Study of quantum algorithms and their implementations

Giancarlo Ponte Gamberi
Faculdade de Computação e Informática
Universidade Presbiteriana Mackenzie
São Paulo, Brazil
giangamberi@hotmail.com.br

Calebe de Paula Bianchini
Faculdade de Computação e Informática
Universidade Presbiteriana Mackenzie
São Paulo, Brazil
calebe.bianchini@mackenzie.br

Abstract—In this work, we studied some quantum algorithms, and selected Grover search problem, it will then receive an in-depth analysis of how the problem works and how it can be adapted to its process in a quantum computer, and a quantum solution using python qiskit library will be elaborated.

Keywords— quantum, algorithms, qiskit, qubit, oracle, superposition, Hadamard, Bloch sphere, state vector

I. Introduction

Quantum computing is currently a central topic of research in basic science and frontier technology. In the last two decades, interest in the area came from the motivation that quantum algorithms provide considerable computational gains, in relation to classical analogues. As a result, in recent years we have experienced technological advances that allow us to begin to overcome some practical problems for the scalable manufacturing of quantum computers [1].

With the approach of quantum supremacy[1] it becomes increasingly interesting and necessary to develop algorithms, whether these are exclusive to quantum computing, or whether they are known problems with a classical solution, to be executed using the new paradigm presented by the new processors.

With this need in mind, this paper consists of researching quantum search algorithms, understanding them in depth, employing one of them and finally trying to propose a new algorithm based on the one implemented. The algorithm selected during the research is Grover's search algorithm, and two different quantum algorithms have been proposed to try to solve the search problem that was solved by Grover.

II. Quantum computing

Before we begin our study, it is necessary to understand the state of the art and some concepts used by the new paradigm and that will be used during the work. These concepts, as they are related to a new technology, are still little disseminated, and, therefore, must be conceptualized before we begin the study.

Starting with the qubits, or quantum bits as they may be often referred, they are the quantum computers equivalent of the bits in classical computers, the big difference between them and their classical version, is that bits only assume two classical positions, logical 0 and 1, and is necessarily in one of these positions. The qubits, on the other hand, can assume a superposition, being able to operate in any state within this superposition. The logical representation of this superposition is made through a Bloch sphere.

A qubit, unlike bits in which we always know their states, needs to be measured for us to know its position, and in this measurement the state where the qubit was, is lost, and the result does not give us the position of this qubit, but rather the probability of it being in state 0 or 1, therefore despite the greater computational power that a qubit provides us, we have to be careful with the data output, since the reading of the processing does not represent the same freedom as the processing, despite still giving us more capacity than classically.

As mentioned, the sphere is the logical representation of the superposition of a qubit (Fig. 1), this sphere has three axis: *x*, *y* and *z*. However, the axis that really deserves attention is z, as states 0 and 1 are found at the extremity of this axis, where the probability is calculated during the measurement. Note that, despite the freedom to work in any quadrant of the axis *x* and *y*, in the end it is the position in *z* that counts for our measurement.

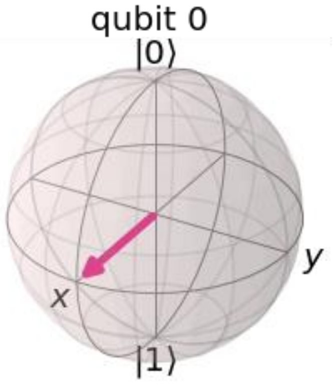

Fig. 1. Bloch sphere generated by a qiskit function, the red arrow pointing to the symbol "X" representing the actual state of the qubit, note that this is in a "perfect" superposition, because in relation to the z axis (which is not explicitly represented, being the axis with the extremes being represented by the vectors $|0\rangle$ and $|1\rangle$) it is exactly in the center.

Another concept that we must keep in mind when we talk about quantum computing is that of oracles, this is not a new concept nor exclusive to quantum computing, however it is a concept frequently used when we talk about quantum algorithms, and, therefore, we need to have a glimpse to better understand the topic.

