. We would describe the main system as a tensor product of both subsystems. We use

$$\otimes$$

to denote tensor product. Hence the main system would be denoted as $|\Psi_i\rangle \otimes |\Psi_{ii}\rangle$.

(*Tensor product | Quantiki*, 2015)

3 Qubits

In this chapter we are going to describe the core concept of quantum computing-a qubit.

3.1 Qubit states

Qubits are very simple quantum systems described by a two-dimensional Hilbert space. Just like in binary computing systems, qubits are measured in two states called 1 and 0 state. However, being quantum systems allows qubits to display properties described by quantum mechanics. One of the properties allowing qubits to be in a combination of the two measurable states is called superposition.(Hughes *et al.*, 2021; 'Qubit', 2021)

3.2 Superposition

Superposition of qubits allows quantum computers to achieve speedup over classical computers in certain algorithms. We can describe a superposition as an ability for qubits or quantum systems in general to be in multiple states until they are measured. Quantum computers achieve this state with the use of logic gates.

Qubit in a superposition has an equal chance of being measured as 1 or 0. Until then, it stays in a form of all possible states. State of this qubit is $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ or $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$ depending on the state of initialization. We refer to superpositions as $|+\rangle$ and $|-\rangle$.(Forcer *et al.*, 2002)

4 Logic gates

Qubit's can be operated on in the same manner any vector can. We utilise this in creating logic gates which can be described as a matrix and operate on qubits with matrix multiplication.

4.1 Acting on single qubits

4.1.1 X gate

Matrix of an X logic gate $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ multiplied by a qubit in a state $|0\rangle$ returns $|1\rangle$. The same applies for $|1\rangle$ which multiplied by X gate returns $|0\rangle$.

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \times \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}; \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \times \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$ (Hagouel and Karafyllidis, 2012)

4.1.2 Hadamard gate

Hadamard gate is used to put qubits into superposition.

$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \times \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{|0\rangle+|1\rangle}{\sqrt{2}} = |+\rangle$; $\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \times \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \frac{|0\rangle-|1\rangle}{\sqrt{2}} = |-\rangle$ (Hagouel and Karafyllidis, 2012)

Having a qubit with state $|0\rangle$ in a circuit, there should be equal probabilities for measuring $|0\rangle$ and $|1\rangle$ after H gate is applied. This gate is can be visualised as seen on Figure 1. (Forcer *et al.*, 2002)

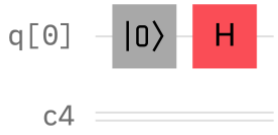

Figure 1 Hadamard gate circuit

Using the simulator predictor on IBM quantum composer, in Figure 2, we can see that the probabilities correspond with our hypothesis.

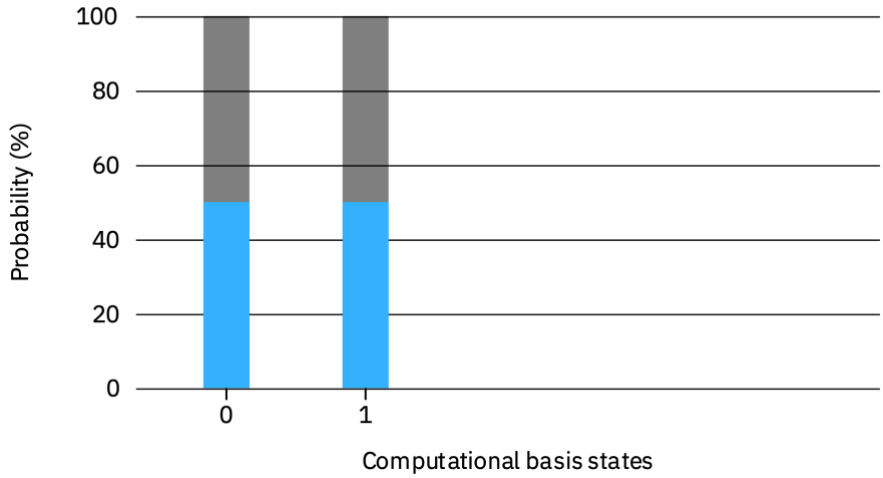

Figure 2 Resulting probabilities after applying H gate

4.1.3 Z gate

Z gate flips the qubit around its Z axis. When applied to a $|0\rangle$ state qubit, no changes are measured. Z gate applied to a $|1\rangle$ state qubit will flip the qubit to $|-1\rangle$. This state measures as $|1\rangle$, but qubit has different properties and interacts differently. $|-1\rangle$ is also written as $|1\rangle$ with its phase angle rotated by π. On a Bloch sphere in Figure 3 used to visualise the state of a qubit, we can see this rotation.

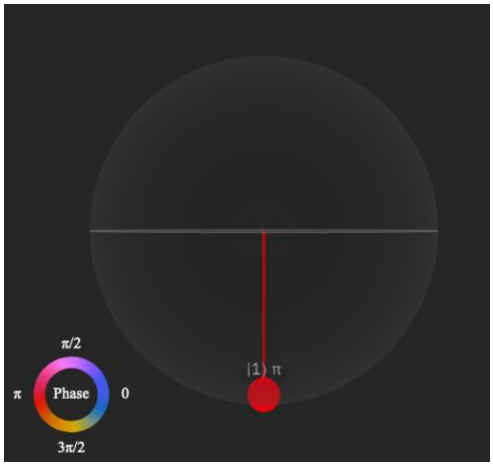

Figure 3 Bloch sphere Z gate

4.2 Acting on multiple qubits

Systems with two qubits are described similarly to single qubit systems and have four possible states $|00\rangle; |01\rangle; |10\rangle; |11\rangle$. Qubit systems can be described in vectors; we describe their collective states using the tensor product.

$$|ba\rangle = |b\rangle \otimes |a\rangle$$

4.2.1 Hadamard gate on two qubits

Two qubits after an H gate is applied simultaneously are both in the state of $|+\rangle$. Their collective state is $|+ +\rangle$. Probability of measurement is equal for all four states with each qubit in a superposition. Bloch sphere visualisation in Figure 4 shows the state probabilities. (*Multiple Qubits and Entangled States*, 2022)

