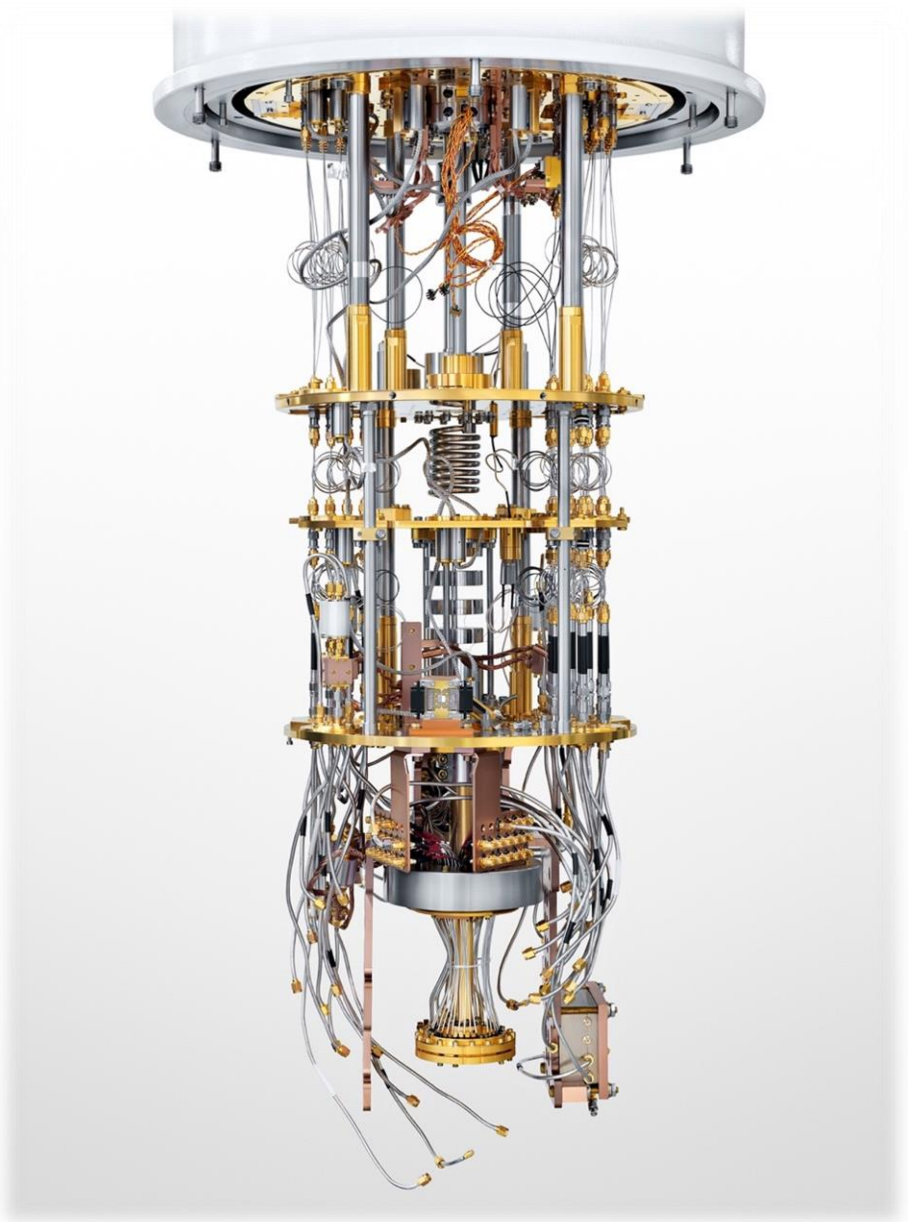


Solving Graph Coloring on a Simulated Quantum Computer

AN APPLICATION OF GROVER'S ALGORITHM

JONATHAN MAILLEFAUD | COMPUTER SCIENCE | 2020-2021

MAITRE ACCOMPAGNANT : M. PAUL MALEY

Abstract

Imagine having to find the shortest path to visit a certain number of places, without crossing the same place twice, like a travelling salesman. Mathematically, this problem can be viewed as coloring a graph and can only be completed in exponential time. In this research, we will look at one way to solve this problem faster using properties of quantum computers, more specifically using Grover’s Algorithm. We will provide a detailed analysis of the solution we successfully designed and implemented.

Acknowledgments

I want to thank M. Maley, my *Travail de Maturité* correspondent, for accompanying me through this project.

I also want to thank to Microsoft Quantum Kata Team and the Quantum Computing StackExchange community for the time taken to answer all my questions on quantum computing.

Contents

1 Introduction

When quantum computer scientist Peter Shor[1] showed his integer factorization algorithm using quantum computers in 1994[2], the public learned for the first time about the usefulness of quantum physics properties applied to computers. The algorithm could factorize large integers in logarithmic speed, making common encryption methods, like the RSA encryption method, unsafe.

Two years later, Lov Grover[3], another well-known computer scientist, released a paper in which he demonstrates a fast database search algorithm using quantum computers[4]. This algorithm possesses a broader range of usage than Shor's algorithm, and it can even be defined as inverting a function. This algorithm could search an unsorted database is time $O(\sqrt{x})$, where x is the size of the problem, which is faster than the time $O(x)$ used by classical computers[5]. This brings a considerable speedup to certain problems, especially the NP problems.[6] One of these problems is the graph coloring problem.

1.1 PROBLEMATIC

In this research work, we will look at a new take on graph coloring using quantum computing to speed up certain processes. The question guiding our reflection will be : "To which extent can Grover's algorithm simulated on a classical computer,

[1] Peter Shor (1959-), professor of applied mathematics at MIT, known for his work on quantum computation, mostly the discovery of a fast integer factorization algorithm

[2] SHOR, Peter. *Algorithms for quantum computation: discrete logarithms and factoring*. Proceedings 35th Annual Symposium on Foundations of Computer Science, 1994. [online] [Consulted 12 December 2020] Available at : https://ieeexplore.ieee.org/document/365700

[3] Lov Grover (1961-), computer scientist specialized in quantum computing, developed a quantum algorithm which can be used to solve many problems in a shorter time than classical computers

[4] GROVER, Lov. *A fast quantum mechanical algorithm for database search*. Proceedings, 28th Annual ACM Symposium on the Theory of Computing (STOC), May 1996, pp. 212-219 [online] [Consulted 5 January 2021] Available at : https://arxiv.org/abs/quant-ph/9605043

[5] In this research work, we will denote as "classical", elements which do not possess the properties of the quantum world.

[6] NP is a complexity class in theoretical computing, which is defined as the class of problems where the solution can be checked in polynomial time. For more information see *Annex 1*

help to solve graph coloring problems ?". The usage of quantum computers being very costly and not available to the public at large scale, we will use a programming language called Q# to simulate the quantum parts of the algorithm. This will limit our capabilities to simulate large instances of the algorithm, but we will have a better understanding of what is happening at any given moment, as we will be able to look at the state of the emulated quantum computer at any time[7]. We will start by explaining what graph coloring is and how to solve graph coloring on a classical computer. Next, we will explain the basics of quantum computers needed to understand the algorithm. We will then look at Grover's Algorithm and find a usage for the algorithm in the coloring of graphs. Finally, we will review the advantages and disadvantages of our method to solve the graph coloring problem and discuss key numerical results.

2 Graph Coloring

2.1 DEFINITION OF A GRAPH

Graph theory is a wide and complex subject. To prevent any ambiguous cases from happening, we will limit ourselves to undirected simple graphs. We will define a graph as follows :

As per the definition by S. Gill Williamson,[8] a graph, named G, is a pair $G = (V, E)$ composed out of :

- V, a set of vertices (also called node or points)
- $E \subseteq \left\{\{x,y\} \middle| \; x,y \in V \text{ and } x \neq y\right\}$, a set of edges which link a pair of vertices, these vertices must be part of V and cannot be the same

A simple example would be this graph with 8 vertices and 14 edges.

[7] In a quantum computer, measuring the state of a system changes the state itself and only offers a glimpse of what is really happening.

[8] WILLIAMSON, S. G. *Lists, Decisions and Graphs.* San Diego, 2010. p, 148

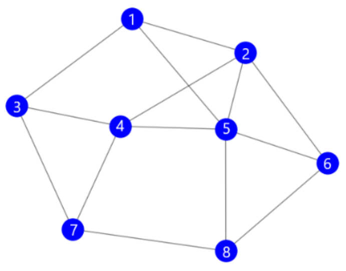


Figure 1 : A simple graph with 8 vertices and 14 edges

We will assign a number to each vertex, starting from 0, from left to right, from top to bottom. In this case, $V = \{0, 1, 2, \cdots, 6, 7\}$ where 0 is the top left vertex, 1 is the top right vertex and 7 the bottom right vertex. Furthermore, $E = \{0,1\}, \{0,2\}, \{0,4\}, \{1,3\}, \{1,4\}, \{1,\ 5\}, \{2,3\}, \{2,6\}, \{3,4\}, \{3,6\}, \{4,5\}, \{4,7\}, \{5,7\}, \{6,7\}$ where the first pair represents the edge connecting vertex 0 to vertex 1, the second pair represents the edge connecting vertex 0 to vertex 2, and the last pair represents the edge connecting vertex 6 to vertex 7. We now have a way to represent graphs.

2.2 COLORING GRAPHS

Coloring graphs means assigning a color to every vertex, so that no two vertices sharing the same edge, have the same color.[9] Since naming colors can be long and tedious, we will replace colors by numbers, which will simplify our results. A coloring will be represented with an array with as many elements as there are vertices. A quick verification shows that *Figure 1* can be colored with minimum 3 colors. For example, a valid coloring could be $\{0,1,1,0,2,0,2,1\}$ which means that the first vertex (vertex 0) could be colored using color 0, vertex 1 could be colored using color 1 and vertex 6 using color 2.

[9] Graph coloring. [online]. 7 August 2020 [Consulted 9 August 2020 13:34]. Available at : https://en.wikipedia.org/w/index.php?title=Graph_coloring&oldid=971607267

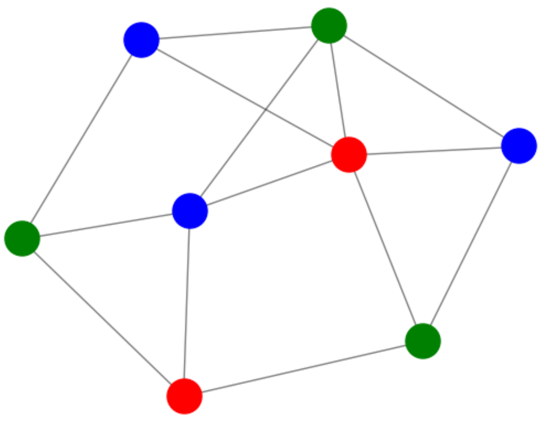


Figure 2 : A colored graph

On classical computers, to color undirected simple graphs, the general solution[10][11] requires to brute force the solution. This implies to test every possible solution until a correct one is found. Brute forcing is inefficient, as the problem takes exponentially more time, with the problem complexity increasing.