An oracle is a “black box” within an algorithm, the idea is that during the process, our data passes through this “box” and comes out of it, without having a complete idea of what was done in it, whether the data was processed or not, and what type of process was performed. In practice, and here we consider how it was commonly used in the researched algorithms, and how we can think to facilitate understanding, it is an unknown algorithm, where its functionality is sometimes mentioned. A fictitious example, but based on those that will be seen, would be the mention of an oracle that orders a vector within our algorithm, we do not necessarily need to know the algorithm used to order this vector, what interests us is the fact that there is a need within our algorithm, to order the vector.

Lastly, let's talk about Qiskit, as the library developed by IBM for Python is called and used for simulating algorithms and quantum circuits, it is the library we will be using to implement the algorithms, with this library we can create quantum circuits and run them, whether via simulation (the library offers some different types of simulators for quantum processors, in this case we will use the most used and recommended by the library's own documentation, the *aer simulator*), or via an IBM cloud quantum processor (the process of executing this is time-consuming, because we use a public queue, we will therefore mostly use simulation).

III. Algorithm Study

Imagine a telephone directory containing *N* names, arranged in random order. To find someone's phone number with a probability of ½, any classical algorithm (whether deterministic or probabilistic) will need to observe at least *N/2* names. [6]

Grover in 1996 proposed a search algorithm to be applied on quantum systems for the classic problem of searching in an unordered vector, the vector in this case will be the vector of states of the qubits.

This algorithm is classified as oracular, that is, there will be use of an oracle in the middle of the algorithm, this oracle, by definition, is a “black box”, that is something that our system will go through , which may or may not perform some processing or change in our system, in practice, we can define it as some algorithm (whether known or not, or which has some clear purpose, or purely for testing) that will be executed prior to Grover’s search algorithm whose specific result must be found.

The algorithm starts with the application of the oracle in our uniformly superposed system, the next step needs to be executed $O(\sqrt{N})$, where *N* is the number of qubits:

We rotate the states we want to look for by π radians, while the others we leave unchanged, then we apply the diffusion matrix, defined by Grover as:

$$\mathrm{D} = \mathrm{WRW} \tag{1}$$

Where W would be the Hadamard gate, most frequently used to put the system in superposed state, and R is defined as a diagonal matrix, where the diagonal can be defined as a vector where the first element is *1*, and the remainder *-1*:

$$\begin{gathered} R_{i,j} = 0 \text{ if } \mathrm{i} \neq \mathrm{j} \\ R_{i,i} = 1 \text{ if } \mathrm{i} = 0 \\ R_{i,i} = -1 \text{ if } \mathrm{i} \neq 0 \end{gathered} \tag{2}$$

Once the algorithm is finished, we measure the system, and we can observe that the state that was rotated right at the beginning has a probability of at least ½.

This Grover algorithm has become widely used in conjunction with other algorithms as it incredibly facilitates finding a specific state that is rotated by π radians, and due to its wide use, this will be the algorithm implemented in section III of this work.

IV. Developing a Solution

Starting with the implementation of the algorithm proposed by Grover, this application used a python ipynb notebook, and for the quantum part the qiskit library.

Initially we run the algorithm with an oracle that will simply mark (here, marking refers to the rotation of π radians that is applied to the state we want to find) some superposition state of random choice, and we apply the diffusion matrix, where then we observed the results, after these steps, a more elaborate oracle, with a more practical example was created, to have a better idea of the capacity and usability of the algorithm.

Once the libraries qiskit, numpy (to assist in data manipulation) and pyplot from matplotlib (to visualize the results) have been imported, and as previously mentioned any state was selected, we created a circuit using the library quantum, in this example we use a 3-qubit circuit.

Here, we come to some ways to perform the necessary rotation, the ideal method to perform would be using a Z gate, which precisely performs the operation we need, however for our case of 3 qubits, rotating the qubit generates an undesirable situation with more than one state marked, because of this we prefer to apply a diagonal matrix specifying only one state.

To apply diagonal matrices to our circuit, the qiskit library has a simplified function for this, requiring only the qubits where it will be executed, and a vector that represents the diagonal of the matrix.

