# An Introduction to Quantum Computing for Computer Scientists, with SAT

Francesco Piro

March 22, 2020

**Abstract**

This is the paper I have produced during my study of quantum computing for solving the SAT problem...

## Introduction

Quantum computation is a wide area involving several disciplines that is having always more success in nowadays applications thanks in particular to the development of the technology and its incredible results. Quantum physics' principles are the fundamentals on which the entire theory is based: thanks to their properties, new architectures allow to define devices that can solve classical problems in surprisingly reduced time and space complexities. However we always have a trade-off to consider, in particular now that these technology are still emerging.

This paper aims at giving a description for computer scientists of what a quantum computer is and which are its real impacts and advantages with respect to the classical ones. Hence I will try to provide a description of the state of the art of quantum computing with an approach that allows to understand how from the basic principles of quantum mechanics we are able to have an algorithm that is faster with respect to its classic counterpart. In order to do so in the first chapter (1) I will start with the basic linear algebra needed to study the quantum physics' principles we need to define a quantum computer. The definition of the *quantum computer* is fundamental both to understand how and algorithm is executed but also to have a comparison with the classic Turing machine that allows us to determine conclusions on computational complexity (chapter 2). In the first chapter we will also have practical examples realized with the **qiskit library** in order to clarify also with some lines of code the concepts. Once the background on quantum computation and computational theory are well consolidated by the reader, the last chapters provide a practical example that is used to prove the speedup for the particular **satisfiability problem.** I have used an efficient classical solver for the *k-SAT* as I could compare it with my quantum implementation, both realized in python. In the end I provide some important conclusions for quantum computing that I was able to conclude thanks to my study, in particular in the papers listed in the references.

# 1 Quantum Computing

This chapter aims at providing first the fundamentals needed in order to deal with quantum mechanics and second at defining a quantum computer thanks to the principles previously identified. With the quantum device we will be able to make a comparison with the classical one to understand with examples how the basic operations are realized in order to use them to implement complete algorithms. Further in the chapter, a section is completely dedicated to the main quantum search algorithm that is fundamental to solve the SAT with a quantum algorithm. Also here we will start from a classical version to compare it with its quantum counterpart. All the arguments related in this chapter, together with the ones in the next (2) are fundamental to give a significant interpretation to the comparison between the classic and quantum implementation of the algorithm able to solve the **satisfiability problem** (2.1).

## 1.1 Fundamentals

The study of quantum computers requires the knowledge of the decimal and binary representation of integers, probability notions and in particular linear algebra fundamental definitions like the ones of: *vectors, spaces, bases, linear systems, tensor product....* In this section are presented the basic concepts needed to face the quantum physics principles that we need in order to realize our quantum computer.

### 1.1.1 Linear Algebra

Basic principles of linear algebra are assumed to be well known by the reader, I want now to remark only the most important operators and definitions that we need to face the definition of the following quantum mechanical theorems we need to define a quantum device. The most important notions we need to acquire from this section are: *tensor product, Hilbert space, bra-ket notation.*

In order to realize a quantum computer we need understand how to define a state that is able to contain information that can be used to obtain a certain objective. As we will see in the next section, the state of a quantum device is a quantum state, thus a mathematical model that lives into a specific *vector state* whose dimension depends on the amount of information it needs to take care of. Typically, significant states contain information that results from the composition of several spaces combined together thanks to a particular operator called tensor product.

**Definition (Tensor Product):** *Given two vector spaces $V$ and $W$ over a field $K$ with bases $e_1, ..., e_m$ and $f_1, ..., f_n$ respectively, the tensor product $V \otimes W$ is another vector space over $K$ of dimension $mn$. The tensor product space is equipped with a bilinear operation $\otimes : V \times W \to V \otimes W$. The vector space $V \otimes W$ has basis $e_i \otimes f_j \forall i = 1, ..., m, j = 1, ..., n$.*

Typically we are going to work with complex Euclidean vector spaces of the form $\mathbb{C}^n$ and, by choosing the standard basis in the origin vector spaces, then the tensor product is nothing more than the Kronecker product.

