Investigation and Implementation of Quantum Algorithms

Georgakopoulos Chris

A thesis submitted in partial fulfilment to the requirements of Digital Systems Department, University of Thessaly,
for the degree of Msc

Jan-2023

Declaration

I hereby declare that the work presented in this thesis has not been submitted for any other degree or professional qualification, and that it is the result of my own independent work.

Georgakopoulos Chris

14/01/2023

Date

Abstract

Of all the implications of quantum theory, the most technologically interesting is potentially the quantum computer

The development of computers stronger than today's requires either the improvement of existing technology or the development of computers, the operating principles of which will be fundamentally different from the laws of classical physics.

A promising field of research is quantum computing, i.e. the development of devices and algorithms based on the principles of quantum theory or quantum mechanics, i.e. the theory that studies the behavior of matter at the subatomic level.

By utilizing quantum properties, such as the superposition of states, the wave-particle dualism of matter, quantum entanglement, etc., we can achieve unprecedented computing power.

In this thesis, we will be concerned with the investigation and implementation of quantum algorithms.

The principles of quantum mechanics and especially superposition of states and quantum entanglement allow the design of algorithms radically different from classical algorithms.

In this thesis we will be concerned with 3 of the most well-known quantum algorithms: Grover's, Simon's, Bernstein's.

We investigate the principles of quantum mechanics and how they can be utilized in the theoretical construction and implementation of algorithms

We run simulations in qiskit and implemented the algorithms in real quantum devices in IBM Quantum Experience cloud platform.

The results we reached indicate, on the one hand, the power and potential of quantum computation and, on the other hand, the limitations and problems associated with quantum computing, such as noise handling and untrustworthiness of machines.

Publications associated with this research

The core of this thesis is the investigation and implementation of Grover's, Simon's, and Bernstein-Vazirani quantum algorithms.

Consequently, the following original papers are cited:

Grover

Grover K. Lov (1996). A fast quantum mechanical algorithm for database search. v3

Available: https://arxiv.org/pdf/quant-ph/9605043.pdf

Simon

Simon, R. Daniel (1997). On the Power of Quantum Computation

Available: https://epubs.siam.org/doi/epdf/10.1137/S0097539796298637

Bernstein-Vazirani

Bernstein Ethan, Vazirani Umesh (1993). Quantum Complexity Theory

Available: https://dl.acm.org/doi/pdf/10.1145/167088.167097

Acknowledgements

I would like to thank the following people for their role and support that lead to the completion of this thesis.

My supervisor, Prof Ilias Savvas -Department of Digital Systems, University of Thessaly- who proposed this topic to me and willingly answered my questions and discussed my suggestions, giving constructive feedback until the completion of this thesis.

My lead and project coordinator for their tolerance and support during the writing of this thesis, as well as, my colleagues, who with their experience helped me to grow as a software engineer.

My family for their caring encouraging.

Last but not least, my friends and my teammates, without whom this thesis would be completed much earlier!

Table of contents

List of figures

List of tables

Introduction

Background and scope

The purpose of this thesis is to explore the basic concepts of Quantum Computing and how these concepts apply in the investigation and execution of algorithms specifically designed to run on Quantum Computers – known as Quantum Algorithms.

Quantum Algorithms outperform classical algorithms, finding the solutions to the problems they are designed to apply to, with the use of fewer resources – that being time, iterations etc – or solve problems that are unsolvable with the use of classical computers.

The algorithms that the writer studied theoretically and executed are: the Grover Algorithm, the Simon's algorithm and the Bernstein-Vazirani algorithm.

Through the study and the implementation of these algorithms, it is shown the efficiency and superiority of these algorithms compared to classical ones.

Thesis structure

While authoring this thesis, I constantly kept in mind the diversity of possible readers as quantum computing is a multidisciplinary field, consisting of computer scientists, programmers, physicists, materials scientists etc.

It seems proper to me to explain, to a certain extent, the ideas these algorithms are based upon, starting from quantum physics, continuing to quantum computing and quantum circuits and their building components -gates and qubits- before moving to investigate and implement the algorithms themselves. It that way, the mental flow towards the algorithms is smoother compared to just mention the algorithms out of the blue.

Consequently, this thesis structure is the following:
Chapter 2: Quantum physics – Ideas and mathematical formulation
This part includes the basic ideas and the mathematical formulation of quantum theory. It can be skipped from those with a background in quantum physics or those who are purely interested in quantum computing and are familiar with the mathematical formulation of quantum theory
Chapter 3: Quantum computing – Circuits and components
This part delves into the basic components of quantum computing: quantum circuits and their building components: quantum bits(qubits) and quantum gates
LIMITATIONS???
Chapter 4: My Algorithms –
Here the algorithms are explained. Considering the problem they solve, the steps and the ideas used, the classical way of confronting the problem and the reasons the quantum algorithms dominate over their classical counterparts
Chapter 5: Implementation
Details about the implementation of the algorithms in IBM quantum computers.
Chapter 6: Results
Presentation of the results
Chapter 7: Discussion

This chapter consists of the discussion and analysis of the results I got from the implementation of the algorithms (Chapter 5)

Chapter 8: Conclusion

Final conclusions and thoughts

Chapter 2: Quantum theory – Ideas and mathematical formulation

Introduction

Until the last decades of 19th century and first decades of the 20th century the framework of classic physics was enough to explain almost every known phenomenon. Physics was considered almost a dead or complete science with every aspect of it properly explored and only a matter of time before the remaining details would be specified.

However, reality had a different opinion and the two experiments that now are considered benchmarks in the evolution of physics would take place and give birth to what is known as modern physics.

The experiment of Michelson-Morley would give Einstein the trigger to work on his theory of relativity.

Quantum physics on the other hand was shaped collectively by a group of the most brilliant minds of the era who worked either together or separately.

The black body problem and the resulting ultraviolet catastrophe was solved by Max Planck and suggested the existence of quanta. The photoelectric effect and the double-slit experiment helped us understand the light wave-particle duality.

Probably the most influential experiment that shaped our initial research towards the formulation of quantum mechanics was the double-slit experiment. In the next section, we present a simplified version of that experiment that uses an experiment apparatus known as Mach-Zehnder Interferometer (Mosca, 1999)

Ideas and the necessity for quantum physics

The Mach-Zehnder experiment

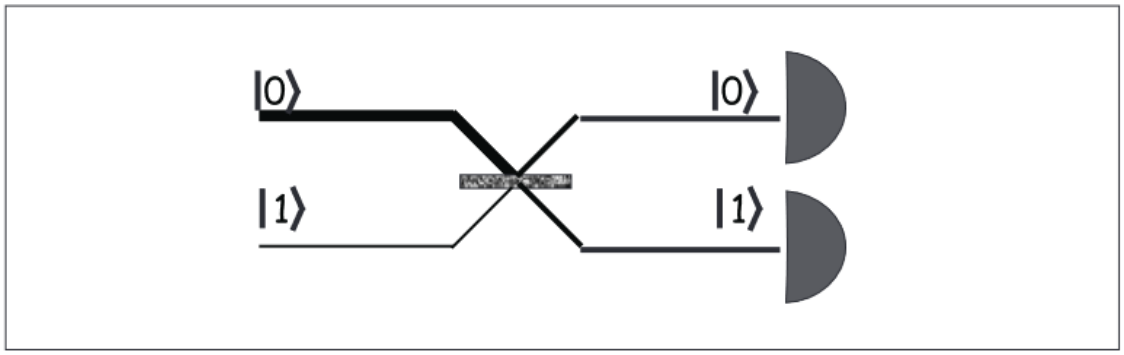

Figure 1. One-slot beam-splitter (Mosca, 1999)

Let us assume we have two beams of photons. We denote that the photons of the one beam are in state 0 and the photons of the other beam in state 1 and two detectors 0 and 1. In the path between the beam generators and the detectors, we place a beam-splitter. The expected outcome is that half the times a photon passes through the beam-splitter it will travel to be detected by 0 and half by 1. Thus, it is an outcome with equal 50% probability - a coin flip outcome.

However, the result is quite different when we construct the Mach-Zehnder Interferometer:

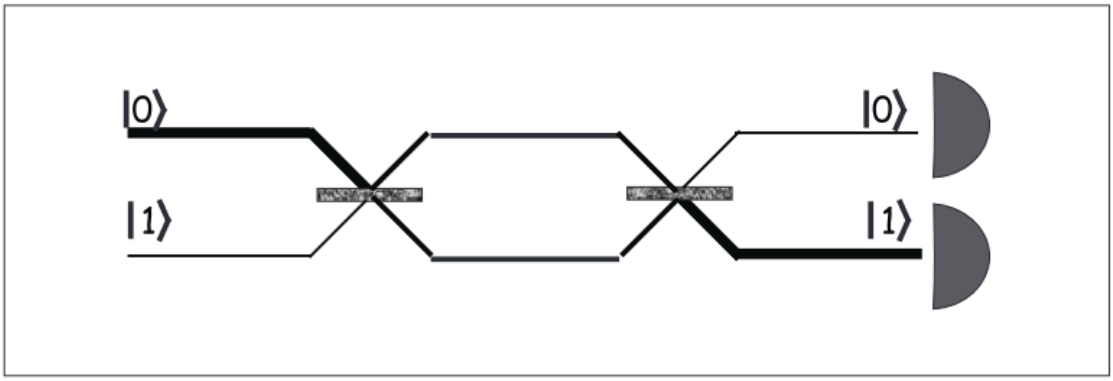

Figure 2. Two single-slot beam-splitters (Mach-Zehnder experiment) (Mosca, 1999)

We expect that if a photon takes a random path after the encounter with a beam splitter, with enough repetitions, we will measure 50% of the photons in detector 0 and 50% in the detector 1.

However, this is not the case.

All the photons will be measured in the detector 1.

Furthermore, if we place a phase-sifter of π in the route 0 of the photon, we observe that the measured outcome is always 0. Thus, it is always possible to detect the presence of the phase shifter.

$$\Psi = sin(\frac{\varphi_0 + \varphi_1}{2})|0> + cos(\frac{\varphi_0 + \varphi_1}{2})|1>$$

$$P_0 = sin^2\left(\frac{\varphi_0 + \varphi_1}{2}\right)$$

$$P_1 = cos^2(\frac{\varphi_0 + \varphi_1}{2})$$

This result is inexplainable in the context of classical physics.

However, it is perfectly explainable, if we accept the existence of superposition that quantum physics suggests. After each split the system comes into a superposition of states

Each beam-splitter maps each state to a new state of superposition:

$$|0> \rightarrow \frac{i}{\sqrt{2}}|0> + \frac{1}{\sqrt{2}}|1>$$

$$|1> \rightarrow \frac{1}{\sqrt{2}}|0> + \frac{i}{\sqrt{2}}|1>$$

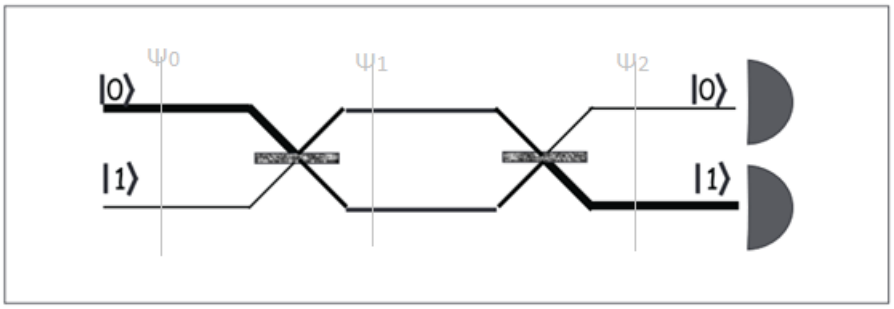

Figure 3. The evolution of Ψ in Mach-Zehnder experiment

$$\Psi 0 = |0>$$

$$\Psi 1 = \frac{i}{\sqrt{2}}|0> + \frac{1}{\sqrt{2}}|1>$$

$$\Psi_2 = \frac{i}{\sqrt{2}}(\frac{i}{\sqrt{2}}|0> + \frac{1}{\sqrt{2}}|1>) + \frac{1}{\sqrt{2}}(\frac{1}{\sqrt{2}}|0> + \frac{i}{\sqrt{2}}|1>)$$

$$\Psi_2 = (\frac{-1}{2}|0> + \frac{i}{2}|1>) + (\frac{1}{2}|0> + \frac{i}{2}|1>)$$

$$\Psi_2 = i|1>$$

Consequently,

$$P_2 = 1$$

Superposition

Superposition is one of the main ideas in quantum mechanics. Superposition and entanglement are what gives quantum computation its advantage over classical computation.

Superposition is a property of every quantum system to get into every possible state when it is not observed.

It is an outcome of Schrodinger's equation, which is linear, thus, any linear combination of its solutions is also a solution.

For example, let's assume a system with possible measured states, |0> and |1>. When it is not observed, it can be in a superposition of these two linear independent basic states.

If |0> is the initial state and the system with the application of suitable operations gets in superposition its initial state is mapped to a linear combination of all possible states:

$$|0> \rightarrow a|0> + b|1>, a, b \in C$$

By observation, we do not necessarily refer to a human being or sensor observing the quantum system, but to interactions with external factors, in general.

For, example, in a system of qubits, if a qubit interacts with an air molecule, it loses the superposition of states it was in, and, consequently, affects the computation accordingly, even though this interaction was not in the intents of the human observer.

This view is in accordance with the probabilistic interpretation of quantum theory. Just as regular waves can interact in classical physics, probability waves can also interact - either constructively or destructively- with each other, affecting the probability to get a specific outcome when the system is measured.

Probabilistic interpretation

Entanglement

Entanglement is a property of quantum systems that allows them to interact and bind their states together. The state of one system affects the state of the other.

In quantum computation, we entangle states with Controlled gates. For example, CNOT entangles the states of target and control qubits. The state of target qubit is bound to that of control qubit. However, let's not forget we are dealing with quantum entities. Contrary to classical gates, where target and control are always distinctive, in quantum gates this distinction is not always possible. Absurd as quantum physics is, some of quantum gates have direct equivalents in classical gates, while others have their own properties, that are not possible to be found outside of the quantum context. For example, the Controlled-Z gate, where the states of control and target qubits are bound, but it is the control qubit that changes state, based on target's value!

Mathematically speaking, an entangles state is state that can not be written as a product of each component's partial state, but only as a linear combination of those.

Fe

$$a(|0> + |1>)\ b|1>, a, b \in C$$

is an unentangled state, while

$$a|0>\ |0> +\ b|1>|1>, a, b \in C$$

is an entangled state, as it cannot be written as a product of the basic states |0> and |1>

Of theoretical significance is the Bell states that are all entangled

Theorem of no cloning

Phase - local vs global

Since the theoretical and mathematical interpretation of a quantum system is

The most prominent interpretation of quantum mechanics is the probabilistic interpretation. The mathematical formulation of quantum mechanics Is built around Schrodinger equation, which describes the state of a quantum system in terms of wave theory with imaginary coefficients.

The idea of phase is crucial in the description of waves. In quantum systems of two or more members, the idea of phase is useful as it can help us describe the constructive or destructive interference of wavefunctions of each member.

For a system of 2 qubits in possible states 0 and 1. Let's bring it in superposition.

$$\Psi = \Psi_1 * \Psi_2 = a_0|00> + a_1|01> a_2|10> a_3|11>$$

Notice that like wave theory

$$\Psi = -\Psi$$

Meaning that any global phase can be ignored.

Phase kickback

Like, classical waves what affects the system state is the difference of phase between the interfering waves or local phase. This is the same with quantum waves

Transformations

$$an \rightarrow -an$$

Leave the probability of each possible state unchanged, but can affect the outcome of a system if we apply specific transformations.

Let the system in state

$$\Psi_1 = \frac{1}{\sqrt{2}}|00> + \frac{1}{\sqrt{2}}|01> + \frac{1}{\sqrt{2}}|10> + \frac{1}{\sqrt{2}}|11>$$

$$\Phi_1 = \frac{1}{\sqrt{2}}|00> - \frac{1}{\sqrt{2}}|01> + \frac{1}{\sqrt{2}}|10> - \frac{1}{\sqrt{2}}|11>$$

If we observe the system the probability of each outcome is the same :

$$\Psi_1 = \frac{1}{\sqrt{2}}|00> + \frac{1}{\sqrt{2}}|01> + \frac{1}{\sqrt{2}}|10> + \frac{1}{\sqrt{2}}|11>$$

To edit the above line

However, if we apply a CNOT gate in Psi 1 and F1, we get

$$CNOT|\Psi_1 = \Psi_2 = \frac{1}{\sqrt{2}}|00> + \frac{1}{\sqrt{2}}|01> + \frac{1}{\sqrt{2}}|10> + \frac{1}{\sqrt{2}}|11>$$

$$CNOT|\Phi_1 = \Phi_2 = \frac{1}{\sqrt{2}}|00> - \frac{1}{\sqrt{2}}|01> + \frac{1}{\sqrt{2}}|11> - \frac{1}{\sqrt{2}}|10>$$

Let

$$|+>= \frac{1}{\sqrt{2}}|0> + \frac{1}{\sqrt{2}}|1>$$

$$|->= \frac{1}{\sqrt{2}}|0> - \frac{1}{\sqrt{2}}|1>$$

Then,

$$\Psi_2 = |++>$$

$$\Phi_2 = |+->$$

Now, if we apply a special transformation to the system and observe it, we have:

$$\Psi_3 = |00>$$

$$\Phi_3 = |01>$$

We note, that while the only difference between phi and psi were the phases of some states when in superposition the final state is different. The phase can be kicked to the result. This trick. To kick the phase in specific qubits, is a trick that is widely used in quantum algorithms. It is used in Bernstein-Vazirani and Grover algorithms studied ion this thesis.

Introduce each new section to the reader, then use more sub-sections as required.

THINGS/ SECTIONS TO WRITE IN THE MATH BACKGROUND

- States representations(, Hermitian and unitary gates)
- Theorem of no cloning

Mathematical formulation

Dirac notation

Dirac notation is a convenient way to represent matrices or vectors.

In quantum mechanics the state of a quantum system is given by the equation

$$|\psi> = \sum_{i=0}^{v} ai|\psi i>$$

Where ψi is the state vector

It is $|\psi> = \begin{bmatrix} a0 \\ a1 \\ \vdots \\ an \end{bmatrix}$

And the adjoint of this vector is

$$<\psi| \; = \; [a*0 \quad a*1 \quad \cdots \quad a*n]$$

Where * denotes complex conjugation

Consequently, the inner product of two vectors is

$$<\varphi|\psi> = [\varphi 0 \quad \varphi 1 \quad \cdots \quad \varphi n] \begin{bmatrix} \psi 0 \\ \psi 1 \\ \vdots \\ \psi n \end{bmatrix} = \sum_{i=0}^{n} \varphi i * \psi i$$

The outer product of two vectors is

$$|\psi><\varphi \; = \begin{bmatrix} \psi 0 \\ \psi 1 \\ \vdots \\ \psi n \end{bmatrix} [\varphi 0 \quad \varphi 1 \quad \cdots \quad \varphi n]$$

Dirac notation can be used for describing infinite-dimensional vectors in a continuous Hilbert space. However, as in quantum computing, only finite-dimensional Hilbert spaces are used, this will not concern us.