To specify the state, we developed the function *converteNumeroDiagonalOraculo* which receives an integer that represents a state, and the number of qubits in the system, and returns a vector ready to be executed in conjunction with the library's diagonal function, the application we can observe in Code 4, the circuit generated from the oracle in Figure 2 and the reading of the states in figures 2 and 3.

```
def converteNumeroDiagonalOraculo(number,  qubits):
    if (2**qubits) < (number-1):
        return -1
    aux = np.ones(2**qubits,dtype=int)
    aux[number] = -1
    return aux
```

Fig. 2. Definition of function *converteNumeroDiagonalOraculo*, which receives the variables *number* and *qubits*, respectively the position of the number to be "marked" and the number of qubits to operate, and then return the variable *aux*, which contains an array of size *2**qubits* with only the number in position *number* as negative.

```
#aplica hadamard em um circuito para a lista de qbits especificada
def inicializaSobreposicao (qc, qubits):
    for q in qubits:
        qc.h(q)
    return qc
```

Fig. 3. Auxiliary function to apply the Hadamard gate to the specified qubits, when we talk about uniform superposition, we talk about the application of these gates.

Arriving then at the core of Grover's algorithm, we need to apply the diffusion matrix, which as mentioned is defined as $D = WRW$, with W being the application of the matrix that leaves the states in uniform superposition, which is the application of the Hadamard gate to the qubits necessary, if you notice well, our auxiliary function initializaSobreposicao (Code 3) performs exactly this operation, so we only have the matrix R left, which we need to elaborate.

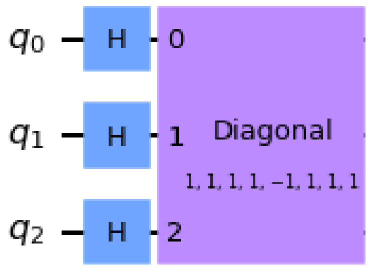

Fig. 4. Representation in the form of a diagram used by the qiskit of our oracle.

$$|\psi\rangle = \begin{bmatrix} 0.35355 \\ 0.35355 \\ 0.35355 \\ 0.35355 \\ -0.35355 \\ 0.35355 \\ 0.35355 \\ 0.35355 \end{bmatrix}$$

Fig. 5. Vector of states that was obtained through the aer simulator, from qiskit, note that all states are identical, except for state 100, which is negative, and is the one we want to look for.

As the matrix R is a diagonal matrix, we apply the diagonal function already embedded in the library, that is, we create the vector that represents the diagonal, for this, we create using numpy a vector of size *2^n* filled with numbers *1*, where *n* is the number of qubits (in the case of this example: *3*) and we invert the sign of all, except the first.

Now that we have W and R, we can easily apply the matrix D, and measure the results of the execution, as per figure 6 below:

```
def matrizDifusao(qc, qubits):
    # D = WRW
    # Matriz de difusao = Hadamard -> diagonal[1,-1,-1,-1] -> Hadamard
    R = np.ones(2**len(qubits),dtype=int)
    for i in range(1,2**n):
        R[i] = -1
    inicializaSobreposicao(qc,qubits)
    qc.diagonal(list(R), qubits)
    inicializaSobreposicao(qc,qubits)
    return qc
```

Fig. 6. Function that applies the diffusion matrix to the circuit

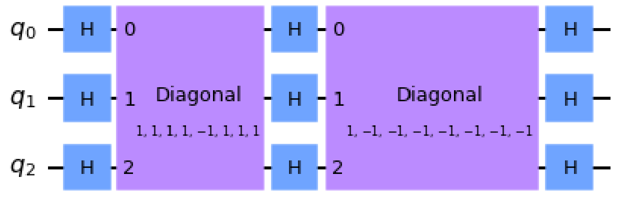

Fig. 7. Circuit after applying the oracle and diffusion matrix.