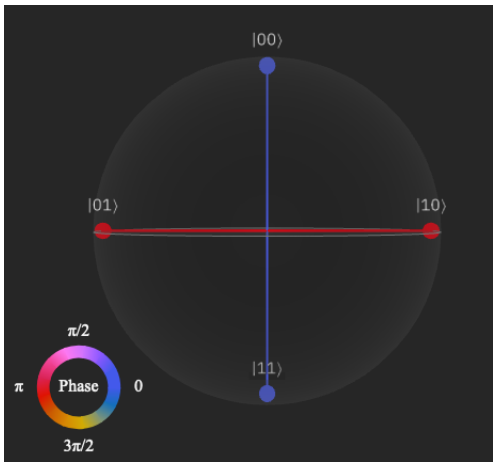

Figure 4 Bloch sphere H gate on 2 qubits

4.2.2 CNOT gate

Controlled not gate flips the target (first) Qubit only if the control qubit is in a state of |1⟩. This principle allows for entangled systems when combined with a Hadamard gate. We can analyse this principle further with a circuit drawn in Figure 5.(Hagouel and Karafyllidis, 2012)

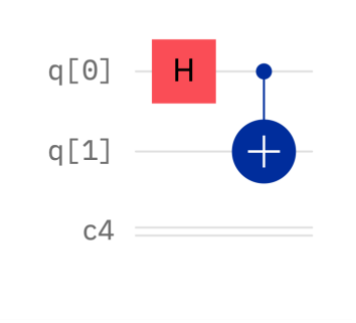

Figure 5 CNOT circuit

We can observe that two outcomes arise in Figure 6. Either the state of q[0] is measured as |1⟩ and q[1] will flip and measure as |1⟩ or q[0] and q[1] will both measure as |0⟩.

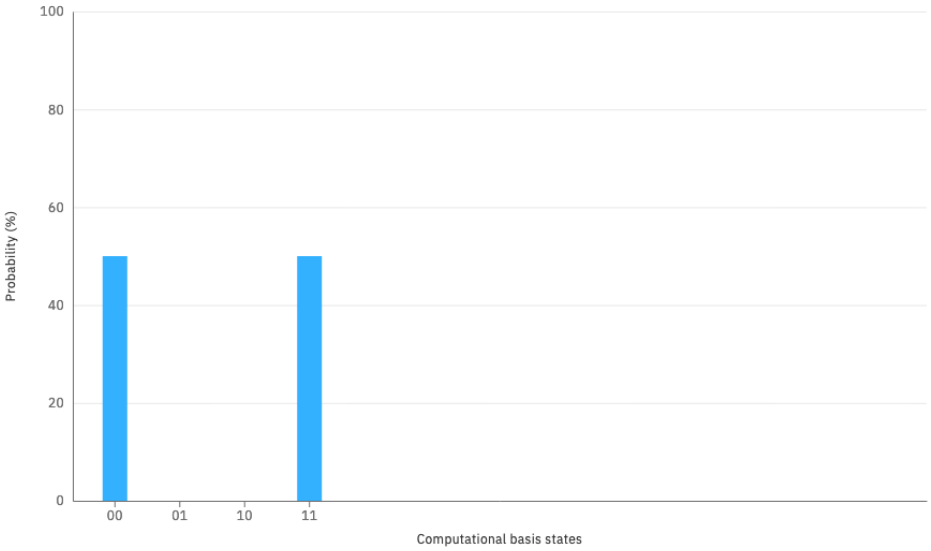

Figure 6 CNOT probabilities

This is known as Bell state and is the basic principle behind entanglement. Scientists have confirmed that if one qubit is measured, it's entangled counterpart will collapse its superposition to a state with respect to the first qubit. This principle is nearly instantaneous, about four magnitudes faster than the speed of light, and hypothetically works with infinite distance between the two qubits. The speed of quantum entanglement doesn't violate theory of relativity. (Gisin, 2019)

4.2.3 CZ gate

CZ gate is symmetrical and doesn't differentiate between the control and target qubit. Visualising probabilities of two qubits in superposition on a Bloch sphere in Figure 7, we have equal probabilities, between following four states. (Hagouel and Karafyllidis, 2012)

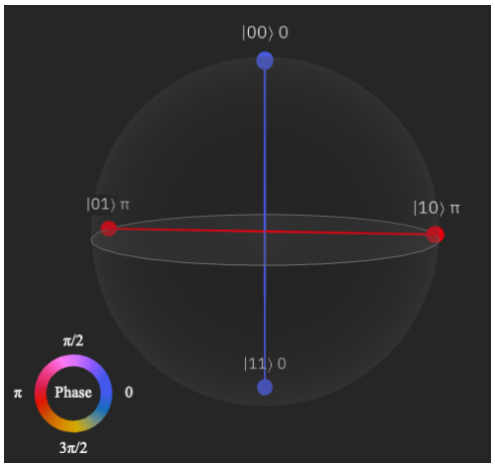

Figure 7 CZ gate scenario no.1

The same circuit with CZ gate added shows that if both qubits are $|1\rangle$, Z gate is applied to the whole system and its phase angle will measure as π; $-|11\rangle$ on a Bloch sphere in Figure 8.

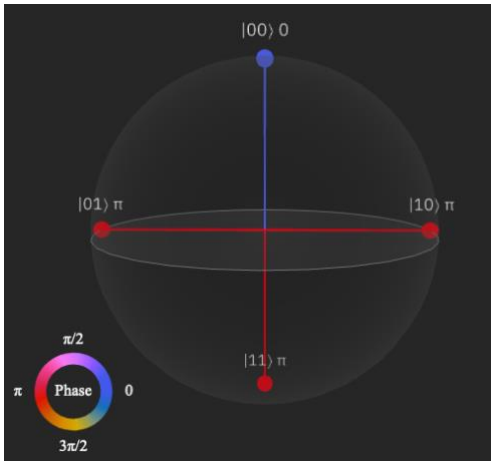

Figure 8 CZ gate scenario no.2

5 Search algorithms on a classical computer

5.1 Time complexity

In order to describe how fast an algorithm is, we measure how many elementary operations a computer performs. Time complexity of an algorithm will not differ depending on computers abilities, but the time, in which it is completed, will.

We will denote the time complexity by big-O notation, which describes the worst-case scenario.

Time complexity of an algorithm that takes one number and increments it will be one because we only had to perform 1 elementary operation. We would denote such time complexity by big-O notation O(1). (*Big O Notation Explained with Examples*, 2020)

5.2 Time complexity of search algorithms

5.2.1 Search algorithms for sorted data sets

If we were to search for a specified data entry in an ordered set, we would have multiple algorithms in our disposal. Binary search algorithm runs in O(log n) time and is the most efficient one. We can visualise this time complexity in figure 9(‘Binary search algorithm’, 2022)

Binary search works by splitting a sorted data set in half and seeing if the number is in the first dissection or the second half, it continues until it finds the targeted number. Having an array of numbers $(0; 10)$ for a computer to find the number $n; n \in (0; 10)$, it would first split this array in the middle, check whether six == n. If not, it would check if $n < 6$ or $n > 6$. If the first case were true, it would split the first half of our array and repeat the process. The same holds true for the second option. Below is the pseudo code for this algorithm.