**Definition (Kronecker Product):** *Given $A \in \mathbb{C}^{m\times n}$, $B \in \mathbb{C}^{p\times q}$, the Kronecker product $A \otimes B$ is the matrix $D \in \mathbb{C}^{mp\times nq}$ defined as:*

$$D = A \otimes B = \begin{pmatrix} a_{11}B & \cdots & a_{1n}B \\ a_{21}B & \cdots & a_{2n}B \\ \vdots & \vdots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{pmatrix}$$

Now that we know the notions of vector state and Tensor product, we can use them to define an important space (in particular for the SAT problem we are going to study later) called the **Hilbert space** and denoted with $\mathcal{H}$.

**Definition (Hilbert Space):** *Given the complex space $\mathbb{C}$ we define the Hilbert space $\mathcal{H}$ as the (n + 1)-tuple tensor product:*

$$\mathcal{H} := \bigotimes_1^{n+1} \mathbb{C}^2$$

As we said, in the Hilbert space we will carry the discussion of the SAT problem but to understand what the result of this tensor product actually defines we still need to give probably the most important definition. In order to represent a quantum state we use the so called *bra-ket notation* introduced in 1939 by Paul Dirac.

**Definition (Dirac/bra-ket Notation):** *Given a complex Euclidean space $\mathbb{S} \equiv \mathbb{C}^n$, $|\psi\rangle \in \mathbb{S}$ denotes a column vector, and $\langle\psi| \in \mathbb{S}^*$ denotes a row vector that is the conjugate transpose of $|\psi\rangle$, i.i. $\langle\psi| = |\psi\rangle$. The vector $|\psi\rangle$ is also called a ket, while the vector $\langle\psi|$ is also called a bra.*

The bra-ket notation allows us to define a quantum state, hence a vector that lives into a particular vector space. Its definition intrinsically defines also the result of combining two states living in the same state with the **inner product**, straightforwardly obtained from what we have just said: $\langle\psi|\phi\rangle$. This result is fundamental to define spaces that are higher than one only dimension as the Hilbert state presented before. To understand what $\mathcal{H}$ is we can now use two examples where we define the basis of the first two results obtained by doing the tensor product of the complex space $\mathbb{C}^2$ with itself. Remember that the basis of a space is the smallest set of linearly independent vectors that can be used to represents all the vectors that belong to that space.

**Example 1:** *Considering the basic case of $\mathbb{C}^2$ the basis can be trivially identified as:*

$$|0\rangle = \begin{pmatrix}1\\0\end{pmatrix} \qquad |1\rangle = \begin{pmatrix}0\\1\end{pmatrix}$$

**Example 2:** *Considering a single product, thus $\bigotimes_1^1 \mathbb{C}^2 = \mathbb{C}^2 \otimes \mathbb{C}^2$, we obtain the basis by multiplying in all possible ways the vectors of the basis of the previous example. We now have kets of dimension 2, thus vectors with 4 lines and 1 column:*

$$|00\rangle = |0\rangle \otimes |0\rangle = \begin{pmatrix}1\\0\end{pmatrix} \otimes \begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}1\begin{pmatrix}1\\0\end{pmatrix}\\0\begin{pmatrix}1\\0\end{pmatrix}\end{pmatrix} = \begin{pmatrix}1\\0\\0\\0\end{pmatrix}$$

$$|01\rangle = |0\rangle \otimes |1\rangle = \begin{pmatrix}1\\0\end{pmatrix} \otimes \begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}1\begin{pmatrix}0\\1\end{pmatrix}\\0\begin{pmatrix}0\\1\end{pmatrix}\end{pmatrix} = \begin{pmatrix}0\\1\\0\\0\end{pmatrix}$$

$$|10\rangle = |1\rangle \otimes |0\rangle = \begin{pmatrix}0\\1\end{pmatrix} \otimes \begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}0\begin{pmatrix}1\\0\end{pmatrix}\\1\begin{pmatrix}1\\0\end{pmatrix}\end{pmatrix} = \begin{pmatrix}0\\0\\1\\0\end{pmatrix}$$

$$|11\rangle = |1\rangle \otimes |1\rangle = \begin{pmatrix}0\\1\end{pmatrix} \otimes \begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}0\begin{pmatrix}0\\1\end{pmatrix}\\1\begin{pmatrix}0\\1\end{pmatrix}\end{pmatrix} = \begin{pmatrix}0\\0\\0\\1\end{pmatrix}$$

Thanks to these two examples we can generalize to the n + 1 case and obtain the definition of the Hilbert space we gave before.