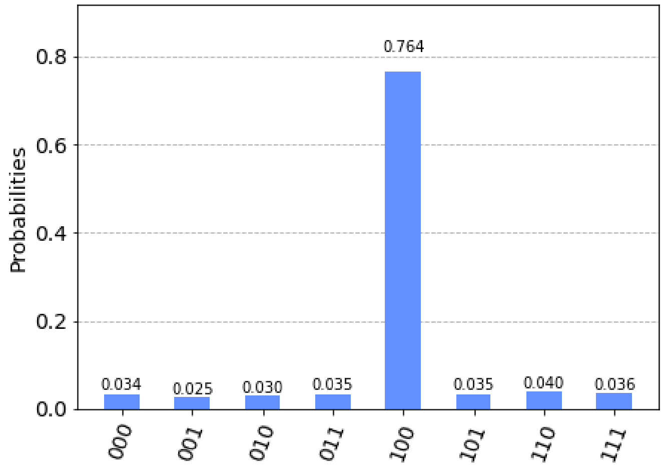

Fig. 8. Simulation plot of observation results, as we can see, our state that had been rotated, and which previously had the same probability as the others, now presents an obvious prominence in relation to the others.

V. Quantum Probability Search Algorithm

The idea behind this proposed algorithm would be to encode the numbers of the input vector in the state vector, the input vector needs to have *2^n-1* elements, so that the number to be searched for is positioned in the last position of the state vector, the proposed port would then subtract each probability by the number sought, resulting in a vector of states where the number sought would have zero probability.

To carry out the above proposal, we start by assembling a *2^n* vector, where the last position contains our searched number, while the rest of the positions have numbers from a vector, with this vector assembled, we can start the process to make it represent amplitudes of a circuit, and then, using qiskit's *StatePreparation* function, we have a circuit that has this vector as its state vector.

Normalization to leave the vector in its state form can be performed by dividing each element by the root of the sum of the quadratic elements, this normalized vector is translated into the circuit using qiskit's *StatePreparation* function.

The elaborated quantum logic gate that would comply with the subtraction of probabilities would be an *n* by *n* matrix where the diagonal will have a value of 1, the rest of the values in the last line will have a value of -1, and the rest of the positions will be composed of 0, also represented per:

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 & 0 \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ -1 & -1 & -1 & \cdots & -1 & 1 \end{pmatrix} \tag{3}$$

However, such a gate would also need to be transformed, to be in the format of a unitary matrix so that we can use it in our circuit. For such conversion, we use the Gram-Schmidt procedure, and apply it to the circuit, which in turn will be ready to be executed.

Some tests were carried out with the following proposed algorithm, in this case the number of qubits used was 3, the vector to be searched was randomly structured as follows: [7,12,4,15,3,13,11] and the number search engine was being changed to analyze the results in the simulations. The logic gate and its unitary matrix, respectively for this test case:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 \end{pmatrix} \tag{4}$$

$$\begin{pmatrix} 0.7071 & -0.4082 & -0.2887 & -0.2236 & -0.1826 & -0.1543 & -0.1336 & 0.3536 \\ 0.0000 & 0.8165 & -0.2887 & -0.2236 & -0.1826 & -0.1543 & -0.1336 & 0.3536 \\ 0.0000 & 0.0000 & 0.8660 & -0.2236 & -0.1826 & -0.1543 & -0.1336 & 0.3536 \\ 0.0000 & 0.0000 & 0.0000 & 0.8944 & -0.1826 & -0.1543 & -0.1336 & 0.3536 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.9129 & -0.1543 & -0.1336 & 0.3536 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.9258 & -0.1336 & 0.3536 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.9354 & 0.3536 \\ -0.7071 & -0.4082 & -0.2887 & -0.2236 & -0.1826 & -0.1543 & -0.1336 & 0.3536 \end{pmatrix} \tag{5}$$

In the first tests performed, using the same simulation configuration used in Grover's algorithm (that is, using qiskit's aer simulator), the algorithm unfortunately did not demonstrate good accuracy after passing through the gate, generating a lot of fluctuation and unsatisfactory results.

