QUANTUM ORACLES - HOW TO TRANSFORM CLASSICAL PROBLEMS INTO QUANTUM ONES

Alexandre Silva
Computer Science
UNIVEM - Centro Universitário Eurípides de Marília

Luis Hilário Tobler Garcia
Computer Science
UNIVEM - Centro Universitário Eurípides de Marília

Maúricio Duarte
Information Technology
Fatec Garça – Deputado Julio Julinho Marcondes de Moura

August 12, 2024

ABSTRACT

Using quantum oracles and other effects, like superposition, *5 mini-projects* were done. The main goal of these projects was to answer if it's possible to bring some problems to the quantum realm and if such translation worth it. After finishing the tests, it was possible to see that, some of the implementations show descent results. However, classical computing is still a fundamental part of quantum algorithms.

1 Introduction

Now days, isn't difficult to hear someone talking about quantum computers, and how these machines will change the world. Nonetheless, the major fraction of these comments come from extrapolations and science fiction present in popular movies and series around the world. In this paper, I'm going to show that quantum computing it's not a magical trick to solve everything, but a tool for solving a distinct group of problems.
To do that, *5 mini-projects* were implemented using qiskit, an open source framework from IBM, exploiting quantum effects and using classical/quantum algorithms to reach the expected outcomes. After that, the results will be here compared with their classical counterparts. Such mini-projects are the following: Quantum File explorer3.1, miles to kilometers conversion3.2, Hanoi towers3.3, Buckshot Roulette 3.4 and QRAM 3.5. All these implementations can be found at my GitHub repository.

2 Oracles

Based on the idea of *Oracle Turing Machines* [1][2][3][4], the Oracles are mathematical modeling tools, used to abstract outlying parts of an algorithms into a black-box, making the algorithm analysis much easier. These machines could also be seen as a function, getting an input x e returning $f(x)$ with time-complexity $O(1)$. Because of these unreachable characteristics for real life, this model can't be implemented, being used only for formal description of decision problems.
However, in quantum computing, it's possible to implement something similar to that, taking advantage of your inner structure and quantum effects to gain a *Speed-up* relative to its classical counterparts, such Speed-up can be seen, for example, in the Deutsch–Jozsa algorithm [5]. Furthermore, the Quantum Oracles have a fundamental role determining the circuit complexity. Some approaches for that are: *depth*, calculating the longest path in the circuit that some information must pass through and *gate counting*, summing up how many gates were applied in the final circuit. Nevertheless, these approaches are very dependent of the *QPU* topology, differing from each *backend* used during the *transpilation* process. To solve this problem, a well-known technique is to pack some parts of the circuit into Oracles,

and then describing the complexity based on how many times they are called, it's also known as *query complexity* [6] [4].

2.1 Types of Oracles

Using the base description of Quantum Oracles, we can classify them based on their structures and how the data is processed.

2.1.1 Phase Oracle

The Phase Oracle, is the most well-known format. Some Algorithms, like Deutsch–Jozsa, Grover, Simon and Bernstein–Vazirani, use it to take advantage over Classical approaches.

2.1.1.1 Default Behavior

As its main characteristic, the Phase Oracle adds a global phase to the circuit, using quantum effects like *Phase Kickback* (basically the phase pass all the way through CNOT's target and is applied in the control Qubit), to change values in superposition.

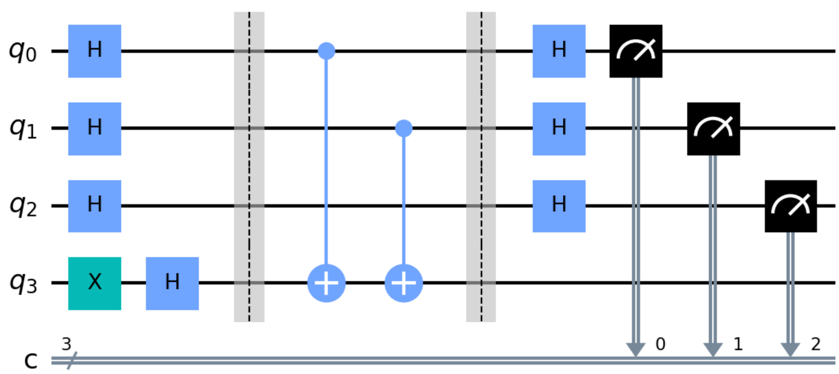
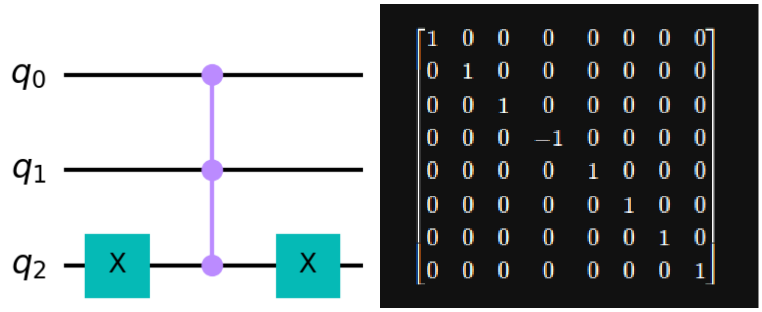


Figure 1: Phase Oracle using Phase-Kickback Example

In the Figure 1, a π phase was added in the auxiliary Qubit ($q3$), taking advantage of the phase $|-\rangle$, which will be the responsible to modify the values in the unitary matrix. In this setup, the $CNOT$ gates act slightly different from its default behavior, this way, the gate invert the value of the target Qubit and, due the π phase, it also acts like a Z gate applied in the control Qubit. So, after applying $CNOT\,|-\rangle\,|+\rangle$ (little-endian), the state becomes $\frac{1}{\sqrt{2}}(|0\rangle\,|-\rangle - |1\rangle\,|-\rangle)$, and after removing the superposition, the final state is: $\frac{1}{\sqrt{2}}(|+\rangle\,|1\rangle - |-\rangle\,|1\rangle)$. This way, the control Qubit is changed due the *Phase Kickback* and its state is flipped from $|0\rangle$ to $|1\rangle$. Taking this effect as and advantage, we can use it to encode some binary values inside these oracles and use them to do some calculations.

2.1.1.2 Minimal Oracle Version

Furthermore, There's another layout possible for implementing a Phase Oracle. Once this one only applies a phase in some bit-strings, the auxiliary Qubit can be removed, and the phase can be provided by CZ gates (or any other gate which can apply a phase for certain bit-strings). Doing that, it's possible to create a phase oracle keep its unitary and reversible nature, in the format of a minimal Oracle.



(a) Phase Oracle - Minimal Oracle Layout Example (b) Figure 2a Resulting Unitary Matrix

Figure 2: Phase Oracle Minimal Version

In the example above 2, a MCP gate was applied adding a π phase and another two X gates were used to invert the Qubits we want to have the value 0 as a trigger for the MCP phase application. Following the little-endian format the final encoded bit-string is: 011_2 or 3_{10}.

Looking at its unitary matrix 2b, it's possible to see the 8×8 identity matrix with a -1 value in the column relative to the bit-string 011_2, representing the encoded value.

2.1.2 Boolean Oracle

The Boolean Oracle has a similar structure with the Phase Oracle. However, this time a phase isn't applied, acting this way a regular boolean function, mapping inputs to outputs.

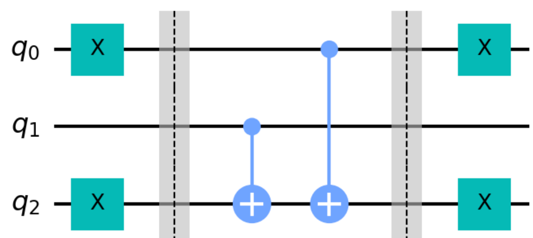


Figure 3: Boolean Oracle Example

The example in the Figure 3, is a well-known balanced Oracle for the Deutsch-Jozsa's algorithm. Nevertheless, to implement it, the Oracle must be transformed into a Phase Oracle.

2.1.3 Minimal Oracle

As previously mentioned, the Minimal Oracle is a function which its essence is unitary and reversible, so no additional Qubits are required. Therefore, this format can be either Boolean or Phase Oracle, depending on its implementation.

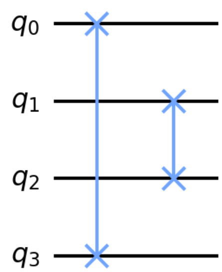


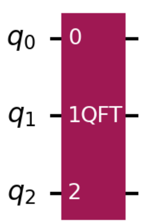
Figure 4: Boolean Minimal Oracle example

The example 4 shows an Boolean Oracle which two $SWAP$ gates were used to invert the Qubits values order. Doing this way, the final matrix keeps its unitary and reversible properties, once the $SWAP$ gates act in both Qubits together, mirroring its action.