Before starting with the quantum physics section where we will start by defining the smallest unit element we use to represent information, the so called **qubit**, we need to give one further definition. In order to perform operations on qubits we will consider (section 1.1.5) only a particular family of matrices called unitary matrices. These matrices allow to perform operations on qubits without modifying the basic properties of the quantum state and are defined as follows.

**Definition (Unitary Matrix):** *A complex square matrix $\mathcal{U}$ is unitary if $U^*U = UU^* = I$.*

Unitary matrices have significant importance in quantum mechanics because they are **norm-preserving**, this will be fundamental to identify the two main features of quantum operations that can now be introduced as: *apply a unitary matrix on a quantum state, thus a vector whose norm will be preserved.*

### 1.1.2 Quantum Physics

Quantum mechanics is a fundamental theory in physics describing the properties of nature. We do not need entirely its entire power for our purpose but, starting from some basic concepts we will exploit some conclusions that are useful to design an algorithm that is faster than its classical counterpart. As we do when we start studying computer science, we want to identify the smallest, most basic element that allows us to represent the information. From a classical point of view we have the **bit** whose values can be either 1 or 0. From a quantum point of view, instead, we have the **qubit**, complex variables that can assume values ranging from 0 to 1 (in modulus), identified in a complex space over a surface called the *Block Sphere.*

**Definition (Qubit:)** *The qubit is the smallest unit of measurement used to quantify information in quantum computing. It identifies the bit in a superposition, hence both 0 and 1 values are considered. Formally it is a vector of the space $\mathbb{C}^2$ represented as a linear combination of the elements contained in its basis (1.1.1). A qubit $\psi$, with $\alpha_0, \alpha_1 \in \mathbb{C}$, is defined as:*

$$|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle = \alpha_0 \begin{pmatrix}1\\0\end{pmatrix} + \alpha_1 \begin{pmatrix}0\\1\end{pmatrix}$$

To understand better the definition of the qubit we now provide its representation in a tridimensional space whose directions are obtained from basic linear algebra principles that are not now relevant. We now just need to understand that, thanks to the coefficients $\alpha_0$ and $\alpha_1$ belonging to the complex space $\mathbb{C}$ we are able to consider the basic element of our computation as one of the infinite points that live over the surface of a sphere. We can grasp in this concept a first hint in the advantages that quantum states provide with respect their classic counterpart.

**Definition (Block Sphere):** *The Block Sphere is the geometrical representation of the pure state space of a two-level quantum mechanical system. In other words it represents all the possible vectors that can be obtained by combining the vectors of the basis for a quantum register of 1 qubit.*

Consider the following examples to understand how vectors are represented in the Bloch sphere; we present the trivial cases for both $|0\rangle$ and $|1\rangle$ and the respective orthogonal vectors identifying the x and y axes.

**Example 3:** *Consider the qubit $\psi$, with $\alpha_0, \alpha_1 \in \mathbb{C}$, such that: $|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$. (Check code at 1)*