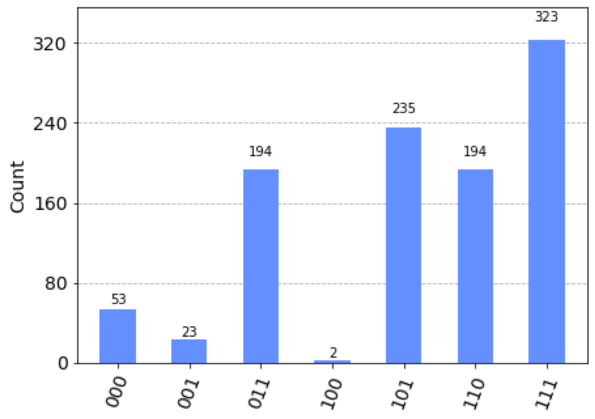

Fig. 9. Simulation results looking for the number 7 (First position of the vector)

As a result of measuring the final circuit using the aer simulator, the idea would be that the number sought would have the lowest probability, in this case, 000, but we clearly see that this was not the case, since 001, 010 and 100 had lower probabilities.

$$[0.25032, 0.42912, 0.14304, 0.5364, 0.10728, 0.46488, 0.39336, 0.25032] \tag{6}$$

$$[-0.2148, 0.13376, -0.05145, 0.42439, 0.06214, 0.46633, 0.45646, -0.56881] \tag{7}$$

State vector (6) encoded in the probabilities, note that the last element is the same as the first, as the value 0.25 would represent the number 7. State vector (7) after the circuit passes through the proposed gate, the expected result would be that the number in the first position is the closest to zero, but numbers with smaller values (in this case numbers *4* and *3*, in the third and fifth position, respectively) ended up being closer to zero, and, therefore, giving us an erroneous result.

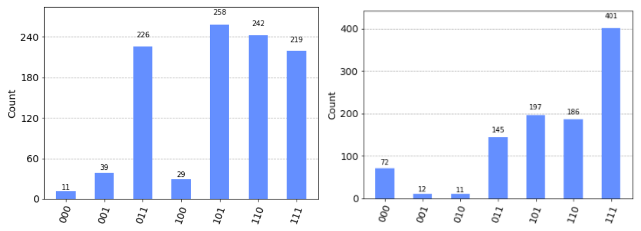

Fig. 10. Search results for numbers 3 and 13 in the same vector, as we can see, the results are not as expected.

We then concluded that the problem lies in the proposed gate, which, because it was converted to a unitary matrix, lost the precision that would be necessary to give us a correct result, however, during the development of possible corrections for this algorithm, the idea of using it was considered. if the rotations of the qubits to store and manipulate the data, instead of using the probabilities of the measurements, Quantum probability search algorithm was later discarded and we kept with Quantum rotation search algorithm, which will be demonstrated next.

VI. Quantum Rotation Search Algorithm

The idea of this algorithm is to encode the vector numbers in each qubit along the perimeter of the first quadrant of the qubit's y-axis. Once this data is encoded, we apply a second equal rotation to each qubit, with an angle defined by the number sought, in such a way that the qubit that holds the number corresponding to the one we are looking for is in a perfect superposition, and finally we apply Hadamard gates to each one, to facilitate the measurement of the probabilities in each qubit, as our desired qubit would result in 100 % chance of being at 0, while other qubits with values different from those sought will not provide such certainty in the measurement.

Visually, that is, by converting our qubits to the visualization of their respective Bloch spheres, we can better understand the proposed algorithm, which is why the circuit was created, and using the qiskit function that generates a visualization in Bloch spheres of our circuit we can follow the result of each rotation applied.

Still considering the rule used previously, we work with numbers from *0* to *15*, if we normalize them, and multiply by 90 radian degrees (*π/2*) we obtain the radian degrees we need to encode the data in qubits, such rotation is easily achieved using qiskit's *ry* gate.

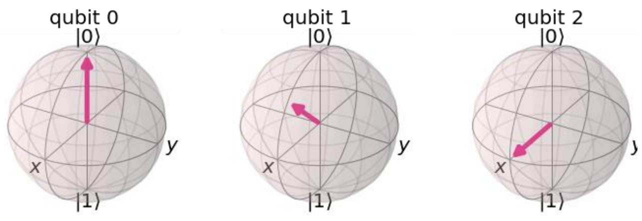

Fig. 11. Above three Bloch spheres representing three different qubits, in them we used the coding step of our proposed algorithm to represent the values 0, 8 and 15, respectively. Note that, as the values increase, the "arrow" (representation of the state of the qubit) rotates on the y-axis.