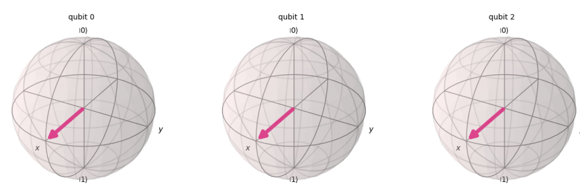
2.1.4 QFT(Quantum Fourier Transform)

The QFT is, in a nutshell, a quantum algorithm based on the Discrete Fourier Transform, used for quantum states period finding, projecting values from the computational basis onto the X basis (also know as Fourier basis).

Even though it's an algorithm by itself, its application is done in the format of Oracles in quantum circuits.



(a) QFT Oracle example



(b) Values Mapped onto Fourier basis example (quantum state $|000\rangle$)

Figure 5: QFT Oracle

2.1.5 Other kinds of Oracles

In addition to the oracles mentioned above, it's possible come across with different mentions of Oracles, like the Simons's, Deutsch-Jozsa's, etc. However, these are just implementations of models already shown in this paper. Also, the most relevant for the following projects are the Phase and Boolean ones, due that, there's no use to keep discussing about many others sub-categories of Oracles here.

3 Development

3.1 File Explorer

Imagine a quantum computer, but instead of one of those we already have in real life, imagine a different kind, very similar to those we have at home, with a quantum operational system, quantum files, etc. This imaginary version could be thought as a hybrid computer as well, taking advantage of both quantum and classical capabilities. Using this idea, the first project implements a file explorer for this ideal system.

3.1.1 Algorithms Used

3.1.1.1 Grover

The Standard algorithm for search problems is the Grover's algorithm. This one, realizes searches on unsorted "databases" (bit-string in this case) with $O(\sqrt{2^n})$, being n the number of Qubits. Its tenets are based on amplitude amplification, using a Phase Oracle to mark some bit-strings and then using a Diffuser to amplify the probability to find these values after measuring it.

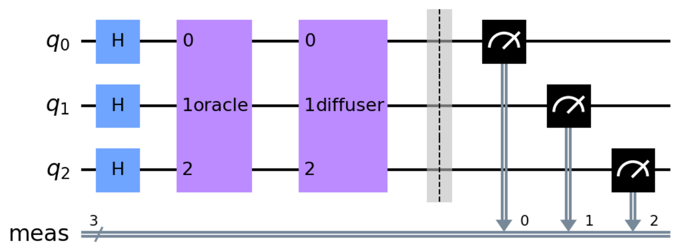


Figure 6: Example Grover's algorithm with 3 Qubits

For building the circuit, is needed a joint of $Oracle + Diffuser$ applied k time, which $k \approx \frac{\pi}{4\sqrt{\frac{a}{2^n}}} - \frac{1}{2}$, being a the number of values marked by the Oracle. Once we want to find just a single file, the use of this relation is not required, being need to apply the algorithm once to find the bit-string.

Even this being the best option for bit-string finding, in this project different rotations were tested trying to find a better superposition which increases the chances to find the value we want.

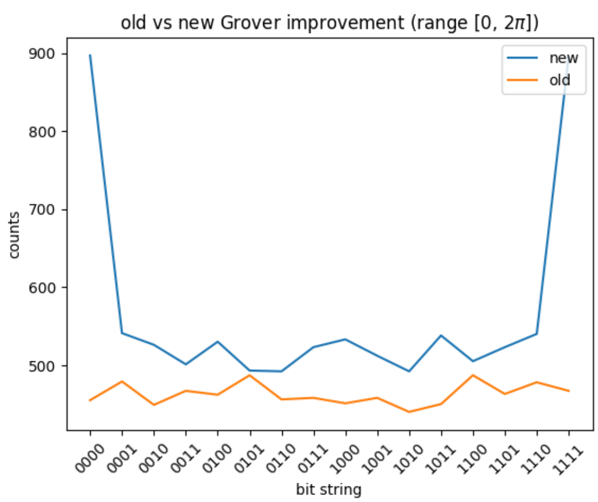


Figure 7: Comparing the Standard Grover's algorithm with the version modifying the rotation angles $[0, 2\pi]$ for each 4 bits bit-string

Using specific rotations for each bit-string, it's possible to improve the outcomes, and even stand out the default rotation.

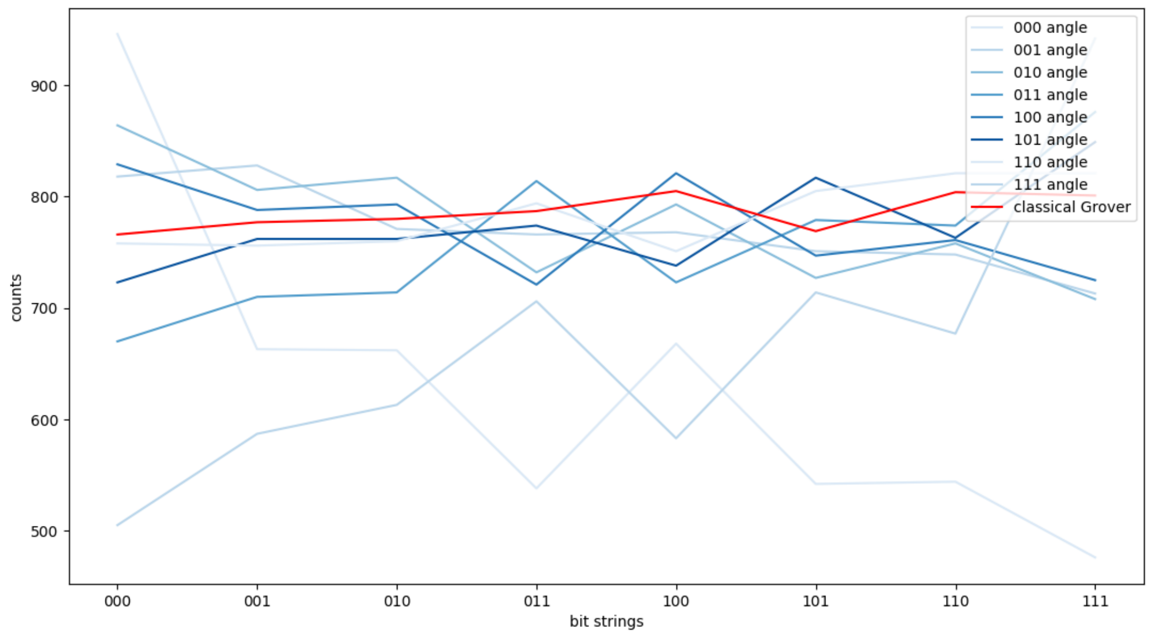
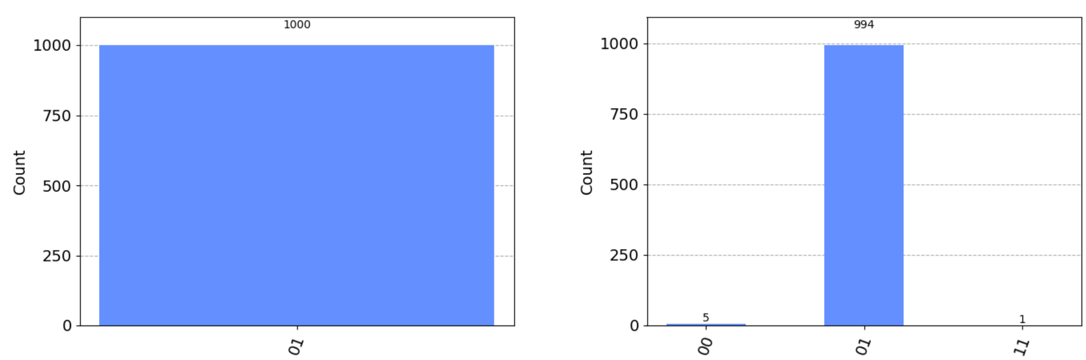


Figure 8: Test Using the best angles of each bit-string in different ones

However, using the best angle of a distinct bit-string as the source of superposition in the algorithm, isn't enough to overcome the classical Grover implementation, and also the probability distribution becomes irregular. So, for a general purpose implementation, the default rotation is the best one for the most part of the cases. This behavior is even weird with 2 Qubits, being the H gate rotation the best ever for that.



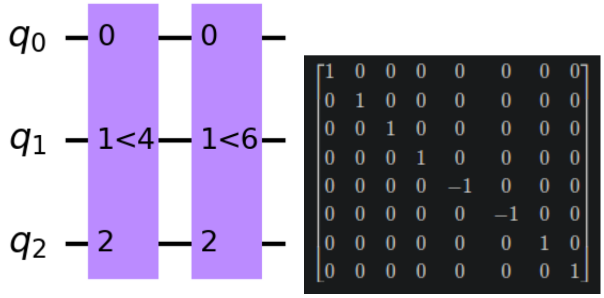
(a) Results for Grover with 2 Qubits (default rotation) (b) Results for Grover with 2 Qubits (modified rotation)

Figure 9: Comparing Grover with 2 qubits using the default and modified rotation

For our purpose in this project, the hybrid approach was used. So during the circuit setup, the searched value pass through a Hash-Table will optimal angles. This way, the algorithm can achieve better outcomes at the end of the execution, finding, this way, the file we want.