1. $\alpha_0 = 1, \ \alpha_1 = 0$

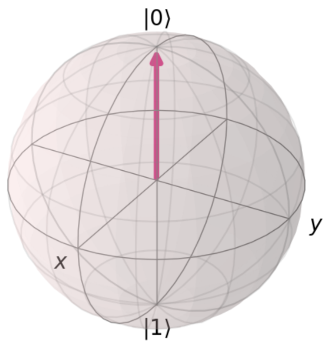

Figure 1: $|\psi\rangle = |0\rangle$

2. $\alpha_0 = 0, \quad \alpha_1 = 1$

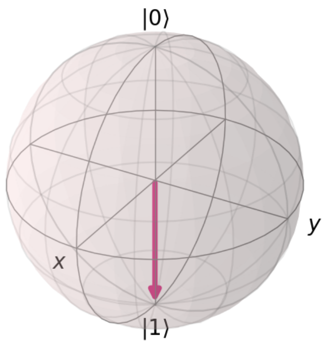

Figure 2: $|\psi\rangle = |1\rangle$

3. $\alpha_0 = \frac{1}{\sqrt{2}}, \quad \alpha_1 = \frac{1}{\sqrt{2}}$

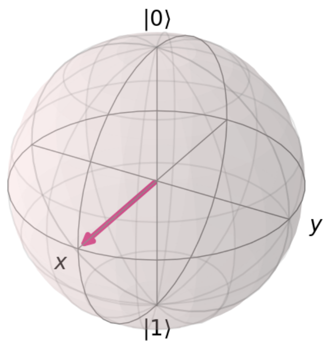

Figure 3: $|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$

4. $\alpha_0 = \frac{1}{\sqrt{2}}, \quad \alpha_1 = \frac{i}{\sqrt{2}}$

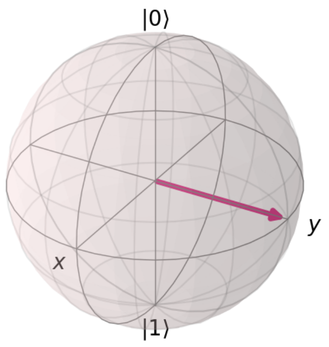

Figure 4: $|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + i|1\rangle)$

Obviously we can also represent vectors that belong to spaces with a higher dimension than 2, in this case we will have their representation over a hypersphere (check the plot_bloch_multivector method in the qiskit library). As in classical computer we haver registers, defined as a sequence of bits, in quantum computing we have *quantum-registers* composed of a sequence of qubits. Typically a quantum computer has a single quantum-register made up of qubits (see section 1.2). It is now important to remark the first important conclusion on quantum computing, that derives from the definition of quantum-register that we have just provided. We see that $\bigotimes_1^{n+1} \mathbb{C}^2$ is a $2^n$ dimensional space. This is sharp in contrast with what happens for classical registers: given n classical bits, their state is a binary string in $\{0, 1\}^n$, thus an n-dimensional space. In other words we arrive to the first important conclusion.

**Conclusion 1:** *the dimension of the state space of quantum registers grows exponentially in the number of qubits, whereas the dimension of the state space of classical registers grows linearly in the number of bits.*

### 1.1.3 Superposition

The mathematical definition of the qubit we have just seen is useful to understand the first difference between the representation of a state either in classical or quantum fields. We want now to define from a more physical point of view what means for a qubit to assume both the value 0 and 1. The basic physical principle behind this property is the *Heisenberg indetermination principle*, but without going too deep in the details let's try to understand it with a simple example.

**Example 4:** *We have seen that bits can assume at a certain time instant one and one value only. Considering now a qubit we could say that it assumed both the values 1 and 0 because its definition is based on Heisenberg's principle, basically stating that particles can assume at the same time different positions. This is a physics principle: electrons can be at the same time in different positions. That is why considering the position 0 and the position 1 we can say that a qubit is situated in both of them. The sad reality is that we do not know the real state of the qubit until we perform a specific operation on it, called measurement, which makes it collapse either to a 0 or a 1 "boring" bit. The following picture tries to illustrate the principle of an electron whose real position is not deterministic until this operation.*

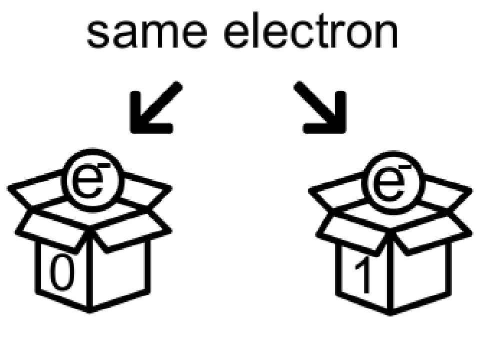

Figure 5: Superposition of the electron

As we may have understood superposition is a very interesting property, also because it can be generalized to the case of n qubits rather than just one. We will se in the implementation of the SAT algorithm how operations on multiple qubits are able to exploit superposition in order to compute the solution of the problem. With this property we can start guessing how algorithms will be run on our quantum computer: from a certain initial state we will perform operations that exploit the indetermination principle over the qubits, thus considering several states at the same time, until the end when performing the measurement we will make them collapse to a single string of 1s and 0s which is our result.