Next, we apply the same normalization and conversion rule to radians in our number to be sought, however we need to obtain the complementary angle to it, to do this, we have two ways to make it work, we can subtract our number by 15 before normalizing it, or subtract it by 90 radians after being converted to radians, regardless of which one we choose, we have to obtain its modulus, with the number to be sought properly treated, we apply a rotation in y to each qubit with that value.

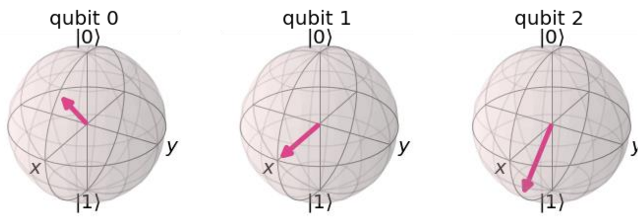

Fig. 12. Continuing the example from the previous figure, we will look for the number 8, which is represented in the second qubit, here we have already applied the rotation of our number to be looked for using our algorithm, note that the second qubit is in a perfect superposition.

Finally, we apply *hadamard* gates to each qubit, this will cause our qubit containing our number to be searched to undo the superposition, while the others will still maintain superpositions, so when we measure our circuit, the qubit searched will be the only one with certainty of being at 0, therefore being easily identifiable.

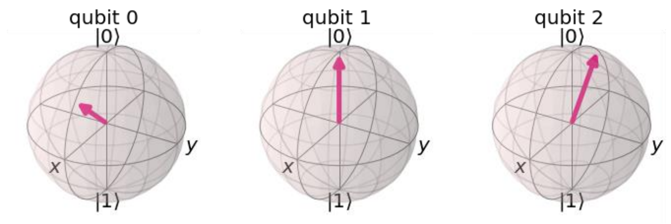

Fig. 13. Circuit in figure 12 after passing each qubit through the Hadamard gate.

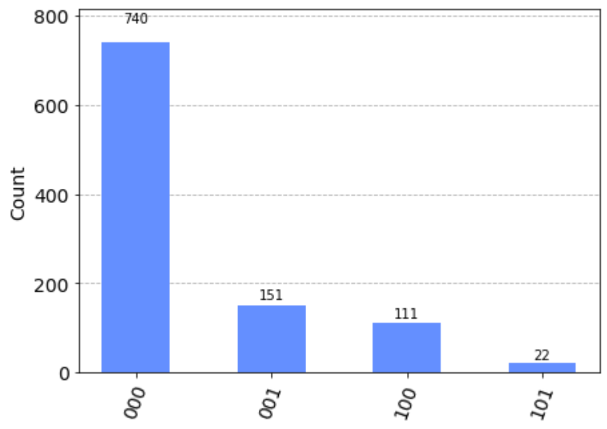

Fig. 14. Measurement result of the example circuit using the aer simulator, note that while the first and third qubits have uncertain measurements, the second qubit, which maintained the number we were looking for, results in all cases as 0.

Several tests were developed using this algorithm, changing both the number sought and the vector that would be sought for various scenarios, presenting satisfactory results.

Analyzing the algorithm and the results, we noticed that numbers with a value close to what we are looking for can present false positives, while numbers that are more spaced out maintain much better accuracy. This could be resolved if we reduced the range of numbers used, since as we saw previously, when we encode numbers in qubits, decreasing the operating range would increase the angle of difference between the numbers, and therefore increase precision in these cases where we have very close numbers.

Another point that we can identify from the results is the ability of a qubit to store and process information in relation to classical bits, in our algorithm, each qubit could store a hexadecimal value, and processing it by itself, and despite the superiority of a quantum processor is not the point under discussion, this capability is worth noting.

Starting with a complexity analysis, and thinking that for a vector to be searched of size n we also have n qubits so that our algorithm does not have to undergo adaptations (which in this case would consist of separating our vector to reuse the qubits, but that which is not our intention to address in this work), we noticed a large cost in the data coding stage, which, due to the need to process the data, has a cost of n, however, once the data is coded, the stage of applying the rotation to find our number and the application of our Hadamard gate, which can be integrated into the circuit in advance, and therefore executed in constant time.