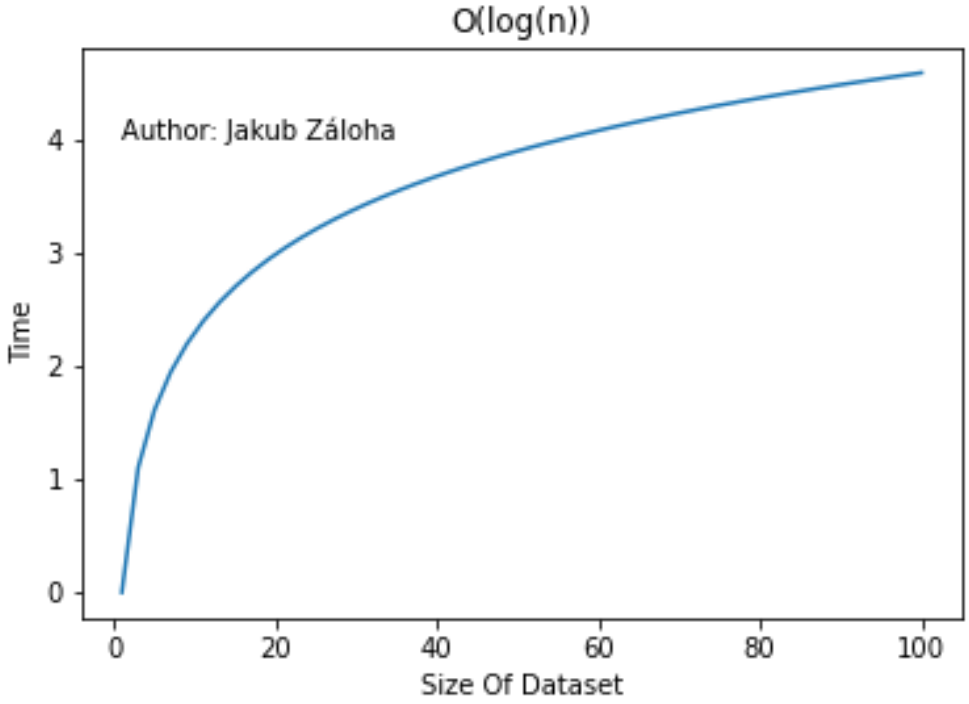

Figure 9 O(log(n)) time complexity graph

```
Procedure binary_search
   A ← sorted array
   n ← size of array
   x ← value to be searched

   Set lowerBound = 1
   Set upperBound = n

   while x not found
      if upperBound < lowerBound
         EXIT: x does not exists.

      set midPoint = lowerBound + ( upperBound - lowerBound ) / 2

      if A[midPoint] < x
         set lowerBound = midPoint + 1

      if A[midPoint] > x
         set upperBound = midPoint - 1

      if A[midPoint] = x
         EXIT: x found at location midPoint
   end while

end procedure
```

('Binary search algorithm', 2022)

5.2.2 Search algorithms for unsorted data sets

When searching for n in unordered data set, there is no pattern for the algorithm to take advantage of, hence it checks every entry in that set exactly once and checks if that entry == n. This method is called linear search and its time complexity is O(n) visualised in figure 10.

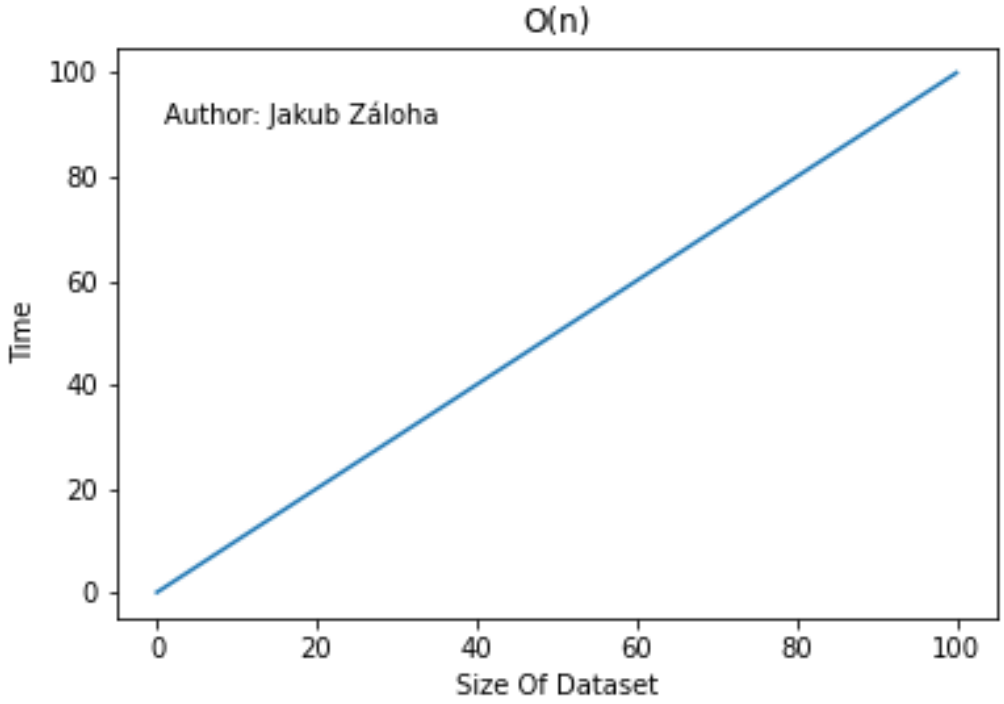

Figure 10 O(n) time complexity graph

```
procedure linear_search (list, value)

   for each item in the list
      if match item == value
         return the item's location
      end if
   end for

end procedure
```

6 Sudoku

6.1 The game of sudoku

For simplification purposes, two by two squares sudoku had been chosen for our practical part. For the sudoku to be declared as successfully solved, all missing numbers must be filled within following constraints. No column can contain the same value twice, neither can a row. While this sudoku might seem trivial it demonstrates the fundamentals of quantum computing clearly and should provide us with a speedup.

6.2 Sudoku solver on a classical computer

Algorithms use common computer science methods when solving sudoku. These methods include backtracking and recursion.