3.1.1.2 Sets Difference

Overlapping two distinct Phase Oracles, each one with a range of values encoded. The remaining Phases, are the difference of the sets encoded by them [7].



(a) Sets Difference Algorithm Example (b) Unitary After applying the Sets Difference algorithm

Figure 10: Sets Difference example

In the last example 10a, the set $\{000, 001, 010, 0110\}$ was encoded in the first Oracle and $\{000, 001, 010, 011, 100, 101\}$ in the second one.

After Overlapping the phases 10b, only the values $\{100, 101\}$ keep the -1 value. This way, we can do the set operation just applying some oracles with the values range we want.

3.1.2 Final Solution

As a solution for this problem, a hash function was made, mapping the path of some file p to a specific bit-string b, $F : p \rightarrow b$. With this function, we can use a set returned values and encode them into a Phase Oracle, creating a sort of quantum Look-Up-Table for files in that machine. Moreover, a second Oracle containing all the bit-strings, with the exception of the searched one, is applied for difference of sets algorithm, this way the phases will be overlapped, and only the one we want will pass to the Grover's algorithm.

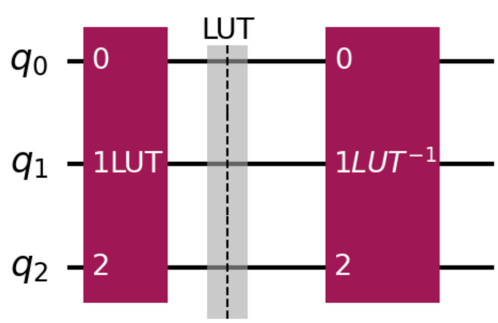


Figure 11: Look-up-Tables applied

Finally, the modified version of the Grover's algorithm is applied.

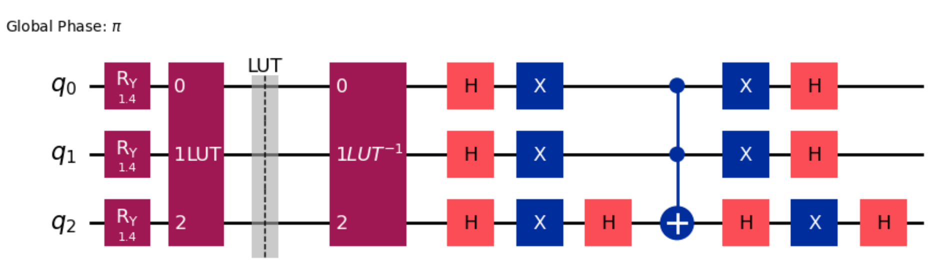


Figure 12: File explorer final circuit

This setup, maximizes the probability to find the queried file, presenting the following distribution after 1000 shots.

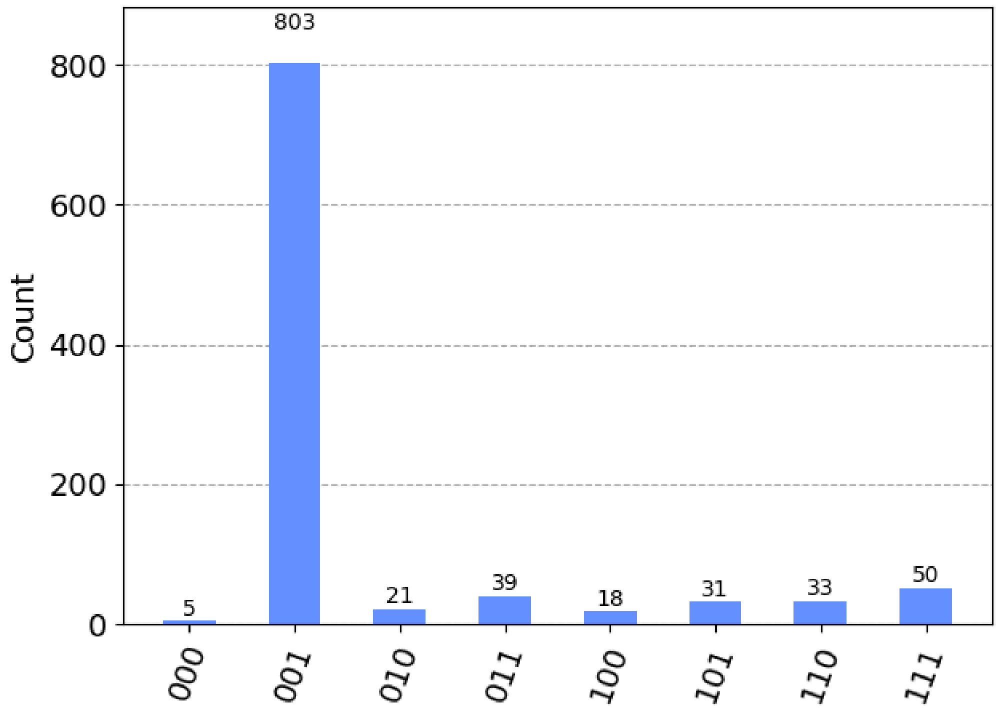


Figure 13: File Explorer Results after running with Qiskit-AER

3.1.3 Results

For this specific ideal case, it is, probably, one of the best known ways to do the search between "quantum files".

However, it's not the best alternative as a way to speed-up classical searches. It happens because storing a large Look-Up-Table for encoded files and another one for keeping track of each optimal angle for every bit-string contained between 2^n combinations, could be really costly and slow. Also, it would be required a hash function with a tiny probability of collision.

An alternative for that could be using the standard Grover's algorithm, but it still the need to encode the hard drive files to a big table. Also, comparing the classical tree approach with Grover's, their worst cases scenarios are $O(log(n))$ and $O(\sqrt{n})$, which implies that the first one still faster than the later. As a last point, the quantum version for this problem, could present the wrong file after searching due errors and the quantum randomness.

Keeping that in mind, the final algorithm is a right choice for a quantum system as describe at the beginning of this section, but it can't overcome the best algorithm in a classical computer.

3.2 Miles to Kilometers Conversion

The second problem that was tested here, was the conversion from miles to kilometers. This idea came out after finding a quantum algorithm capable of calculate the Fibonacci sequence, which is an essential piece for this very project.

3.2.1 Used Algorithms

3.2.1.1 Quantum Fibonacci

The quantum version of Fibonacci was presented in [8]. This paper shows that, using a quantum circuit with all bit-string in superposition, and then apply controlled rotations to remove values that contains consecutive ones, is possible to approximate the result of the *n*-th Fibonacci sequence value.

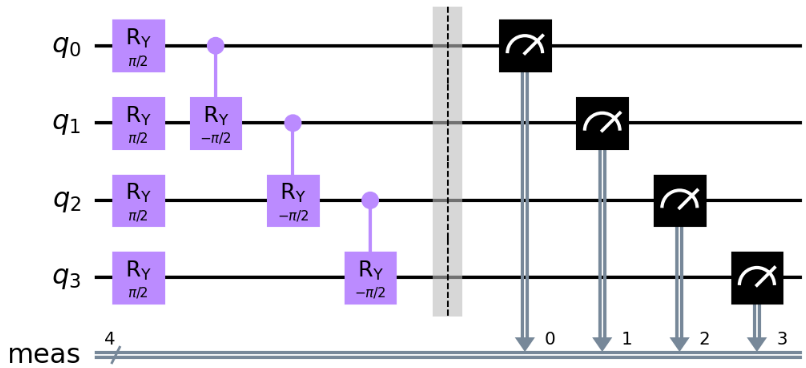


Figure 14: Quantum Fibonacci Example

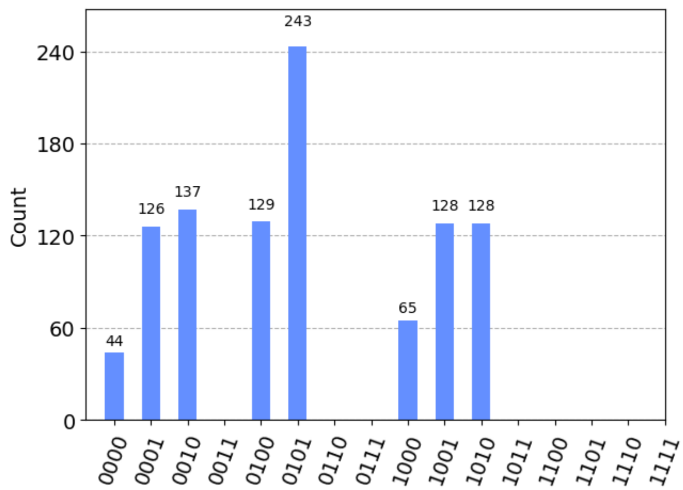


Figure 15: Quantum Fibonacci result - F(4)

After running the circuit k times, the amount of unique bit-string that appeared during the experiment represent the n-th Fibonacci value, which n is the total of Qubits were used in the circuit. So, for the example 14 above, the position n we want, is 4, and the result after counting the bars of which values are $\neq 0$ give us the result of $F(4) = 8$.