### 1.1.4 Entanglement

Entanglement is the second main feature of quantum computers that, together with superposition, differentiates them from quantum computers. These are in fact the two main features that are exploited in algorithms to find the solution of a problem.

To understand entanglement we start by answering the following question:

$\underline{What\ do\ we\ gain\ by\ moving\ from\ single\ qubits\ spaces\ to\ multiple\ qubits\ spaces?}$

The answer has two motivations, the first regarding the states representation the second the property of entanglement:

(i) As we described before, also in quantum computing we will need to define registers that contain more than 1 qubit. To do so it suffices to compose a vector of n qubits which identify the state obtained by the result of their quantum product.

(ii) Linear algebra definitions, in particular for what concerns the tensor products, show that not every state that we consider can be represented as the tensor product of n-qubits. Whenever we have to deal with a state of that kind we will say that the quantum state is entangled.

More formally the definition of entanglement is the following:

**Definition (Entanglement):** *A quantum state* $|\psi\rangle \in \bigotimes_1^{n+1} \mathbb{C}^2$ *is a product state if it can be expressed as a tensor product* $|\psi_0\rangle \otimes |\psi_1\rangle \otimes \cdots \otimes |\psi_n\rangle$ *of n 1-qubit states. Otherwise, it is entangled.*

To understand better the definition let's consider the simplest example possible: considering a state living in the $\mathbb{C}^4$ complex space we want to check if it is a product state or entangled by looking for two 1-qubit states whose tensor product is the state that we are dealing with. We will see next how the entanglement feature can be used, in particular what still happens when 2 qubits are entangled together.

**Example 5:** *Consider the following 2-qubit state:*

$$\frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle$$

*This is a product state because we can find two states whose tensor product is the starting one:* $\frac{1}{2}(|0\rangle + |1\rangle) \otimes \frac{1}{2}(|0\rangle + |1\rangle)$. *By contrast, the 2-qubit state:*

$$\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$$

*is an entangled state, because it can not be expressed as the tensor product of two 1-qubit states.*

The actual meaning of entanglement can be fully appreciated only once we have defined the measurement, but we can start grasping its importance thanks to the algebraic definition that we have just provided. Having an entangled state means that we can not find two qubits whose tensor product result is the one that we are considering, this because there are not two "linearly independent" states satisfying such property. In conclusion the qubits that compose an entangled state are in some way related one with the other, in other words: *when two or more qubits are entangled, they affect each other, and measuring one qubit changes the probability distribution for the other qubits.*

### 1.1.5 Operations on Qubits

The definition of qubit should be clear now, moreover we have also understood that a quantum computer is composed of a quantum register where multiple qubits are used in order to move from a state to another. We want now to understand how to perform operations on qubits (without breaking the basic properties of the quantum state) so that we can implement algorithms that allow to solve problems incrementally going through different quantum states. First of all we have to give the definition of a quantum operation on n qubits:

**Definition (Quantum Gates):** *An operation performed by a quantum computer with n qubits, also called a gate, is a unitary matrix in* $\mathbb{C}^{2^n \times 2^n}$.

Thus, for an n-qubit system, the quantum state is a unit vector $|\psi\rangle \in \mathbb{C}^{2^n}$, while a quantum operation is a matrix $U \in \mathbb{C}^{2^n \times 2^n}$, and the application of U onto the state $|\psi\rangle$ is the unit vector $U|\psi\rangle \in \mathbb{C}^{2^n}$. This leads to the following important features of quantum gates:

- Quantum operations are **linear**
- Quantum operations are **reversible**

Every significant operation on a quantum state must be represented as a unitary matrix, this may seem very restrictive but it has been proved that these two features do not remove any power to the quantum computer we want to design. We can now give the following important conclusion.

**Conclusion 2:** *A universal quantum computer is Turing-complete.*

Now that we know how quantum operations are formally defined we have to show first how they operate over a set of qubits and second which are the most important gates that we need to implement quantum algorithms. A quantum algorithm is implemented with a quantum circuit: a quantum circuit is represented by indicating which operations are performed on each qubit or group of qubits. For a quantum computer with n qubits, we represent n qubits lines and operations as blocks taking as input a set of qubits and with output the same input lines. Consider the following picture as the first trivial representation where a general unitary matrix $U$ is applied over all the qubits of the quantum computer.