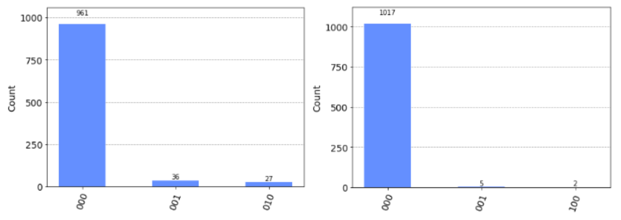

Fig. 15. Results for tests with the respective vectors: [7,14,11] and [4, 5, 6] searching for the respective numbers: 11 and 5

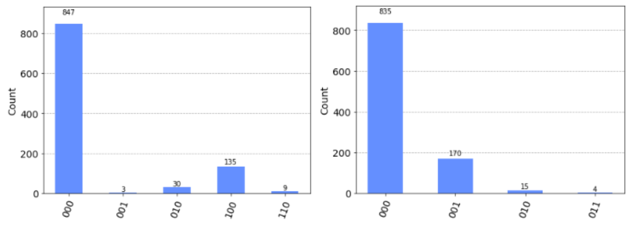

Fig. 16. Results for tests with the respective vectors: [4, 8, 12] and [15, 10, 7] searching for the respective numbers: 5 and 7.

VII. Final Considerations

The algorithm we chose to work and study was Grover's search algorithm, and the proposed algorithms were also search algorithms, we easily noticed a great difficulty in encoding classical data into qubits, where depending on the amount of classical data we have to work with, these will be responsible for most of the complexity of the algorithm.

Among the proposed algorithms, these were approached using two quantum strategies, one through probabilities, and one through rotations, starting with probabilities. We can see difficulties in working with this type of approach, since quantum systems have noise (still which decreases as processors develop) and easily lose precision if not worked with care, but still have plenty of potential.

Regarding the approach through rotations, we were able to analyze and better understand the functioning of qubits and the great difference they present in relation to their classical versions. The python notebook where these algorithms were executed and worked on is published on GitHub[1].

References

[1] Brylinski, Ranee K.; Chen, Goong (2002) "Mathematics of Quantum Computation", In: 1. ed. New York: Chapman and Hall/CRC. 448 p. ISBN 9780429122798.

[2] Divicenzo, David P. (1998) "Quantum gates and circuits", In: Royal Society, [s. l.], year 1998, v. 454, ed. 1969, january 8 1998. DOI https://doi.org/10.1098/rspa.1998.0159.

[3] Pasquale, R.; Bianchini, C. (2020) "Um estudo exploratório das falhas de segurança de algoritmos de criptografia na era da computação quântica" In: Jornada de Iniciação Científica e Mostra de Iniciação Tecnológica - ISSN 2526-4699, Brazil, December 2020.

[4] Rabelo, Wilson R.M. e Costa, Maria Lúcia M. (2018) "Uma abordagem pedagógica no ensino da computação quântica com um processador quântico de 5-qbits", In: Revista Brasileira de Ensino de Física [online]. 2018, v. 40, n. 4.

[5] Arute, Frank. "Quantum supremacy using a programmable superconducting processor", In: Nature, Nature, year 2019, v. 574, p. 505-510, 23 October 2019. DOI https://doi.org/10.1038/s41586-019-1666-5. Disponível em: https://rdcu.be/cBpMi.

[6] Peter W. Shor. (1997) "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer", In: SIAM Journal on Computing, 26(5):1484-1509, 1997. DOI https://doi.org/10.48550/arXiv.quant-ph/9508027.

[7] Lov K. Grover (1996) "A fast quantum mechanical algorithm for database search", In: https://arxiv.org/abs/quant-ph/9605043v3. DOI https://doi.org/10.48550/arXiv.quant-ph/9605043. Originally published in Proceedings, 28th Annual ACM Symposium on the Theory of Computing (STOC), May 1996, pages 212-219.

[8] Josn Preskill (2012) "Quantum computing and the entanglement frontier", In: https://arxiv.org/abs/1203.5813v3. DOI https://doi.org/10.48550/arXiv.1203.5813.

[1] URL to the notebook repository with implementations: https://github.com/giangamberi/Estudo-de-algoritmos-quanticos-e-suas-implementacoes