It may seem wrong, however, the quantum setup differ slightly from the classical one, once the first value is 2 instead of 1, this way, the sequence is: $F(1) = 2, F(2) = 3, F(3) = 5, F(4) = 8, F(5) = 13, F(6) = 21, ...$, which implies that it worked for this example.

3.2.1.2 Aproximação de Milhas para Quilômetros usando Fibonacci

Para aproximar o valor de milhas para quilômetros, podemos utilizar a sequência de Fibonacci com a seguinte relação: $F_{km} = F_{milhas}(n + 1)$, sendo aqui F a versão clássica de Fibonacci com $F(1) = 1$ e $F(2) = 2$. Dessa forma, se a posição n é conhecida, valor aproximado em quilômetros será dado em $n + 1$.

milhas	km
1	2
2	3
3	5
5	8

Table 1: valores aproximados de Milhas para Quilômetros

Valores não presentes na sequência, podem ser aproximados repartindo o valor em partes menores. Por exemplo, para transformar 10 milhas em quilômetros, podemos fazer: $8 + 2 = 10miles \rightarrow F(5) + F(2) \rightarrow F(5+1) + F(2+1) = 13 + 3 = 16km$, aproximando então do valor mais preciso de ≈ 16.0934

3.2.2 Implementação do circuito

Com essa formulação, o algoritmo final segue esse fluxo:

Algorithm 1 Algoritmo quântico para a conversão

partes = $quebraValor(valorDeEntrada)$
for parte in partes **do**
    Aplique o Oracle $F(parte)$
    Faça as medições nos qubits
    Reset os qubits usados
end for
verifique o resultado de cada bit-string
Multiplique cada resultado com o valor i correspondente

Nesse formato, é necessário pré-processamento utilizando um algoritmo clássico para dividir o número em partes menores. Este então, retorna tuplas mapeando a entrada para o valor n_i e a quantidade de vezes que é necessário a sua aplicação p, $(n) \rightarrow ((n_1, p_1), (n_2, p_2), ...)$.
A partir disso, a parte quântica segue com a aplicação do algoritmo de Fibonacci em formato de Oracle no circuito para o valor n_i, em seguida as medições nos qubits usados pelo Oracle e por fim o reset deles, seguindo esse ciclo para cada valor n.
Após terminar, basta pegar os resultados, e, com um pouco de pós-processamento, agrupar as partes e multiplicar pelo seus valores p, retornando então o valor aproximado em quilômetros.

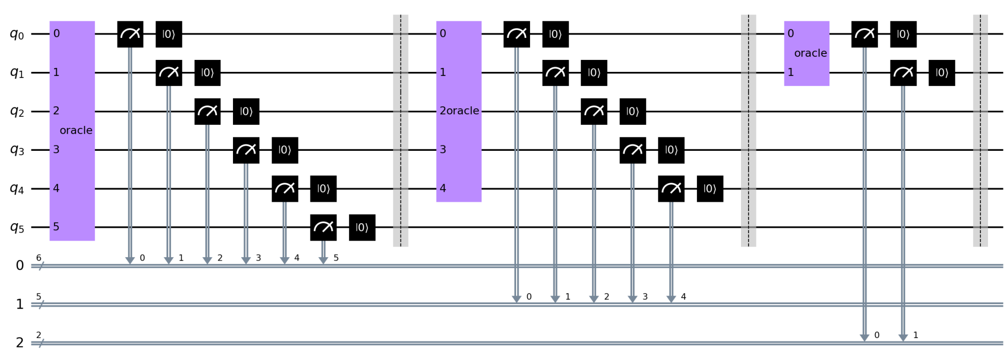


Figure 16: Circuito de conversão

3.2.3 Resultados

Usando esse método, é possível alcançar os valores esperados. Contudo existem alguns pontos que tornam esse método inviável:

1. **Quantidade necessária de medições e tempo de execução**
   Para cada medição do circuito, é necessária uma quantidade alta de *shots* (valores entre 5000 e 10000 foram testados localmente usando o Qiskit AER e, para os testes no hardware da IBM, foram usados apenas 1000 por questões de extrema demora e erros durante os experimentos) para alcançar melhores resultados, aumentando também o tempo necessário para finalizar a execução.

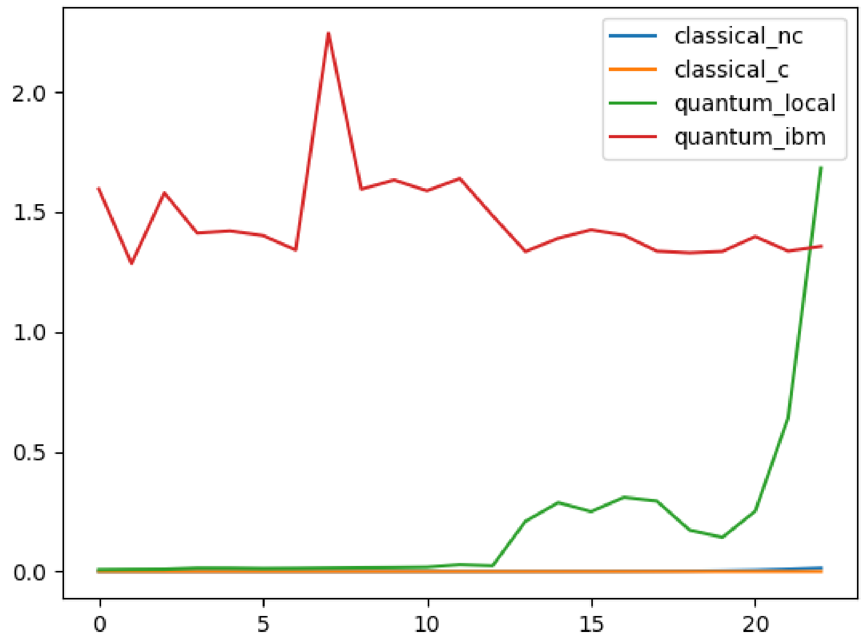


Figure 17: Comparação tempos de execução

Como mostrado em 17, o tempo das versões clássicas, com e sem memoization, possuem tempos praticamente constantes em relação as versões quânticas.

2. **Erros**
   Como a maioria dos algoritmos Quânticos da era NISQ(noisy intermediate-scale quantum), os erros também estão presentes, e por serem utilizados inúmeros gates multi-qubits, esses erros podem ainda se intensificar de acordo com hardware usado.