6.2.1 Backtracking

Backtracking technique tries to build a solution by filling data with no logic. It does so one by one and removes the last data point when solution fails to meet all the constraints. (*Backtracking | Introduction - GeeksforGeeks*, 2018)

6.2.2 Recursion

Recursion is an algorithmic technique. We declare a recursion algorithm when it contains a function that calls upon itself. For recursion algorithms to be safe, the feedback loop cannot be infinite.

6.2.3 Time complexity

Because recursion runs solely on the principle of trial and error, in the worst-case scenario, it will have tried all the possible variations before it finishes. The time complexity of such algorithms is large.

"Time complexity of such algorithm is $O(n^m)$, where n is the number of possibilities for each square and m is the number of squares that are blank." (*Sudoku Solver*, 2020) In a four-by-four sudoku, n will always be four and we visualise it's time complexity in Figure 11.

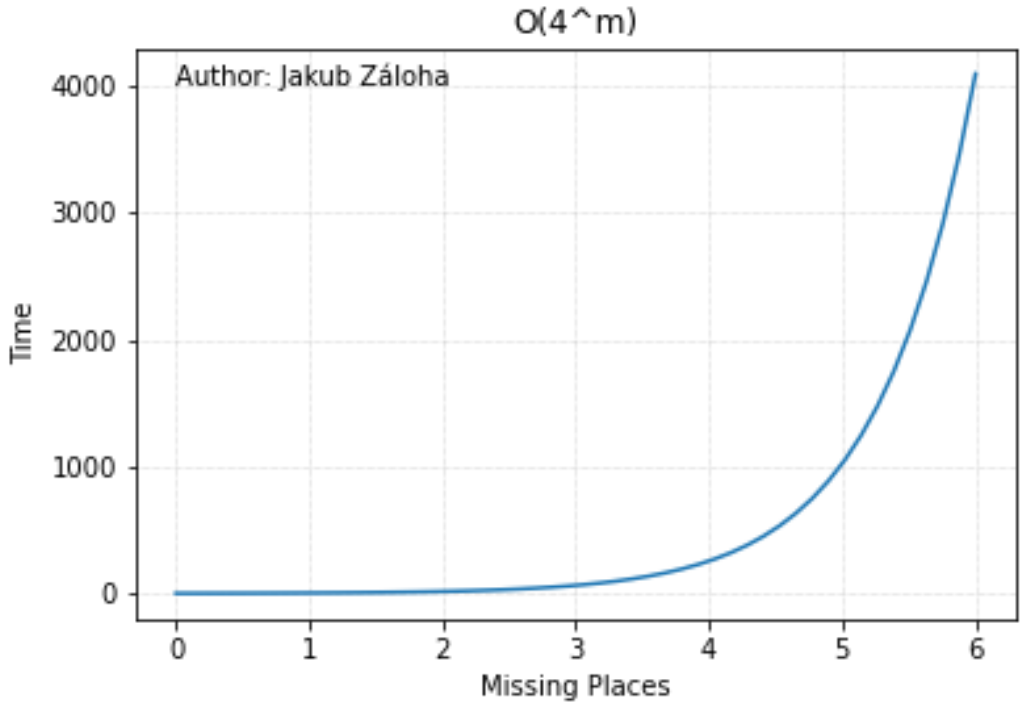

Figure 11 O(4^m) time complexity graph

The most places a standard nine by nine sudoku can miss to still be solvable is sixty-four. Time complexity of such sudoku as seen in Figure 12 demonstrates how complex some computational problems are to solve.

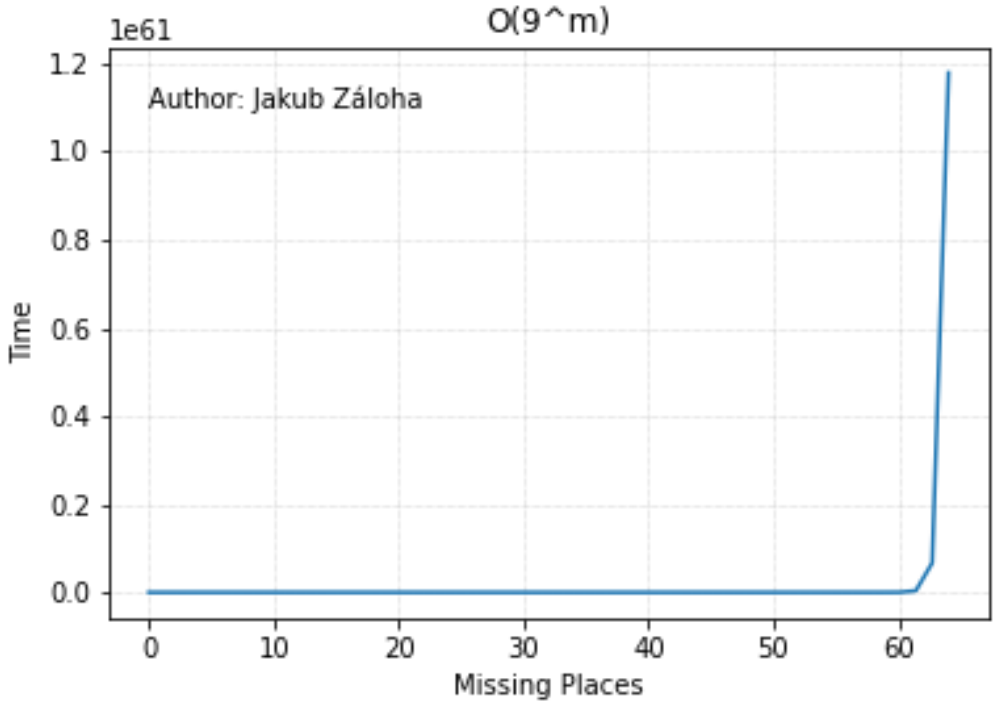

Figure 12 O(9^m) time complexity graph

Time complexity of the worst-case scenario with 64 missing numbers is 9^{64}.

7 SAT search algorithm

7.1 Grover's algorithm

Grover's algorithm is a search algorithm suitable for finding results in a large, unsorted dataset. The area where Grover's algorithm is proposed to provide speedup over classical search algorithm is SAT, in other words Boolean satisfiability problems.