In our representation of a graph, to verify the correctness of a solution, we have 3 inputs :

- The list of vertices which are named with numbers from 0 to n (where n is the number of vertices). These numbers can be simplified to one number, as the list always starts at 0. So, when there are 5 vertices, we know that they are named $\{0, 1, 2, 3, 4, 5\}$.
- The list of edges where an edge is defined by the vertices it connects. The edge connecting vertices 1 and 2 is defined as $\{1, 2\}$. The vertices need to be in the list of vertices and cannot be the same.

[10] To find if the graph can be colored with 0, 1, 2 or 4 colors can be computed in polynomial time, however, most of the graphs can be colored with 3 colors which is NP-complete. Currently, the best algorithm computes a 3 coloring in $O(1.3289^n)$.

[11] BEIGEL, R. and D. EPPSTEIN. *3-Coloring in Time O(1.3289ˆn)*. Computer Research Repository, arXiv:cs/0006046, 2000. [Online] [Consulted 12 December 2020] Available at : https://www.sciencedirect.com/science/article/abs/pii/S0196677404001117

- The colouring to test. It is in the format of a list where the n^{th} element corresponds to the colour of the n^{th} vertex.

To check if the coloring is correct, we will look at every edge to ensure that the color of the vertex on one side is different from the vertex on the other side. If all colors are different, the coloring is correct.

This is the way to color graphs with a classical computer. We will now look at quantum computers and how they can help to solve this problem faster.

3 Properties of Quantum Computers

3.1 DIFFERENCES BETWEEN CLASSICAL AND QUANTUM COMPUTERS

Classical computers operate with bits. Bits are the element which compose any data structure in computers.[12] A bit can be in two states: on, represented by a 1, and off, represented by a 0. Putting several bits together can create a structure, called system, which can be in several states. For example, 3 bits can be in the states 000, 001, 010, 011, 100, 101, 110, 111. The amount of states a system can be in is called N and is equal to 2^n where n is the number of bits.[13]

Quantum computers have the same information base, only that quantum bits, named qubits, have two more capabilities : they can be in a superposition and they can be entangled[14]. Since we do not make use of entanglement in our problem, we will omit it from our explanation.

3.2 SUPERPOSITION

We already know that with n bits we can create 2^n states, called basis states. A state $|\psi\rangle$ is in superposition when it is in a "blending" of these states.[15] For example, for 2 qubits, the basis states are 00, 01, 10 and 11. When a state $|\psi\rangle$[16] is in superposition of some or all these basis states, it is a certain amount in each state. In the example $|\psi\rangle = \frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle$, the system is the same amount in every basis state with 2 qubits. Such a state is named an equal superposition[17]. More generally, for n qubits, a state in superposition can be written as $|\psi\rangle = c_0|x_0\rangle + c_1|x_1\rangle + \cdots +$

[12] SHANNON C. E. *A mathematical theory of communication*. The Bell System Technical Journal, vol. 27, no. 4, July 1948, pp. 379-423. DOI: 10.1002/j.1538-7305.1948.tb00917.x.
[13] DE WOLF, R. *Quantum Computing: Lecture Notes*. 2019 [online] [consulted 12 August 2020] Available at : https://arxiv.org/abs/1907.09415
[14] PERRY, Riley, 2012. *Quantum Computing from the Ground Up*. Singapore : World Scientific Publishing. p. 116
[15] YANOFSKY, Noson and Mirco MANNUCCI. *Quantum computing for computer scientists*. Cambridge: Cambridge University Press, 2018. p, 107
[16] The usage of the symbols "|" and "⟩", called Dirac Notation, is explained in *Annex 4*
[17] Qubit. In : *Wikipedia: The Free Encyclopaedia* [online]. 15 April 2020 [Consulted 18 May 2020]. Available at : https://en.wikipedia.org/w/index.php?title=Qubit&oldid=951161218

$c_{n-1}|x_{n-1}\rangle$, where $c_0, c_1,\ldots, c_{n-1}$ represent the complex amplitudes of the states $x_0, x_1,\ldots, x_{n-1}$, which are the basis states.[18] When all complex amplitudes are equal, we call the state an equal superposition.

3.3 MEASUREMENTS

Measuring is an operation that seems trivial in classical computing but must be paid attention to in quantum computing. This is due to quantum properties, such as Heisenberg's uncertainty principle which states that only a certain amount of knowledge can be gained from a quantum system.[19] Measuring a quantum system that is in a basis state does not change the system at all.[20] For example, if $|\psi\rangle = 1|000\rangle$, every time we measure the system, we will find it in the basis state $|000\rangle$. The change occurs when a system in superposition is measured.[21] In this case, measurement will force the system to "collapse" to one of the basis states.[22] As another example, if $|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$, when the qubit is measured, we can measure either $|0\rangle$ or $|1\rangle$. The probability of measuring a basis state is given by the square of its complex amplitude: $P(x_n) = |c_n|^2$. In this case, we can measure $|0\rangle$ or $|1\rangle$ with the same probability of $\frac{1}{2}$. The sum of the probabilities must always be equal to 1.[23]

3.4 GATES

We still need to know how to affect all these states. To modify states, we use gates. All single qubit gates are represented by a unitary matrix.[24] We can see the effect of a gate on basis states by looking at the columns of a matrix. If we take a gate A,

[18] YANOFSKY, Noson and Mirco MANNUCCI. *Op cit.* p, 144
[19] HEISENBERG, W. *Über den anschaulichen Inhalt der quantentheoretischen Kinematik und Mechanik*. Zeitschrift für Physik 43, pp. 172-198, March 1927. [online] [Consulted 04 January 2021] Available at : https://doi.org/10.1007/BF01397280 [Access with subscription]
[20] Since the complex amplitude of a basis state is 1, we will always measure it as $|1|^2 = 1$
[21] YANOFSKY, Noson and Mirco MANNUCCI. *Op cit. p, 107*
[22] *Ibid. p, 107*
[23] NIELSEN, Michael and Isaac CHUANG. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge : Cambridge University Press, 2010. p, 85
[24] YANOFSKY, Noson and Mirco MANNUCCI. *Op cit.* p, 144

represented by the matrix $A = \begin{bmatrix} \alpha & \gamma \\ \beta & \delta \end{bmatrix}$, the effect of the gate on the state $|0\rangle$ will be $A(|0\rangle) = \alpha|0\rangle + \beta|1\rangle$ and the effect on the state $|1\rangle$ will be $A(|1\rangle) = \gamma|0\rangle + \delta|1\rangle$. It should be added that $|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and $|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$. More information about the gates used in my program can be found in *Annex 5*.

3.5 PHASE

Phase is a value that arises from different parts of quantum physics.[25] The phase has a mathematical meaning when we write complex numbers in polar form, $z = re^{i\theta}$, in this case θ is the phase. However, the phase also has a physical meaning. It arises when a state evolves in accordance with the Schrodinger equation, $i\hbar\frac{d}{dt}|\psi\rangle = \widehat{H}|\psi\rangle$.[26] The phase is also responsible for wave interference.[27] In quantum computing, the phase is used as a value which can be changed and used in computations.[28] The phase does not affect the complex amplitude, hence, it does not affect the result of the measurement of a system.[29] There are two types of phases, the global phase, and the relative phase.[30] We will take the $|-\rangle$ state to show an example of both :

3.5.1 Global Phase

A global phase exists when two states differ by a global complex factor.[31] The global phase does not affect measurements and only has a mathematical meaning and can therefore be converted from one to another. This can be seen with the effect of the

[25] NIELSEN, Michael and Isaac CHUANG. *Op cit*. p, 93
[26] FERNANDES, M. C. B., KHANNA, F. C., MARTINS, M., SANTANA, A. and J. VIANNA. *Non-linear Liouville and Shrödinger equations in phase space*. Physica A: Statistical Mechanics and its Applications, Elsevier BV, vol. 389, issue 17, pp. 3409–3419, 2010. [Online] [Consulted 14 October 2020] Available at : https://doi.org/10.1016/j.physa.2010.04.030
[27] *Ibid.*
[28] CLEVE, R., EKERT, A., MACCHIAVELLO, C. and M, MOSCA. *Quantum Algorithms Revisited*. Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences, vol. 454, pp. 339-354, March 1998 [online] [Consulted 12 December 2020] Available at : https://arxiv.org/abs/quant-ph/9708016
[29] NIELSEN, Michael and Isaac CHUANG. *Op cit*. p, 93
[30] *Ibid.* p, 93
[31] *Ibid.* p, 93

X gate on the $|-\rangle$ state. $X(|-\rangle) = X(\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle) = \frac{1}{\sqrt{2}}|1\rangle - \frac{1}{\sqrt{2}}|0\rangle = -1(\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle) = -|-\rangle$. In this case, -1 is the global phase. It can be safely ignored when we want to measure the state.

3.5.2 Relative Phase

We talk about a relative phase when two complex amplitudes differ by a relative complex factor[32]. For the sake of simplicity, we will only describe it as the operator that separates one basis state from the others. For example, in the $|-\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$ state, the relative phase is of -1, since there is a minus sign between the $|0\rangle$ and the $|1\rangle$.

We now understand the main properties of quantum computers. We will look at Grover's Algorithm and how it helps us solve the graph coloring problem.

[32] NIELSEN, Michael and Isaac CHUANG. *Op cit*. p, 93

4 Grover's Algorithm

The algorithm, discovered by Lov Grover in 1996[33], uses quantum superposition to invert functions[34]. It helps us find an element with specific properties in a database, using a function that returns True when the element is the one searched for. We will use this search method to obtain a solution to our graph coloring problem, which requires that no two vertices connected with an edge have the same color.

An example for this algorithm is to find a needle in a haystack. Usually, each element of the haystack has to be looked at and analyzed, whether it is a haulm or a needle. In the example, the unsorted array is the haystack, the correct element is the needle, all the other elements are the haulms and analyzing an element is a query. To come back to the computer science version of the problem, we have an unsorted array of length x and in the worst case we have to do x queries to find the correct element (if it is the last element we look at).[35] This is the method used with classical computers.

However, if we use quantum computers, the number of queries required to be sure to find the correct element is $\sqrt{x}$.[36] This means Grover's Algorithm provides quadratic speedup to the search. This speedup is possible thanks to the usage of quantum superposition.