3. **Imprecisão**

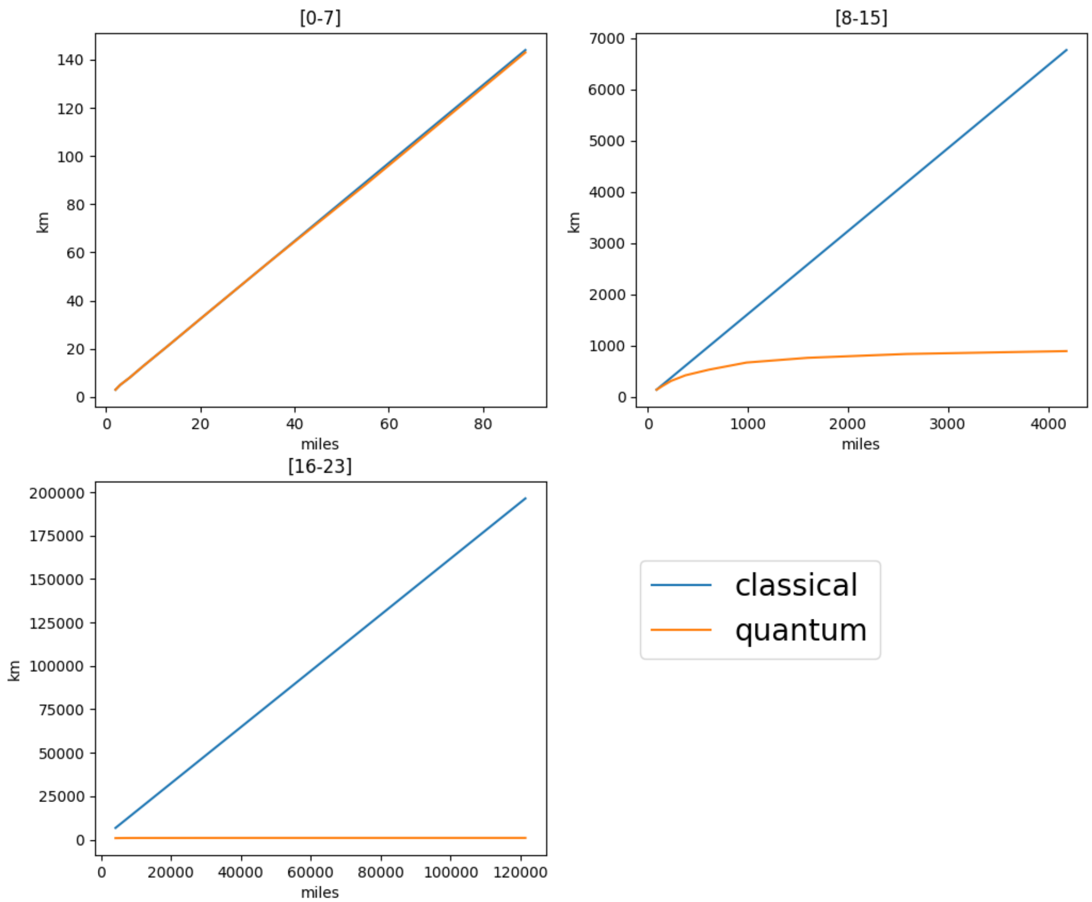


Figure 18: Comparação resultados versão clássica e quântica

Como mostrado em 18, valores pequenos possuem uma boa precisão com os números esperados, mas a partir de certo ponto, eles começam a se distanciar e perdem totalmente a precisão.

4. **Necessidade de intervenção clássica**
   Por requisitar pré e pós processamento clássico e apenas uma pequena parcela ser de fato processamento quântico, a necessidade de utilizar esse algoritmo se reduz a zero.

Sendo assim, esse algoritmo não consegue se sair bem como a versão clássica, além de ser mais custoso na maioria dos casos. Para evoluir essa implementação, será necessário remodelá-lo para um versão com pouca, ou nenhuma, computação clássica, priorizando a maneira como dados podem ser encodados e transformados no circuito.

3.3 Torres de Hanoi

Para a criação das torres de Hanoi, foi pensado em uma maneira de encodar a posição dos discos na torre utilizando seus valores binários e o Phase Oracle como meio de armazenamento.

3.3.1 Implementação

Para esse projeto, são necessários $(\lfloor \log_2 x \rfloor + 1) * 3$ qubits, sendo x o número de discos. Estes seguem a ordem $|t_{n-1}t_{n-2}...t_0\rangle\, |a_{n-1}a_{n-2}...a_0\rangle\, |s_{n-1}s_{n-2}...s_0\rangle$, sendo s,a,t a primeira, segunda e última torre respectivamente, e

$n = \frac{nqubits}{3}$.
Com essa configuração, os números de 1 à x são codificados em seu formato binário nos qubits s, utilizando a fase global π. Em seguida, são realizadas operações de $swap$ bit-a-bit para mover os valores dos n qubits menos significativos para os n mais significativos.
Para realizar essas operações, é necessário pré-calcular, classicamente, a sequência de movimentos usados [9] [10] [11] [12]. Dessa forma, essa versão quântica age como um jogador com uma lista de passos a serem seguidos, executando-os um-a-um.

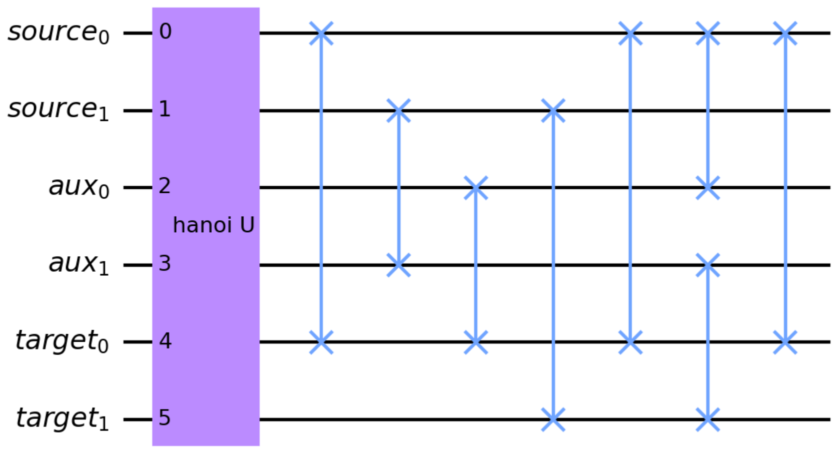


Figure 19: Torre de Hanoi com 3 discos

Nesse circuito, pode-se utilizar algoritmos adicionar, como o algoritmo de Grover, para verificar o resultado, ou executar outras operações nos valores.

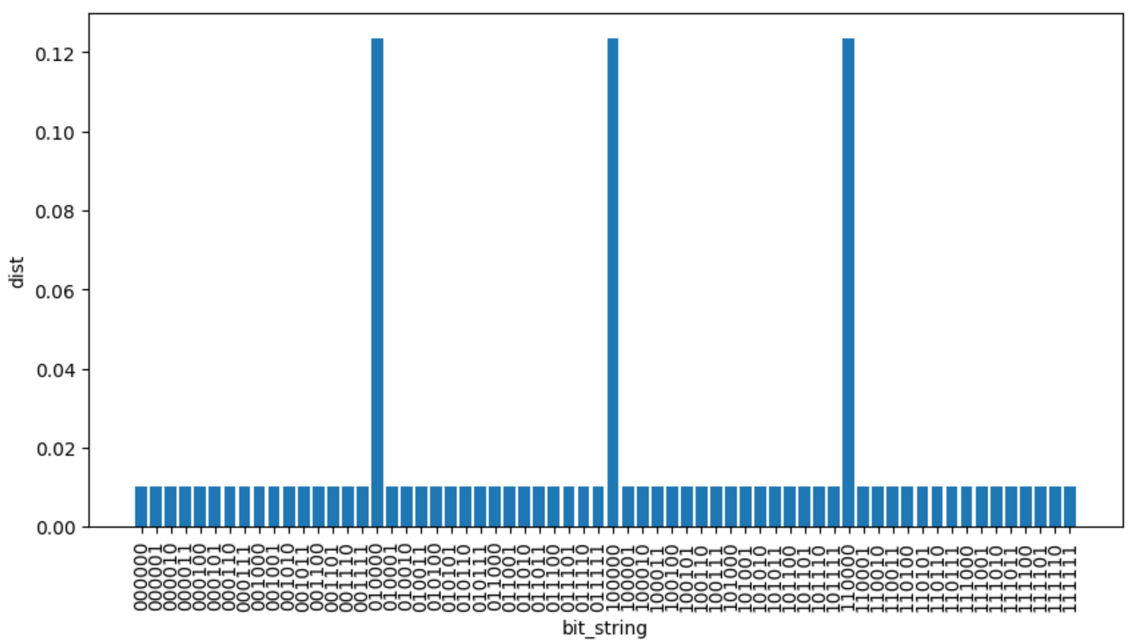


Figure 20: Resultado usando Grover - Torre de Hanoi com 3 discos

Em 20, os 3 maiores resultados obtidos são as bit-strings com 01, 10 e 11 nos bits mais significativos. Sendo assim, o resultado esperado para uma torre com 3 discos, foi atingido.

3.3.2 Resultados

Nessa versão, é seguida a mesma sequência do algoritmo clássico, necessitando, inclusive, de pré-processamento para conseguir a sequência de ações.
Em uma versão clássica, o movimento de retirar um disco de uma torre e move-lo para a próxima requer também esse pré-processamento, podendo ser realizado um-a-um ou tudo de uma vez antes da partida. Dessa forma, a versão clássica e quântica se igualam, não tendo ganhos ou perdas expressivas.

3.4 Buckshot Roulette

Buckshot Roulette é um jogo de computador feito pelo desenvolvedor Mike Klubnika, tomando como base a premissa de reinventar a infame roleta russa. No jogo, você é desafiado por um demônio (dealer), e caso você ganhe, uma recompensa lhe será dado, caso contrário o jogo reinicia e você pode tentar novamente.
Nesse projeto, foi tomado como objetivo analisar a primeira rodado do jogo e tentar encontrar a melhor estratégia para maximizar os ganhos do jogador. O motivo da escolha da primeira rodada se dá pela sua simplicidade, sendo direto ao ponto, sem power-ups ou fatores que dificultariam as simulações, mas, ainda assim, mantendo a essência do jogo.

3.4.1 Dinâmica

Na rodada, são colocadas 2 balas falsas e 1 bala verdadeira na arma, sendo o player o primeiro a jogar. Ambos os jogadores podem escolher entre atirar em si mesmo, ou em seu oponente. Assim, a próxima ação é estritamente depende das probabilidades de ser uma bala real ou falsa. A partir dai, a dinâmica funciona da seguinte forma:

Algorithm 2 Possíveis jogadas

```
if jogador escolhe atirar no dealer then
    if bala for real then
        Jogador ganha a rodada
    else
        Dealer joga a próxima
    end if
else
    if bala for real then
        Jogador perde
    else
        Player joga a próxima
    end if
end if
```

Essa dinâmica se repete a cada jogada, sendo válida tanto para o dealer, como para o player.