These problems are usually easy to prove, hard to solve. Let's look at an example of such problem. We have a list of all restaurants in the world. Find a restaurant that suits all the following criteria. A restaurant must be nearby. A restaurant must be cheap. A restaurant must have four or more stars. A restaurant must have Vietnamese food. These conditions can be written as binary expression a AND b AND c AND d. This algorithm initiates all the restaurants to value 0, performs criteria check on each restaurant and if the restaurant meets all the criteria, it switches value of the restaurant to 1. We can assume this algorithm would run close to O(mn) time, where $m$is the number of criteria and n is the set of restaurants. In this chapter we will look at the Grover's algorithms mechanisms and analyse the innerworkings of a quantum computer which should provide us with a quadratic speedup.('Boolean satisfiability problem', 2022)

7.2 The principle behind Grover's algorithm

Grover's algorithm works on a principle of increasing the chance of measuring the targeted state. This means that a system with multiple qubits, each being in a superposition will have a high chance of measuring the correct answer. While a system with two qubits has an equal probability of measuring each possible state, amplitude amplification increases the amplitude (probability squared) of the correct answer while simultaneously decreasing the amplitudes of other states.(Portugal, 2013)

8 Practical part

In this chapter we will attempt to write an algorithm for solving a two-by-two sudoku. We will compare multiple sources and try to construct an algorithm that will provide us with further information on building quantum gates and working with this novice technology. While constructing this algorithm. We will analyse individual parts and provide reader with necessary details and methodologies. If we succeed in building such algorithm, we will provide it for free on github.com as a mean to contribute to the research community. We will also provide a discussion on the viability of this technology with the gathered information.

8.1 Research questions

To what extent is quantum computing a viable option when performing time complex algorithms?

8.2 Goals

Contribute to the research community by creating a Boolean satisfiability search algorithm.

Analyse the process of programming for quantum algorithms.

9 Methodology

We will use the programming language python for constructing our algorithm. Python is a language commonly used for data science and applications with large levels of abstraction. Python is a user-friendly language with a network of contributors. Developers often use packages. These are modular application built on top of a programming language. Packages are an optimal solution when there is no need for optimising every component of an application. We will use a software development kit qiskit developed by IBM. Development kits are a collection of packages, in this case designed for interacting with quantum computers. Qiskit is also an open-source project, which means that all the core components can be examined by the users, and everyone can contribute to this project. Another reason for using qiskit as opposed to other alternatives is its extensive documentation and a collection of guides. (*Welcome to Python.org*, 2022; *Qiskit*, 2022)

Matplotlib is a python-based library of visualisation tools which can be used in anything from creating complex graphs and data visualisation to plotting the circuits in our algorithm. It is also opensource and has a large community around it. We use matplotlib not only in the practical part of this paper, but also in the theoretical part.(Hunter, Darren and Firing, 2022)

Jupyter notebook is a web-based development environment we will use for developing our algorithm. It has seamless integration with qiskit, matplotlib and IBMQ. It is the gold standard development environment in physics and math research. It is also opensource. (Bektas, 2022)

IBM Quantum Services or IBMQ is a cloud-based solution for experimentation and real-life development on quantum computers. It offers a wide array of quantum systems that can be used for no fee. It also provides the user with a build in jupyter notebook.(*IBM Quantum Computing*, 2022)

GitHub is a cloud-based application used for maintaining code and projects. It is a place where you can find most of the open-source projects.(*GitHub: Where the world builds software · GitHub*, 2022)

10 Sudoku

10.1 Rules

Our algorithm should provide us with a solution that respects the following constraints: $\alpha_0 \neq \alpha_3, \alpha_1 \neq \alpha_2$, drawn in Figure 13. This sudoku is modified to allow numbers in the same rows and columns but not diagonals. Number value must be in the Boolean form 0 or 1.

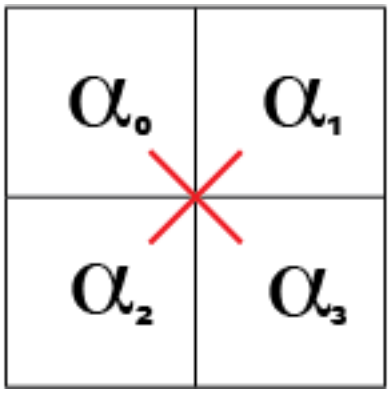

Figure 13 Modified sudoku rules

This means that results are valid in the following forms:

$$\alpha_0 = 0;\ \alpha_1 = 0; \alpha_2 = 1; \alpha_3 = 1;$$

$$\alpha_0 = 0;\ \alpha_1 = 1; \alpha_2 = 0; \alpha_3 = 1;$$

$$\alpha_0 = 1;\ \alpha_1 = 0; \alpha_2 = 1; \alpha_3 = 0;$$

$$\alpha_0 = 1;\ \alpha_1 = 1; \alpha_2 = 0; \alpha_3 = 0;$$

10.2 Overview of Grover's algorithm

10.2.1 Initialization

We initialize N qubits in $|s\rangle$. Our qubits will be in a state of uniform superposition $\frac{1}{\sqrt{N}}\sum_{x=0}^{N-1}|x\rangle$. We also know that the uniform superposition without correct solutions will be in a right angle to the correct solution. This gives us a clear way to visualise the whole process of Grover's algorithm in a two-dimensional vector space $\mathbb{C}^N$ which we do in Figure 14. (Cotrini, 2019)

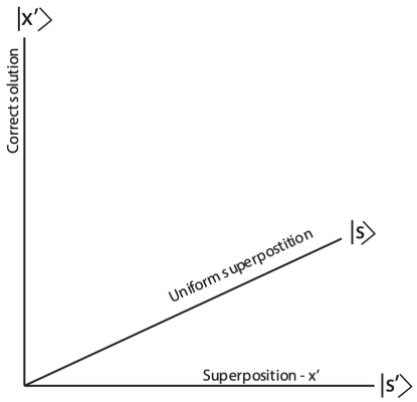

Figure 14 Grover process no.1

10.2.2 Oracle

Oracle is a stage responsible for correctly identifying the solution to our problem. Logic gates are responsible for checking conditions. If all clauses are met, the solution is marked by rotating it around the π. Phase angle of the solution will be $-|x\rangle$. The function for marking the correct solution will equal to one if correct and zero otherwise. We can describe this function as $\int(x') = 1; \int(x) = 0;$ x' == correct solution.