Grover's algorithm assigns to each basis state of a superposition an element of the array of size m. This can be achieved by applying the H gate on every qubit of the array. Since n qubits can make 2^n basis states. We will need at least $n \approx \log_2(m)$, and we will always have to round up. The algorithm then repeats a process to increase the probability of measuring the correct state, this means that it increases the complex amplitude of the correct state. This process is called Grover's iteration and can be separated into two parts :

[33] GROVER, Lov. *op cit.*
[34] *Ibid.*
[35] YANOFSKY, Noson and Mirco MANNUCCI, 2018. *Op cit.* p, 196
[36] *Ibid.* p, 196

4.1 GROVER'S ITERATION PART 1 : DESCRIPTION OF U_ω

The first part aims at separating the correct answers from the remaining ones. It requires a phase flipping oracle, called U_ω. An oracle is a function which recognizes if the input is correct[37]. In our case the oracle will flip the phase if the input is correct.

We will need a function : $f(x) = \begin{cases} 1, & x \text{ is true} \\ 0, & x \text{ is false} \end{cases}$. This function will return 1 when the answer is correct and 0 for an incorrect answer. The phase flipping can be viewed with this equality : $|x\rangle = (-1)^{f(x)}|x\rangle$. $|x\rangle$ represents the whole system which must be in a superposition. Flipping this phase can be complicated in quantum computations, since it requires careful handling of superposition states with complex gates. Another type of oracle is easier to implement : a marking oracle.

A marking oracle is an oracle which flips the state of a qubit, when the answer is correct.[38] This is easier to execute than a phase flipping oracle, as it only requires the X gate. The goal is then to have the marking oracle produce the same effect as the phase flipping oracle. This can be achieved using the phase kickback trick. When the X gate is used on the state $|-\rangle$ we obtain the state $-|-\rangle$. We can observe that we only multiply it by -1, this means that we applied a relative phase of -1[39]. Now we know how to implement a phase flip with a marking oracle.

Finally, we have a way to differentiate the correct element from the remaining ones, since its phase is negative compared to all others. However, since the phase does not affect the probability of the system collapsing to a specific state, we still have the same probability of measuring the correct state as the probability of measuring any other state. Consequently, since there are more incorrect elements than correct ones, only applying U_ω has a much higher probability of returning an incorrect element than a correct one.

[37] VAN MELKEBEEK, D. *Randomness and Completeness in Computational Complexity.* ACM Doctoral Dissertation Award Series, LNCS 1950. Springer-Verlag, 2000.
[38] VAN MELKEBEEK, D. *Op cit.*
[39] See *3. Properties of quantum computers*

4.2 GROVER'S ITERATION PART 2 : DESCRIPTION OF U_s

To increase the probability of measuring a correct element, we use Grover's diffusion operator. The operator is defined as $U_s = |\psi\rangle\langle\psi| - I$. The operator takes the average of all complex amplitudes and flips each one around this mean.[40] Consequently, the probability of measuring the incorrect elements decreases, since the average will always be closer to the incorrect elements as there are more incorrect elements than correct ones. At the same time, given that the correct elements are further away from the mean, the probability of measuring correct elements increases, as their complex amplitude is negative, whereas the complex amplitude of incorrect elements is positive.

$|\psi\rangle\langle\psi| - I$ does not explain how to implement this operator with gates we know. We already know that the system is in an equal superposition of all the basis states, which we can get from applying the H gate to every qubit. Therefore, we can convert $|\psi\rangle\langle\psi| - I$ to $H^{\otimes n}(|0\rangle\langle 0| - I)H^{\otimes n}$. With calculations that are detailed in *Annex 2*, we get the result that $|0\rangle\langle 0| - I$ is equal to applying the X gate to every qubit, doing a controlled Z gate, and applying X to every qubit again. In this case, all except the last qubit are 'control' and the last qubit is 'target'.

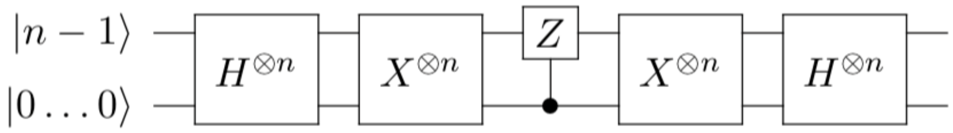


Figure 3 : Grover's diffusion operator in basic gates

4.3 FURTHER OBSERVATIONS ON GROVER'S ALGORITHM

Proving the number of Grover's iterations required is beyond the scope of this paper, however, we can get an intuition of its value. After $\sqrt{N}$ iterations, the complex amplitude of the correct state is so high that, when the incorrect states are flipped around the mean, their complex amplitude becomes negative. Any further iteration will increase the complex amplitude again and therefore increase the probability of measuring the incorrect elements.[41]

[40] GROVER, Lov. *op cit.*
[41] *Ibid.*

The attentive reader may notice the uncertainty of measuring a correct element. For this reason, we must verify the solution with the oracle after each measurement and, in the case of an incorrect solution, we will try again.

A further question that arises is how the oracle deals with multiple correct items. We will try every natural number of iterations, until we find a solution, starting from 1. It was proven that the algorithm finds a solution in time expected in $O(\sqrt{\frac{N}{t}})$ where t is the number of solutions.[42] However, as we do not know the exact number of colorings our graph has, we cannot use this fact. Given the sizes of graphs I considered, the number of iterations was always between 1 and 10.[43]

4.4 SUMMARY OF THE ALGORITHM

Now that we have all the parts of the algorithm, we can look at all the steps together :

1) Start in state $|0\rangle$
2) Repeat with $t \in [1, \infty[$ until correct answer is found
   a) Apply H gate to every qubit ($H^{\otimes n}$)
   b) Repeat t times :
      i) Apply operator U_ω
      ii) Apply Grover diffusion operator U_s
   c) Measure the system
   d) Test if the measurement is correct

[42] GROVER, Lov. *op cit*.

[43] A formal proof of Grover's algorithm can be found in : GROVER, L. K. *From Schrödinger's equation to the quantum search algorithm*. Pramana, Springer Science and Business Media LLC, vol. 56, pp. 333–348, 2001 [online] [Consulted 12 December 2020] Available at : https://arxiv.org/abs/quant-ph/0109116

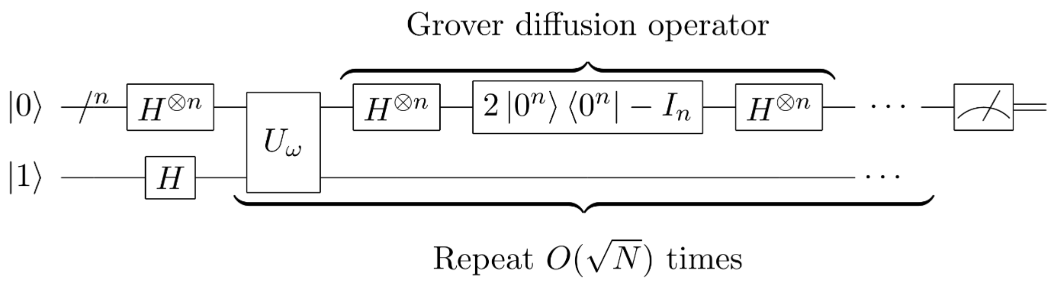


Figure 4 : Quantum Circuit representation of Grover's Algorithm[44]

4.5 THE MARKING ORACLE

We can now use Grover's algorithm to solve graph coloring with a quantum computer. To perform the algorithm, we need to provide it with an oracle, that can be executed on a quantum computer.

The inputs of the oracle are a) the edges of the graph, b) the register of qubits which represents a specific color, c) a target qubit to flip if the coloring is correct. We will create an array which contains a qubit for every edge, and if this edge has the same colors on both sides, we will flip the corresponding qubit. After we have analyzed every edge, we verify that no qubit has been flipped, and in that case, we would flip another qubit. This is the quantum method to check whether every edge is correctly colored. The qubit we flip will be specially prepared to create a phase flip, like described in the definition of U_ω.

Since every qubit must be released in state $|0\rangle$, we must apply the edge check twice.

[44] Bender2k14 - *Created in LaTeX code using Q-circuit.*, CC BY-SA 3.0, [online] Available at : https://commons.wikimedia.org/w/index.php?curid=13524800

5 My Implementation : Coding the Coloring Process

The first element I coded was the operation implementing Grover's Algorithm. It requires an oracle (flipping a qubit if the solution is correct), a register containing a correct element and the number of iterations applying U_ω and U_s. Decomposing $U_s = |\psi\rangle\langle\psi| - I$ into basic gates was challenging for me and required linear algebra tools, see *Annex 2*. Finally, I decomposed it in X gates, H gates and a controlled Z gate. These operations are not complicated to execute on a quantum computer, making this a good solution to produce the effect of Grover's diffusion operator. After having implemented Grover's Algorithm, the next step was to create a phase flipping oracle to create U_ω.

The phase flipping was complicated for me, as it requires turning $|\psi\rangle$ into $-|\psi\rangle$. Initially, I could not find a simple and efficient way to flip the phase, once the correct basis state was found. After further reading, I found a solution using the phase-kickback trick. The phase flipping can be converted into a state flipping, using the fact the $X(|-\rangle) = -|-\rangle$. This enabled me to use a state flipping oracle, which is easier to do than creating a phase flipping oracle.

The oracle checks whether the colors of every edge are different. As the code has to be executed by a quantum computer and the information has to be stored on qubits, the usable operations are restricted to quantum gates. I chose to represent the colors with two qubits, since two qubits can be in 4 different basis states. To compare both colors, I found CNOT gates to be the most practical. CNOT gates, when given two qubits, turn the second qubit into state $|0\rangle$, if both qubits are equal, or into state $|1\rangle$ if they are different. For the two colors : $a = [q_{a1}; q_{a2}]$ and $b = [q_{b1}; q_{b2}]$; I applied CNOT on q_{a1} ; q_{b1} and on q_{a2} ; q_{b2}. This resulted in turning b into $[|0\rangle; |0\rangle]$ if $a = b$. To check if $b = [|0\rangle; |0\rangle]$, I decided to use a controlled operation flipping a target qubit, for which I had to create an array of qubits with one qubit for every edge. The qubit would be flipped if the coloring of the edge was wrong. When initialized, the array is in state $|0 \ldots 0\rangle$. This means that every time an edge with a wrong coloring is found, the corresponding qubit is turned into a $|1\rangle$. Using the same trick as before allowed me to check that no edge was affected with a controlled operation. Once I had found a working state flipping oracle, I had to find a way to read the coloring of an array of qubits.