3.4.2 Versão clássica

Para entender melhor a dinâmica, é possível representar cada ação e suas consequências em formato de árvore. Dessa forma, cada jogada leva a partida para mais próximo do fim.

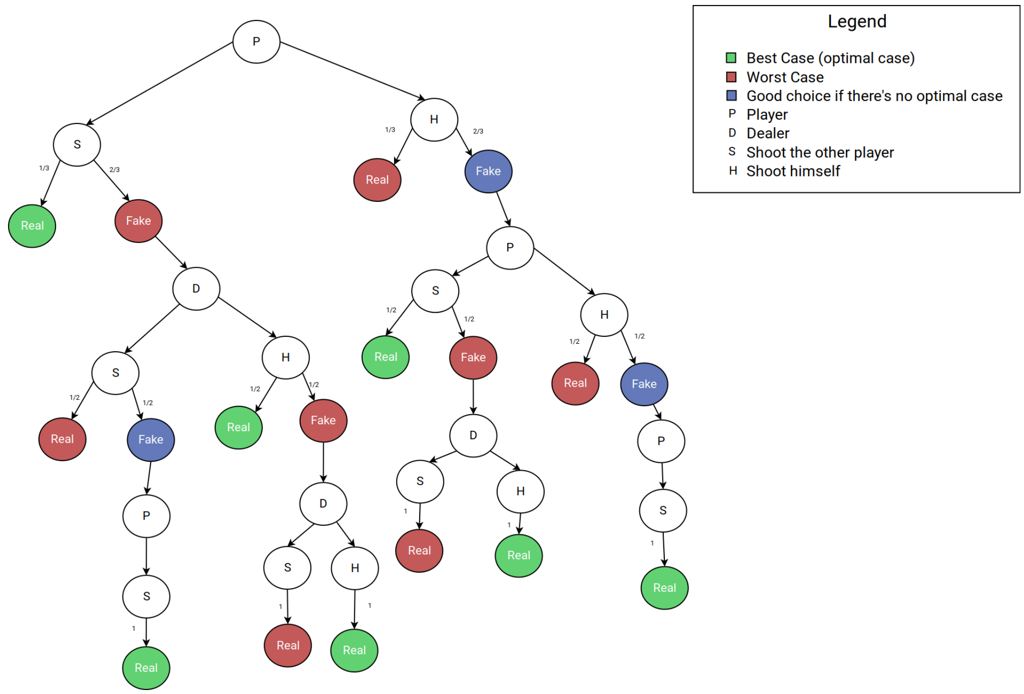


Figure 21: Buckshot Roulette diagrama de árvore

Nessa estrutura, é previsto que o jogador seja um agente racional, e o dealer uma máquina com ações aleatórias. Assim, o jogador sempre visa o seu próprio benefício, enquanto o dealer age pela sorte.Tal comportamento pode ser visto nas folhas da árvore do qual, sempre que o player é o próximo jogador, sua ação é apenas atirar no adversário, enquanto o dealer ainda possui a possibilidade de entregar o jogo atirando em si próprio, mesmo havendo apenas uma bala na arma e, pela lógica do jogo, ser uma bala verdadeira.
Seguindo essa estrutura, podemos simular os possíveis caminhos e verificar a melhor estratégia.

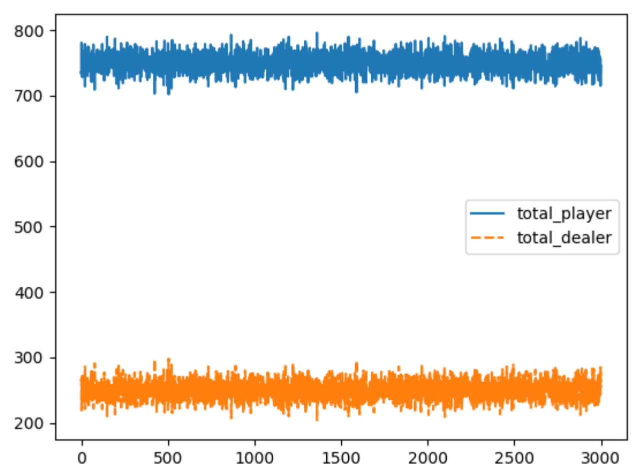


Figure 22: Buckshot Roulette clássico - melhor estratégia

Após testar os caminhos possíveis, o melhor resultado obtido foi esse apresentado acima em 22. Com um pouco de investigação, foi possível entender que essa estratégia se baseia no jogador começar atirando no dealer. Isso acontece, pois, ao seguir tal caminho, ele tem uma chance a menos de perder a rodada ao atirar em si mesmo logo no começo da partida.

rodada	ação	resultado da ação	resultado da partida
1	player atira no dealer	real	player ganha
1	player atira no dealer	fake	-
2	dealer atira no player	real	dealer ganha
2	dealer atira no player	fake	-
2	dealer atira nele mesmo	real	player ganha
2	dealer atira nele mesmo	fake	-
3	player atira no dealer	real	player ganha
3	dealer atira no player	real	dealer ganha
3	dealer atira nele mesmo	real	player ganha

Table 2: melhor estratégia - possíveis resultados

3.4.3 Versão quântica

Para a versão quântica, um circuito foi modelado imitando o funcionamento do game. Nesse algoritmo, um Oracle foi usado para cada jogador, implementando internamente sua estratégia.

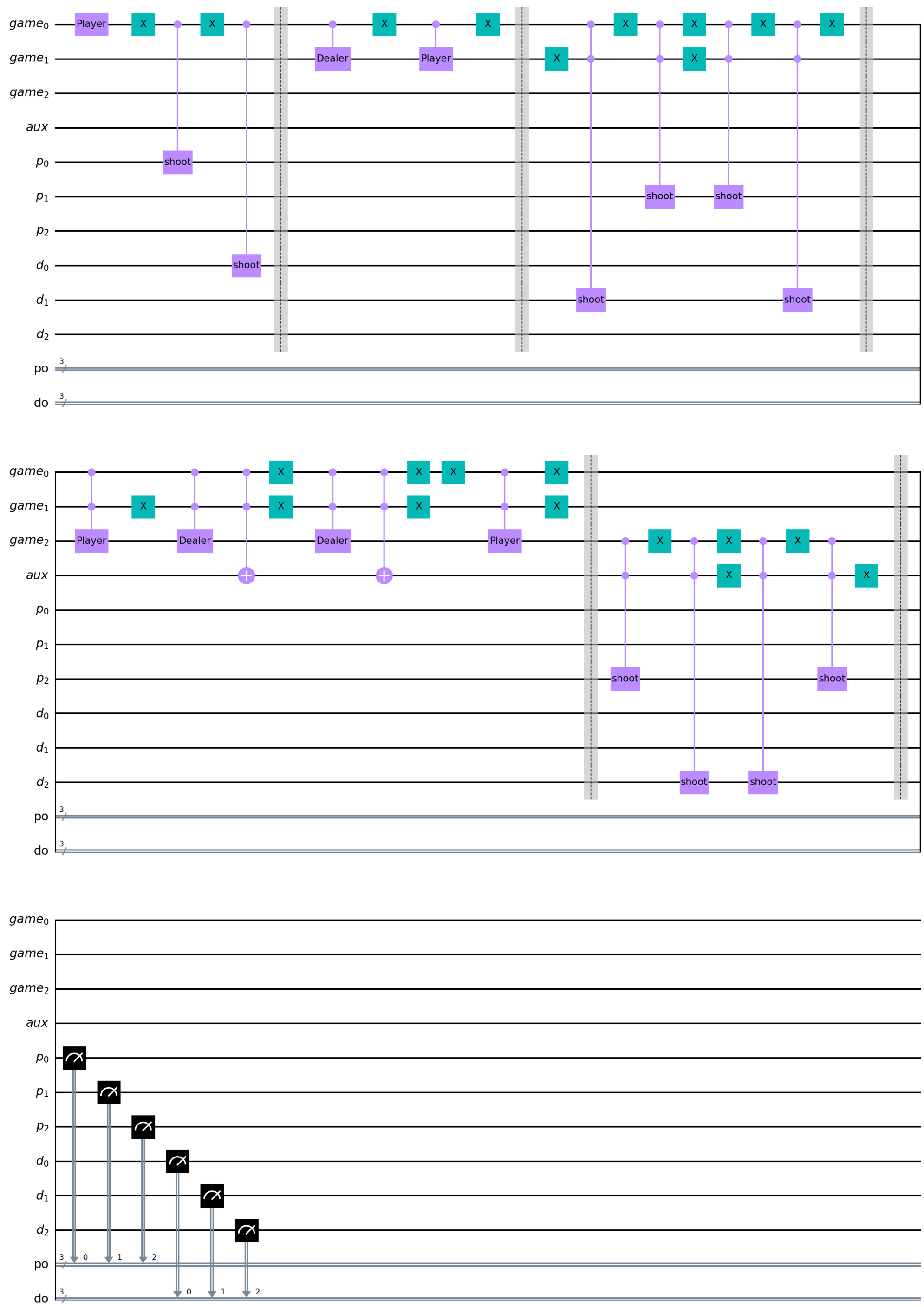


Figure 23: Circuito para o Buckshot Roulette

Além disso, para encontrar a estratégia, foram inseridos dois parâmetros dentro do Oracle do player, sendo possível configurar qualquer valor θ e ϕ para modificar a rotação na Bloch Sphere.
Após verificar os possíveis valores, a rotação que entregou o melhor resultado foi $\theta \approx 3.0853981633974477, \phi \approx 3.7853981633974474$ radianos. Usando essa estratégia, os resultados foram semelhantes a versão clássica.