Quantum states can be described as vectors living in a Hilbert space. In n the number of qubits in a system, this Hilbert space has dimension d = n and in general imaginary

Generally, in quantum theory Hilbert space in of infinite dimension and the vectors have imaginary coefficients.

In practical uses, depending on the system's degrees of freedom and the precision of approximation we can afford, only a finite-dimensional approximation is used.

For example, in atom, the state of the electrons can be described by their spin (that takes values up or down) and the position of the electrons. Position can be anywhere in space, so the Hilbert space where the atom lives in is continuous and therefore infinite-dimensional. In quantum computing, the situation is simpler. Each qubit can take only two discrete values (0 or 1). To efficiently describe a system of qubits, we need a space of 2n dimension.

If we choose to describe the vectors with matrices, things fast get out of hand.

A suitable option sis the Dirac notation

Axioms of quantum mechanics

Quantum mechanics is a mathematical model for describing quantum systems. It can be summarized by its 5 axioms (Preskill, 2015)

Axiom 1

A state is a complete description of a physical system. A state is a ray in a Hilbert space.

What is a Hilbert space?

1. It is a vector space over the complex numbers C. Vectors will be denoted $|\psi\rangle$ (Dirac's ket notation).

2. It has an inner product $\langle\psi|\phi\rangle$ that maps an ordered pair of vectors to C, and that has the properties:

    a. Positivity: $\langle\psi|\psi\rangle > 0$, for $|\psi\rangle \neq 0$.

    b. Linearity: $\langle\phi|(a|\psi_1\rangle + b|\psi_2\rangle) = a\langle\phi|\psi_1\rangle + b\langle\phi|\psi_2\rangle$.

    c. Skew symmetry: $\langle\phi|\psi\rangle = \langle\psi|\phi\rangle*$.

    (The $*$ denotes complex conjugation.)

3. It is complete in the norm $||\psi|| = \langle\psi|\psi\rangle^{1/2.}$

    (Completeness is an important proviso in infinite-dimensional function spaces, since it ensures the convergence of certain eigenfunction expansions. But mostly we will be content to work with finite-dimensional inner-product spaces.)

What is a ray? It is an equivalence class of vectors that differ by multiplication by a nonzero complex scalar. For any nonzero ray, we can by convention choose a representative of the class, denoted $|\psi>$, that has unit norm:

$$< \psi|\psi > = 1$$

Thus states correspond to normalized vectors, and the overall phase of the vector has no physical significance: $|\psi>$ and $e^{i\alpha}|\psi>$ describe the same state, where $|e^{i\alpha}| = 1$.
Since every ray corresponds to a possible state, given two states $|\phi>$, $|\psi>$, another state can be constructed as the linear superposition of the two, $a|\phi> + b|\psi>$. The relative phase in this superposition is physically significant; we identify $a|\phi> + b|\phi>$ with $e^{i\alpha}(a|\phi> + b|\psi>)$ but not with $a|\phi> + e^{i\alpha}b|\psi>$.
We use the notation $<\psi|$ (Dirac's bra notation) for a linear function (a dual vector) that takes vectors to complex numbers, defined by $|\phi> \rightarrow <\psi|\phi>$

Axiom 2

Observables

An observable is a property of a physical system that in principle can be measured. In quantum mechanics, an observable is a self-adjoint operator. An operator is a linear map taking vectors to vectors,

$$A : |\psi > \rightarrow A|\psi >, A\,(a|\psi > + b|\varphi >) = aA|\psi > + bA|\varphi >$$

The adjoint $A^\dagger$ of the operator A is defined by:

$$<\phi|A\psi> = <A^\dagger\phi|\psi>$$

for all vectors $|\phi>$, $|\psi>$ (where here $A|\psi>$ has been denoted as $|A\psi>$).

A is self-adjoint if $A = A^\dagger$, or in other words, if $\langle\phi|A|\psi\rangle = \langle\psi|A|\phi\rangle^*$ for all vectors $|\phi\rangle$ and $|\psi\rangle$. If A and B are self-adjoint, then so is A+B (because $(A + B)^\dagger = A^\dagger + B^\dagger$), but $(AB)^\dagger = B^\dagger A^\dagger$, so that AB is self-adjoint only if A and B commute.
Note that AB + BA and i(AB − BA) are always self-adjoint if A and B are.
A self-adjoint operator in a Hilbert space H has a spectral representation – its eigenstates form a complete orthonormal basis in H. We can express a self-adjoint operator A as:
$A = \sum a_n E_n$
Here each a_n is an eigenvalue of A, and E_n is the corresponding orthogonal projection onto the space of eigenvectors with eigenvalue an. The
En's satisfy
$E_n E_m = \delta_{n,m} E_n$.
$E^\dagger_n = E_n$.
The orthogonal projector onto the one-dimensional space spanned by the vector $|\psi\rangle$ may be expressed as $|\psi\rangle\langle\psi|$, where $\langle\psi|$ is the bra that annihilates vectors orthogonal to $|\psi\rangle$. Therefore, an alternative notation for the
spectral representation of A is

$$A = \sum |n\rangle a_n \langle n|$$

where $\{|n\rangle\}$ is the orthonormal basis of eigenstates of A, with $A|n\rangle = a_n |n\rangle$.
(For unbounded operators in an infinite-dimensional space, the definition of self-adjoint and the statement of the spectral theorem are more
subtle, but this will not concern us.)

Axiom 3

Measurement

A measurement is a process in which information about the state of a physical system is acquired by an observer. In quantum mechanics, the measurement of an observable A prepares an eigenstate of A, and the observer learns the value of the corresponding eigenvalue. If the quantum state just prior to the measurement is $|\psi\rangle$, then the outcome a_n is obtained with a priori probability

$$\mathrm{Prob(an)} = \|\mathrm{En}|\psi\rangle\|^2 = \langle\psi|\mathrm{E_n}|\psi\rangle$$

if the outcome a_n is attained, then the (normalized) quantum state just after the measurement is

$$\frac{\mathrm{E_n}|\psi\rangle}{\|\mathrm{E_n}|\psi\rangle\|}$$

If the measurement is immediately repeated, then according to this rule
the same outcome is obtained again, with probability one. If many identically prepared systems are measured, each described by the state $|\psi\rangle$,
then the expectation value of the outcomes is

$$\langle a \rangle \equiv \sum a_n Prob(a_n) = \sum a_n \langle \psi | E_n | \psi \rangle = \langle \psi | A | \psi \rangle. \quad (2.9)$$

Axiom 4

Dynamics

Dynamics describes how a state evolves over time. In quantum mechanics, the time evolution of a closed system is described by a unitary operator.

In the Schrodinger picture of dynamics, if the initial state at time t is $|\psi(t\rangle$, then the final state $|\psi(t')\rangle$ at time t' can be expressed as

$$|\psi(t')\rangle = U(t', t)|\psi(t)\rangle|\psi(t)\rangle,$$

where U (t', t) is the unitary time evolution operator. Infinitesimal time evolution is governed by the Schrodinger equation

$$\frac{d}{dt}(|\psi(t)\rangle) = -iH(t)(|\psi(t)\rangle)$$

where H(t) is a self-adjoint operator, called the Hamiltonian of the system.

To first order in the infinitesimal quantity dt, the Schrodinger equation can be expressed as

$$|\psi(t + dt)\rangle = (I - iH(t)dt)|\psi(t)\rangle$$

Thus, the operator U (t + dt, t) ≡ I − iH(t)dt is unitary; because H is self-adjoint it satisfies U †U = 1 to linear order in dt. Since a product of unitary operators is unitary, time evolution governed by the Schrodinger equation over a finite interval is also unitary. In the case where H is time-independent we may write U (t', t) = e−i(t'−t)H .

Our final axiom relates the description of a composite quantum system AB to the description of its component parts A and B.

Axiom 5

Composite Systems

If the Hilbert space of system A is H_A and the Hilbert space of system B is H_B , then the Hilbert space of the composite systems AB is the tensor product $H_A \otimes H_B$. If system A is prepared in the state $|\psi\rangle_A$ and system B is prepared in the state $|\phi\rangle_B$, then the composite system's state is the product $|\psi\rangle_A \otimes |\phi\rangle_B$.

What is a tensor product of Hilbert spaces? If $\{|i\rangle A\}$ denotes an orthonormal basis for HA and $\{|\mu\rangle_B\}$ a basis for H_B , then the states $|i, \mu\rangle_{AB} \equiv |i\rangle_A \otimes |\mu\rangle_B$ are a basis for HA ⊗ HB , where the inner product on $H_A \otimes H_B$ is defined by ${}_{AB}\langle i, \mu | j, \nu\rangle_{AB} = \delta_{ij}\,\delta_{\mu\nu}$.

The tensor product operator $M_A \otimes N_B$ is the operator that applies M_A to system A and N_B to system B. Its action on the orthonormal basis $|i, \mu\rangle_{AB}$ is
$M_A \otimes N_B |i, \mu\rangle_{AB} = M_A|i\rangle_A \otimes N_B |\mu\rangle_B = \sum_{j,\nu} |j, \nu\rangle_{AB} (M_A)_{ji} (N_B)_{\nu\mu}$.(2.14)
An operator that acts trivially on system B can be denoted $M_A \otimes I_B$, where IB is the identity on HB , and an operator that acts trivially on system A can be denoted $I_A \otimes N_B$.

Representation of quantum states using the Bloch sphere

In quantum computing, the simplest possible system is a system consisting of 1 qubit. This system can be described using the Dirac formulation.

We describe the 1-qubit system first and then we generalize to multi-qubit system

Bloch sphere for 1-qubit systems

In 1-qubit systems the qubit can be described by a statevector. This statevector represents the probability of finding the qubit in a specific state. The system can be either in state $|0\rangle$, state $|1\rangle$ or when in superposition in an intermediate state.

Using Cartesians, the qubit q can be formulated as

$|q\rangle = a|0\rangle + b|1\rangle$, $a, b \in C$.

This means that while the pure states of a quantum system are 1-D subspaces of a Hilbert space, the 1-qubit system is a Hilbert space of dimension 2

To better understand the properties of this simple, but fundamental system it is a good idea to visualize it. Since the coefficients a, b are imaginary numbers it is impossible to visualize the system using a plane. We need to use 3-D visualization.

A 2-D Hilbert space can be visualized using the Bloch sphere.

Since $a^2 + b^2 = 1$ this Bloch Sphere with be of r=1.

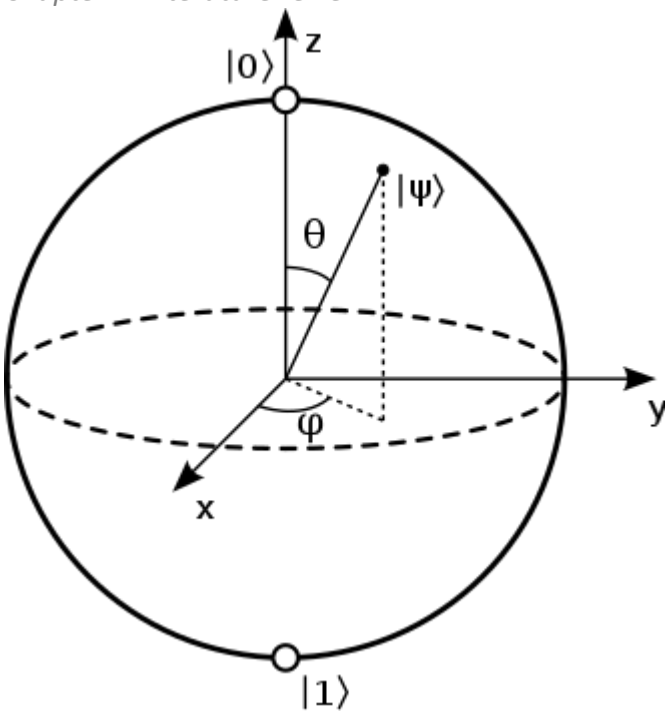

Figure 4. Bloch sphere (Bloch sphere, 2023)

By convention the clear states of a system are represented by the north and the south poles of the Bloch sphere. In applications, the poles can represent the spin-up and spin-down states of an electron, the normal and the excited state of a molecule etc. The points between the poles represent superposition states.

Special attention must be paid to the fact that the Bloch sphere represents a Hilbert space. That means the base vectors -the poles of the Bloch sphere- are Hilbert-orthogonal, even though in the physical space they form an angle of π.

Hilbert orthogonal for a, b means that a*b = 0, where * is their inner product. It is easy to prove that the poles are Hilbert-orthogonal and it is displayed at (Mao, 2023)

The state of random qubit q, can be written as

$|q\rangle = a|0\rangle + b|1\rangle$, $a, b \in C$.

Or using the Bloch sphere as reference:

$$|q\rangle = \cos(\frac{\theta}{2})|0\rangle + e^{i\phi}\sin(\frac{\theta}{2})|1\rangle, \theta, \phi \in R$$

Proof can be found in (Mao, 2023).

An other non-trivial observation is that the points across the equator of the sphere represent points in superposition with equal probability to degenerate in 0 or 1 when measured. This is expected as, in equator θ= π/2, consequently,

$$a^2 = cos^2(\frac{\pi}{4}) = \frac{1}{2}, b^2 = sin^2(\frac{\pi}{4}) = \frac{1}{2}$$

Additional bases in Bloch sphere for 1-qubit systems

As shown in Appendix A 2 opposite points in the Bloch sphere can be used as a basis of the vector space. We conveniently took the vectors that point to the north and south poles as base vectors. In other words, we chose two points across the z-axis. However, there is no constriction at the points we can choose to form a base, as long as they have an angle of π. As we will discuss later, sometimes it is actually more convenient to take as basis vectors across the x- axis or across the y-axis.

In those cases, the base vectors are:

$$|+> = \frac{1}{\sqrt{2}}(|0> + |1>), |-> = \frac{1}{\sqrt{2}}(|0> - |1>)$$

$$|i+> = \frac{1}{\sqrt{2}}(|0> + i|1>), |i-> = \frac{1}{\sqrt{2}}(|0> - i|1>)$$

Bloch sphere for multi-qubit systems

The ideas discussed above are quite easy to be generalized for multi-qubit systems. However, it is of little importance, as even for 2-qubit systems a sphere with more than 3 dimensions is required to depict the system.

For a system with N base vector, the required Bloch hyper-sphere is of dimension d= N^2-1 (Diederik Aerts, 2014)

Henceforth, for a 2-qubit system - N=4 – d = 15.

Consequently, the generalization will not be our concern. The curious reader who is interested in d is directed to (Diederik Aerts, 2014)

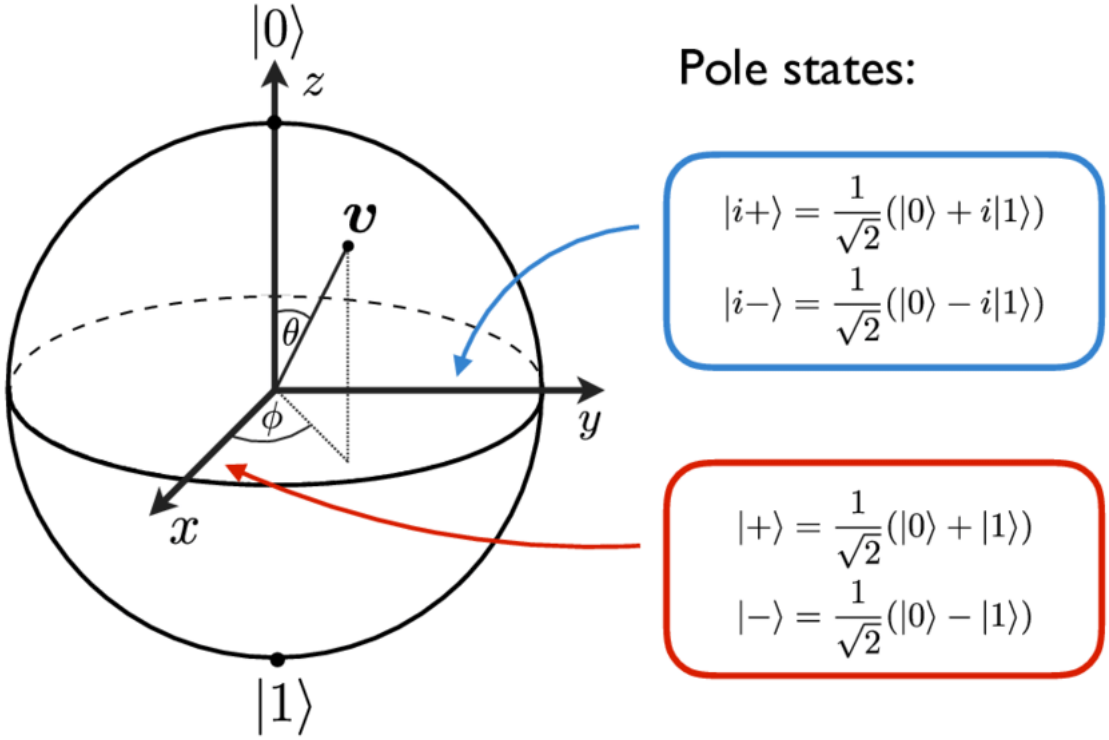

Figure 5. Bloch sphere with base vectors across the x- and y- axes (researchgate, 2023)

Conclusion to this chapter

In this chapter, we reviewed the main ideas of quantum theory and now we possess the necessary theoretical background to proceed to the next chapter.
In the next chapter, we delve into quantum computing by describing quantum circuits and their building parts and we talk about quantum simulators and quantum assembly (QASM) language.

Chapter 3: Quantum computing – Circuits and components

Introduction

In the field of quantum algorithms, a quantum circuit is a model for quantum computation. Every circuit is a set of two -or three, if we want to be strict- elementary components

- Qubits, the carriers of quantum information, the operands of the circuit
- Quantum gates, the operators that act on qubits and transform their state
- Quantum wires can also be considered part of a quantum circuit. Their role is to forward the information from gate to gate and do not affect the state in any way. With their use we can compose complex structures from simple gates. They can be considered as the application of Identity information between gates. This is pretty much anything useful we have to say about them, and we will not discuss much about them.

  Familiarity with the notions involved in the theory of classical circuits is useful but not necessary to follow along this chapter. Everything the reader needs to know is introduced along the way.

  It is of interest that, while quantum circuits, quantum gates and qubits are the equivalents of classical circuits, classical gates and bits respectively, there are differences between each other.