Amplitude of the uniform superposition will be reduced because of the change in phase angle. The Oracle operation takes in $|x\rangle$ and returns $(-1)^{\int(x)}|x\rangle$. Geometrically, $-|x\rangle$ rotates about the uniform superposition - $|x'\rangle$ as seen in Figure 15. (Akanksha Singhal and Chatterjee, 2018)

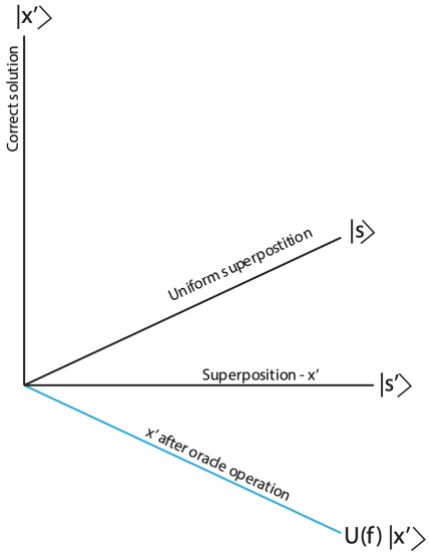

Figure 15 Grover process no.2

10.2.3 Amplification

Final stage of Grover's algorithms increases the amplitude corresponding to the correct state $|x'\rangle$. It does that by performing the diffusion operation $\widehat{D} = 2|s\rangle\langle s| - \hat{I}$. This operation reflects the amplitude corresponding to $|x'\rangle$ above the uniform uniformed superposition $\frac{1}{\sqrt{N}}\sum_{x=0}^{N-1}|x\rangle$ shown in Figure 16. (Katabira, 2021)(Akanksha Singhal and Chatterjee, 2018)

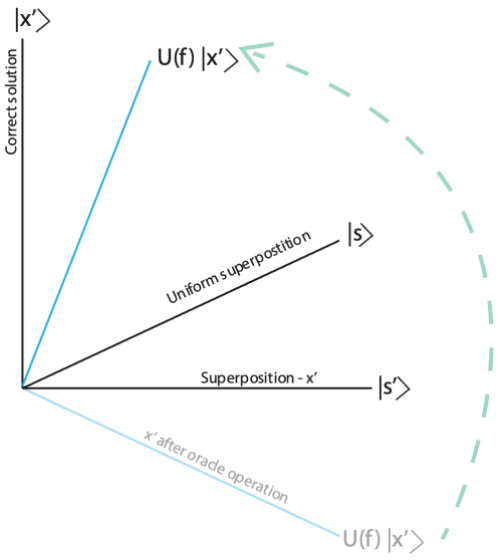

Figure 16 Grover process no.3

10.3 Creating individual components

For our code to be readable and reusable, in other words, to maintain good practices, we will declare individual functions for our algorithm.

10.3.1 Diffuser

Diffuser is a name of function responsible for performing amplitude amplification.

As previously described, we need to rotate the state with a negative phase above the uniform superposition. If the angle between equal superposition and superposition excluding the correct solution is θ, the angle between uniform superposition and negatively marked state is 2θ. This means that we need to rotate the negatively marked state by 4θ to end up with a state closer to the correct state. (Learning, 2021)

We know that $|s\rangle$ is the uniform superposition and now we need to find an operator a that returns the average amplitude of a state.

$$|\Psi\rangle = \sum_x a_x |x\rangle$$

$$A|\Psi\rangle = \sum_x \bar{a}|x\rangle$$

$$\bar{a} = \frac{1}{2^N}\sum_x a_x$$

The diffusion operator is denoted as D

$$D = 2A - I$$

$$D|\Psi\rangle = (2A - I)|\Psi\rangle = (2\bar{a} - 1)|\Psi\rangle = \sum_x (2\bar{a} - a_x)|x\rangle$$

$$A = |s\rangle\langle s|$$

$$D = 2|s\rangle\langle s| - I$$

(Squarishbracket, 2018)

With this information we can describe a function diffuser in python.

First, we put each qubit out of superposition with a Hadamard gate. We apply X gate to each qubit and perform a controlled Z gate. We then apply an X gate and entangle all the qubits with a superposition. We return this function as a quantum gate. The whole function can be viewed as a circuit in Figure 17 (Akanksha Singhal and Chatterjee, 2018; Pal *et al.*, 2020)

```
def diffuser(nqubits):
    qc = QuantumCircuit(nqubits)
    for qubit in range(nqubits):
        qc.h(qubit)
    for qubit in range(nqubits):
        qc.x(qubit)
    qc.h(nqubits-1)
    qc.mct(list(range(nqubits-1)), nqubits-1)
    qc.h(nqubits-1)
    for qubit in range(nqubits):
        qc.x(qubit)
    for qubit in range(nqubits):
        qc.h(qubit)
    U_s = qc.to_gate()
    U_s.name = "U$_s$"
    return U_s
```

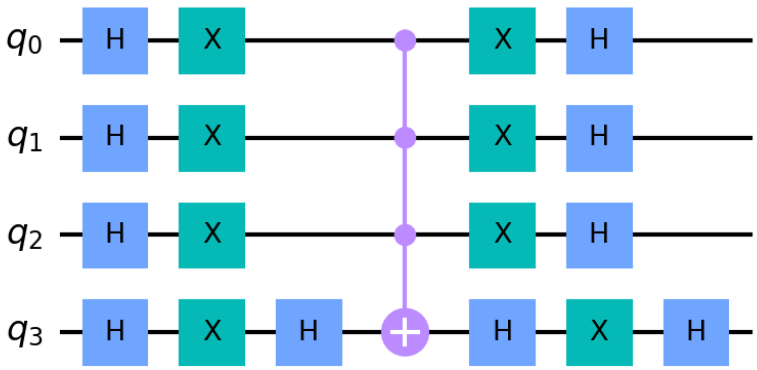

Figure 17 Diffuser function

10.3.2 XOR gate

To be able to check whether two variables have the same value, we must construct a gate. Fortunately, there is a gate that does just that in classical computing, hence we can copy the logic behind it. Our function will need to take in the quantum circuit to which it will be applied, two qubits that it will compare and an output variable to which it will write the result.

```
def XOR(qc,a,b,output):
    qc.cx(a,output)
    qc.cx(b,output)
```

10.3.3 Clause list

We now need to tell our program what constrains the result needs to follow. We will put information into a clause list. Again, our sudoku constraints are $\alpha_0 \neq \alpha_3, \alpha_1 \neq \alpha_2$.

```
clause_list = [[0,3],[1,2]]
```

10.3.4 Sudoku oracle

To build our oracle we will use the XOR gate and check all our clauses

```
def sudoku_oracle(qc,clause_list,clause_qubits):
    i=0
    for clause in clause_list:
        XOR(qc,clause[0],clause[1], clause_qubits[i])
        i +=1
    qc.mct(clause_qubits, output_qubit)
    i = 0
    for clause in clause_list:
        XOR(qc,clause[0],clause[1], clause_qubits[i])
        i +=1
sudoku_oracle(qc, clause_list, clause_qubits)
qc.draw()
```

10.4 Building the circuit

We can now assemble the parts together and build the actual circuit.

```
var_qubits = QuantumRegister(4, name='v')
clause_qubits = QuantumRegister(2, name='c')
cbits = ClassicalRegister(4, name='cbits')
output_qubit = QuantumRegister(1, name='out')
qc = QuantumCircuit(var_qubits, clause_qubits, output_qubit, cbits)
qc.initialize([1, -1]/np.sqrt(2), output_qubit)
qc.h(var_qubits)
qc.barrier()
sudoku_oracle(qc, clause_list, clause_qubits)
qc.barrier()
qc.append(diffuser(4), [0,1,2,3])
qc.measure(var_qubits, cbits)
qc.draw(fold=-1)
```

We first declare all our 9 qubits. 4 will be used for measurement. We call these var_qubits. When we measure the states of var_qubits, we will write their results onto the clause_qubits. Cbits will be used to write the final state of our application.