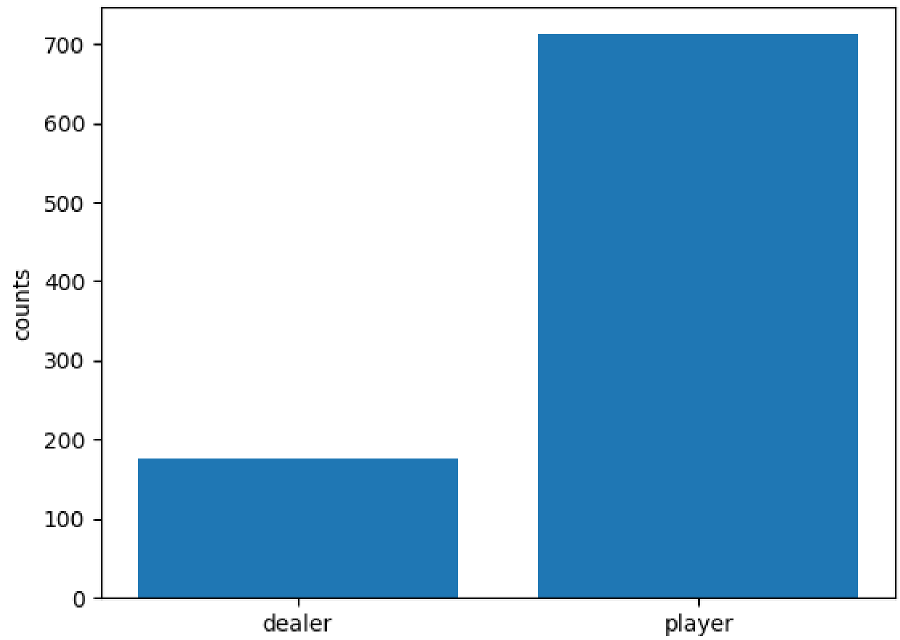


Figure 24: Resultado Buckshot Roulette quântico - Qiskit AER

Observando a Bloch Sphere do estado gerado por essa rotação, é possível ver também que a estratégia se aproxima da versão clássica, com o player preferindo atirar no dealer a maior parte do tempo (o valor 1 representa atirar no outro jogador e 0 em si mesmo).

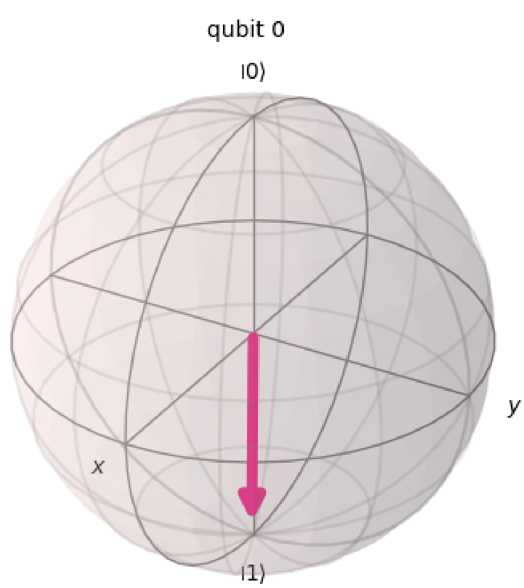


Figure 25: Melhor estratégia Buckshot Roulette quântico - Bloch Sphere

Como uma última nota sobre o circuito, no exemplo 24, o total de partidas ganhas por cada jogador não chega ao total jogado (nesse caso 1000 partidas foram simuladas). Isso acontece devido ao design do circuito, o qual não é possível verificar a jogada do player anterior, acarretando na continuação do jogo mesmo que um dos players já tenha perdido, o que cria a necessidade do uso de pós processamento para limpar os resultados inválidos.

3.4.4 Conclusões

Para esse problema, não há uma competição certa entre as duas versões, já que uma é diretamente inspirada na outra. Contudo, a versão quântica possui ainda a possibilidade de explorar mais valores do que a versão clássica, deixando o player mais aberto a escolha de novas estratégias, o que pode ser visto como um ponto a favor da versão quântica.
Em suma, ambos as simulações atingiram o mesmo resultado e foi demonstrado que é possível usar o quantum Oracle como uma representação de um player dentro do circuito.

3.5 QRAM

Por fim, o último projeto realizado foi o de uma *QRAM* utilizando os Oracles. Nessa versão, foi testado a criação de *QROMs* (com dados estáticos dentro), e uma possível maneira de utilizar uma QRAM hábil para escrita.
Neste projeto, foi tido como objetivo o armazenamento de estados quânticos (superposições), e não apenas de bit-strings clássicas. Isso pois, para garantir a real eficiência da computação quântica, a superposição é indispensável, e seu armazenamento pode ser um ponto chave para algoritmos melhores.

3.5.1 QROM

Para a QROM, são utilizados n qubits para o barramento de endereços e m qubits para a o barramento de dados, sem a necessidade desses valores estarem correlacionados, podendo assim ser utilizado, por exemplo, $n = 3; m = 10$. Nessa estrutura, podemos mapear diversas superposições diferentes e aplicá-las quando certo endereço for chamado. Sendo assim, o algoritmo armazena os valores a partir da configuração de gates controlados interiores ao Oracle, criando uma superposição apenas quando certo valor de entrada é inserido, seguindo o formato: $|0\rangle^{\otimes m}|a_{n-1}a_{n-2}...a_0\rangle$.

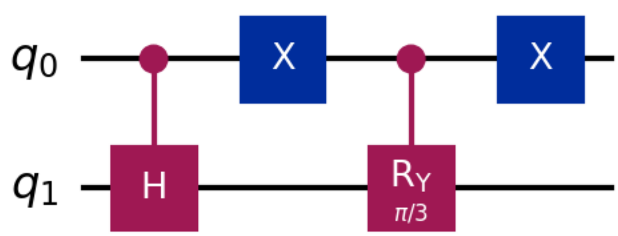


Figure 26: Exemplo circuito - QROM

Em 26, q_0 age como o barramento de endereços, enquanto q_1 como o barramento de dados. Aqui configuramos para mapear o endereço $0 \rightarrow RY(\frac{\pi}{3})$ e $1 \rightarrow H$. Sendo assim, para n qubits no barramento de endereços é possível mapear para 2^n estados, e com os m qubits é possível criar estados mais complexos aumentando sua quantidade e utilizando outros gates acionados para um mesmo endereço.
A partir da abstração desse circuito para um Oracle,é possível utilizar a QROM em um circuito maior, chamando-o novamente sempre que for necessário um certo estado. Além disso, no formato de Oracle, há a possibilidade de colocar os endereços em superposição e ter uma mistura de estados na saída.

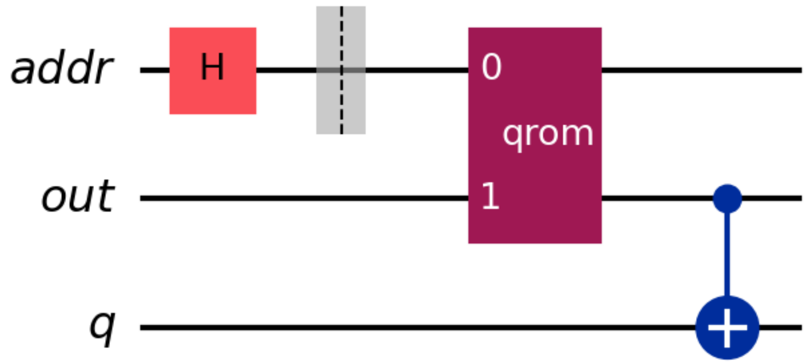


Figure 27: Exemplo circuito usando a QROM com endereços em superposição

Nesse exemplo 27, os endereços são colocados em superposição no qubit *addr* e assim os estados em internos do Oracle são colocados em uma sobreposição de $50 - 50$ no qubit *out*. Com isso, pode-se aproveitar do resultado de *out* em outros qubits, como nesse caso o qubit q.
Contudo, devido ao no-cloning-theorem, não é possível copiar esse estado para outro qubit alvo. Sendo assim, não é possível ter dois qubits com o mesmo estado a partir daquele armazenado, podemos apenas pegar o resultado de uma superposição e utilizar o valor binário como trigger para outra operação.
Uma opção para solucionar isso, é utilizar o teleporte quântico, destruindo assim o estado interno do Oracle e movendo-o para outro qubit desejado.