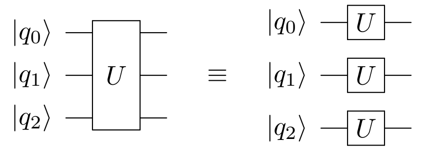

Figure 6: Trivial equivalence of quantum circuits

In the picture above we have used a slight abuse of notation for what concerns the multigate $U$ applied over the 3-qubit state. In mathematical terms, in fact, the equivalence holds when we consider the gate $U \otimes U \otimes U$ applied to the state composed of 3 qubits. Hence, the gate of the first circuit has to be interpreted as the unitary matrix obtained from the tensor product of 3 $U$ unitary matrices, thus living in the $\mathbb{C}^8$ space.

Before going to study the fundamental gates we need to implement quantum algorithms it is very important to understand how to interpret the representation of a quantum circuit. Circuit diagrams are read from left to right, but because each gate corresponds to applying a matrix to the quantum state, the matrices corresponding to the gates should be written from right to left in the mathematical representation. In the following picture, for example, the result of the circuit is the state $BA\,|\psi\rangle$, obtained by first applying gate $A$ and then $B$.

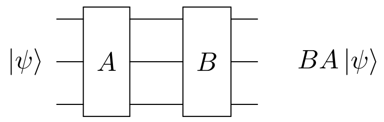

Figure 7: Quantum circuit interpretation

So far it seems that every kind of unitary gate is allowed to be used in a quantum circuit, but as we are not allowed in classical algorithms to define every kind of function also in quantum computing we will define operations as unitary gates by combining a set of nice matrices which are efficiently specifiable and implementable. The only set of nice matrices we will consider in our study (which suffices to implement the SAT algorithm) are called the *Pauli Operators* and they are defined as follows.

**Definition (Pauli Operators):** *The Pauli operators are four single-qubit unitary matrices $I, X, Y, Z$ forming a basis for $\mathbb{C}^{2\times 2}$ such that: $XYZ = iI$. The four matrices are:*

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \qquad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

*With the definition it can be checked trivially that all $I, X, Y, Z$ are unitary.*

Now that we know the Pauli set we can start to give the list of the most important operators we need to implement quantum algorithms by comparing them with the respective counterpart operations in classic computation:

- The $X$ gate is the equivalent to the NOT gate in classical computers. Thus we have: $X|0\rangle = |1\rangle$ and $X|1\rangle = |0\rangle$.
- The $Z$ gate has no equivalent in classical computers because it performs a phase-flip on the target qubit. Thus we have: $Z|0\rangle = |0\rangle$ and $Z|1\rangle = -1$.
- Another single-qubit fundamental gate is the Hadamard gate $H$. $H$ is still a unitary matrix that belongs to the class of the **Clifford** gates, the characteristic property of a Clifford gate is to transform a Pauli operator in another Pauli operator. The Hadamard gate is defined as:

  $$H := \tfrac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

  Applying a Hadamard to a qubit brings it to a superposition, we will see later that this is the fundamental initialization of several quantum algorithms. In fact: $H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$.

So far we have considered only single-qubit gates, let's continue our list with multiple qubit gates, starting from the basic ones and continuing with those obtained by computing a tensor product of the single qubit gates just seen.

- The CNOT gate, also called "controlled NOT", acts on a 2-qubit state. The two qubit are called *control* and *target* qubits, and the CNOT gate works as follows: the target qubit is inverted if and only if the control qubit value is $|1\rangle$. The unitary matrix representing the CNOT gate is defined as:

$$CNOT := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

As we may have guessed we have a clear analogy with the XOR gate in classical computers, in fact, as we see in the circuit representation the result is nothing more than the XOR between the control and the target qubit.

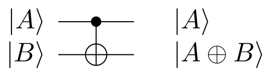

Figure 8: The CNOT gate

- The SWAP gate is used to swap the control with the target qubit and it can be obtained by using a sequence of three CNOT gates. The SWAP operation on a quantum state maps it to a new quantum state in which every basis state has its i-th and j-th digit permuted. The circuit definition is provided as an equivalence with the sequence of CNOT gates that allow to implement it.