  For example, qubits can be in a mix of states simultaneously, using the quantum property of superposition of states – or simply superposition -, while bits can only be in a pure state of either 0 or 1. Quantum gates are demanded to be reversible. Although they can be equivalent to specific classical gates, there are quantum gates that have no classical analogue.

  Considering the circuits as total, input and output are not physically separated, thus allowing us to express unitary operation carried out by the circuit in a more coherent way (Dan C. Marinescu, 2012)

  Let’s review each component in further detail.

Qubits

A qubit is the quantum equivalent of a bit.

In a classical computer, the bit is the fundamental unit of information stored or processed by a computer. Signals in the form of voltage pass through logical gates and are transmitted through wires in a semiconducting silicon chip. High voltages represent 1's, and low voltages represent 0's; storing a bit is accomplished by charging a capacitor with the appropriate polarity. Each bit can either be a 1 or a 0. The logical gates transform the state of a bit either by changing its state or keeping it the same. In a multibit system, more complex transformations can be performed, like changing the state of a bit based on the state of the previous bit etc

Theoretically, any quantum system that can take two discreet states can be a qubit (various spins on a strand of a polymer, ions in an ion trap, photons in a pure state)

A quantum circuit has a structure that both resembles and differs from a classical circuit.

In a quantum gate array, the qubits stand still, and logical gates (selective radiofrequency pulses, coherent laser radiation, beam-splitters, and phase-shift media) are applied, thus changing the states of individual qubits and modulating their interactions with one another. Readout of the result of a computation is accomplished at the end of the experiment, by performing a measurement (recording radiofrequency emissions, stimulating fluorescence via quantum-jump methods,

interferometrically determining the phase of a photon) on the quantum system.

In other words, as in a classical system, in a quantum system, gates are transformations that apply to a qubit and change its state either by keeping it, changing it or make operations based on the state of other qubits in the system, for example, the CNOT that changes the state of a qubit only if the previous qubit is in state 1.

As in any quantum system, preservation of probability is necessary. That means that the total probability of finding the system in any of its possible states is 1 before the computation, during the computation and after the computation.

What makes the quantum circuits so different compared to classical is the superposition. Superposition is the ability of a quantum system to be in all its possible

states at the same time. That means that if we apply appropriate gates in a 1-qubit system, it can be in a superposition of 0 and 1 at the same time, a 2-qubit system can be in a superposition of 00, 01, 10, 11 at the same time etc. In general, when in superposition a n-qubit system can be in 2^n states at the same time.

To get a measurement, we need to apply special gates that bring the system back to a single state. Of course, random interactions can break the superposition can destroy or reduce the precision of the computation. For example, interactions with air molecules can break the superposition.

This is a very practical architecture for quantum computation: all that is required for implementing quantum computation on a particular physical system is to determine the most easily implemented quantum mechanical operations (the elementary gates) for that system, and then to compile the desired algorithm down to those gates. The computation can be executed in a finite amount of

time and often is readily analysable.

Ancilla qubits (or auxiliary qubits)

Quantum algorithms as we know them would be impossible without the use of ancilla qubits.

Ancilla qubits are the equivalent of ancilla bits in classical reversible computing

Ancilla bits are bits that serve a specific computational purpose and are not part of the input, nor the output. In quantum computation, ancilla qubits are qubits that are used to ensure reversibility. In quantum mechanics, we demand that the operators are unitary -to preserve probability. Consequently, they , also, must be reversible. Since quantum computers must be in accordance with the mathematical formulation of quantum mechanics, we demand that the operations act on them in the form of quantum logical gates are reversible, as well. To ensure such a reversibility, it is sometimes necessary to use an ancilla qubit or even a register of ancilla qubits to ensure reversibility and zero loss of information in the process.

To our knowledge, ancilla qubits are used to achieve entanglement and, even though they are discarded before the end of the computation(although measuring them would

be possible, but usually offers no purpose), they are crucial part of the computation as part of the oracle

Gates

Quantum logical gates or quantum gates are transformations that apply to the state of the system and change it (the identity operation can be considered also as a transformation

Details about quantum gates can be found at: (Christidis, 2016) and (Crooks, 2012)

The figures 6-12 are taken from (Crooks, 2012).

Implementing the gates in real-world circuits can be particularly challenging and require advanced physical implementations. However, from a developer's perspective, the idea of a gate is quite simple. A gate is a matrix that operates to the circuit and transforms its state. The most common gates are those that follow:

Gates that work for 1 or more qubits

Hadamard

This gate is crucial part of every non-trivial quantum circuit. It is the gate that allows a circuit to be set from a solid state to superposition and from superposition back to a solid state. It can be applied to one or more qubits. When Hadamard is applied to a single bit of a multi-qubit system, we consider that Hadamard

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$$

The coefficient in front of the matrix is to ensure that $\Psi^2=1$

In that case we consider that Hadamard is applied to one qubit while the identity operator is applied to the other.

In general, for a n-qubit quantum system the 2^nx2^n Hadamard matrix H_n *has* matrix entries

$$H^n_{x,y} = 2^{-\frac{n}{2}}(-1)^{\vec{x}\vec{y}}$$

where *x, y* , are vectors corresponding to the digits of *x*, *y* written in binary. For example, the
entry in the 3rd row and 5th column of H^7 is
$2^{-7/2}\,(-1)^{0000011\,\times\,0000101} = -2^{7/2}$.

Another way to express the result of Hadamard gate in a qubit a is

$$H\left|a>=\frac{1}{\sqrt{2^n}}\sum_{x\in\{0,1\}^n}(-1)^{a\cdot x}\right|x>$$

Where x is a base vector(or base qubit) on the n-dimensional qubit space

Pauli-I (Identity)

Identity gate I keeps unchanged the qubit-s it is applied upon. Its matrix is a matrix that has every element in its diagonal equal to 1 and every other element equal to 0

$$I^n=\begin{bmatrix}1 & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & 1\end{bmatrix}$$

In applications, we consider that the identity gate is applied when the wavefunction of a qubit is not changed. That means that either no stimulation is applied on the qubit or that is applied, but does not affect its state.

Pauli-X (NOT)

When applied to a qubit in a clear state it changes its state from 0 to 1 and vice versa.

In terms of Bloch sphere, it corresponds to a rotation by π around the x-axis.

In registries of more than one qubit its effect is the same as if we applied an X gate to every individual qubit.

Fe, X|01> = |10>

Its general formula is:

$$X^n=\begin{bmatrix}0 & \cdots & 1\\ \vdots & \ddots & \vdots\\ 1 & \cdots & 0\end{bmatrix}$$

Please note that the Pauli-X acts as a classical NOT only for qubit systems in a clear state. In other words, it maps a point in Bloch spere to its opposite for clear states.

However, a system in arbitrary superposition can not be mapped to its opposite point in general.

The first example that comes in mind is a system that has H gate applied to it.

Ψ = H∑|x>

Since Pauli-X corresponds to rotation across the x-axis, application of Pauli-X will have no effect on the system.

A general proof is provided on Appendix B.

Pauli-Y

Similarly to Pauli-X, we can define a gate that performs a qubit around the Y-axis. That is called Pauli-Y or simply Y. For 1-qubit system:

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$$

and its general formula is

$$Y^n = \begin{bmatrix} 0 & \cdots & \pm i \\ \vdots & \ddots & \vdots \\ \pm i & \cdots & 0 \end{bmatrix}$$

Pauli-Z

Similarly to Y-gate, we can define a gat that performs a qubit around the Y-axis. That is called Pauli-Z or simply Z. For 1-qubit system:

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

and its general formula is

$$Z^n = \begin{bmatrix} \pm 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \pm 1 \end{bmatrix}$$

P(phase gate)

The P-gate performs a rotation of φ around the Z-axis direction. The P-gate (phase gate) is *parametrised,* that is, it needs a number (φ) to tell it exactly what to do. It has the matrix form:

$$P = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\varphi} \end{bmatrix}$$

General Universal gate U

Every 1-qubit gate can be considered as a special case of the Universal gate

$$U(\theta,\phi,\lambda) = \begin{bmatrix} \cos\left(\frac{\theta}{2}\right) & -e^{i\lambda}\sin\left(\frac{\theta}{2}\right) \\ e^{i\varphi}sin\left(\frac{\theta}{2}\right) & e^{i(\varphi\ +\ \lambda)}cos\left(\frac{\theta}{2}\right) \end{bmatrix}$$

However, in circuits we prefer to write the specific gates instead of the universal as it is easier to read

However, to give some examples of some common gates expressed as special cases of the U gate:

$$U(\frac{\pi}{2},0,\pi) = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = H$$

$$U(0,0,\lambda) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\,\lambda} \end{bmatrix} = P$$

Identities

Among the Pauli gates, the following identities are valid:

For brevity, Pauli-X = X, Pauli-X = Y, Pauli-Z = Z,

XY = −YZ = iZ

YZ = −ZY = iX

ZX = −XZ = iY

XYZ = Ii

Please, note that the Pauli gates are Hermitian Gate = Gate†, square to the identity $Gate^2$ = I and that the X, Y, and Z gates anti-commute with each other.

Gates that work for 2 or more qubits

Pauli-I(Identity)

Identity matrix for a 2-qubit system is.

$$\mathrm{I} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

CX (CNOT, controlled-X, CX, Feynman)

CNOT needs two qubits to work, the first one called the control qubit and the second one the target qubit. It maps the value of the target qubit to its counter state, based on the value of the control qubit.

Assuming clear states, if the control qubit is |1>, it flips the target qubit state, else if the control qubit is |0> it does nothing. In other words, if the two qubits are not in superposition, the action of the CNOT gate is equivalent to the classical XOR gate.

Generalizing for circuits in superposition, CNOT has the following effect on the system:

CNOT(a|00> + b|01> + c|10> + d|11>) = a|00> + b|01> + d|10> + c|11>

Its formula is:

$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Its representation in circuits is:

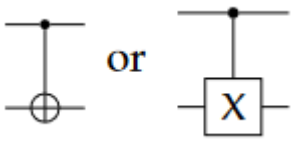

Figure 6. CNOT-gate

In circuits it is possible to use the second qubit as control qubit and he first qubit as target(reversed CNOT). This can be done, by applying CNOT and Hadamards or CNOT and the SWAP gate.

Its representation in circuits is:

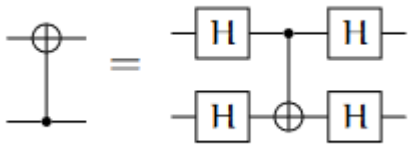

Figure 7. Reversed CNOT

CY

CY is a gate logically equivalent to CX

Its formula is:

$$\mathrm{CY} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -\mathrm{i} \\ 0 & 0 & +\mathrm{i} & 0 \end{bmatrix}$$

Its representation in circuits is:

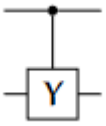

Figure 8. CY-gate

However, in circuits CX and CZ are more commonly used because they only include real coefficients

CZ

CZ is a control gate that has no equivalent in classical gates. It is immune to permutations of the control and target qubits. It is a quantum gate and its reasoning to a circuit maybe is not as obvious as CX. However, as it handles control and target qubit as equivalent it can be useful

Its formula is:

$$\mathrm{CZ} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

Its representation in circuits is:

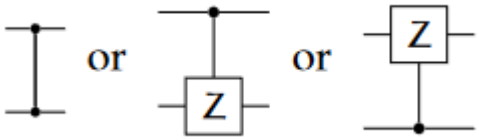

Figure 9. CZ-gate

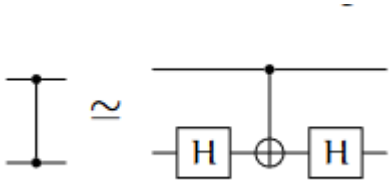

Figure 10. CZ identity

Swap gate

It is useful gate that simplifies circuits in case of consequent CNOTs or reversed CNOTs

Its formula is:

$$\mathrm{SWAP} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Its representation in circuits is:

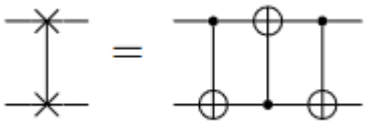

Figure 11. Swap gate

Gates that work for 3 or more qubits

A general 3-qubit gate has $(2^3)^2$=64 parameters and there is not any easy way to visualize it. As a result, 3-qubit gates appear rarely to applications or theory compared to 1- and 2- qubit gates. However, there are a few examples that are of some interest

Toffoli gate (controlled-controlled-not, CCNot)

A 3-qubit gate with two control and one target qubits. Originally studied in the context of reversible classical logic, where 3-bit gates are necessary for universal computation. The target bit flips only if both control bits are one. We often encounter this gate when converting classical logic circuits to quantum circuits.

Its formula is:

$$\mathrm{CCNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Its representation in circuits is:

Figure 12. CCNOT-gate (Toffoli)

Black box

Sometimes, when we study the algorithms theoretically and we are not interested in the implementation specifics yet, we represent a part, or all the transformations, as a black box or oracle. This means that we ignore the inner workings of a set of transformations and are interested only in the result. In other words, an oracle is a setup that transforms input to output, and we have no knowledge of *how* it does so. We only know *what* it does. For example, for the circuit of Bernstein-Vazirani algorithm, we have:

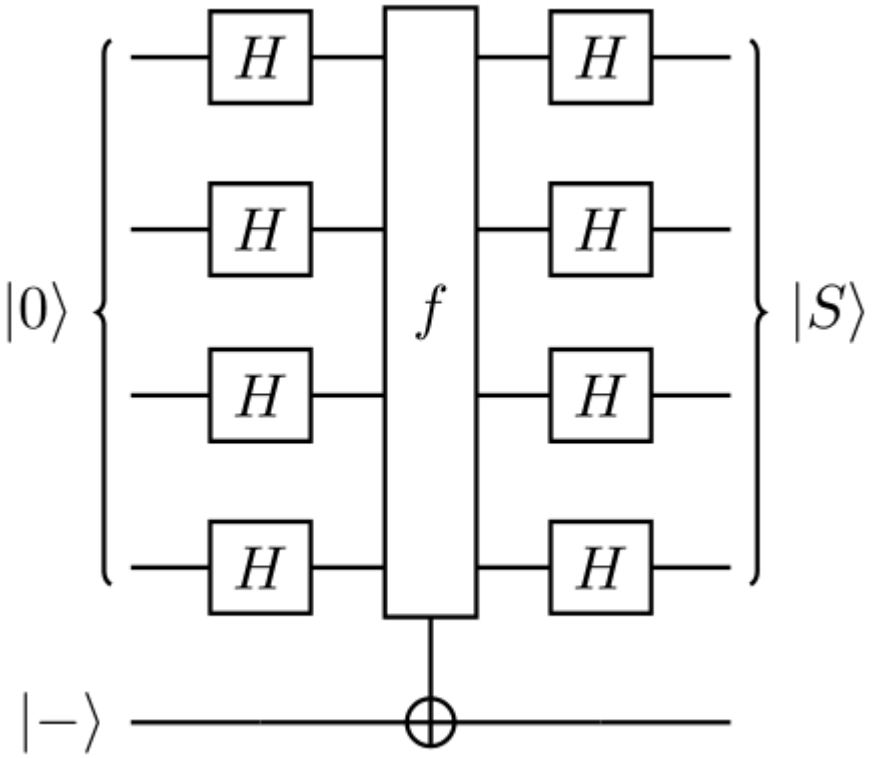

Figure 13. Circuit with oracle function f (qiskit-bv-img, 2023)

Here, with f we denote the oracle. When we decide to implement the algorithm for a specific key, let's say 1101, we can be more specific and use the proper gates:

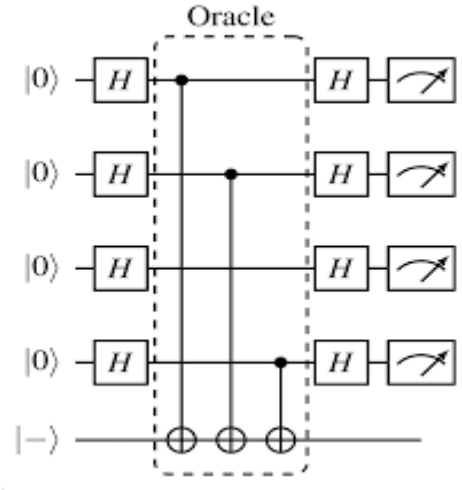

Figure 14. The implementation -of an oracle- with actual gates (circuit-with-oracle, 2023)

Circuits

A quantum circuit has the following structure:

Reading the circuit from left to right, we assume that the operations on the left side take place first, following by those to their right and so on.

In general, the general structure of a quantum circuit is acyclic and consists of the following parts:

1. Input of Initial qubits in a well-known
2. Transformations: Quantum gates that apply to every or only to a subset of the qubits and change the system's state. Sometimes, when we study the algorithms theoretically and we are not interested in the implementation specifics yet, we represent a part of all the transformations as a black box or oracle.
3. Output of classically controlled gates

To demonstrate in a picture the structure of a quantum circuit and further explain its parts, we use the circuit that implements the quantum teleportation algorithm.

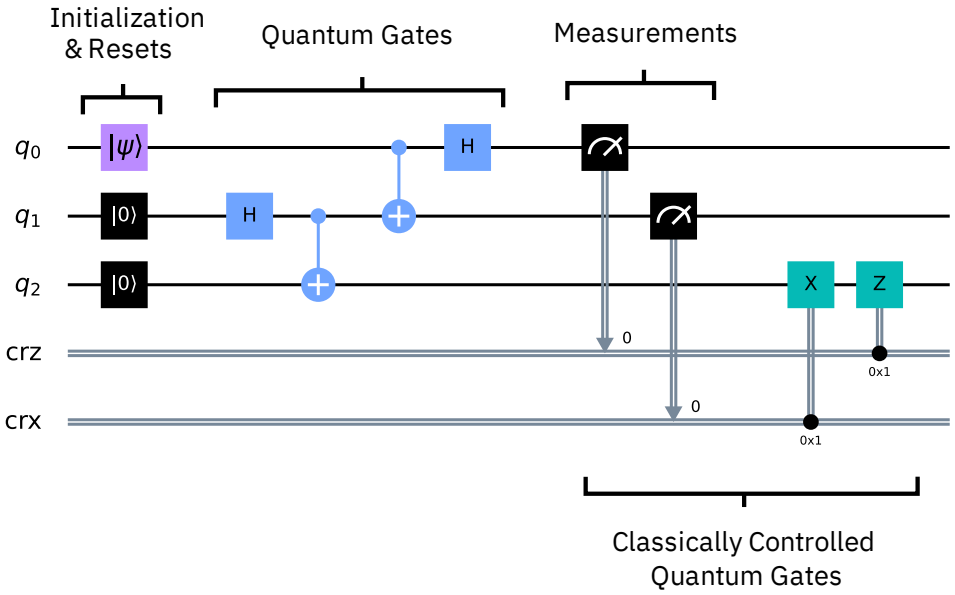

Figure 15. Structure of a quantum circuit (qiskit-general-circuit-img, 2023)

The quantum circuit uses three qubits and two classical bits. There are four main components in this quantum circuit (qiskit-general-circuit, 2023).