We then append these qubits and classical bits onto our quantum circuit qc. The output qubit is initialized in the state |Ψ⟩ as explained previously. After that we can put all our var_qubits in the superposition with the H gate. Just for clarity we append a visual barrier to our quantum circuit. Next, we utilise the sudoku oracle and draw another barrier. After that we append the diffuser and measure the var_qubits. We write the measurements on the cbits.

We have now built our application and can draw the final circuit as seen in Figure 18.

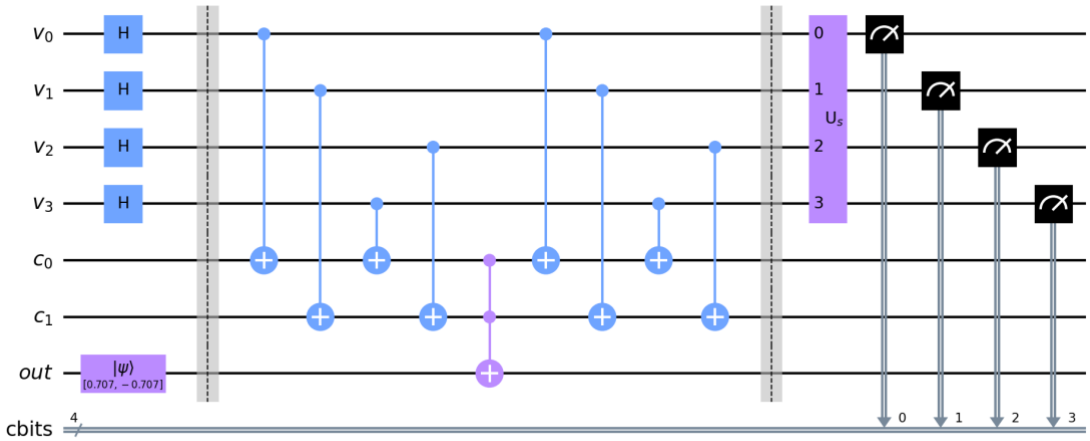

Figure 18 Grover circuit

11 Running the application and analysing results

We can now finally run our application on a real quantum computer. We have chosen ibm_washington system which supports operations on up to 127 qubits. It is one of the most powerful quantum computers available. When running the final application, the computer had been calibrated with specifications you can download here. We have run our application with 1024 shots. Each shot represents one execution of our algorithm. This technique of using multiple shots is used to further validate the results.

Results can be seen in Figure 19.

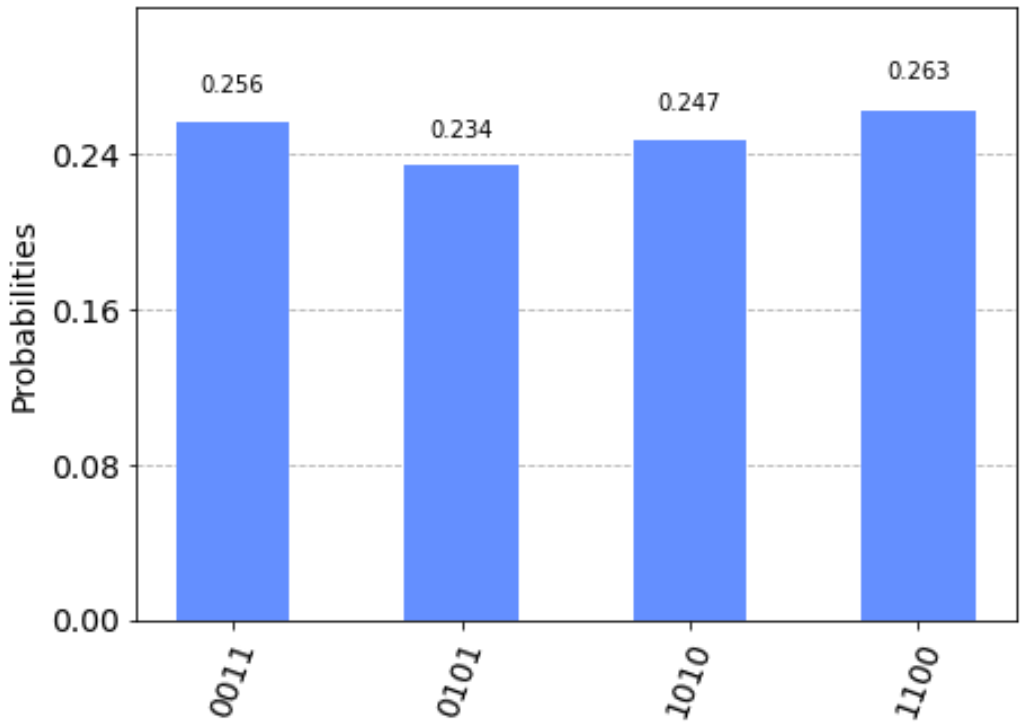

Figure 19 Job results

When comparing these results to the correct results, we can see that the four possible outcomes are indeed equivalent to what the quantum computer had returned.

12 Discussion

The use of python and qiskit was a good choice. Every problem that arose was easily resolved with the help of large community around these development tools.

Matplotlib had an unexpectedly steep learning curve, but after a while, it became clear that it is much better at creating graphs than any other code-less application.

Using IBMQ was not a good experience. The application provided is unreliable, there is no proper documentation nor community to reference when dealing with an issue.

The results confirmed our hypothesis and the process of developing the algorithm provided us with an insight into quantum development. In my opinion, for such a novice and niche technology, tools like qiskit provide a surprisingly user-friendly way to understand an abstract topic like quantum computing.

The experiment provides a new insight into the relationship between the process of designing an algorithm from a practical standpoint and the theoretical field of quantum physics.