To read the coloring of an array of qubits, I had to transform the array. As the colors are represented by 2 qubits, I first separated the coloring into chunks of size 2. Then, I measured the chunks of 2 in the PauliZ basis. However, the colors would still be in binary form. To make them easier to read, I converted them to base 10 integers. This only left me to link all the parts together.

To summarize the process, I ran Grover's algorithm, providing it with a) an array of two qubits for each vertex, b) an oracle which flips a qubit if the answer is correct and c) the number of iterations to apply U_ω and U_s. However, Grover's algorithm only finds the correct solution with a certain probability, as the array could still collapse to a wrong state, when I measured it. Consequently, after measuring the array I had to apply the oracle again and measure its target qubit. If the measurement returned the state $|1\rangle$, I knew the answer was correct. If the measurement returned the state $|0\rangle$, I knew the answer was incorrect and I had to retry with one more iteration.

I wanted my implementation to resemble the regular architecture of a quantum processor, in which the quantum processor is used as a secondary processor to execute certain tasks. So, I created a python host program which would import the compiled Q# program and be used to transform the data it outputs. Additionally, I used two python libraries to draw the graphs in a plot. This is how I solved graph coloring with a simulated quantum computer. The code I created can be seen in *Annex 3*. To have a basic understanding of its structure, it is helpful to consult *Annex 6*.

6 Results

My implementation of coloring graphs has always successfully returned a correct coloring if given enough time and machine resources for graph sizes below 8 vertices. With 9 vertices and above my computer was not powerful enough.

There is a theoretical restriction to my implementation, which I did not encounter during my test runs. Even with infinitely low probability, there is always a chance that my program could run infinitely. This is due to the measurement in the main part of the implementation. The measurement makes the system collapse to a specific state with a certain probability. In our case, the measurement will collapse the system to the state representing the correct coloring with high probability, but there is still a chance for the system to collapse to an incorrect state. In such a case, I would have had to reapply Grover's Algorithm with one more iteration.

To reflect further on the results and to ultimately compare the simulation with a real quantum computer, we will look at the numerical results of my implementation. To calculate the execution time on a quantum computer based on the simulation time, we will look at the performance of my program and introduce key concepts to facilitate the understanding of the results.

6.1 PERFORMANCE

One of the best ways to quantify the performance of an algorithm is to calculate its complexity. The complexity can be defined as the function which gives the resources used when given the size of the inputs. This function is often an approximation as calculating the exact resource consumption is often impossible.[45] However, when the program can be decomposed into basic elements, for example gates, the complexity can be calculated more easily, since only an approximation of resource consumption of the basic element is needed. This can be done for the parts of the program requiring no measurement, mainly the oracle and Grover's algorithm. As a resource estimation cannot simulate probabilities, we cannot perform any measurements without providing a probability for the measurement. However, measuring does not count as a gate and

[45] WIKIPEDIA CONTRIBUTORS. *Big O notation*. Wikipedia, The Free Encyclopedia. [online] [Consulted 21 December 2020] https://en.wikipedia.org/w/index.php?title=Big_O_notation&oldid=994900730.

we can simply omit it from the resource estimation. As counting gates by hand is tedious, I used a tool provided by the Q# environment to decompose our program into a set of basic gates, which need to be universal.

6.2 UNIVERSAL GATES

Classical circuits can always be decomposed into a pattern of NAND gates or a pattern of NOR gates because the NAND and NOR gates are universal gates.[46] Quantum computers also have sets of universal gates. Taking all the gates in one of these sets, any quantum circuit can be created. One of these groups is the Clifford + T group, which is the one I decomposed my program in.[47] Clifford gates were of minor importance when I calculated the difference in speed between my implementation and the quantum computer, as they can be simulated on a classical computer with the same speed and efficiency as they can be run on a quantum computer, as stated by the Gottesman-Knill theorem.[48] Therefore, the interesting number is the number of T gates. This number depends on the number of iterations our program has and on the size of the graph we provide.

6.3 T GATES

We will start with an example, a 10 vertex graph with 21 edges can be coloured with 2436 T gates, with 3 iterations. This could seem like a large number, but with an execution time of a couple milli-seconds for a T gate on a regular quantum computer, the execution time is still reasonable. Furthermore, the number of T gates is growing linearly with a growing number of iterations. This means the number of T gates can be

[46] SHEFFER, H. M. *A set of five independent postulates for Boolean algebras, with application to logical constants*. Transactions of the American Mathematical Society, vol. 14, pp. 481-488, July 1913. [online] [Consulted 12 December 2020] Available at : https://www.ams.org/journals/tran/1913-014-04/S0002-9947-1913-1500960-1/S0002-9947-1913-1500960-1.pdf
[47]GOTTESMAN, Daniel. *The Heisenberg Representation of Quantum Computers*. Group22: Proceedings of the XXII International Colloquium on Group Theoretical Methods in Physics, eds. S. P. Corney, R. Delbourgo, and P. D. Jarvis, pp. 32-43 (Cambridge, MA, International Press, 1999), July 1998. [online] [Consulted 12 December 2020] Available at : https://arxiv.org/abs/quant-ph/9807006
[48] AARONSON, S. and D. GOTTESMAN. *Improved simulation of stabilizer circuits*. Physical Review A, American Physical Society (APS) vol. 70, 2004. [online] [Consulted 12 December 2020] Available at : https://arxiv.org/abs/quant-ph/0406196

expressed with a function $f(i) = \alpha * i$ where i is the number of iterations and α a coefficient depending on the size of the graph. The linear growth means that a higher number of iterations, caused by “bad luck” during the measurements, does not add as much runtime as an exponential growth, for example the growth caused by the coloring of a graph with more edges or vertices. The question is how to estimate the execution time of a T gate.

First, the execution time of a T gate is dependent on the type of quantum computer, a number of quantum computers are even created with CCNOT gates implemented natively, making the creation of T gates unnecessary. This could ease our program as we have a “controlled not” gate for every edge, plus one gate, every time we invoke the vertex coloring oracle. I took a scenario and imagined that we do not have a natively implemented CCNOT gate. T gate execution time depends on two factors, the space-time cost, due to the need to do a magic distillation[49] of the T gate, and the reaction time cost, because teleporting through a T gate applies its inverse half of the time.[50] Whereas these two factors add a lot to the execution time of a T gate, they do not give us an exact value. The bottom line is that there is no real time estimation of a T gate easily available yet. Google, however, in their quantum supremacy article, said that they sampled a random state 2 million times in 2 hundred seconds, with an average time of 0.1 milliseconds per sample.[51] If we use the same approximate time for a T gate, this gives us a reasonable estimation of the runtime of a T gate and therefore for our program on quantum hardware. To come back to our previous example, this means we have a runtime of $2436 * 0.001 = 4$ minutes to solve the graph on a quantum computer which seems reasonable.

[49] BRAVYI, S. & KITAEV, A. *Universal Quantum Computation with ideal Clifford gates and noisy ancillas*. Physical Review A, American Physical Society (APS), vol. 71, 2005. [online] [Consulted 12 December 2020] Available at : https://arxiv.org/abs/quant-ph/0403025

[50] JOZSA, R. *An introduction to measurement based quantum computation*. arXiv:quant-ph/0508124, 2005 [online] [Consulted 12 December 2020] Available at : https://arxiv.org/pdf/quant-ph/0508124.pdf

[51] ARUTE, F.; ARYA, K.; BABBUSH, R., et al. *Quantum supremacy using a programmable superconducting processor.* Nature, vol. 574, pp. 505-510, 23 October 2019 [online] [Consulted 24 December 2020] Available at : https://www.nature.com/articles/s41586-019-1666-5

6.4 NUMERICAL RESULTS

Table 1 (below) indicates several numerical values I retrieved from my simulations. It shows sizes of graphs, an example of a graphical representation with the coloring, the time it takes to simulate the coloring and a theoretical time of execution on a quantum computer.

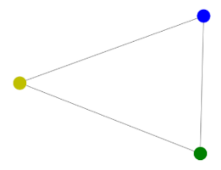
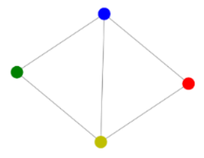
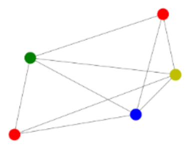
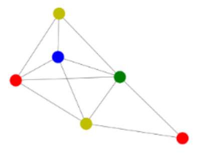
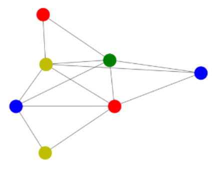
Graph size	Graphical representation	Simulation time	Theoretical time
3 vertices		0.02766 seconds (1000 random graphs tested)	98 clifford gates + 296 T-gates ≈ 0.03648 seconds
4 vertices		0.04104 seconds (500 random graphs tested)	296 clifford gates + 1032 non-clifford gates ≈ 0.11235 seconds
5 vertices		0.91531 seconds (100 random graphs tested)	706 clifford gates + 2664 non-clifford gates ≈ 0.45815 seconds
6 vertices		2 minutes + 21,88106 seconds (5 random graphs tested)	864 clifford gates + 3324 non-clifford gates ≈ 29.60299 seconds
7 vertices		40 minutes + 8,47753 seconds (1 random graph tested)	1022 clifford gates + 3984 non-clifford gates ≈ 8 minutes + 12.1011648 seconds

Table 1 : A table showing the coloring times for different graphs

As I now had the T-gate count for my program, I could estimate the time it takes to run the coloring on a quantum computer. I multiplied the simulation time by the ratio between the clifford gates and the total number of gates. To this I added the number of T-gates, with an approximate time of 0.0001 seconds per gate. This gave me a final formula of : $quantumRuntime = t_{simulation} * \frac{cliffordGates}{totalGates} + TGates * 0.0001$. This value can be seen in the theoretical time column of the table.

It shows that the theoretical quantum computer surpasses the classical computer from 5 vertices on. This is due to the increasing amount of non-clifford gates that are more difficult to simulate on a classical computer than to execute on a quantum computer, as proven by the Gottesman-Knill theorem.[52] I was not able to simulate the coloring of an 8-vertex graph, because of RAM limitations. An 8-vertex graph has a minimum of 14 edges which means that more than 30 qubits are needed to simulate it. This requires more than 2^{30} complex numbers to be stored in the RAM, which is more than the 16 gigabytes my computer has.

The 7-vertex graph takes more than 40 minutes to color, which is nearly 20 times the time it takes to color a 6-vertex graph. This exponential growth indicates that the growth will make the quantum computer even more efficient with larger graphs. This difference in growth can be seen in the logarithmic graph in *Figure 5*. For the values between 3 and 4, the simulation time is shorter than the execution time on a quantum computer. This reinforces our theoretical approach, which states that the simulation time can be represented with an exponential function, and the execution time on a quantum computer can be represented with a square root function, as for small values the square root function is larger than the exponential function.