Initialization and reset

First, we need to start our quantum computation with a well-defined quantum state. This is achieved using the initialization and reset operations. The resets can be performed by a combination of single-qubit gates and concurrent real-time classical computation that monitors whether we have successfully created the desired state through measurements. The initialization of q_0 into a desired state $|\psi\rangle$ can then follow by applying single-qubit gates.

Quantum gates

Second, we apply a sequence of quantum gates that manipulate the three qubits as required by the teleportation algorithm. It is worth noticing that bringing a quantum system to a superposition requires advanced physical operations. But, as a programmer, bringing superposition only requires a simple operation: applying Hadamard gate to at least one qubit of the system.

In this case, we only need to apply single-qubit Hadamard (H) and two-qubit Controlled-X ($\oplus$) gates. When the quantum gates apply to the system the total probability is 1. In other words, $|\psi\rangle^2 = 1$, no matter how many or what kind of gates we apply to the system.

Measurements

Third, we measure two of the three qubits. A classical computer interprets the measurements of each qubit as classical outcomes (0 and 1) and stores them in the two classical

bits. Getting out of superposition is as simple as entering: application of a Hadamard gate to the system.

Classically conditioned quantum gates

Fourth, we apply single-qubit Z and X quantum gates on the third qubit. These gates are conditioned on the results of the measurements that are stored in the two classical bits. In this case, we are using the results of the classical computation concurrently in real-time within the same quantum circuit.

Construction of circuits using simulators and real quantum devices

Qiskit simulator

The simulators on this thesis were run using qiskit simulator. Qiskit uses *Python* (python, 2023) to simulate quantum circuits, draw them and plot the results. It Is useful to get familiarized with the behaviour of quantum systems, have an idea of the expected results, and visually show and present our results.

As a matter of interest qiskit can also simulate noise in quantum systems. However, as this feature is no tin the scope of this thesis it is not used, but is of interest whatsoever.

Instructions about the download and installation of qiskit and documentation can be found here (qiskit, 2023)

Quantum assembly in IBM Quantum Composer

The cloud quantum devices that were used is this thesis are part of the IBM Quantum Experience ecosystem (IBM Quantum Experience, 2023)

Quantum assembly(QASM) is a low-level language used by machines and simulators to apply quantum gates on qubits. There are many versions of QASM. IBM uses the OpenQASM version. So, far the common QASM (cQASM) standard is proposed, that would standardize the composition of quantum circuits. Setting a standard to Transpile from intermediate version of QASM to cQASM would increase compatibility between various quantum systems. So far this is only a proposition, not a widely used standard (Palmieri, 2019)

An example of quantum assembly to build the Bernstein-Vazirani circuit for key ‘1011‘ with the use of OpenQASM is the following:

```
include "qelib1.inc";

qreg q[5];
creg c[4];

h q[0];
h q[1];
h q[2];
```

```
h q[3];
x q[4];
h q[4];
cx q[3], q[4];
cx q[1], q[4];
cx q[0], q[4];
h q[0];
h q[1];
h q[2];
h q[3];
measure q[0] -> c[0];
measure q[1] -> c[1];
measure q[2] -> c[2];
measure q[3] -> c[3];
```

Conclusion to this chapter

In this chapter, we examined the structure and the building components of quantum circuits.
We pointed out the probabilistic nature of qubits -operands- and how they interact with gates -operators.
We pointed out most common quantum gates -the operators of the circuit-, their mathematical formulation and properties. Furthermore, we examined identities between them and saw that while some of them have their classical equivalents, others are of clearly quantum.
Finally, we composed the aforementioned knowledge to describe quantum circuits: sets of qubits, quantum gates and quantum wires.
We pointed out the similarities and the differences between classical circuits and their components versus the quantum.
In the next chapter, we investigate the algorithms that form the heart of this thesis: Grover Search Algorithm, Simon Algorithm and Bernstein-Vazirani, Algorithm.
We investigate them theoretically and compare them to the classical algorithms for the same problems.

As a trailer to the Implementation chapter, we implement the classical algorithms and the quantum algorithms in our local simulator to better demonstrate their structure and different approach.

Chapter 4: My Quantum Algorithms

Introduction

In this chapter, 3 of the most well-known quantum algorithms are studied. Each one is important for the reasons described below and offers an insight into a different aspect of the quantum computing superiority over the classical.

The quantum algorithms investigated are the following:

Grover's, Simon's and Bernstein-Vazirani's

Grover's solves the problem of searching through unstructured data. This algorithm can speed up an unstructured search problem quadratically, but its uses extend beyond that; it can serve as a general trick or subroutine to obtain quadratic run time improvements for a variety of other algorithms (Phillip Kaye, 2006)

Simon's algorithm provided the first example of an exponential speedup over the best-known classical algorithm by using a quantum computer to solve a particular problem. The problem is described in detail in the next section.

Bernstein-Vazirani's algorithm extends knowledge from Deutsch's algorithm into a more complex problem.

The algorithms

Grover

Grover's algorithm (or quantum search algorithm) is a process of steps to investigate very quickly and find an element in a set of values.

When we say very quickly, we mean that with Grover's algorithm we can investigate a set of N elements with $O(N)^{1/2}$) evaluations, while using a classic algorithm, we must do O(N) evaluations. Grover's algorithm provides acceleration by root rather than exponential, like other quantum algorithms.

Consequently, it cannot solve in polynomial time NP-complete problems, since the root of an exponential function remains an exponential function - and not a polynomial.

Despite its limitations, it remains faster than classical algorithms.

Practical applications exist. While finding an element in a set of elements may not initially sound particularly interesting, the truth is that it is a field with applications in cryptography, since cryptanalysis consists of finding an element - the one that breaks the cryptogram and we call it a key - from a set of possible keys.

The Grover's problem

In general, problems in computer science can be considered problems where we have a set of values and want to find a subset of one or more members that satisfy a given condition, often called the key. Furthermore, we are interested in following a procedure with as less complexity as possible, ideally in polynomial time or faster.

Grover's problem assumes no structure for the set of possible keys.

For an unstructured list of N items where one of them is the key, in general, the complexity of the algorithm is O(N). At least of a classical algorithm. Grover asked whether using a quantum algorithm is possible to reduce the complexity to something better than O(N). Indeed, it is possible to reduce complexity to $O(N^{1/2})$ and Grover's algorithm uses a clever trick named amplitude amplification to do so.

To summarize, suppose we have a system of $N = 2^n$ possible states $S_0, S_1, \ldots, S_N$. Each state is represented by a binary string of length n. We want to determine a state Sk that satisfies a condition C for which:

$$C(Sx) = \begin{cases} 0, & x \neq k \\ 1, & x = k \end{cases}$$

The Grover's algorithm

Classical approach

If we want to find a key in a list with 100% certainty, we need to check 1 element in best case and n elements in worst case. The complexity is O(N), with N/2 tries in average.

Quantum approach

Description

Grover's algorithm solves the Grover's problem with the following steps:

1) Bring the system to a superposition of N states each one with the same amplitude
2) Repeat the following procedure for $N^{1/2}$ times:
   a) In case x=k, rotate the phase by π rad, Else, do nothing
   b) Apply the diffusion matrix which is defined as follows:

$$D_{ij} = \begin{cases} \frac{2}{N}, & i \neq j \\ 1 - \frac{2}{N}, & i = j \end{cases}$$

The diffusion matrix can be implemented as D = HRH

Where H is the Hadamard gate and R is a rotation gate defined as follows:

$$R_{ij} = \begin{cases} 0, & i \neq j \\ 1, & i = 0: \ -1, \quad i \neq 0 \end{cases}$$

3) Measure the state. If there is a state with $C(S_k) = 1$, the state will be S_k with probability at least 50%

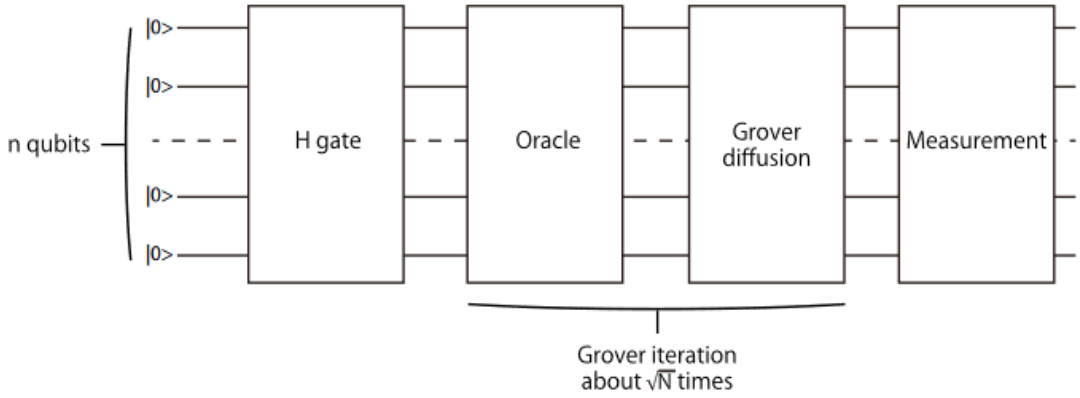

Figure 16. Grover circuit with oracle

Implementation

We implement the algorithm to better demonstrate its principles. We use a local simulator with qiskit, before eventually run the algorithm in a real quantum device in Chapter 5: Implementation

- Initialization.

```
from qiskit import QuantumCircuit, execute, Aer, BasicAer
from qiskit.visualization import plot_histogram
import numpy as np
import matplotlib.pyplot as plt

n=3
gr_key='111'

# 1. We need a circuit with n qubits. Initialize the system in state |0> ^n and put in a superposition of n equal-weigth states
grover_circuit = QuantumCircuit(n)
for q in range(n):
    grover_circuit.h(q)
```

Figure 17. Importing the necessary libraries and initiating the registers.

- Define the necessary gates ccz and diffusion.

```
# 2. Define the Grover iteration
#  a. define the phase transformation
#there is no ccz in qiskit, so compose one from the identity CCZ = HCCXH
def ccz():
    grover_circuit.h(2)
    grover_circuit.ccx(0,1,2)
    grover_circuit.h(2)
    grover_circuit.i([0,1])

#  b. define the diffusion transformation
def diffusion():
    for q in range(n):
        grover_circuit.h(q)
        grover_circuit.x(q)

    ccz()

    for q in range(n):
        grover_circuit.x(q)
        grover_circuit.h(q)
```

Figure 18. Define the necessary gates ccz and diffusion.

- Apply the gates to the circuit

```
# 2. Apply the Grover iteration  2 times, since 1 < sqrt(3)
#   a. apply the phase transformation
#  b. apply the diffusion transformation

for i in range(2):
    ccz()
    diffusion()

grover_circuit.draw("mpl")
```

Figure 19. Apply the gates to the circuit

- Draw the circuit. The iterations of step 2 are separated by the barriers

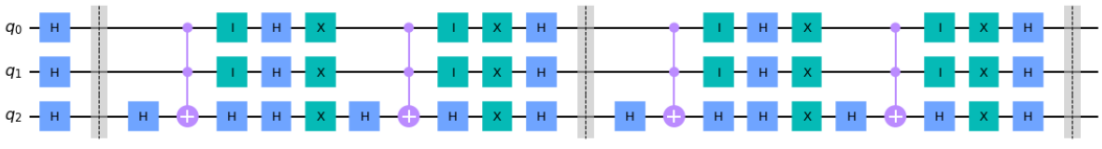

Figure 20. Grover circuit for key '111'

- Step 3: measurement. We measure the circuit. The circuit is ready to run in the local simulator. We can finally draw the full circuit

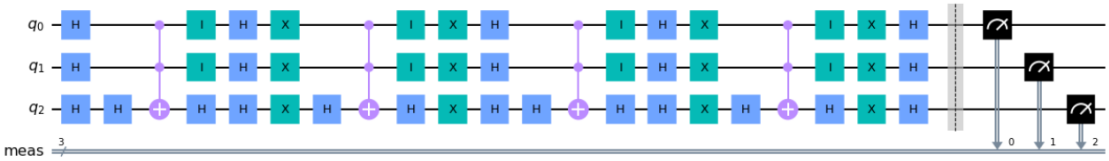

Figure 21.Grover circuit for key '111' with measurement

- Using the local simulator. Running it many times to make statistically insignificant any uncertainties in the result

```
backend = BasicAer.get_backend("qasm_simulator")
job = execute(grover_circuit, backend, shots=100)
results=job.result()
answer = results.get_counts()
```

Figure 22. The commands to run the Grover circuit

- Plot the result

```
# plot the result
myfig = plt.figure()
ax = myfig.add_subplot(111)

#   set label names
ax.set_ylabel('Y-axis')

#   set label colors
ax.xaxis.label.set_color('lightblue')
ax.yaxis.label.set_color('lightblue')
#   set values(tick_params) colors
ax.tick_params(axis='x', colors='yellow')
ax.tick_params(axis='y', colors='pink')
#   set bg color
ax.set_facecolor("#e2e2e2")


#   draw the result
plot_histogram(answer, ax=ax )
```

Figure 23. Customization of plot to visually present the Grover result

- The graph

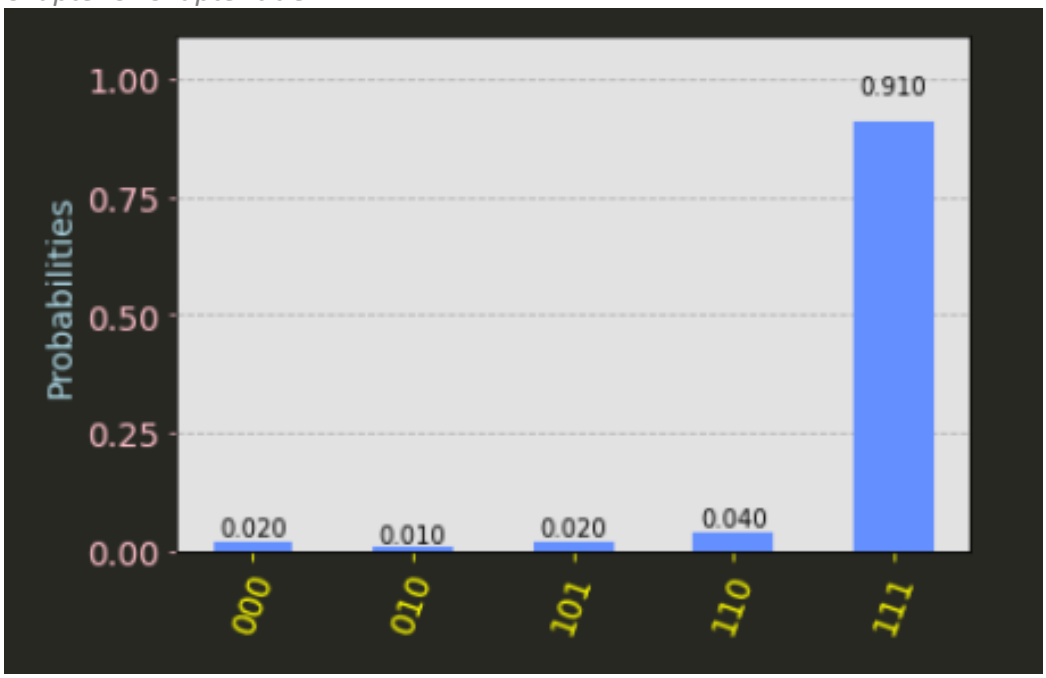

Figure 24. Probabilities of the key for Grover algorithm - 2 iterations

We notice that in the result there is some noise, but it is insignificant.

If we iterate only once the accuracy is lower as expected

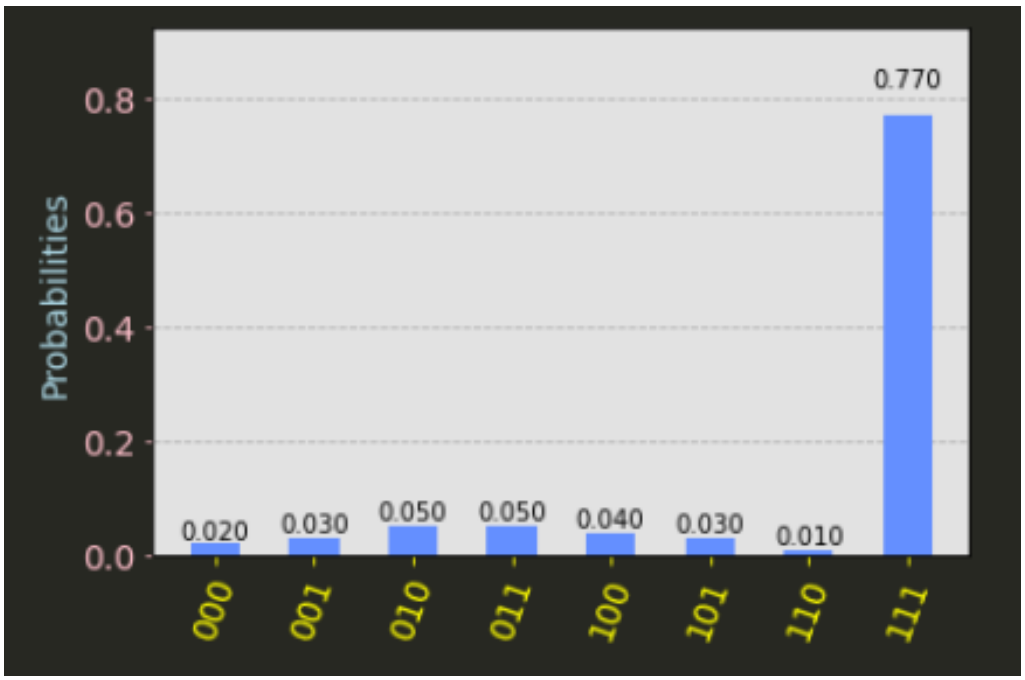

Figure 25. Probabilities of the key for Grover algorithm - 1 iteration

Simon

Simon's algorithm is a process of steps to solve the Simon problem.

The problem is this:

suppose we have a function that accepts as input- independent variable - sequences of digits of length n and gives as output- dependent variable - sequences of digits of length n.

In addition, the digits of both the input and the output can be either 0 or 1, $\{0,1\}^n \rightarrow \{0,1\}^n$. Our function has the property of being either 1-to-1 or 2-to-1, but we have not been told, which of the two is true.