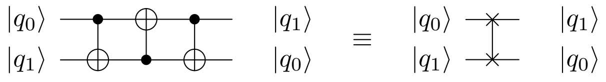

Figure 9: The SWAP gate

- The CNOT gate can be extended to more than two qubits only. If we consider two bits as control bits and one as target we obtain the "double controlled NOT" gate also known as the Toffoli gate. The generalized meaning of the CCNOT gate is: the control qubit is flipped if and only if both the control qubits values are $|1\rangle$. The unitary matrix representing the CCNOT gate is defined as:

$$CCNOT := \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

If we consider as target qubit $|0\rangle$ we clearly see the analogy between the CCNOT gate and the AND classical gate. The result of applying it to two general control qubits $|A\rangle$ and $|B\rangle$ yields in fact to the logical formula shown in the circuit representation.

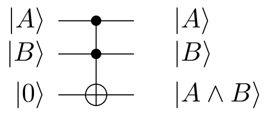

Figure 10: The CCNOT gate

- In general, as we saw at the beginning of this section we can build quantum gates by computing the tensor product of a set of nice unitary matrix, so to obtain another unitary matrix that maintains the basic properties of the quantum state on which we are acting. The most important example is to compute the Hadamard gate on all the n qubits of which a quantum computer is composed. This operations allows to bring the initial state in a *superposition* so that we can continue performing quantum gates to implement our quantum algorithm. The formal definition of an n-Hadamard gate is:

$$\bigotimes^n \mathcal{H} = \tfrac{1}{\sqrt{2}} \begin{pmatrix} \bigotimes^{n-1} \mathcal{H} & \bigotimes^{n-1} \mathcal{H} \\ \bigotimes^{n-1} \mathcal{H} & -\bigotimes^{n-1} \mathcal{H} \end{pmatrix}$$

Now that we know the basic quantum gates and their relative counterpart in classical computers we can start to play with them in an example. Knowing in fact how to define a NOT and an AND gate we can do everything by exploiting the De Morgan laws.

**Example 6:** *Consider the following clauses defining a 3-SAT problem over the variables $X_1, X_2, X_3$:*

$$C_0 = \{\neg X_1, X_2, X_3\}$$
$$C_1 = \{X_1, \neg X_2, X_3\}$$

*The problem is trivially satisfiable, but for now we are interested in its representation by using the gates that we have defined so far, knowing that the definition of satisfiability yields to check the possibility to assign the true value to the CNF of the clauses defining the problem. In our case the CNF becomes:*

$$CNF = (\neg X_1 \vee X_2 \vee X_3) \wedge (X_1 \vee \neg X_2 \vee X_3)$$

*And the quantum circuit corresponding to this instance can be obtained with the implementation available at the path:* /Code/Quantum/DecisionVersion. (Check the snippet of code for this example at (2))