[52] GOTTESMAN, D.. *Op cit.*

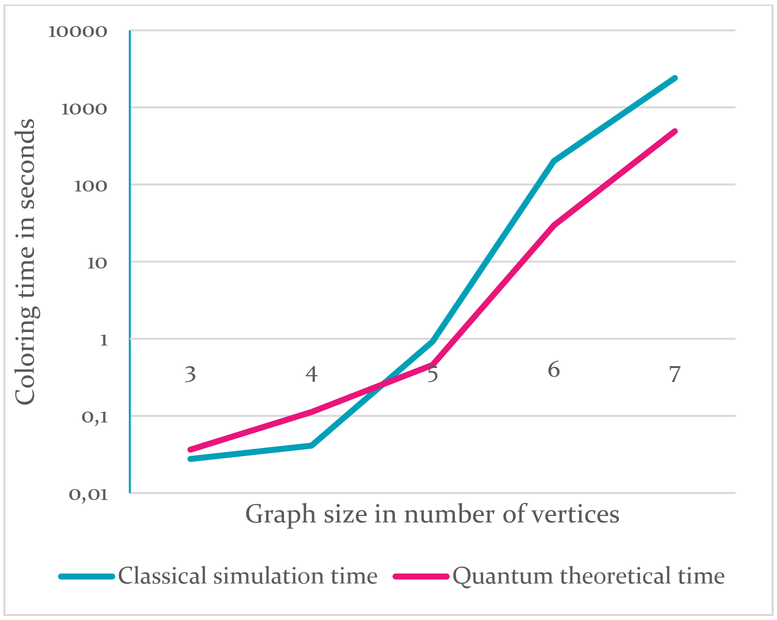


Figure 5 : Coloring time in function of the graph size

I tested a multitude of graphs and a few reflections can be made on their properties.

First, it is necessary to have each vertex connected with at least one edge, as coloring a single vertex is trivial and can be ignored. Secondly, to calculate the efficiency of my program, I studied a worst-case scenario. The worst-case scenario is a graph where all the vertices are connected to all the other vertices. This is the graph with the most edges and is called a complete graph. A complete 10-vertex colored graph can be seen in *Figure 6*. By definition, we need 10 colors to color such a graph. Coloring a complete graph takes the longest since our oracle must check the most edges, which also requires the most qubits. This leads us to another restriction of my simulation.

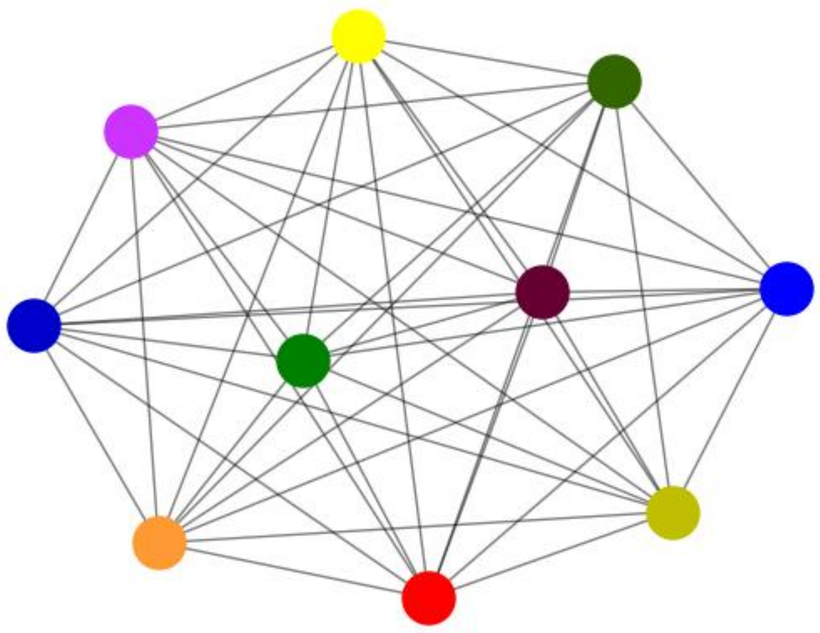


Figure 6 : A 10-vertex colored complete graph

6.5 REPRESENTING COLORS

The biggest challenge in quantum computing is to stabilize systems with large amounts of qubits[53]. For example, Google's silica quantum processor, was built to support 54 qubits.[54] However, when one of its qubits broke, the system was unrepairable and building another one was too costly and time consuming.[55] This shows the complexity of large qubit systems. This is even more visible when simulating large amounts of qubits. My program requires $V * C + E$ qubits, where V is the number of vertices, E the number of edges and C the number of qubits to represent the colors. Since V has a coefficient of C, the biggest number of qubits is used to store the colors of the vertices. Two qubits can be in 4 different states, $\{|00\rangle,\ |01\rangle,\ |10\rangle, |11\rangle\}$, so I could only represent 4 colors. Three qubits can be in 8 different states[56], so I could represent 8 colors. However, the number of qubits used will increase faster when

[53] ARUTE, F.; ARYA, K.; BABBUSH, R., et al. *Op cit.*
[54] *Ibid.*
[55] *Ibid.*
[56] $\{|000\rangle, |001\rangle, |010\rangle, |011\rangle, |100\rangle, |101\rangle, |110\rangle, |111\rangle\}$

representing 8 colors than 4, see *Figure 8*. To have a reasonable runtime of less than one hour, my simulation could use maximum ~30 qubits. Even after I tried running the program with more RAM, the exponential growth of the resources consumed to simulate the algorithm is too high to have any significant result with a higher runtime. This means that I was quickly limited on the graph size, if I wanted to represent 8 colors, to the point that it would become too costly in computer resources. Consequently, I chose to only represent 4 colors with 2 qubits.

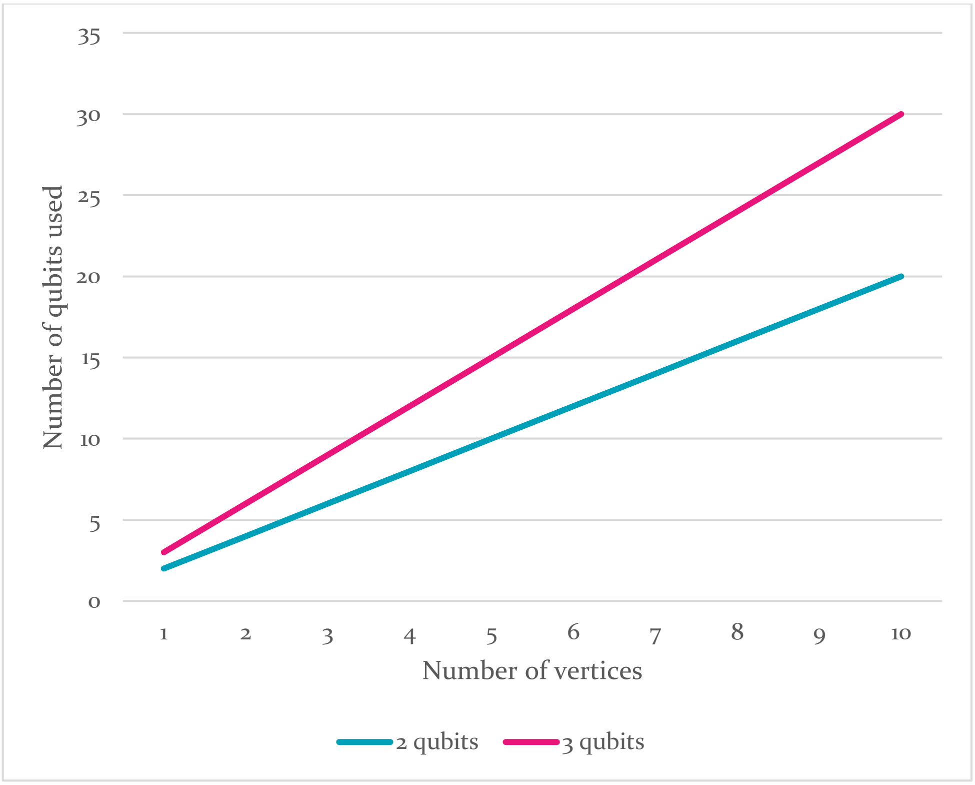


Figure 8 : Number of qubits used with increasing number of vertices

6.6 FOUR COLOR THEOREM

This choice combines well with the 4-colour theorem. The 4 color theorem states that every planar graph can be colored with 4 colors[57]. A planar graph is a graph which can be drawn so that no edges cross each other. This means that our program can color every planar graph with V vertices if $2 * V + E \leq 30$.

[57] XIANG, L, 2009. *A Formal Proof of the Four Color Theorem*. [online] [Consulted 12 December 2020] Available at : https://arxiv.org/abs/0905.3713

7 Analysis of Results

It is important to remind the usefulness of simulations in quantum computing. Since quantum computers can be highly unstable and hard to correct.[58] Finding errors or bugs can be complicated as they could arise from the instructions given or simply from the hardware. It is therefore useful to simulate quantum computers to check, for small instances, whether the algorithm developed is working without errors. Then, once the algorithm is thoroughly tested, when running on quantum hardware, potential errors will mostly lie in the hardware[59].

My program meets these criteria well. We can only run small instances of the problem as shows the rule $2 * V + E \leq 30$, which states that the addition between the double of the number of vertices and the number of edges must be inferior or equal to 30. Since simulating more than 30 qubits require more than 2^{30} complex numbers, we reach the limits of the hardware of most classic computers. This limit could be surpassed by increasing the resources available in our computer, but also by finding new strategies in the programing language to further increase the performance. For example, a big part of the runtime comes from the allocation of qubits. If this process could be improved that the runtime would be reduced, and the performance enhanced.

I was able to deduce a theoretical runtime on a quantum computer from the simulation time by using the functions provided by the Q# environment. This allowed me to analyze the difference between a classical simulation and a quantum computer, as the results showed that the quantum computer would be more efficient for larger graphs.

8 Conclusion

We can conclude that my simulation was a success, because I was able to simulate the coloring of a multitude of graphs with a reasonable runtime of 40 minutes for the most complex ones. I also accomplished the goal of a simulation of a quantum

[58] BRAVYI, S. & KITAEV, A, *Op cit.*
[59] As the field of quantum hardware is still in a very early stage, access to quantum hardware is very restricted and quantum computers of an interesting size are mostly operated by large companies, most of the research is currently done in hardware to stabilize large quantum systems.

computer, which is to test the algorithm and to look for errors and bugs. Furthermore, it allowed me to discuss the efficiency of a simulation on a classical computer.