1-to-1: each input corresponds to a unique output. For example, for a function that accepts 4 inputs:

$$f(1) \rightarrow 1,\ f(2) \rightarrow 2,\ f(3) \rightarrow 3,\ f(4) \rightarrow 4$$

2-to-1: 2 inputs correspond to a single output. For example, for a function that accepts 4 inputs:

$$f(1) \rightarrow 1,\ f(2) \rightarrow 2,\ f(3) \rightarrow 1,\ f(4) \rightarrow 2$$

The aim is to determine whether the function f is 1-to-1 or 2-to-1 with as few calculations as possible

Explaining the problem, this time using technical language, the problem is formulated as follows:

Given a function f such that $f: \{0,1\}^n \rightarrow \{0.1\}^n$ and knowing that, for unknown $s \in \{0.1\}^n$, for every $x, y \in \{0.1\}^n$, then $f(x)=f(y)$, if and only if , $x \oplus y \in \{ 0^n , s \}$,

where $\oplus$ denotes the binary addition mod2.

The problem is to determine whether the function f is 1-to-1(i.e., $s = 0^n$) or 2-to-1($s \neq 0^n$) with as few calculations as possible.

It is a problem that has exponential complexity when solved in a classical way, but polynomial complexity when solved in a quantum way. Simon's algorithm is an algorithm that offers exponential growth over classical computers.

If we want to be 100% sure of our solution, we need to do $2^{n-1}+1$ calculations. If we are lucky, i.e., if the meeting is 2-to-1 and we find the same outputs as the first test we will only do one act. But if we are unlucky, we should exhaust the $2^{n-1}+1$ i.e., check half the inputs plus one more value.

Even if we use probabilistic methods and accept a little uncertainty in the result, it may turn out that we need to make comparisons of the order of $2^{n/2}$

On the contrary, using Simon's algorithm we can solve the problem with polynomial complexity.

The Simon algorithm

Classical approach

Description

Classically, if we want to know what s is with 100% certainty for a given f, we must check up to $2^{n-1}+1$ inputs, where n is the number of bits in the input. This means checking just over half of all the possible inputs until we find two cases of the same output. If f is 2-1 and we are lucky and find the two same outputs in the first tries, we could solve the problem with just 2 queries. But if f is one-to-one, or get *really* unlucky with an f that's two-to-one, then we need to check the full $2^{n-1}+1$ outputs. There are known algorithms that have a lower bound of Ω(2n/2) (see Reference 2 below), but even in that case the complexity is of exponential order.

Quantum approach

Using an errorless quantum computer, we can solve this problem with 100% confidence after only n call to the function f(x).
The quantum Simon algorithm to find the hidden bit string is the following:

1. Initialize n input qubits plus a n-qubit ancilla register in state $|0\rangle^n$
2. Apply Hadamard gates to registers
3. Query the oracle
4. Measure the ancilla register
5. Apply Hadamard gates to the input register
6. Measure the input register

Between steps 3 and 4, we can measure the ancilla register as well. This is not necessary, but it was a step Simon did when he first investigated his algorithm (Simon, 2021).

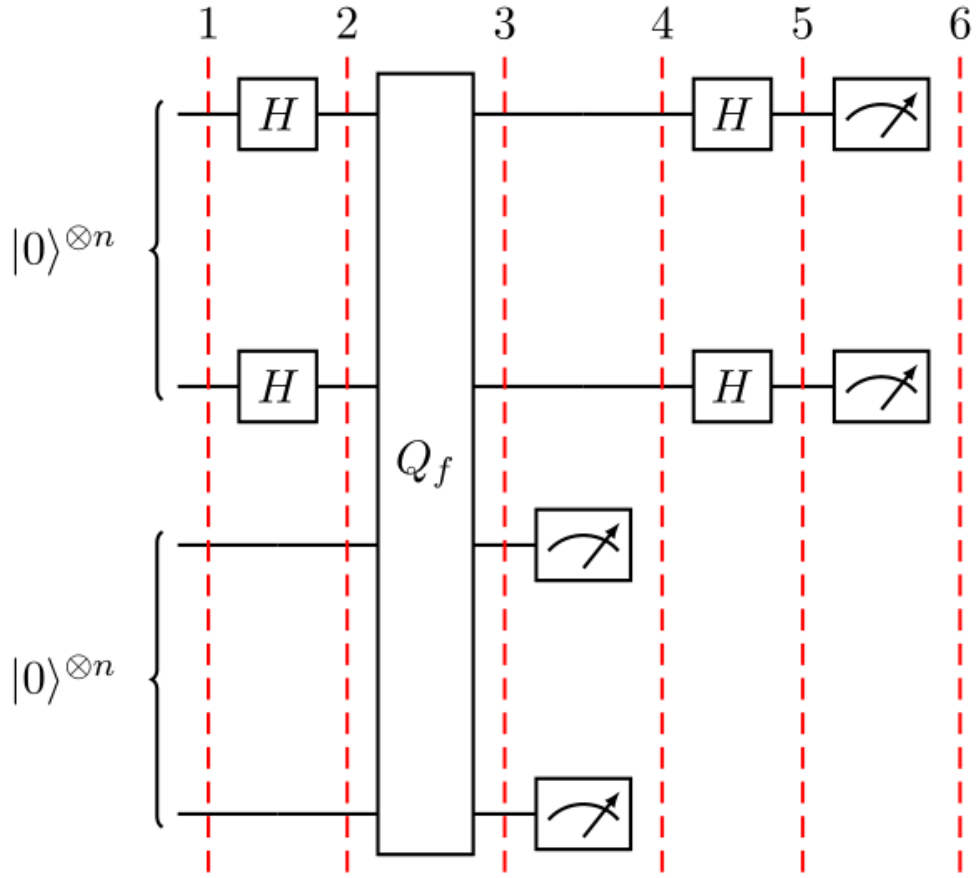

Figure 26. Simon cirquit with oracle (qiskit-simmon-with-oracle, 2023)

1. Initialize n input qubits and n ancilla qubits in state $|0>^{n,}$

   $\Psi_1 = |0>^n|0>^n$

2. Apply Hadamards to both registers

$$\psi_2 = \frac{1}{\sqrt{2^n}}\sum_x |x>|O>n$$

3. Apply the Oracle to both registers

$$\psi_3 = \frac{1}{\sqrt{2^n}}\sum_x |x>|f(x)>$$

4. Measure the ancilla register. The measured value can either be x or y= x $\oplus$ s

$$\psi_4 = \frac{1}{\sqrt{2}}(|x> + |y>)$$

5. Apply Hadamard to the input register

$$\psi_5 = \frac{1}{\sqrt{2^{n+1}}}\sum_z((-1)xz|x> + (-1)yz)|z>$$

6. Measure the input register. Measuring the first register will give an output only if:

$$(-1)^{xz} = (-1)^{yz}$$

$$xz = yz$$

$$xz = (\mathrm{x} \oplus \mathrm{s})z$$

$$xz = xz \oplus sz$$

$$sz = 0\, mod 2$$

Repeating the algorithm n times we get n different z values, thus having the pairs

$$sz_1 = 0$$
$$sz_2 = 0$$
$$sz_n = 0$$

We can find s using Gaussian elimination

Implementation

We implement the algorithm to better demonstrate its principles. We use a local simulator with qiskit, before eventually run the algorithm in a real quantum device in Chapter 5: Implementation

- Importing the necessary libraries and initiating the registers.

```
from qiskit import QuantumCircuit, execute, Aer, BasicAer
from qiskit.visualization import plot_histogram
import numpy as np
import matplotlib.pyplot as plt

n=6
p=int(n/2)
bs='001'

# 1. We need a circuit with n/2 qubits, plus one ancilla register of n/2 qubits
#    the out put will be p qubits
simon = QuantumCircuit(n,p)
```

Figure 27. Importing the necessary libraries and initiating the registers.

- Construct the oracle

```
n=6
p=int(n/2)
bs='001'
n=6

so=QuantumCircuit(n)

# construst the oracle

for i in range(p):
    so.cx(i,i+p)

k=0

for i in range(p-1,-1,-1):
    if bs[i]=='1':

        m=p
        for j in range(p-1,-1,-1):
            print(j,i)
            if bs[j]=='1':
                so.cx(k,m)
            m+=1

        break
    k+=1

so.draw('mpl')
```

Figure 28. Implementing the oracle

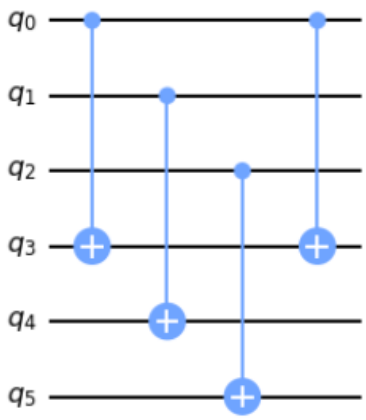

Figure 29. The Simon oracle for key = '001'

- Steps 2 – 5 of the algorithm

```
# 2. Apply H-gates
for qubit in range(p):
    simon.h(qubit)

simon.barrier()

# 3. Oracle to implement bit string multiplication
simon=simon+so

simon.barrier()

# 4. Apply Hadamard gates to the input register after querying the oracle
for qubit in range(p):
    simon.h(qubit)
simon.barrier()
```

Figure 30. Assembling the circuit from its components

- Step 6: measurement. The circuit is ready to run in the local simulator. We can finally draw the full circuit

```
# 5. Measurement
for i in range(p):
    simon.measure(i, i)

# Display circuit
simon.draw('mpl')
```

✓ 0.7s

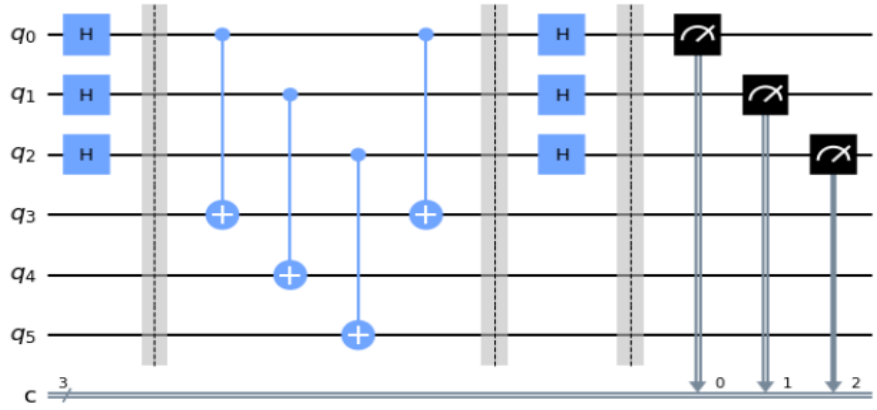

Figure 31. Visual representation of Simon circuit

- Using the local simulator. Running it many times to make statistically insignificant any uncertainties in the result

```
# use local simulator
simulator = BasicAer.get_backend('qasm_simulator')
shots = 5000
noisy_results = execute(simon, backend=simulator, shots=shots).result()
noisy_counts = noisy_results.get_counts()
```

Figure 32. The commands to run the Simon circuit

- Plot the result

```
# plot the result
myfig = plt.figure()
ax = myfig.add_subplot(111)

#   set label names
ax.set_ylabel('Y-axis')

#   set label colors
ax.xaxis.label.set_color('lightblue')
ax.yaxis.label.set_color('lightblue')
#   set values(tick_params) colors
ax.tick_params(axis='x', colors='yellow')
ax.tick_params(axis='y', colors='pink')
#   set bg color
ax.set_facecolor("#e2e2e2")


#   draw the result
plot_histogram(noisy_counts, ax=ax )
```

Figure 33. Customization of plot to visually present the Simon result

- The graph

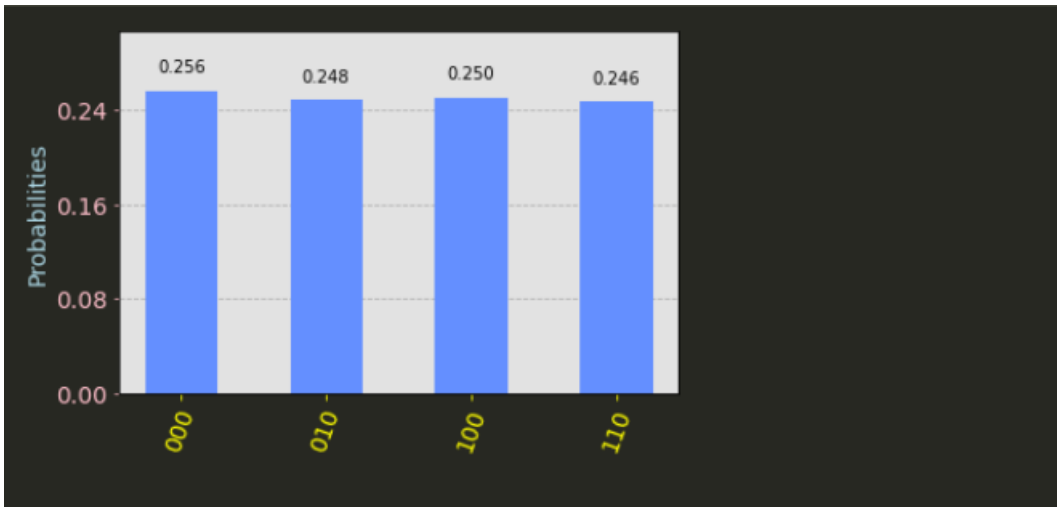

Figure 34. Probabilities of the key for Simon circuit with key '001'

Bernstein – Vazirani

The Bernstein – Vazirani problem

Description

The algorithm of Bernstein– Vazirani is a process of steps to solve the Bernstein–Vazirani problem .

The problem is this:

Given a function f such that f: $\{0,1\}^n \rightarrow \{0.1\}$ and knowing that, for unknown s $\in$ $\{0.1\}^n$, for every x $\in$ $\{0.1\}^n$, with f(x)= x*s = x 1*s 1 $\oplus$ x 2*s 2 … $\oplus$ x n*s n, find s,

where $\oplus$ denotes the internal product mod 2.

The problem is to identify each bit of s with as few calculations as possible.

It is a problem that has polynomial complexity O(n) when solved in a classical way, but complexity O(1) when solved in a quantum way. Simon's algorithm is an algorithm that solves the problem with a single call to the f-function.

The Bernstein – Vazirani algorithm

Description

We should determine s. Therefore, for known length of n of x, f(x) we should take the x, f(x) and given their values, reconstruct s. However, the approach found in the bibliography is a different one. We are given x, f(x). We pick an s. For that s, we perform the multiplication f(x) = x*s and we extract the conclusions. The approach in the bibliography is a demonstrational one and a bit unintuitive. However, as it is the standard one, we will follow along.

Classical Approach

Description

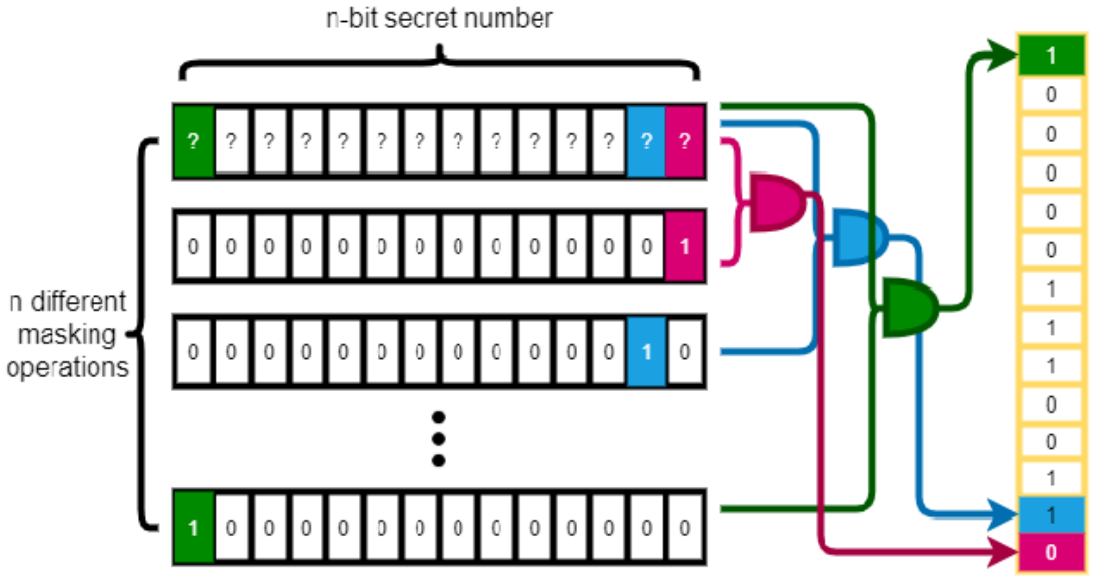

Figure 35. A visual representaion of Bernstein-Vazirani algorithm inner working (Loizidid, 2021)

This method takes advantage of the table (table of bitwise product)

Steps

The method consists of the following steps

- Construct the input x
- Construct f(x)
- Find s based on the s & s = f(x)

Description

We construct n strings x of length n. The strings have the following form:

x_0 = 100…

x_1= 010…

…

In general, the k-th bit is found by setting

x_k = 00… 1 …0

In other words, for the k-th input x, every bit is 0 except the k-th number being 1.

Then we proceed to find the 0^{th} bit

So, to find the 0-th bit we, check the input x and the output f(x). Based on table we have the following table:

Table 1. The classical algorithm to find the key bit-by-bit in Bernstein-Vazirani

x	F(x)	S_k
00000000000001	10000011100110	0
00000000000010		1
10000000000000		1

In this way, we need at exactly n tries to find s.

Implementation

Implementing the abovementioned algorithm

```
let mask = 1, result ='', s;
s='001'; //demonstration for a random s

for (let i=0; i<s.length; i++){
hit = mask & parseInt(s[i])  //bitwise product of mask,
                             //aka k-th bit of input x and k-th bit of s
result = result+hit
console.log(result)
}
```

Figure 36. code the classical implementation to solve the Bernstein-Vazirani problem

For this implementation, we use the approach of the bibliography. We know, the s we choose inputs x and expect the result to equal to the actual s. If the secret key s was held by another person, we could apply the same reasoning by asking the keyholder to bitwise-AND the input x and the s and give us the result.

Please, note that we don't use vectors, but instead we use only the k-th element of input x as mask. Then, we bitwise-AND the mask and the k-th element of s.

Quantum Approach

Description

Using an errorless quantum computer, we can solve this problem with 100% confidence after only one call to the function f(x).
The quantum Bernstein-Vazirani algorithm to find the hidden bit string is very simple.