It is beyond the scope of this study to analyse where quantum computing can be in the next decades. But the data contributes a clearer understanding of how quantum computing can provide speedup over classical computing even in the early stages of this technology.

Further research is needed to establish the constraints on amount of variable conditioning that can be implemented in quantum algorithms.

13 Conclusion

Going back to the research question, the results confirm that a quantum computer can be a viable option for running time complex algorithms. However, for the general public it will probably be a long time until it becomes an option. This is due to the high development time cost and steep learning curve. For the middle-sized corporations, in my opinion, quantum computing is not a good choice, because of its high likelihood of errors, which is only minimalised by running the program multiple times to filter out the anomalies. Quantum computer is already a viable choice for financial analysis, where datasets can reach the size of petabytes. Quantum computing also contributes to society in the form of microbiological, physics and even chemistry research previously impossible on classical computers. These are all fields that benefit from the uncertain behaviour of quantum computers, because it helps them simulate real world systems from forming new molecules to discovering structures of new highly specialised materials. Simulating such systems on classical computers is slow for the number of variables that need to be accounted for.

When analysing the outcome of our algorithm from a theoretical standpoint, a quantum computing technology was able to run a program in a smaller time complexity, than will ever be possible on a classical computer.

We were also able to contribute to the community by developing an algorithm that is usable and helpful. This algorithm will be published on GitHub under the standard MIT opensource license allowing anyone to use it, modify it and share it freely.

As for developing for quantum systems, the experience was nearly not as bad as we had expected it to be when we had started this project. After learning all the prerequisite mathematics and physics, the programming methods and practices mostly did not differ.

References

Akanksha Singhal and Chatterjee, A. (2018) ‘Grover’s Algorithm’. doi:10.13140/RG.2.2.30860.95366.

Backtracking | Introduction - GeeksforGeeks (2018). Available at: https://www.geeksforgeeks.org/backtracking-introduction/ (Accessed: 9 March 2022).

Bektas, M. (2022) *Project Jupyter*. Available at: https://jupyter.org (Accessed: 21 April 2022).

Big O Notation Explained with Examples (2020) *freeCodeCamp.org*. Available at: https://www.freecodecamp.org/news/big-o-notation-explained-with-examples/ (Accessed: 8 March 2022).

‘Binary search algorithm’ (2022) *Wikipedia*. Available at: https://en.wikipedia.org/w/index.php?title=Binary_search_algorithm&oldid=1074688665 (Accessed: 8 March 2022).

‘Boolean satisfiability problem’ (2022) *Wikipedia*. Available at: https://en.wikipedia.org/w/index.php?title=Boolean_satisfiability_problem&oldid=1075904932 (Accessed: 14 March 2022).

Bradben (2022) *Linear algebra for quantum computing - Azure Quantum*. Available at: https://docs.microsoft.com/en-us/azure/quantum/overview-algebra-for-quantum-computing (Accessed: 26 October 2021).

Cotrini, C. (2019) ‘An introduction to quantum computing’, p. 14.

Forcer, T.M. *et al.* (2002) ‘Superposition, entanglement and quantum computation’, *Quantum Information and Computation*, 2(2), pp. 97–116. doi:10.26421/QIC2.2-1.

Gisin, N. (2019) ‘Entanglement 25 years after Quantum Teleportation: testing joint measurements in quantum networks’, *Entropy*, 21(3), p. 325. doi:10.3390/e21030325.

GitHub: Where the world builds software · GitHub (2022). Available at: https://github.com/ (Accessed: 21 April 2022).

Hagouel, P.I. and Karafyllidis, I.G. (2012) ‘Quantum computers: Registers, gates and algorithms’, in *2012 28th International Conference on Microelectronics Proceedings*. *2012 28th International Conference on Microelectronics (MIEL 2012)*, Nis: IEEE, pp. 15–21. doi:10.1109/MIEL.2012.6222789.

Hilbert spaces | Quantiki (2015). Available at: https://www.quantiki.org/wiki/hilbert-spaces (Accessed: 26 October 2021).

Hughes, C. *et al.* (2021) *Quantum Computing for the Quantum Curious*. Cham: Springer International Publishing. doi:10.1007/978-3-030-61601-4.

Hunter, J., Darren, D. and Firing, E. (2022) *Matplotlib documentation — Matplotlib 3.5.1 documentation*. Available at: https://matplotlib.org/stable/index.html (Accessed: 21 April 2022).

IBM Quantum Computing (2022). Available at: https://www.ibm.com/quantum-computing (Accessed: 21 April 2022).

Katabira, J. (2021) *GROVER'S ALGORITHM IN QUANTUM COMPUTING AND ITS APPLICATIONS*.

Learning, F.Z.| Q.M. (2021) *A Practical Guide To Quantum Amplitude Amplification*, *Medium*. Available at: https://towardsdatascience.com/a-practical-guide-to-quantum-amplitude-amplification-dbcbe467044a (Accessed: 20 March 2022).

Multiple Qubits and Entangled States (2022). Available at: https://community.qiskit.org/textbook/ch-gates/multiple-qubits-entangled-states.html (Accessed: 11 March 2022).

Pal, A. *et al.* (2020) 'Solving Sudoku game using a hybrid classical-quantum algorithm', *EPL (Europhysics Letters)*, 128(4), p. 40007. doi:10.1209/0295-5075/128/40007.

Portugal, R. (2013) *Quantum Walks and Search Algorithms*. New York, NY: Springer New York. doi:10.1007/978-1-4614-6336-8.

Qiskit (2022). Available at: https://qiskit.org/ (Accessed: 21 April 2022).

'Qubit' (2021) *Wikipedia*. Available at: https://en.wikipedia.org/w/index.php?title=Qubit&oldid=1055034395 (Accessed: 17 November 2021).

Shafi (2020) 'Quantum Computing — required Linear Algebra', *Analytics Vidhya*, 14 September. Available at: https://medium.com/analytics-vidhya/quantum-computing-required-linear-algebra-f11c6b2a766f (Accessed: 26 October 2021).

Squarishbracket, ~ (2018) *Building the diffusion operator*, *Rising Entropy*. Available at: https://risingentropy.com/building-the-diffusion-operator/ (Accessed: 20 March 2022).

Sudoku Solver (2020). Available at: https://afteracademy.com/blog/sudoku-solver (Accessed: 9 March 2022).

Tensor product | Quantiki (2015). Available at: https://www.quantiki.org/wiki/tensor-product (Accessed: 30 October 2021).

Welcome to Python.org (2022). Available at: https://www.python.org/ (Accessed: 21 April 2022).