Mainly, I have :

- Implemented a state flipping oracle which can recognize a correct coloring of a graph and flip a qubit if it is the case
- Implemented an operation which transforms a state flipping oracle in a phase flipping oracle
- Implemented the algorithm discovered by Lov Grover to search the correct coloring for a graph
- Linked all the parts together with basic error handling to create a unique Q# operation call to color the graph.

My program still presents some minor flaws. First, it has an extremely small chance to run infinitely as a measurement is not perfect and the system could collapse to an incorrect state. Secondly, it is very machine resources costly as simulating n qubits take 2^n complex numbers, meaning that a 30-qubit system requires more than 1 trillion complex numbers to be simulated, which is very RAM and CPU intensive.

While we simulated graph coloring problems in this research work and showed that the algorithm works correctly, a logical next step would be to run the graph coloring program on a real quantum computer. As an extension of my work, I considered the two offerings from IBM and Microsoft. IBM provides access to quantum computers for the wider public but forces to use a specific programming language. Microsoft also offers access to quantum computers as part of the Microsoft Azure cloud computing platform and supports the programming language Q# that I used for my implementation. Even though this is out of the scope of my work, out of interest, I registered to get access to Microsoft Azure, but I could not complete this test for lack of time.

In addition, the field of quantum hardware is still in its early development and the focus of research is on quantum fault-tolerant computers, the capacity of quantum computers to run larger programs requiring more than 10 qubits with high connectivity[60] is limited. Despite these limitations, to go further in my work, running the algorithm on a real quantum computer would be a consequential achievement to validate that the graph coloring problem can be solved in non-exponential time.

9 Personal Review

Despite this project being challenging and time-consuming, it furthered my interest in this field mixing my favorite scientific fields : computer science, math, and physics.

It required a lot of learning, as I had to advance myself in the math program, learning complex numbers and linear algebra, before the chapter was reached at school. Furthermore, I had to learn a new programming language called Q# which helped me to understand the properties of the quantum world in a comprehensive way. I then had to understand Grover's algorithm using university-level textbooks (*Quantum Computation and Quantum Information* by Nielsen and Chuang + *Quantum Computing for Computer Scientists* by Yanofsky and Mannucci), research papers (*A fast quantum mechanical algorithm for database search* by Lov Grover) and many lecture notes (Available on the arxiv physics archive).

I then had to code all the processes using the "Katas"[61] provided by the Microsoft Quantum Team. I did not run into major roadblocks, but the installation of all the tools was complicated as the Q# language was only released shortly before I decided to use it. I also learned to interact with the quantum computing online community through different platforms. First, through the Github Quantum Kata page on which I could use the *Issues* section to get help for problems and questions concerning the Q# programming language. Secondly, the Quantum Computing StackExchange is the main question-answer platform available openly on the internet focusing on quantum computing. I used this to get answers and explanation for questions and concepts I needed more information on. As I became more familiar with the concepts and technics I was learning, I also started contributing content on these

[60] Connectivity is the number of interactions between the qubits possible, for example high connectivity is required to run Grover's Algorithm as there are large superpositions

[61] Katas are small tutorial-like exercises provided by the Microsoft Quantum Team to learn quantum computing using the Q# language

forums for others to benefit, for example to understand how Grover’s diffusion operator could be decomposed into basic gates (see *Annex 2*).

During all the project, I was continuously writing this research work, as I completed the corresponding parts. After the long time spent on learning, I yet had to understand how to properly extract results from my program and present them. I also set one of my goals to show the differences between a quantum computer and its simulation. This permitted me to deduce a theoretical execution time from a simulation time, which enabled me to draw conclusions from my results.

To conclude, more than acquiring knowledge about quantum computing, I also learned how to do research work in autonomy, find the resources I needed in a recent scientific field and write a formal paper.

Annex 1 : Further Information on Algorithm Complexity

On May 24th, 2000, the Clay Mathematics Institute of Cambridge, Massachusetts put a one-million-dollar prize on seven mathematical problems that, if solved, would make great advances in research. Although most of them feature complex mathematics which are not made for the public, one problem stood out for its difference and simplicity to understand: P vs NP.[62] This problem addresses computing complexity. In computing, not only creating a program for a computer to solve a problem is important, but also to find a solution which is executable in an acceptable time. For example, designing a program to play chess can be an achievement, but it has no use, if the program cannot choose the correct move to play in acceptable time for the players to stay concentrated and for the audience to enjoy watching the game. Therefore, it is important to find optimal algorithms for a computing problem.[63]

Often, solving a problem in an acceptable time is defined as solving it in polynomial time. This means that the time it takes to solve the problem, can be expressed as a polynomial : $a_n x^n + \cdots + a_2 x^2 + a_1 x + a_0$, where x is the size of the input in bits and n is a constant. The particularity of a polynomial is that it has no variable at the exponent, and it will not grow as fast as an exponential function which has a variable at the exponent. This is tightly linked to the P = NP problem.

P and NP are two classes of computing complexity. P is defined as the group of problems for which an answer can be found in polynomial time.[64] P contains problems like identifying a number as prime[65] or finding the smallest common divisor of a

[62] CMI, 2000. P vs NP Problem [Online]. 14 December 2020. [Consulted 23-November-2020]. Available at : https://www.claymath.org/millennium-problems/p-vs-np-problem
[63] MENABREA (Count.), L. F. *Sketch of the Analytical Engine invented by Charles Babbage ... with notes by the translator. Extracted from the 'Scientific Memoirs,' etc. [The translator's notes signed: A.L.L. ie. Augusta Ada King, Countess Lovelace.]*. R. & J. E. Taylor, 1843.
[64] AARONSON, Scott. *P=?NP*. Open Problems in Mathematics (Springer), ECCC TR17-004, 2016. [Online] [Consulted 12 December 2020] Available at : https://www.scottaaronson.com/papers/pnp.pdf
[65] AGRAWAL, Manindra, Neeraj KAYAL and Nitin SAXENA. *PRIMES is in P* [online]. Ph. D. : Mathematics : Indian Institute of Technology Kanpur, 2002 [Consulted 18 May 2020]. Available at : https://doi.org/10.4007/annals.2004.160.781

number[66]. NP is defined as the group of problems that allow to check, if a solution is correct in polynomial time.[67] These include problems like sudokus, that are correct if every row, column, and 3 by 3 square contains all the numbers from 1 to 9, or coloring graphs, which requires every point to have a different color to all adjacent points. However, finding solutions for NP problems can take an exponential time. Since finding a solution in polynomial time, implies that checking if the solution is correct can be done in polynomial time, by definition, $P \subseteq NP$.

The NP class also contains a sub-group called NP-complete problems. These problems can be converted into all other NP problems[68]. Hence, if they can be solved in polynomial time, all NP problems can be solved in polynomial time.

Graph coloring is an NP-complete problem and can be converted into any NP problem, so solving graph coloring quickly means solving any NP problem quickly. Furthermore, solving graph coloring in polynomial time would prove that $P = NP$ and reward 1 million dollars.

[66] P (complexity). In : *Wikipedia: The Free Encyclopaedia* [online]. 9 November 2019 04:06 [Consulted 18 May 2020 10:38]. Available at : https://en.wikipedia.org/w/index.php?title=P_(complexity)&oldid=925296575
[67] AARONSON, Scott. *Op cit.*
[68] GAREY, M. R. and D. S. JOHNSON. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

Annex 2 : Transformation of Grover's Diffusion Operator into X and Z Gates

$$2|0\rangle\langle 0| - \mathrm{I} = 2\begin{bmatrix}1 & 0 & \cdots & 0\\ 0 & 0 & \cdots & 0\\ \vdots & \vdots & \ddots & 0\\ 0 & 0 & 0 & 0\end{bmatrix} - I = \begin{bmatrix}2 & 0 & \cdots & 0\\ 0 & 0 & \cdots & 0\\ \vdots & \vdots & \ddots & 0\\ 0 & 0 & 0 & 0\end{bmatrix} - \begin{bmatrix}1 & 0 & \cdots & 0\\ 0 & 1 & \cdots & 0\\ \vdots & \vdots & \ddots & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$
$$= \begin{bmatrix}1 & 0 & \cdots & 0\\ 0 & -1 & \cdots & 0\\ \vdots & \vdots & \ddots & 0\\ 0 & 0 & 0 & -1\end{bmatrix}$$

When this operator is applied to $|0\cdots 0\rangle$, it will leave it as is. It will flip the phase of any other state. By applying a -1 global phase, we only need to flip the phase of $|0\cdots 0\rangle$.

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

If we apply the X gate to every qubit, we will only have to flip the state of $|1\cdots 1\rangle$. This is the result of the calculation above. The last matrix represents the operation : Controlled Z. We only need to apply the X gate to every qubit before and after the Z operation.

Annex 3 : My Code

Here is a link to the full code, including testing operations, written in Q# with a python host program : Github link