1. Initialize n input qubits in state $|0\rangle^n$
2. Apply Hadamard gates to the input register
3. Query the oracle
4. Apply Hadamard gates to the input register
5. Measure

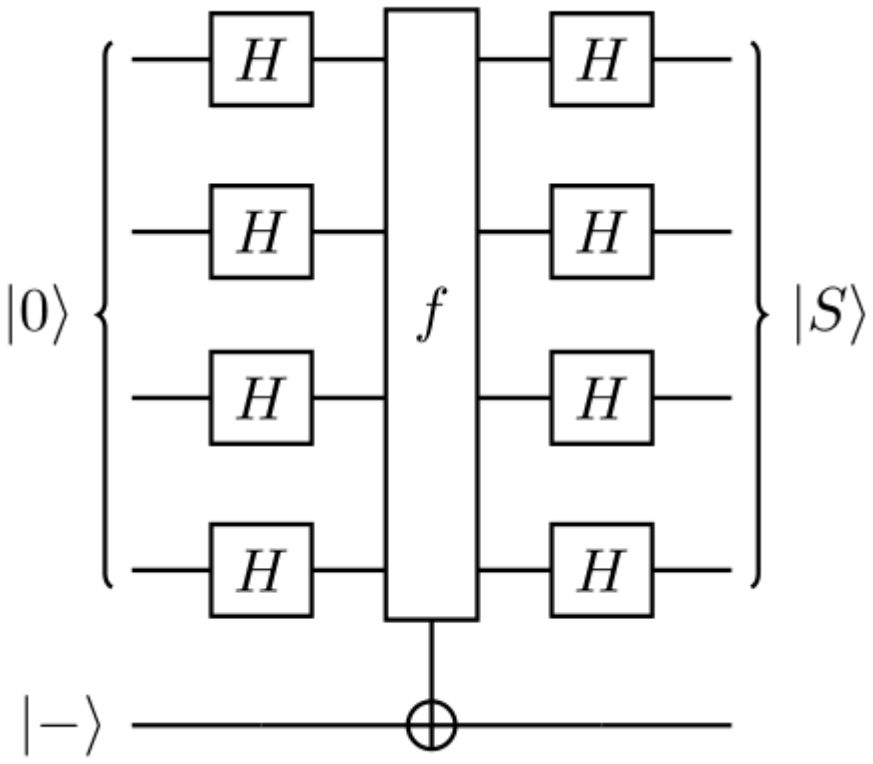

Figure 37. Bernstein-Vazirani circuit with oracle (qiskit-bv-img, 2023)

1. So, for a n- qubit system the initial state is |0…0>

2. By applying Hadamard to the input we bring the system to a superposition state. From

$$H\left|a>=\frac{1}{\sqrt{2^n}}\sum_{x\in\{0,1\}^n}(-1)^{a\cdot x}\right|x>$$

We can derive that the system will be in state

$$H\left|0\ldots0>=\frac{1}{\sqrt{2^n}}\sum_{x\in\{0,1\}^n}(-1)^{0\ldots0\cdot x}\right|x>$$

Since 0...0 x =1, we have

$$H\left|a>=\frac{1}{\sqrt{2^n}}\sum_{x\in\{0,1\}^n}\right|x>$$

3. Then, we apply the oracle to the system

   Using phase kickback:

$$f\left|x>=\frac{1}{\sqrt{2^n}}\sum_{x\in\{0,1\}^n}(-1)^{s\cdot x}\right|x>$$

4. Applying Hadamard to the system, we get:

$$H|a>=|a>$$

5. We measure the state of the system. The wavefunction of the system is now

   Ψ= |s>

We implement the algorithm to better demonstrate its principles. We use a local simulator with qiskit, before eventually run the algorithm in a real quantum device in Chapter 5: Implementation

- Importing the necessary libraries and initiating the registers.

```
from qiskit import QuantumCircuit, execute, Aer, BasicAer
from qiskit.visualization import plot_histogram
import matplotlib.pyplot as plt


n =4  # number of qubits used to represent s
s = '1011'   # the hidden binary string

# 1. We need a circuit with n qubits, plus one ancilla qubit
#    Also need n classical bits to write the output to
bv_circuit = QuantumCircuit(n+1, n)
```

Figure 38. Importing the necessary libraries and initiating the registers.

- Steps 2 – 4 of the algorithm

```
# 2. Apply Hadamard gates before querying the oracle
for i in range(n):
    bv_circuit.h(i)



# put ancilla in state |->
bv_circuit.x(n)
bv_circuit.h(n)

# Apply barrier
bv_circuit.barrier()

# 3. Oracle to implement bit string multiplication
 # reverse s to fit qiskit's qubit ordering
i=n-1
for q in s:
    if q == '1':
        bv_circuit.cx(i, n)
    i-=1

# Apply barrier
bv_circuit.barrier()

# 4. Apply Hadamard gates after querying the oracle
for i in range(n):
    bv_circuit.h(i)
```

Figure 39. Assembling the circuit from its components

- Step 5: measurement. The circuit is ready to run in the local simulator. We can finally draw the full circuit

```
# Measurement
for i in range(n):
    bv_circuit.measure(i, i)

bv_circuit.draw('mpl')
```

✓ 1.4s

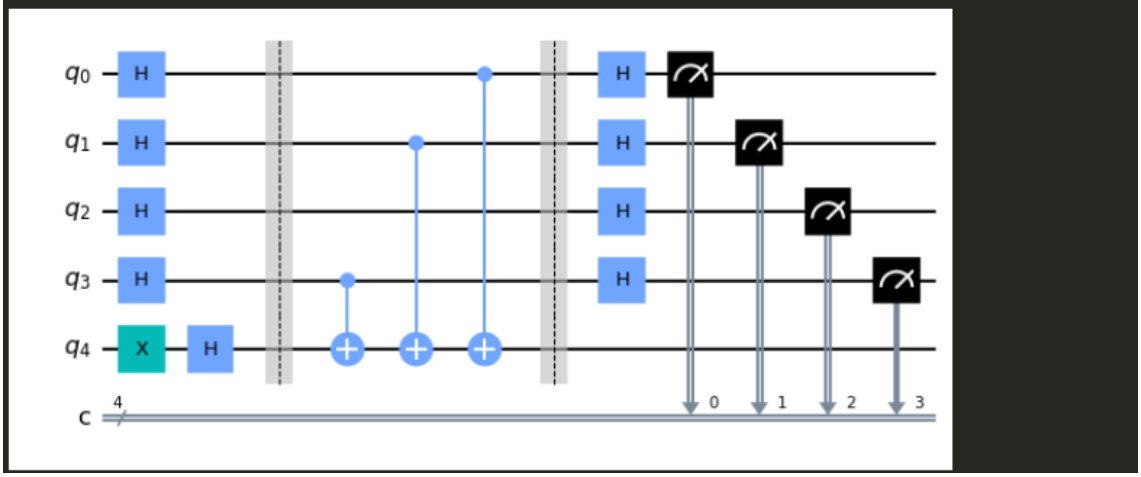

Figure 40. Visual representation of Bernstein-Vazirani circuit

- Using the local simulator. Running it many times to make statistically insignificant any uncertainties in the result

```
# use local simulator
backend = BasicAer.get_backend('qasm_simulator')
shots = 1024
job = execute(bv_circuit, backend=backend, shots=shots)
results=job.result()
answer = results.get_counts()
```

Figure 41. The commands to run the Simon circuit

- Plot the result

```
# plot the result
myfig = plt.figure()
ax = myfig.add_subplot(111)

#   set label names
ax.set_ylabel('Y-axis')

#   set label colors
ax.xaxis.label.set_color('lightblue')
ax.yaxis.label.set_color('lightblue')
#   set values(tick_params) colors
ax.tick_params(axis='x', colors='yellow')
ax.tick_params(axis='y', colors='pink')
#   set bg color
ax.set_facecolor("#e2e2e2")


#   draw the result
plot_histogram(answer, ax=ax )
```

Figure 42. Customization of plot to visually present the Simon result

- The graph: 100% probability to get s = 1011

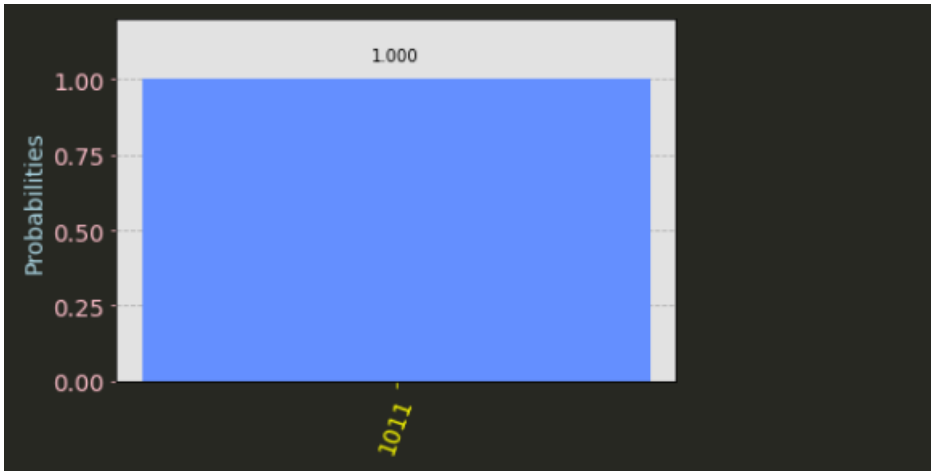

Figure 43. Probabilities of the key for Bernstein-Vazirani circuit with key '1011'

Conclusion to this chapter

In this chapter, we declared the algorithms, compared the classical and the quantum approach and implemented the quantum solutions in qiskit local simulator. In the next chapter, we implement the algorithms in real quantum devices.

Chapter 5: Implementation

Introduction

The real test for any quantum program is to run it in a quantum computer.

In the previous chapter, we understood the ideas behind the quantum algorithms, compared them to the classical solutions to the same problems and implemented them, in quantum simulators.

Now the time has come to evaluate the programs in real quantum devices.

Procedure to run programs in IBM quantum computers

For our purposes, we used the IBM quantum computers.

One first needs to create an account to have access to the IBM ecosystem. After that, constructing the circuits is even easier than the qiskit simulator, as we don't have to write any code, just to drag and drop gates to form a circuit.

IBM Quantum Composer offers the possibility to write the circuit in OpenQASM. However, we chose to construct the circuit through drag n' drop, as it is easier and less prone to faults.

To create an account, just visit https://quantum-computing.ibm.com/ and follow the instructions. The step-to-step instructions is provided by the page are straight-forward. Among the suggested programs we choose the free one. This offers access to various machines around the world. Each one permits access to up to 7 qubits.

Each algorithm was run to at least 2 different machines. The machines were chosen based to waiting time.

The IBM quantum ecosystem

After signing in we had access to the machines shown below

Table 2. The available IBM quantum experience machimes at the time of writing this thesis (11/2022 – 01/2023)

Number of qubits	Machines
5	ibm_lima, ibm_belem, ibm_quito, ibm_manila
7	ibm_oslo, ibm_nairobi

Table 3. Information about the machines used.

Algorithm	Machine	used qubits	Available machine qubits	Shots
Grover -1-iteration	ibm_belem	3	5	1024
	ibm_manila		5	
Grover -2-iterations	ibm_belem		5	
	ibm_oslo		7	
	ibm_manila		5	
Simon	ibm_oslo	6	7	
	iIbm_nairobi		7	
Bernstein-Vazirani	ibm_belem	5	5	
	ibm_manila		5	

Grover

Grover's algorithm was run for 3 qubits. The key was '111'. First, we constructed the circuit and run it for <√3 iterations (1 iteration) and then for >√3 iterations (2 iterations). This was to check that the accuracy of the algorithm gets higher as we reach √N

1 iteration

We constructed the circuit by drag and drop the appropriate gates in the circuit canvas.

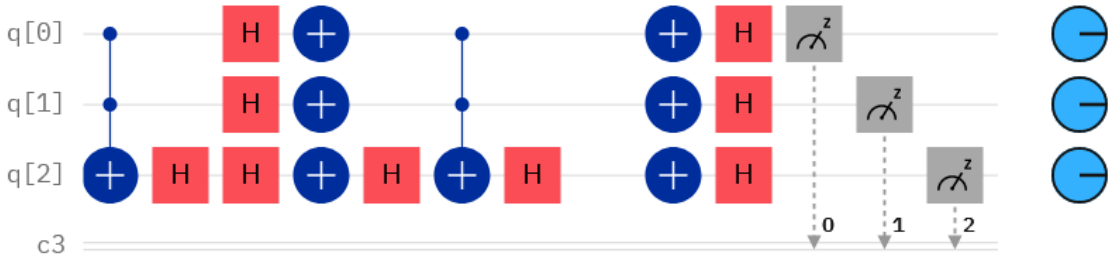

Figure 44. Circuit for Grover algorithm-1 iteration in ibm quantum composer

Since the canvas offered no option for a CCZ gate, we used the identity CCZ = H_t-CCX-H_t

Where H_t denotes that we apply Hadamard gate to the target qubit of CCX

Based, on the availability of each machine we run the algorithm to the machines as follows:

Table 4. Machines the Grover algorithm – 1 iteration- was run on.

Algorithm	Machine	used qubits	Available machine qubits
Grover -1-iteration	ibm_belem	3	5
	ibm_manila		5

More details about the execution:

Table 5. Transpiling and running info for Grover algorithm – 1 iteration.

Machine	Transpile time(ms)	Validation time(ms)	In queue	Running(s)	Total time
ibm_belem	604	608	2m 1,8s	2,9	2m 16,7s
ibm_manila	595	825	1m 43,9s	3,9	2m 1,5s

2 iterations

We constructed and ran the circuit for 2 iterations of the Grover algorithm.

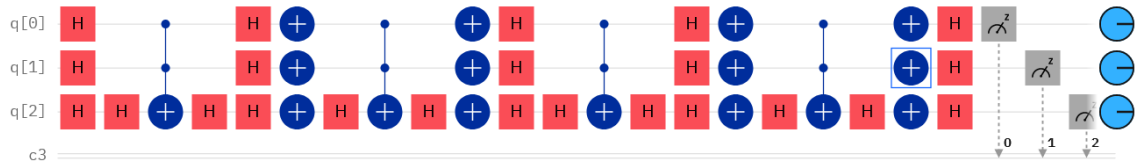

Figure 45. Circuit for Grover algorithm -2 iterations- in ibm quantum composer

Based, on the availability of each machine we run the algorithm to the machines as follows:

Table 6. Machines the Grover algorithm – 2 iterations- was run on.

Algorithm	Machine	used qubits	Available machine qubits
Grover -2-iterations	ibm_belem	3	5
	ibm_oslo		7
	ibm_manila		5

More details about the execution:

Table 7. Transpiling and running info for Grover algorithm - 2 iterations.

Machine	Transpile time(ms)	Validation time(ms)	In queue	Running(s)	Total time
ibm_belem	572	680	1m 34,2s	2,8	1m 46,8s
ibm_oslo	723	541	9m 55,3s	4,7	10m 11,7s
ibm_manila	696	674	33s	4,4	1m 21,9s

Simon

Simon's algorithm was run for 3 qubits. With the use of 3 ancilla qubits, 6 qubits were needed.

We constructed the circuit by drag and drop the appropriate gates in the circuit canvas.

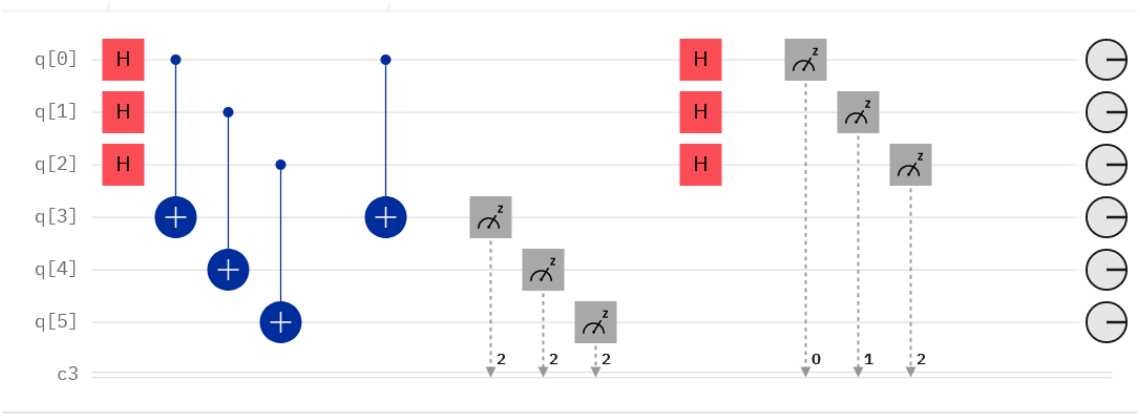

Figure 46. Circuit for Simon algorithm in ibm quantum composer

Since only 2 of the offered machines had at least 6 qubits and were capable of running the circuit, we ran as follows:

Table 8. Machines the Simon algorithm was run on

Algorithm	Machine	used qubits	Available machine qubits
Simon	ibm_oslo	6	7
	ibm_nairobi		7

More details about the execution:

Table 9. Transpiling and running info for Simon algorithm.

Machine	Transpile time(ms)	Validation time(ms)	In queue	Running(s)	Total time
ibm_oslo	882	567	9m 54s	4,4	10m 2,8s
ibm_nairobi	724	733	3h 53m 40,5s	4,6	3h 53m 50,9s

Bernstein-Vazirani

Simon's algorithm was run for 4 qubits. With the use of 1 ancilla qubit, 5 qubits were needed.

We constructed the circuit by drag and drop the appropriate gates in the circuit canvas.

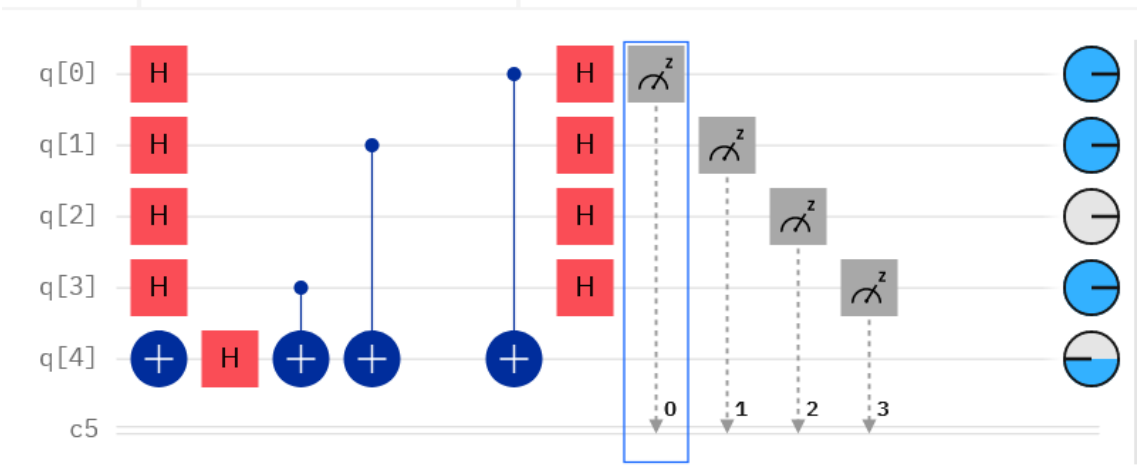

Figure 47. Circuit for Bernstein-Vazirani algorithm in ibm quantum composer

Based, on the availability of each machine we run the algorithm to the machines as follows:

Table 10. Machines the Bernstein-Vazirani algorithm was run on

Algorithm	Machine	used qubits	Available machine qubits
Bernstein-Vazirani	ibm_belem	5	5
	ibm_manila		5

More details about the execution:

Table 11. Transpiling and running info for Bernstein-Vazirani algorithm.