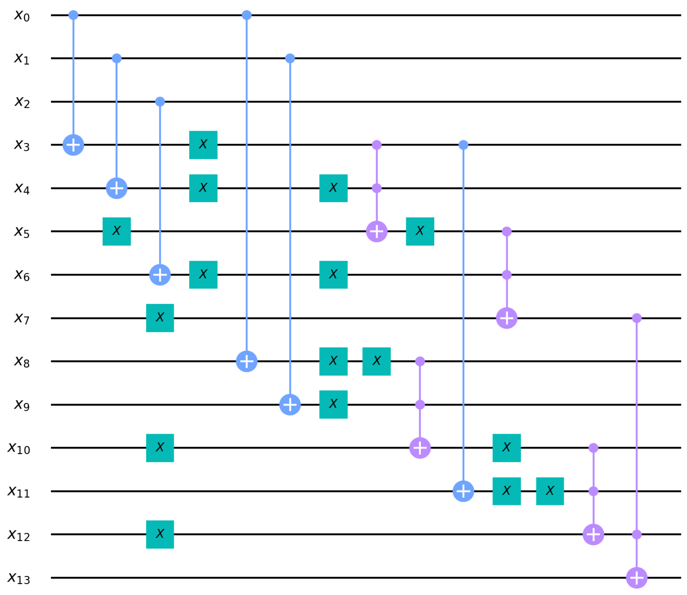

Figure 11: CNF circuit

### 1.2 The Quantum Computer

### 1.3 Grover's Algorithm

## 2 Computational Theory

### 2.1 SAT Problem

## 3 SAT Implementation

### 3.1 Classical

### 3.2 Quantum

### 3.3 Classical vs. Quantum

## 4 Conclusions

# Appendices

## A Qiskit

## B Code with Qiskit

This section contains the list of the snippets of code needed to represent the examples used in the paper. To replicate them consider ti import the following libraries:

```python
from matplotlib import pyplot as plt
import numpy as np
from qiskit import *
from qiskit.visualization import *
```

1.
   ```python
   figure1 = plot_bloch_vector([1, 0, 0], title="zeroket")
   figure1.savefig('zeroket.png')

   figure2 = plot_bloch_vector([-1, 0, 0], title="oneket")
   figure2.savefig('oneket.png')

   figure3 = plot_bloch_vector([0, 1, 0], title="xket")
   figure3.savefig('xket.png')

   figure4 = plot_bloch_vector([0, 0, y], title="yket")
   figure4.savefig('yket.png')
   ```

2.
   ```python
   # First I define the dimensions of the quantum circuit:
   # 3 qubits for the variables +
   # + 5 ancilla qubits for each clause
   # + 1 = 14 qubits
   qc = QuantumCircuit(QuantumRegister(14, 'x'))

   # A quantum circuit once initialized has all qubits set to 0
   # Hence we need to invert them when storing the clauses

   # Encoding of the first clause: C_0 = notX_1, X_2, X_3
   qc.cx(0, 3)
   qc.cx(1, 4)
   qc.x(4)
   qc.x(3)
   qc.x(4)
   qc.x(5)
   qc.ccx(3, 4, 5) # notX_1 or X_2 now encoded on qubit 5
   qc.cx(2, 6)
   ```

```
qc.x(6)
qc.x(5)
qc.x(6)
qc.x(7)
qc.ccx(5, 6, 7) # notX_1 or X_2 or X_3 now encoded on qubit 7

# Encoding of the second clause: X_1, notX_2, X_3
qc.cx(0, 8)
qc.cx(1, 9)
qc.x(8)
qc.x(8)
qc.x(9)
qc.x(10)
qc.ccx(8, 9, 10) # X_1 or notX_2 now encoded on qubit 10
qc.cx(3, 11)
qc.x(11)
qc.x(10)
qc.x(11)
qc.x(12)
qc.ccx(10, 11, 12) # X_1 or notX_2 or X_3 now encoded on qubit 12

# Now that we have both partial results
# we can compute the and of the two clauses
qc.ccx(7, 12, 13)
```

## References

[1] Qiskit textbook. `https://qiskit.org/textbook/`.

[2] Scott Aaronson. Np-complete roblems and physical reality. 2005.

[3] Stephen A. Cook. The complexity of theorem-proving procedures. 1971.

[4] V. Kreinovich E. Dantsin and A. Wolpert. On quantum versions of record-breaking algorithms for sat. 2005.

[5] Richard P. Feynman. Quantum mechanical computers. 1986.

[6] Lov K. Grover. A fast quantum mechanical algorithm for database search. 1996.

[7] T. Hertli and R. A. Moser. Improving ppsz for 3-sat using critical variables. 2018.

[8] S. Tamaki K. Makino and M. Yamamoto. Derandomizing hssw algorithm for 3-sat. 2011.

[9] K. Kutzkov and D.Scheder. *Using Constraint Satisfaction To Improve Deterministic 3-SAT*. 2018.

[10] Giacomo Nannicini. An introduction to quantum computing, without the physics. 2020.

[11] K. Y. Chwa O. Cheong and K. Park. *Algorithms and Computations*. Springler, 2010.

[12] M. Ohya and N.Masuda. Np problem in quantum algorithm. 2018.

[13] M. Ohya and Igor V. Volovich. New quantum algorithm for studying np-complete problems. 2018.