The following code is the part that is simulated like a secondary quantum processor :

```
namespace GraphColoring
{
    open Microsoft.Quantum.Convert as Convert;
    open Microsoft.Quantum.Intrinsic;
    open Microsoft.Quantum.Canon;
    open Microsoft.Quantum.Diagnostics;
    open Microsoft.Quantum.Measurement as Measurement;
    open Microsoft.Quantum.Arrays as Arrays;

    /// # Summary
    /// Given a qubit register, will return the integer
    /// corresponding the the binary in little endian format.
    operation measureColor (register : Qubit[]) : Int {
        return Convert.ResultArrayAsInt(Measurement.MultiM(register));
    }

    /// # Summary
    /// Given a register of multiple colors in little endian format
    /// and the number of colors in the array,
    /// will return an array with the corresponding integers.
    operation measureColoring (numberElements : Int, register : Qubit[]) : Int[] {
        let elementSize = Length(register)/numberElements;
        let splicedArray = Arrays.Chunks(elementSize, register);
        return Arrays.ForEach(measureColor, splicedArray);
    }

    /// # Summary
    /// Flips a target qubit if 2 colors are equal
    /// # Input
    /// ## c0
    /// Register of qubits representing the first color
    /// ## c1
    /// Register of qubits representing the second color
    /// ## target
    /// target qubit to flip if the colors are equal
    /// # Remarks
    /// CNOT has the same effect as the classical OR gate
    operation ColorEqualityOracle_Nbit (c0 : Qubit[], c1 : Qubit[], target : Qubit)
 : Unit is Adj+Ctl {
        for ((q0, q1) in Arrays.Zip(c0, c1)) {
            CNOT(q0, q1);
        }
        (ControlledOnInt(0, X))(c1, target);
        for ((q0, q1) in Arrays.Zip(c0, c1)) {
            CNOT(q0, q1);
        }
    }

    /// # Summary
    /// Converts a bit-flipping oracle in a phase-flipping oracle.
    ///
    /// # Description
    /// Applying a bit-flip to the |-> state converts it to -|->
    /// only flipping the phase.
    ///
    /// # Type Parameters
    /// ## Qubit[]
    /// The register to analyse
    /// ## Qubit
    /// The target qubit to flip
    operation oracleConverter (markingOracle : ((Qubit[], Qubit) => Unit is Adj), r
egister : Qubit[]) : Unit is Adj {
        using (target = Qubit()) {
            X(target);
            H(target);
            markingOracle(register, target);
            H(target);
            X(target);
        }
    }

    /// # Summary
    /// Applies grovers algorithm when provided an bit-
flip oracle, a register and the the number of iterations
    ///
    /// # Description
    /// View https://en.wikipedia.org/wiki/Grover%27s_algorithm
    operation groverAlgorithm (markingOracle : ((Qubit[], Qubit) => Unit is Adj), r
egister : Qubit[], iterations : Int) : Unit is Adj {
        let phaseOracle = oracleConverter(markingOracle, _);
        ApplyToEachA(H, register);
        for (i in 1..iterations) {
            phaseOracle(register);
            ApplyToEachA(H, register);
            ApplyToEachA(X, register);
            (Controlled Z)(Arrays.Most(register), Arrays.Tail(register));
            ApplyToEachA(X, register);
            ApplyToEachA(H, register);

        }
    }

    /// # Summary
    /// The oracle to test the coloring of the graph
    ///
    /// # Description
    /// Will look at every edge and look if the colors of the two vertices are diff
erent,
    /// if all the colors are different, will flip the target qubit
    operation vertexColoringOracle (V : Int, edges : (Int, Int)[], colorsRegister :
 Qubit[], target : Qubit) : Unit is Adj+Ctl {
        let numberEdges = Length(edges);
        using (correctness = Qubit[numberEdges]) {
            for (i in 0..numberEdges - 1) {
                let (v0, v1) = edges[i];
                ColorEqualityOracle_Nbit(colorsRegister[v0*2 .. v0*2+1], colorsRegi
ster[v1*2 .. v1*2+1], correctness[i]);
            }
            (ControlledOnInt(0, X))(correctness, target);

            for (i in 0..numberEdges - 1) {
                let (v0, v1) = edges[i];
                Adjoint ColorEqualityOracle_Nbit(colorsRegister[v0*2 .. v0*2+1], co
lorsRegister[v1*2 .. v1*2+1], correctness[i]);
            }
        }
    }

    /// # Summary
    /// When given an oracle and the number of vertices, will return a valid colori
ng if possible
    ///
    /// # Description
    /// Will try up to 10 iterations to find a valid coloring using @"groverAlgorit
hm" by :
    /// - Applying grovers algorithm with i iterations
    /// - Measuring the register
    /// - Verifying the solution with another qubit and the oracle
    /// - If the solution is True outputing the oracle using @"measureColoring", el
se repeating
    ///
    /// # Input
    /// ## oracle
    /// A black-box oracle which flips a qubit if the result is correct
    /// ## V
    /// The number of vertices of the graph
    ///
    /// # Output
    /// An array of integers representing the colors
    ///
    /// # Remarks
    /// The color register has to be of size 2*V since the maximum number of colors
 is 4 which can be stored in 2 bits
    /// (see : https://en.wikipedia.org/wiki/Four_color_theorem)
    operation graphColoringMain (V : Int, edges : (Int, Int)[]) : Int[] {
        mutable coloring = new Int[V];
        let oracle = vertexColoringOracle(V, edges, _, _);

        using ((register, output) = (Qubit[2 * V], Qubit())) {
            mutable correct = false;
            mutable iterations = 1;
            repeat {
                /// Message($"Trying iteration {iterations}");
                groverAlgorithm(oracle, register, iterations);
                let temp = Measurement.MultiM(register);
                oracle(register, output);
                if (Measurement.MResetZ(output) == One) {
                    set correct = true;
                    set coloring = measureColoring(V, register);
                }
                ResetAll(register);
            }
            until (correct or iterations > 10)
            fixup {
                set iterations += 1;
            }
            if (not correct) {
                fail "No valid coloring was found";
            }
        }
        return coloring;
    }

    operation graphColoringMainBis (V : Int, edges : (Int, Int)[], iterations : Int
) : Int[] {
        let coloring = new Int[V];
        let oracle = vertexColoringOracle(V, edges, _, _);
        using ((register, output) = (Qubit[2 * V], Qubit())) {
            groverAlgorithm(oracle, register, iterations);
        }
        return [1];
    }

    operation vertexColoringOracleBis (V : Int, edges : (Int, Int)[], colorsRegiste
r : Bool[]) : Unit is Adj+Ctl {
        using ((target, colouring) = (Qubit(), Qubit[2 * V])) {
            ApplyPauliFromBitString(PauliX, true, colorsRegister, colouring);
            vertexColoringOracle(V, edges, colouring, target);
        }
    }

    // markingOracle : ((Qubit[], Qubit) => Unit is Adj), register : Qubit[],
    operation groverAlgorithmBis (iterations : Int, registerSize : Int) : Unit is A
dj {
        // let phaseOracle = oracleConverter(markingOracle, _);
        using (register = Qubit[registerSize]) {
            ApplyToEachA(H, register);
            for (i in 1..iterations) {
                // phaseOracle(register);
                ApplyToEachA(H, register);
                ApplyToEachA(X, register);
                (Controlled Z)(Arrays.Most(register), Arrays.Tail(register));
                ApplyToEachA(X, register);
                ApplyToEachA(H, register);
            }
        }
    }
}
```

Annex 4 : Dirac Notation

In quantum physics, as a lot is represented with state vectors, Dirac notation is used to quickly represent vectors, also called ket-bra notation. A *ket* represents a vector, the ket $|0\rangle$ represents the vector $\begin{bmatrix}1\\0\end{bmatrix}$ and the ket $|1\rangle$ represents $\begin{bmatrix}0\\1\end{bmatrix}$. Since these states are basis states, we can write any vector/qubit as $\begin{bmatrix}\alpha\\\beta\end{bmatrix} = \alpha|0\rangle + \beta|1\rangle$. An arbitrary vector is normally represented by $|\psi\rangle$. There are also 4 more common states :

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

The effect of a gate A on a state is written as $A|+\rangle = |\psi\rangle$. Furthermore, the effect of a gate $A = \begin{bmatrix}\epsilon & \zeta\\\eta & \mu\end{bmatrix}$ on the basis states is $A|0\rangle = \epsilon|0\rangle + \eta|1\rangle$ and $A|1\rangle = \zeta|0\rangle + \mu|1\rangle$.

The *bra* of a *ket* is its Hermitian adjoint and is written as $\langle\psi| = |\psi\rangle^\dagger$. The *ket* is a column vector and its corresponding *bra* is a row vector.

This gives us a good way to represent inner- and outer-products. The inner product of the vectors $|\phi\rangle$ and $|\psi\rangle$, is the matrix product of $\langle\phi|$ and $|\psi\rangle$ written as $\langle\phi|\psi\rangle$. The same goes for outer products which can be written as $|\phi\rangle\langle\psi|$.

We can also decompose gates into *bra-ket* notation. $X = \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix} = \begin{bmatrix}0 & 1\\0 & 0\end{bmatrix} + \begin{bmatrix}0 & 0\\1 & 0\end{bmatrix} = \begin{bmatrix}1\\0\end{bmatrix}\begin{bmatrix}0 & 1\end{bmatrix} + \begin{bmatrix}0\\1\end{bmatrix}\begin{bmatrix}1 & 0\end{bmatrix} = |0\rangle\langle 1| + |1\rangle\langle 0|$. In general, any gate $A = \begin{bmatrix}\epsilon & \zeta\\\eta & \mu\end{bmatrix}$ can be written as $\epsilon|0\rangle\langle 0| + \zeta|0\rangle\langle 1| + \eta|1\rangle\langle 0| + \mu|1\rangle\langle 1|$. This makes us able to perform math with *bra-ket* notation : $X|0\rangle = (|0\rangle\langle 1| + |1\rangle\langle 0|)|0\rangle = |0\rangle\langle 1|0\rangle + |1\rangle\langle 0|0\rangle = |0\rangle(0) + |1\rangle(1) = |1\rangle$.

Annex 5 : Specific Gates

We will look at the main gates that we needed in our implementation, but only mention the most general knowledge required to fit the resolution of the problem, namely X, H and Z gates:

X GATE

The X gate[69] can be compared to the classical NOT gate. It turns $|0\rangle$ into $|1\rangle$ and $|1\rangle$ into $|0\rangle$. More generally, to a state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ into the state $|\psi\rangle = \beta|0\rangle + \alpha|1\rangle$. It is represented by the matrix $X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$. For us, the most critical consideration is how it affects the $|-\rangle$ state as $X(|-\rangle) = -|-\rangle$.

H GATE

The H gate is one of the most important gates. It is the gate that creates superposition. It is defined by the matrix $H = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$. We can see the result of applying the gate on the basis states : $H(|0\rangle) = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle = |+\rangle$ and $H(|1\rangle) = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle = |-\rangle$. We may notice that the difference between $|+\rangle$ and $|-\rangle$ is a minus sign. This is called the relative phase and is explained in the section *Phases*.

Z GATE

The Z gate has a less visible effect, as it only affects the phase. It is defined as $Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$. Z does not change the complex amplitudes, as $Z(\alpha|0\rangle + \beta|1\rangle) = \alpha|0\rangle - \beta|1\rangle$.

[69] The descriptions of these gates can be found in *Quantum computation and quantum information* by Nielsen and Chuang.

CONTROLLED GATES

A controlled operation is a gate that is applied to a “target” qubit, only when all the “control” qubits are in state $|1\rangle$.[70] This operation is necessary in quantum computing since every operation must be reversible.[71] A controlled gate is the same as calculating a tensor product between the gate U and the identity matrix I_2. For example, doing a controlled gate of the H gate is equal to $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix}$.

This gate will apply the H gate if the control qubits are in state $|1\rangle$.

[70] MONROE, C., et al. *Demonstration of a Fundamental Quantum Logic Gate.* Physical Review Letters, vol. 75, pp. 4714–4717, 1995. [online] [Consulted 20 December 2020] Available at : https://link.aps.org/doi/10.1103/PhysRevLett.75.4714.
[71] NIELSEN, Michael and Isaac CHUANG. *Op cit.* p. 93

Annex 6 : Q# Basics

I used the Q# programming language to simulate quantum programs. Q# is an open-source programming language for developing and running quantum algorithms. It uses elements from python, C#, and F#, coupled with new functions specific to quantum computing. It also provides libraries for Chemistry, Machine Learning and Numeric uses. We will look at Q# basics to properly understand the code which will simulate Grover's Algorithm to solve graph coloring problems.