Machine	Transpile time(ms)	Validation time(ms)	In queue	Running(s)	Total time
ibm_belem	398	572	2m 32,5s	2,7	2m 38,3s

Ibm_manila	691	703	2m 26,6s	4,2	3m 22,4s

Conclusion to this chapter

In this chapter, we described the implementation of our algorithms in the IBM quantum devices. In the next chapter, we present and analyse the results.

Chapter 6: Results

Introduction

In this chapter the results of the implementation are presented. It Is useful to analyse if the results were the expected, if not, what were the possible sources of error and to what extent the programs run in a real quantum computer differ from those expected theoretically or the results we got in the simulations.

Grover

We run Grover algorithm for a system of 3-qubits and key = '111'. For a system of 3 qubits, we need to run the algorithm √3 ~ 1,7 times. We run it for 1 iteration and for 2 iterations to confirm that a little above √N times may give error and a little above √N times will give a credible result.

1-iteration

We run the algorithm in 2 different quantum machines to ensure that the result is not machine-specific:

Table 12. Machines the Grover algorithm – 1 iteration- was run on.

Algorithm	Machine	used qubits	Available machine qubits
Grover -1-iteration	ibm_belem	3	5
	ibm_manila		5

The results are presented in the following pictures:

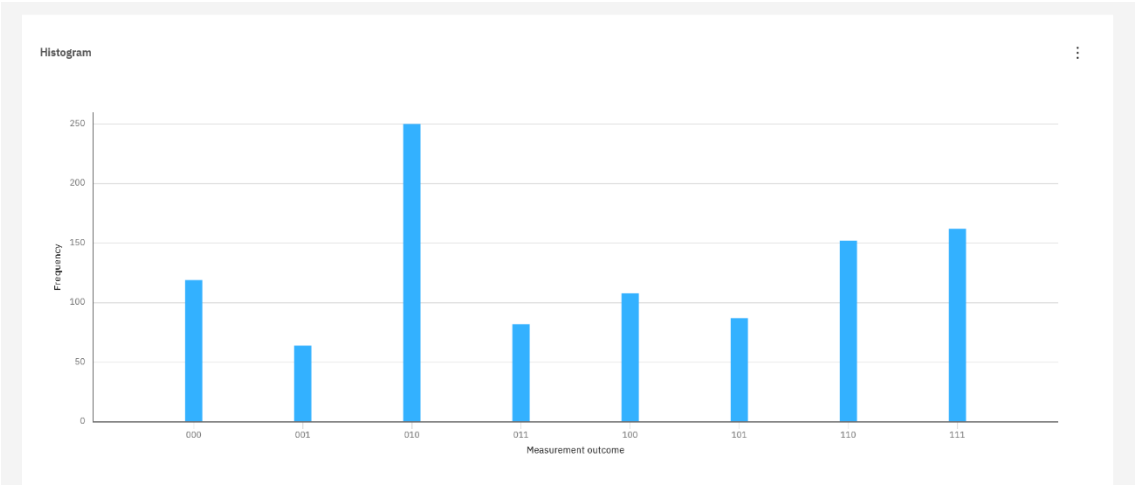

Figure 48 The probability distribution in ibm_belem was not the expected. Noisy and the key was not found correctly.

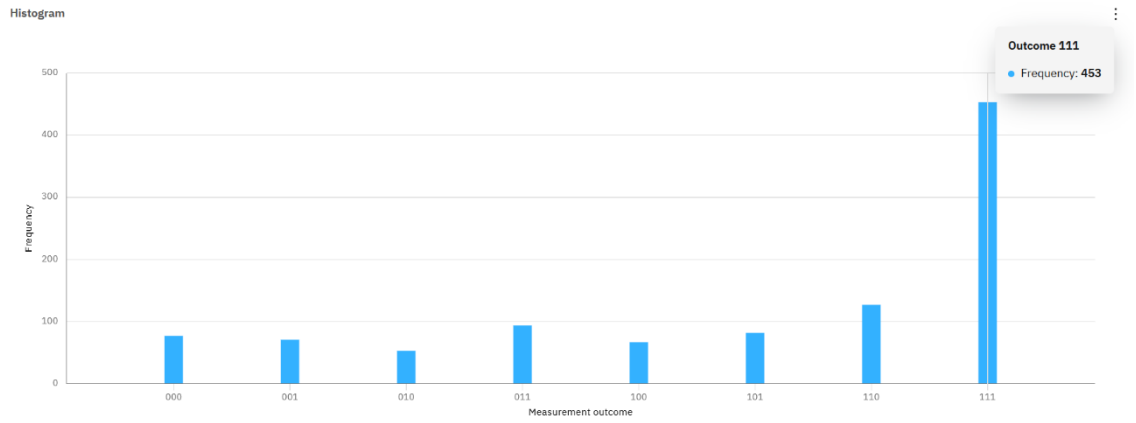

Figure 49 The probability distribution in ibm_manila was the expected.

2-iteration

System info

The algorithms where run in the following machines:

Table 13. Machines the Grover algorithm – 2 iterations- was run on.

Algorithm	Machine	used qubits	Available machine qubits
Grover -2-iterations	ibm_belem	3	5
	ibm_oslo		7
	ibm_manila		5

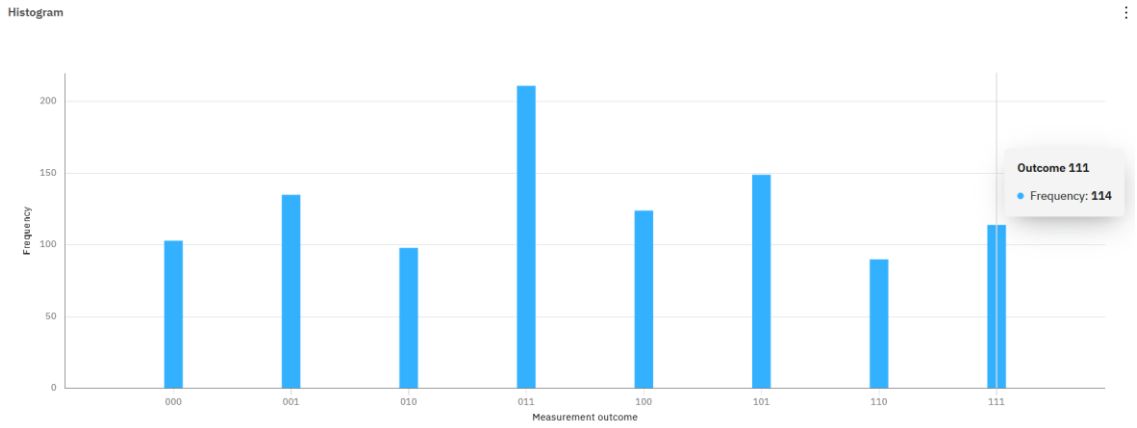

Figure 50. The probability distribution in ibm_belem was not the expected. . Noisy and the key was not found correctly.

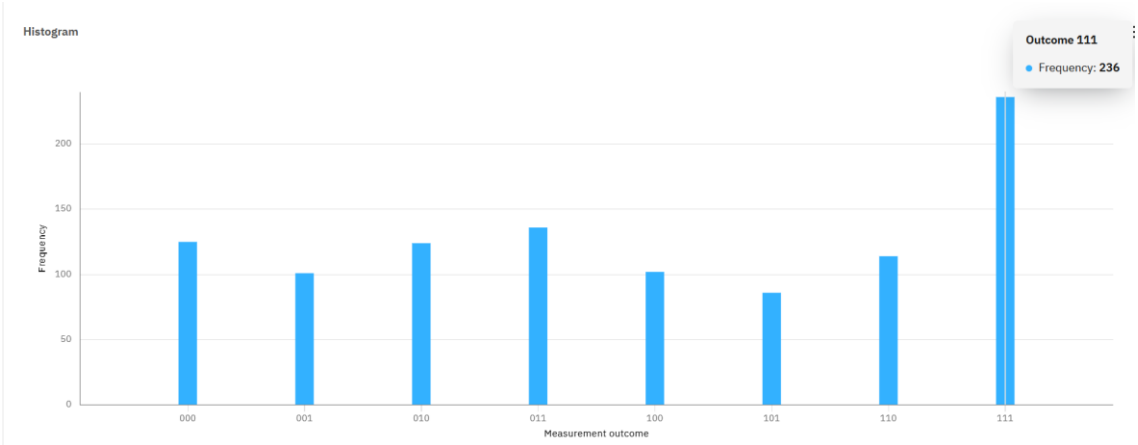

Figure 51. The probability distribution in ibm_oslo was the expected

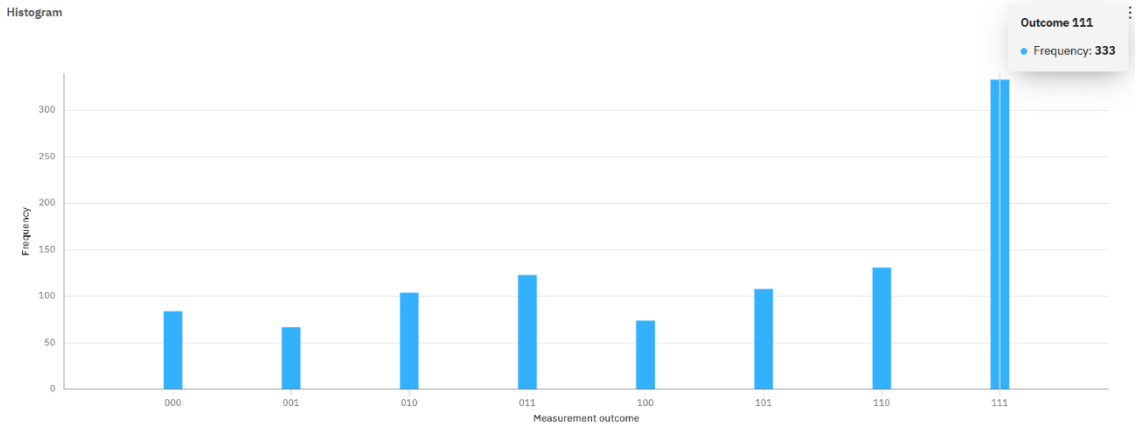

Figure 52. The probability distribution in ibm_manila was the expected

Analysis

It is useful to compare the results we got in the quantum machine vs the results from the simulation

The results of the Belem computer are the not the expected. Both in 1-iteration and in 2-iterations the output is not the expected. Consequently, we will not include them in the analysis.

The results are summarized in the following table

Table 14. Statistical analysi showed that the key is an outlier and can be distinguished from the noise

	Machine		
	Simulation	ibm_oslo	ibm_manila
Prob. Of key (%)	100	23,0	32,5
σ	-	0,04	0,08
z-index of key	-	2,68	2,64

We note that the result in actual quantum machines contains noise, but the expected outcome is clearly dominant.

For the Belem machine we have:

Table 15. The negative z-score of the expected key in ibm_belem denotes that we should exclude this result from our analysis

	Machine	
	Simulation	ibm_belem
Prob. Of key	100	11,0
σ	-	0.03793
z-index of key	-	-0.12873

Obviously, the key is not only an outlier as expected, but also below average. Obviously, the result is untrustworthy

The low σ and the value of z-score of the key (>2,5) show that the key is clearly an outlier of the dataset

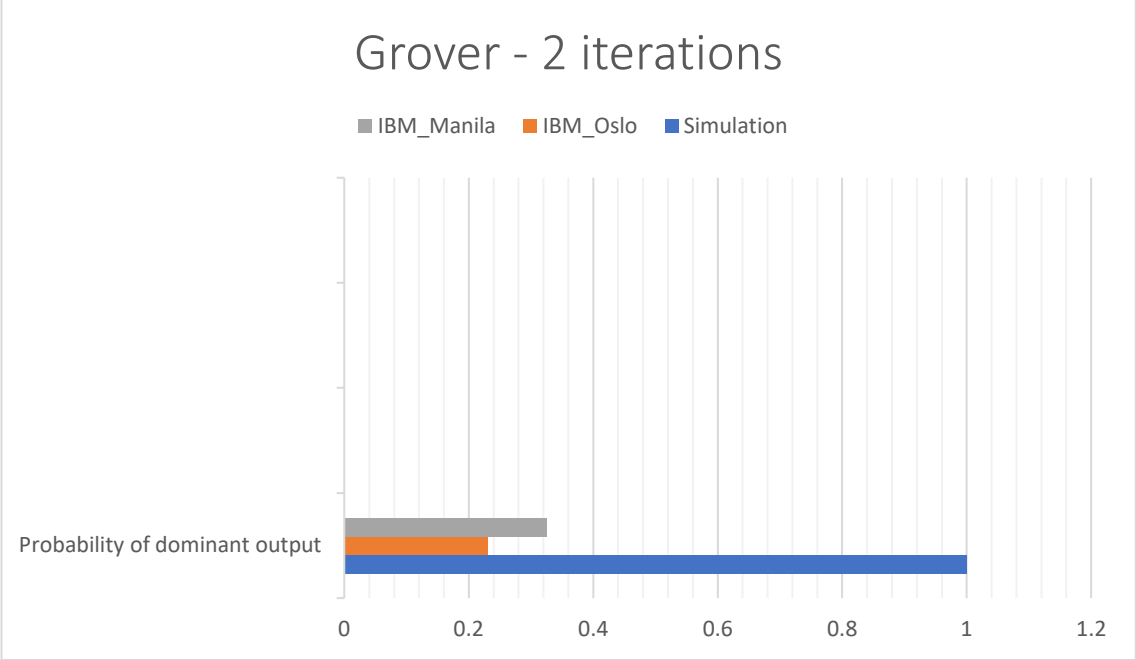

Figure 53. Probability of dominamt output (key) in simulation and in ibm quantum devices

Simon

System info

The algorithms where run in the following machines:

Table 16. Machines the Simon algorithm was run on.

Algorithm	Machine	used qubits	Available machine qubits
Simon	ibm_oslo	6	7
	ibm_nairobi		7

The results of the execution are the following:

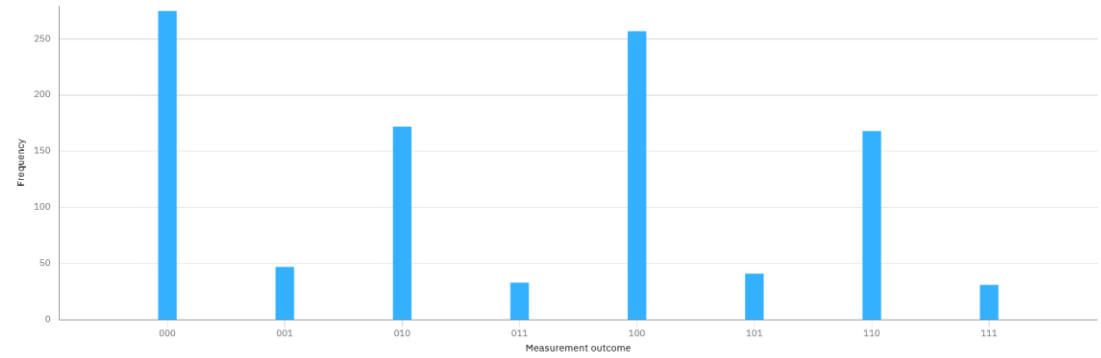

Figure 54. When run in ibm_oslo the quantum part of Simon algorithm gives 4 dominant output and some noise

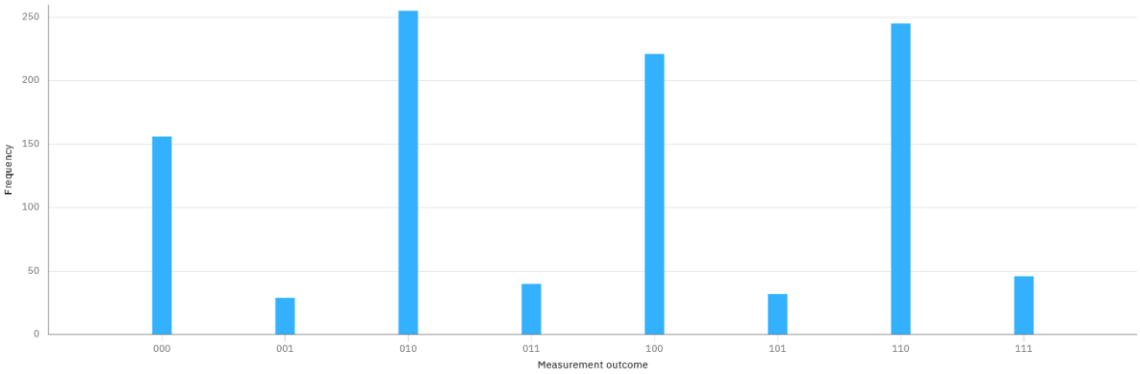

Figure 55. When run in ibm_nairobi the quantum part of Simon algorithm gives 4 dominant output and some noise

Analysis

It is useful to compare the results we got in the quantum machine vs the results from the simulation

The results from the quantum part of Simon algorithm are the expected

In the simulator we got 4 results (000, 010, 100, 110).

In the real device, we have 4 outputs with high probability and 4 with lower.

The two groups are:

Table 17. Outputs x on group 1 satisfy the condition x*key = 0, contrary to outputs on group 2that do not and are considered noise.

Output

Group 1	Group 2
000, 010, 100, 110	001, 011, 101, 111
Satisfy condition	Do not satisfy condition

The results with high probability satisfy the condition x*key = 0

Thus, we attribute the existence of the red outputs to noise, and we can ignore them as they do not affect the result.

With the use of z-score, we can reason solidly about the division in 2 groups we chose

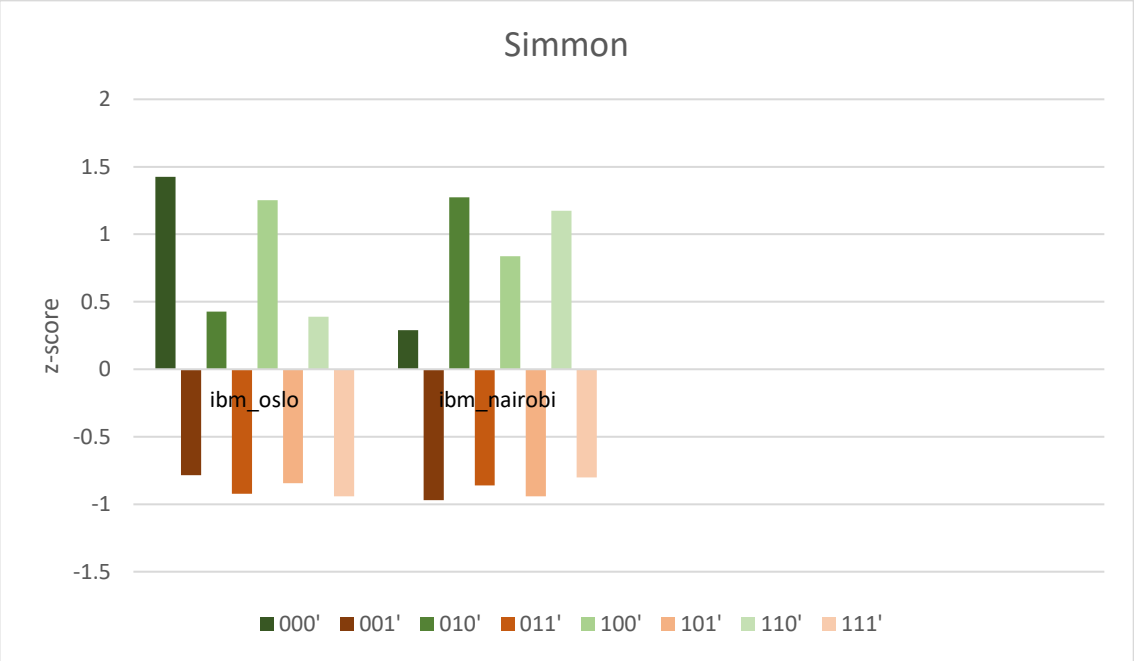

Figure 56. The z-score of noise is negative, contary to the z-score of actual results that is positive

The values with positive z-index satisfy the condition, while those with negative, do not.

Bernstein-Vazirani

System info

The algorithms where run in the following machines:

Table 18. Machines the Bernstein-Vazirani algorithm was run on.

Algorithm	Machine	used qubits	Available machine qubits
Bernstein-Vazirani	ibm_belem	5	5
	ibm_manila		5

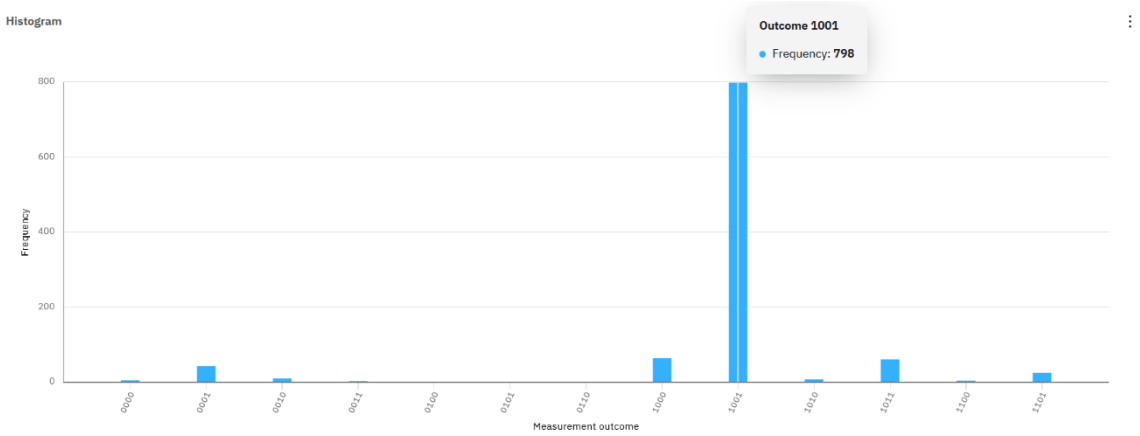

Figure 57. The dominant result -'1001'- in ibm_belem is not the expected key –'1011'- of Bernstein-Vazirani algorithm

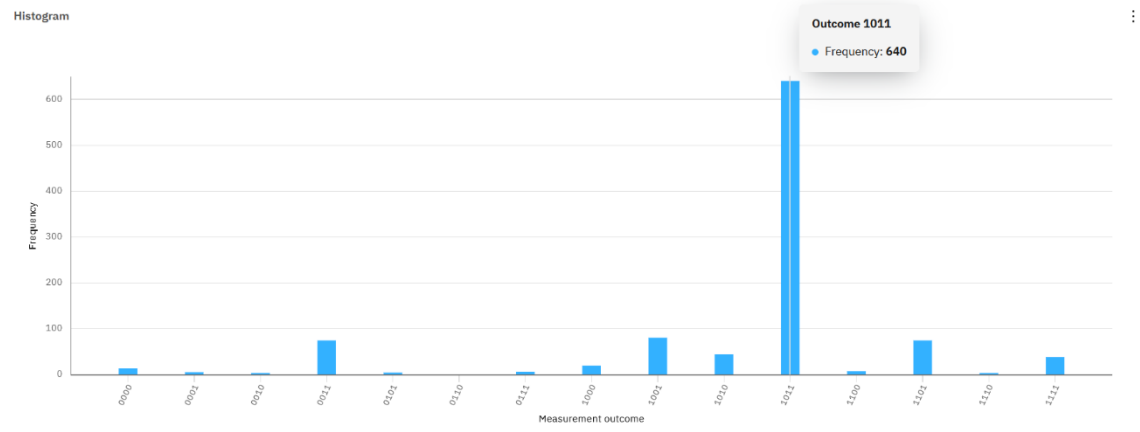

Figure 58. The key -'1011'- of Bernstein-Vazirani algorithm was dominant with ignorable noise in ibm_manila

Analysis

It is useful to compare the results we got in the quantum machine vs the results from the simulation

The results are summarized in the following table

Table 19. Statistical analysis of the results denotes that the key was found incorrectly in the ibm_belem machine

	Machine		
	Simulation	ibm_manila	ibm_belem
Prob. Of key (%)	100	62,5	5,7
σ	-	0,15	0,19
z-index of key	-	3,69	-0.025

We notice that the Belem machine is untrustworthy again. The manila result is the expected. We exclude Belem result from the graph

We see that the result in quantum contains noise, but the expected outcome is clearly dominant

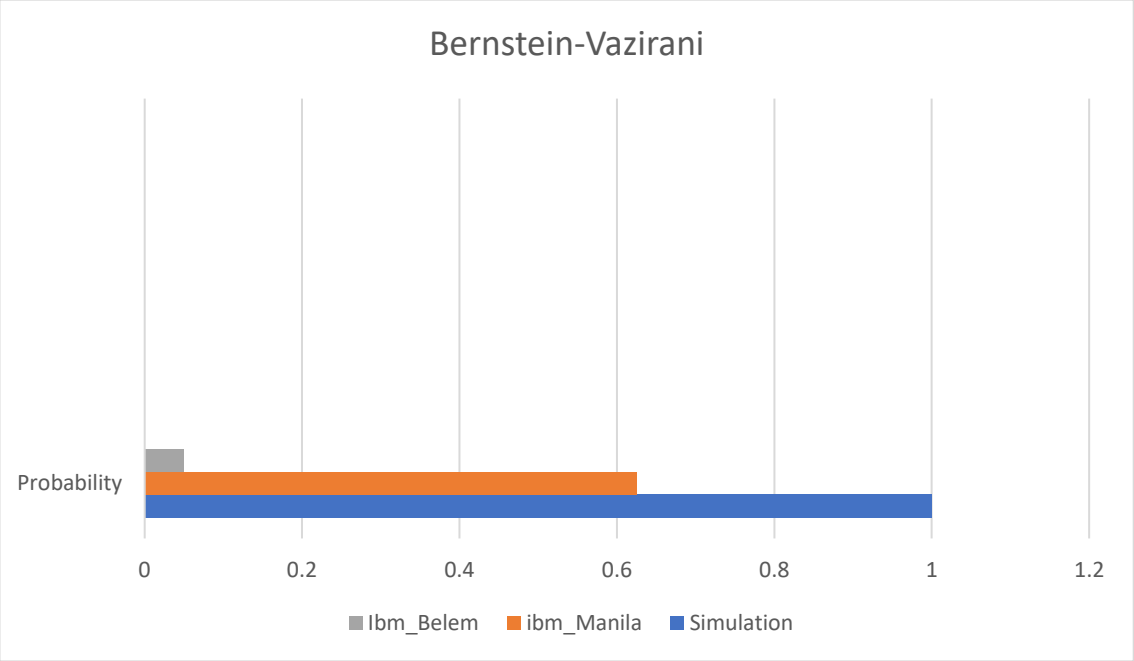

Figure 59. Bernstein-Vazirani on ibm_manila gave the less noisy result of the devices and algorithms tested, while ibm_belem was again untrustworthy aand the simulation gave 100% certainty

Conclusion to this chapter

In this chapter, the results of the implementation in real quantum devices are presented and analysed. We observe that we get the expected results. However, it is also existent the presence of noise in the result, but with simple statistical handles, we can confirm the dominance of the expected outcomes compared to the noise, which can be ignored. One of IBM machines (ibm_belem is untrustworthy at the time of the writing, but the other machines used are reliable, if we judge by the result).

In the next chapter, we further discuss about the results, quantum computation in theory and implementation as was studied in this thesis and what useful information we can obtain so far.

Chapter 7: Discussion

Introduction

We have finally reached to the point where we are able to thoroughly analyse and discuss about the findings of this thesis.

In the previous chapter we presented the results.

In this chapter, we will delve into details about the results, their importance, whether they are the expected and probable causes of error and limitations, and we will explore further the meaning of quantum computation through the prism of Grover, Simon and Bernstein-Vazirani algorithms

Restatement of the question

After so many pages of calculations and theory, it is useful to restate the purpose of this thesis and the questions it investigates.

The purpose of this thesis is to explore the principles of quantum computation through the investigation and implementation of Grover, Simon and Bernstein-Vazirani algorithms. Furthermore, to reason the superiority of quantum algorithms over classical ones for specific problems and to understand how from the general theoretical algorithms, we can construct specific circuits and implement suitable oracles to answer the problems these algorithms are dealing with.

Summary

We found that the quantum algorithms are superior to classical ones for specific problems we studied.

Quantum computation is built on different physical principles than the classical. The unique properties that manifest themselves in quantum scale, make it possible to take advantage of aspects of the physical world that are unreachable in bigger scales, where quantum properties are neglectable.

Different physics lead to different computational mechanisms.

The mathematical formulation of quantum mechanics – and we accept this formulation is the one aptly describes the physical quantum reality, consequently we can base our computational models on it- suggests that our circuits shall be reversible. The existence of an ancilla qubit or ancilla registers are demanded in many cases to achieve reversibility and achieve the computational goals.

The ideas of entanglement and superposition are vital and set the foundation blocks of the building of quantum computation.

While a classical system can check one outcome at a time, a system in superposition can examine exponentially more outcomes at once.

BV algorithm solves the Bernstein problem. The problem is of complexity O(n) with classical algorithms and O(1) with quantum algorithm.

It uses an ancilla qubit to achieve phase kickback

Grover algorithm uses amplitude amplification to iteratively increase the amplitude of the solution to Grover's problems. It reduces complexity from O(n) to O(√n)

Simon algorithm solves the Simon problem exponentially faster that any classical algorithm outperforming both classical deterministic and probabilistic algorithms. It is a hybrid algorithm that includes a quantum part with an ancilla register that is measured and discarded and a classical post-processing to solve the equations derived from the quantum part and determine the solution to the problem

Implementation in quantum machines gives noisy results, but with elementary statistical analysis it is easy to discern noise from the key.

Both simulation and implementation require repetitions to ensure trustworthy results.

Interpretations

The purpose of this thesis was successfully fulfilled. The superiority of quantum computing was confirmed for the problems studied. The simulations in our local machine and the implementation in IBM machines were successful. In the most part, the results are those expected. The results of the simulations output the correct result. The results in the implementation gave the expected result in most machines, with one exception. Ibm_belem machine gave wrong key for Grover and Simon algorithm and is considered untrustworthy. Consequently, we excluded the wrong key results from the analysis. Since, the internal workings of the machines are not known to the author, we can only speculate about the cause of the discrepancy. Probably, a different internal structure in the machine results in noisy output.

The unexpected results in the ibm_belem do not affect the reliability of our results and running the algorithms in different machines is the exact reason we took that precaution: to avoid machine-specific errors.

Implications

The findings of this thesis support the literature views on the subject. (Bernstein Ethan, 1993) (Grover, 1996) (Simmon, 1997)

The investigation and implementation follow the known patterns. The results contain a tolerable and expected level of noise. With proper statistical analysis it is easy to determine the evaluation of the results.

Quantum machines are trustworthy. However, one of the machines we used is not trustworthy and its results were not the expected. The key it gave was wrong and the fault persisted among different algorithms.-

Limitations

The results are useful to demonstrate the principles of quantum computation and its implementation to solve problems significantly better than the classical machines.

Grover's algorithm could be used in real life situations to find the key in a list of possible keys, for example to break RSA. However, we should acknowledge that the other algorithms studied, have more of theoretical value than practical.

Recommendations

Further studies could be conducted to determine the sources of noise in IBM's machines and how to eradicate them.

Conclusion to this chapter

In this chapter, we recapped the purpose of this study, the questions it answers, the expected or not of the results, possible sources of error and suggestion for future studied.
After the discussion of all the above, we are ready to move to the Conclusion, where we conclude the current thesis and summarize it as a whole.

Chapter 8: Conclusion

Introduction

We conclude the study. We re-present the results and the goals.

Conclusion

- Quantum computation, indeed, outperforms classical computation
- Different physics, different computation
- Simulations give the expected results, limitations as it is a local machine
- Implementation gives noisy or faulty results. Free available qubits
- Developing field, mostly theoretical, but with potential
- Suggested studies: noise handling, practical construction of bigger machines

References

Bernstein Ethan, V. U. (1993). Quantum Complexity Theory. Retrieved from https://dl.acm.org/doi/pdf/10.1145/167088.167097

Bloch sphere. (2023, 1 11). Retrieved from Bloch sphere: https://tikz.net/bloch-sphere/

Christidis, C. (2016). *Nanoelectronics & quantum gates*. Retrieved 12 15, 2022, from https://eclass.upatras.gr/modules/document/file.php/CEID1069/7%CE%B7%20%CE%94%CE%B9%CE%B1%CE%BB%CE%B5%CE%BE%CE%B7.pdf

circuit-with-oracle. (2023, 11 1). Retrieved from circuit-with-oracle: https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQPx_gTlpJ_m3R65d5vuqDx9S1cUGyzJb3TmkpZuD-m6qJk8nfCoDX8ri_NV7jnCS1yRng&usqp=CAU

Crooks, G. E. (2012). *Gates, States, and Circuits.* Retrieved 12 2022, from http://threeplusone.com/gates

Dan C. Marinescu, G. M. (2012). *Classical and Quantum Information.* Academic Press. doi:https://doi.org/10.1016/C2009-0-64195-7

Diederik Aerts, M. S. (2014, 12). The extended Bloch representation of quantum mechanics and the hidden-measurement solution to the measurement problem. *Annals of Physics, 351*, pp. 975-1025. doi:https://doi.org/10.1016/j.aop.2014.09.020

Grover, K. L. (1996). A fast quantum mechanical algorithm for database search. v3. Retrieved from https://arxiv.org/pdf/quant-ph/9605043.pdf

IBM Quantum Experience. (2023, 1 11). Retrieved from IBM Quantum Experience: https://quantum-computing.ibm.com/

Loizidid, S. (2021). *IMPLEMENTATION AND EVALUATION OF ALGORITHMS ON CLASSICAL AND QUANTUM.* Nicosia: University of Cyprus.

Mao, L. (2023, 1 9). Retrieved from https://leimao.github.io/blog/Qubit-Bloch-Sphere/

Mosca, M. (1999). *Quantum Computer Algorithms.* Oxford: University of Oxford. Retrieved 1 8, 2023

Palmieri, R. (2019). *Implementation and testing of current quantum pattern matching algorithms applied to genomic sequancing.* Torino.

Phillip Kaye, R. L. (2006). An introduction to Quantum Computing. In R. L. Phillip Kaye, *An introduction to Quantum Computing* (pp. 152-178). Oxford: Oxford Academic. doi:https://doi.org/10.1093/oso/9780198570004.003.0011

Preskill, J. (2015). Lecture Notes for Ph219/CS219: Quantum Information. Chapter 2. In J. Preskill, *Lecture Notes for Ph219/CS219.* Retrieved 12 2022, from http://theory.caltech.edu/~preskill/ph219/chap2_15.pdf

python. (2023, 1 11). Retrieved from python: https://www.python.org/

qiskit. (2023, 1 11). Retrieved from qiskit: https://qiskit.org/

qiskit-bv-img. (2023, 1 11). Retrieved from qiskit-bv-img: https://qiskit.org/textbook/ch-algorithms/images/bv1.png

qiskit-general-circuit. (2023, 1 11). Retrieved from qiskit-general-circuit: https://qiskit.org/textbook/ch-algorithms/defining-quantum-circuits.html

qiskit-general-circuit-img. (2023, 1 11). Retrieved from qiskit-general-circuit-img: https://qiskit.org/textbook/ch-algorithms/images/teleportation_labelled.svg

qiskit-simmon-with-oracle. (2023, 1 11). Retrieved from qiskit-simmon-with-oracle: https://qiskit.org/textbook/ch-algorithms/simon.html

researchgate. (2023, 1 11). Retrieved from researchgate: https://www.researchgate.net/profile/Andreas_Ketterer/publication/316828699/figure/fig3/AS:669404337565697@1536609839396/color-online-Bloch-sphere-representation-of-the-Bloch-vector-v-for-qubits-d-2.png

S D Fallek, C. D. (2016). Transport implementation of the Bernstein–Vazirani algorithm with ion qubits. *New Journal of Physics*. doi:doi:10.1088/1367-2630/18/8/083030

Simmon, R. D. (1997). On the Power of Quantum Computation. Retrieved from https://epubs.siam.org/doi/epdf/10.1137/S0097539796298637

Simon, D. (2021, 10 11). *Exploring Simon's Algorithm with Daniel Simon*. Retrieved 1 12, 2023, from Exploring Simon's Algorithm with Daniel Simon: https://aws.amazon.com/blogs/quantum-computing/simons-algorithm/

Στέφανος, Τ. (2013). *Κβαντομηχανική ΙΙ.* Heraklion: University Publications of Crete.

Τραχανάς, Σ. (2011). *Κβαντομηχανική Ι.* Hearaklion: University Publications of Crete.