The scientific community believes that future use of quantum computers will arise under the form of quantum co-processors. The main processor of computers will communicate with the quantum co-processor to execute quantum tasks. The Q# programming language works similarly. We would write a Q# file that would be compiled and act like a library. Then, using a python or C# host program, functions and operations from the Q# file can be called to return data and process it. Therefor the basis of a Q# file is functions and operations:

Q# functions work similarly to mathematical functions: they take inputs and map them to outputs. They are classical routines and are purely deterministic (Q# docs). They do not use any quantum properties; hence, they do not contain any randomness. This means that the output is fixed, and the same input will always return the same output. They can take in any number of elements, but only return one element. A function is created with the ```function``` keyword. A name, the input names and types in brackets and the output type have to be added. The code is then added inside braces. For example, this function would return the square of a number:

```
function Square (x : Double) : (Double) {
     return x*x;
}
```

Operations are like functions; however, they can use other operations and quantum properties. This makes them non-deterministic.

Bibliography

AARONSON, S. and D. GOTTESMAN. *Improved simulation of stabilizer circuits*. Physical Review A, American Physical Society (APS) vol. 70, 2004. [online] [Consulted 12 December 2020] Available at : https://arxiv.org/abs/quant-ph/0406196

AARONSON, Scott. *P=?NP*. Open Problems in Mathematics (Springer), ECCC TR17-004, 2016. [Online] [Consulted 12 December 2020] Available at : https://www.scottaaronson.com/papers/pnp.pdf

AGRAWAL, Manindra, Neeraj KAYAL and Nitin SAXENA. *PRIMES is in P* [online]. Ph. D. : Mathematics : Indian Institute of Technology Kanpur , 2002. [Consulted 18 May 2020]. Available at : https://doi.org/10.4007/annals.2004.160.781

ARUTE, F.; ARYA, K.; BABBUSH, R., et al. *Quantum supremacy using a programmable superconducting processor*. Nature, vol. 574, pp. 505-510, 23 October 2019 [online] [Consulted 24 December 2020] Available at : https://www.nature.com/articles/s41586-019-1666-5

BEIGEL, R. and D. EPPSTEIN. *3-Coloring in Time O(1.3289^n)*. Journal of Algorithms, vol. 54, issue 2, February 2005. [Online] [Consulted 12 December 2020] Available at : https://www.sciencedirect.com/science/article/abs/pii/S0196677404001117

BENDER, Edward and Gill WILLIAMSON. *Lists, Decisions and Graphs*. S. Gill Williamson, n.d, 2010. [online] [Consulted 12 December 2020] Available at : https://cseweb.ucsd.edu/~gill/BWLectSite/Resources/LDGbookCOV.pdf

BOYER, Michel, et al. *Tight bounds on quantum searching*. Fortschritte der Physik, vol. 46, pp. 493-506, February 1998. [online] [Consulted 12 December 2020] Available at : https://arxiv.org/abs/quant-ph/9605034

BRAVYI, S. & KITAEV, A. *Universal Quantum Computation with ideal Clifford gates and noisy ancillas*. Physical Review A, American Physical Society

(APS), vol. 71, 2005. [online] [Consulted 12 December 2020] Available at : https://arxiv.org/abs/quant-ph/0403025

CLEVE, R., EKERT, A., MACCHIAVELLO, C. and M, MOSCA. *Quantum Algorithms Revisited*. Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences, vol. 454, pp. 339-354, March 1998. [online] [Consulted 12 December 2020] Available at : https://arxiv.org/abs/quant-ph/9708016

CMI, 2000. P vs NP Problem [Online]. 14 December 2020. [Consulted 23-November-2020]. Available at : https://www.claymath.org/millennium-problems/p-vs-np-problem

DE WOLF, Ronald. *Quantum Computing: Lecture Notes*. 2009 [online] [consulted 12 August 2020] Available at : https://arxiv.org/abs/1907.09415

FERNANDES, M. C. B., KHANNA, F. C., MARTINS, M., SANTANA, A. and J. VIANNA. *Non-linear Liouville and Shrödinger equations in phase space*. Physica A: Statistical Mechanics and its Applications, Elsevier BV, vol. 389, issue 17, pp. 3409–3419, 2010. [Online] [Consulted 14 October 2020] Available at : https://doi.org/10.1016/j.physa.2010.04.030

GAREY, M. R. & JOHNSON, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co, 1979.

GOTTESMAN, Daniel. *The Heisenberg Representation of Quantum Computers*. Group22: Proceedings of the XXII International Colloquium on Group Theoretical Methods in Physics, eds. S. P. Corney, R. Delbourgo, and P. D. Jarvis, pp. 32-43 (Cambridge, MA, International Press, 1999), July 1998. [online] [Consulted 12 December 2020] Available at : https://arxiv.org/abs/quant-ph/9807006

GROVER, Lov K. *A fast quantum mechanical algorithm for database search*. Proceedings, 28th Annual ACM Symposium on the Theory of Computing (STOC), May 1996, pp. 212-219 [online] [Consulted 5 January 2021] Available at : https://arxiv.org/abs/quant-ph/9605043

GROVER, Lov K. *From Schrödinger's equation to the quantum search algorithm*. Pramana, Springer Science and Business Media LLC, vol. 56, pp. 333–348, 2001 [online] [Consulted 12 December 2020] Available at : https://arxiv.org/abs/quant-ph/0109116

HANSEN, Pierre and Julio KUPLINSKY. *The smallest hard-to-color graph*. Discrete Mathematics, vol. 96, pp. 199–212, 1991. [online] [Consulted 18 November 2020] Available at : http://www.sciencedirect.com/science/article/pii/0012365X9190313Q.

HEISENBERG, W. *Über den anschaulichen Inhalt der quantentheoretischen Kinematik und Mechanik*. Zeitschrift für Physik 43, pp. 172-198, March 1927. [online] [Consulted 04 January 2021] Available at : https://doi.org/10.1007/BF01397280 [Access with subscription]JOZSA, Richard, 2005. *An introduction to measurement based quantum computation*. arXiv:quant-ph/0508124.

KELLY, Adam. *Simulating Quantum Computers Using OpenCL*. arXiv:1805.00988 [quant-ph], November 2018. [online] [Consulted 2 January 2021] Available at : http://arxiv.org/abs/1805.00988.

LAVOR, C.; L. MANSSUR and R. PORTUGAL. *Grover's Algorithm: Quantum Database Search*. arxiv:quant-ph/0301079, January 2003. [online] [Consulted 12 December 2020] Available at : https://arxiv.org/abs/quant-ph/0301079

MENABREA (Count.), Luigi Federico. *Sketch of the Analytical Engine invented by Charles Babbage ... with notes by the translator. Extracted from the 'Scientific Memoirs,' etc. [The translator's notes signed: A.L.L. ie. Augusta Ada King, Countess Lovelace.]*. R. & J. E. Taylor, 1843.

MICROSOFT QUANTUM KATA CONTRIBUTORS. *Microsoft/QuantumKatas*. [online] [Consulted 24 December 2020] Available at : https://github.com/microsoft/QuantumKatas.

MONROE, C., et al. *Demonstration of a Fundamental Quantum Logic Gate.* Physical Review Letters, vol. 75, pp. 4714–4717, 1995. [online] [Consulted 20

December 2020] Available at : https://link.aps.org/doi/10.1103/PhysRevLett.75.4714.

NIELSEN, Michael A. and Isaac L. CHUANG. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

PERRY, Riley tipton. *Quantum Computing from the Ground Up*. WORLD SCIENTIFIC, 2012.

QUANTIKI Contributors. *List of QC simulators — Quantiki, Quantum Information Portal and Wiki*. 2020. [online] [Consulted 30 November 2020] Available at : https://www.quantiki.org/wiki/list-qc-simulators.

SHANNON C. E. *A mathematical theory of communication*. The Bell System Technical Journal, vol. 27, no. 4, July 1948, pp. 379-423. doi: 10.1002/j.1538-7305.1948.tb00917.x.

SHEFFER, H. M. *A set of five independent postulates for Boolean algebras, with application to logical constants*. Transactions of the American Mathematical Society, vol. 14, pp. 481-488, July 1913. [online] [Consulted 12 December 2020] Available at : https://www.ams.org/journals/tran/1913-014-04/S0002-9947-1913-1500960-1/S0002-9947-1913-1500960-1.pdf

SHOR, Peter. *Algorithms for quantum computation: discrete logarithms and factoring*. Proceedings 35th Annual Symposium on Foundations of Computer Science, 1994. [online] [Consulted 12 December 2020] Available at : https://ieeexplore.ieee.org/document/365700

TUSAROVÁ, Tereza. *Quantum Complexity Classes*. CoRR cs.CC/0409051, 2004. [online] [Consulted 12 December 2020] Available at : http://arxiv.org/abs/cs.CC/0409051.

WIKIPEDIA Contributors. *Big O notation — Wikipedia, The Free Encyclopedia*. [online] [Consulted 18 december 2020] Available at : https://en.wikipedia.org/w/index.php?title=Big_O_notation&oldid=994900730.

XIANG, Limin. *A Formal Proof of the Four Color Theorem*. May 2009. [online] [Consulted 12 December 2020] Available at : https://arxiv.org/abs/0905.3713

YANOFSKY, Noson and Mirco MANNUCCI. *Quantum computing for computer scientists*. Cambridge: Cambridge University Press, 2018.

Image Sources

- Cover image : JUSTIN FANTL. *Rigetti – quantum computing.* [online] Available at : http://justinfantl.com/commissions/quantumcomputing/1556/
- *Figures 1, 2, 6, 9* and all the images in *Table 1* have been realised by me in *python* [online] (Available at : https://www.python.org/downloads/release/python-387/) using libraries *NetworkX* [online] (Available at : https://pypi.org/project/networkx/) and *Matplotlib.Pyplot* [online] (Available at : https://matplotlib.org/api/pyplot_api.html)
- *Figure 3* has been realised by me in *Latex* [online] (Available at : https://www.overleaf.com/) using the *Qcircuit* package [online] (Available at : http://physics.unm.edu/CQuIC/Qcircuit/)
- *Figure 5* : Bender2k14 - *Created in LaTeX code using Q-circuit*., CC BY-SA 3.0, [online] Available at : https://commons.wikimedia.org/w/index.php?curid=13524800
- All other figures were created by me using built-in word tools