3.5.2 QRAM

Para criar uma QRAM com a possibilidade de escrita, o teleporte quântico, já citado anteriormente, é um caminho para isso. Com ele, podemos ter n qubits, sendo cada qubit um endereço único, e utilizar do teleporte para mover um estado que estava no circuito, para o domínio da QRAM.

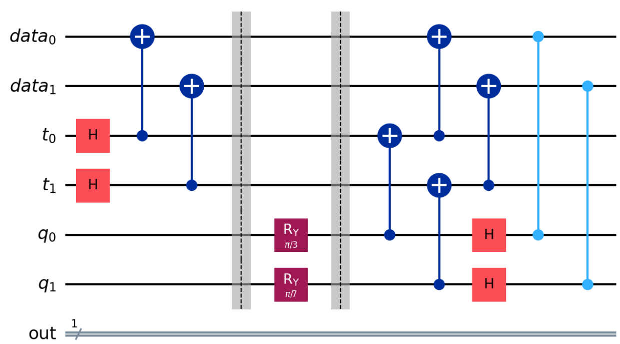


Figure 28: Exemplo circuito - QRAM

Aqui, os n qubits agem tanto como endereços quanto dados (qubits *data*). Além disso, são necessários mais n qubits para o teleporte (qubits *t*).
Com isso, é possível ver que o circuito cresce de forma linear a medida que mais endereços são requisitados, sendo assim $O(2 * n)$ em relação à quantidade de qubits total.
Na configuração acima 28, é possível sobre-escrever valores, assim como interferir com outras superposições apenas teleportando novos valores para o qubit i. Dessa forma, podemos criar uma memória menor e, conforme necessário, remover e adicionar outros valores.

3.5.3 Conclusões

Com esse projeto, e com a literatura usada [13][14], é possível entender que criar versões quânticas de memória é uma tarefa desafiadora, e ainda não é possível tomar proveito de todo o seu potencial usando as superposições e estados de outras bases a não ser a base computacional $(0, 1)$. Fatores como, complexidade de mapear dados, complexidade de utilizar a memória (já que é necessário reaplica-lá toda vez que for requisitado seu uso), no-cloning-theorem, decorrência, etc. Influenciam diretamente na possibilidade de sua criação.Mesmo sendo possível implementar pequenos circuitos que agem como memória, como os mostrados aqui, ainda não é usual e muito menos universal para qualquer tipo de máquina quântica.
Além disso, por esses fatores, a QRAM, pode dificultar a execução de múltiplas tarefas, uma vez que o valor presente nela não pode ser copiado, e ao move-lo para outro qubit, o valor anterior da QRAM é completamente destruído.
Como mostrado na literatura, para resolver esses problemas, o melhor approach para a sua implementação, é a utilização de um hardware especifico para essa finalidade, sem a intervenção de circuitos quânticos.
Em suma, mesmo sendo possível criar pequenos circuitos para implementar uma memória, seu uso está longe de se comparar as versões clássicas.

3.6 Conclusão

Perante o exposto, foi evidenciado que a computação quântica ainda tem muito potencial. No entanto, é possível ver que certos fatores, e a falta de alguns recursos, prejudicam o seu uso no momento.
Como já mostrado pelas inúmeras pesquisas em áreas como, química, machine learning, criptografia, otimização, etc. A computação quântica pode, num futuro próximo, ser um ponto crucial para conseguir resultados mais precisos e, em certos casos, em menor tempo.
No entanto, na era NISQ, para conseguir utilizar todo seu potencial, é necessário ter em conjunto máquinas clássicas para pré e/ou pós processamento, seja para executar alguma tarefa computacionalmente custosa para um computador quântico, ou para o uso de algoritmos de detecção e correção de erros. Como demonstrado aqui, ao utilizar esse conjunto, é possível ter o melhor dos dois mundos, mesmo que na maioria do casos, esse formato de implementação não se sobressaí as versões já utilizadas classicamente, com o tempo e o aperfeiçoamento das técnicas e do hardware darão uma abrangência maior aos usos da computação quântica.
Em resumo, é possível tirar proveito da computação quântica para problemas que conhecemos classicamente. No entanto, é necessário averiguar se há algum fator quântico que pode ser explorado para conseguir alguma vantagem perante a sua versão clássica, se houver, é necessário verificar também se todas as tarefas são mais vantajosas ao serem implementadas usando o algoritmo quântico, ou se ao explorar uma abordagem híbrida os ganhos podem ser maiores.

References

[1] Robert I. Soare. Turing oracle machines, online computing, and three displacements in computability theory. *Annals of Pure and Applied Logic*, 160(3):368–399, 2009. Computation and Logic in the Real World: CiE 2007.

[2] Sadika Amreen and Reazul Hoque. Oracle turing machines.

[3] Subrahmanyam Kalyanasyndaram. mod04lec23 - oracle turing machines, 09 2021.

[4] Niklas Johansson and Jan-Åke Larsson. Quantum simulation logic, oracles, and the quantum advantage. *Entropy*, 21(8), 2019.

[5] Yale Fan. A generalization of the deutsch-jozsa algorithm to multi-valued quantum logic. In *37th International Symposium on Multiple-Valued Logic (ISMVL'07)*. IEEE, May 2007.

[6] Ryan O'Donnell. Lecture 5: Quantum query complexity, 09 2015.

[7] Javier Sanchez-Rivero, Daniel Talaván, Jose Garcia-Alonso, Antonio Ruiz-Cortés, and Juan Manuel Murillo. Some initial guidelines for building reusable quantum oracles, 2023.

[8] Austin Gilliam, Marco Pistoia, and Constantin Gonciulea. Canonical construction of quantum oracles, 2020.

[9] Lídia André. Tower of hanoi – lídia andré, 03 2021.

[10] diptokarmakar47. How to solve the tower of hanoi problem - an illustrated algorithm guide, 01 2019.

[11] Towers of hanoi: A complete recursive visualization, 05 2020.

[12] GeeksforGeeks. Program for tower of hanoi, 05 2014.

[13] Samuel Jaques and Arthur G. Rattew. Qram: A survey and critique, 2023.

[14] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum random access memory. *Physical Review Letters*, 100(16), April 2008.

[15] Dave Bacon. Cse 599d -quantum computing simon's algorithm, 2006.

[16] Robin Kothari. An optimal quantum algorithm for the oracle identification problem. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014.

[17] Ryan O'Donnell. Lecture 13: Lower bounds using the adversary method, 10 2015.

[18] Laurel Brodkorb and Rachel Epstein. The entscheidungsproblem and alan turing, 12 2019.

[19] Martin Davis. Turing reducibility?, 11 2006.

[20] Mahesh Viswanathan. Reductions 1.1 introduction reductions, 2013.

[21] Takashi Yamakawa and Mark Zhandry. Classical vs quantum random oracles. Cryptology ePrint Archive, Paper 2020/1270, 2020. `https://eprint.iacr.org/2020/1270`.

[22] Harry Buhrman, Richard Cleve, and Avi Wigderson. Quantum vs. classical communication and computation, 1998.

[23] Elham Kashefi, Adrian Kent, Vlatko Vedral, and Konrad Banaszek. Comparison of quantum oracles. *Physical Review A*, 65(5), May 2002.

[24] William Zeng and Jamie Vicary. Abstract structure of unitary oracles for quantum algorithms. *Electronic Proceedings in Theoretical Computer Science*, 172:270–284, December 2014.

[25] Alp Atici. Comparative computational strength of quantum oracles, 2004.

[26] Kathiresan Sundarappan. How to build oracles for quantum algorithms, 04 2022.

[27] Zhifei Dai, Robin Choudhury, Jinming Gao, Andrei Iagaru, Alexander V Kabanov, Twan Lammers, and Richard J. Price. View of the role of quantum algorithms in the solution of important problems.

[28] Don Ross. Game Theory. In Edward N. Zalta and Uri Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Spring 2024 edition, 2024.

[29] Tomasz Zawadzki and Piotr Kotara. A python tool for symbolic analysis of quantum games in ewl protocol with ibm q integration. `https://github.com/tomekzaw/ewl`.

[30] Piotr Frackiewicz. Application of the ewl protocol to decision problems with imperfect recall, 2011.

[31] Jens Eisert, Martin Wilkens, and Maciej Lewenstein. Quantum games and quantum strategies. *Physical Review Letters*, 83(15):3077–3080, October 1999.

[32] Muhammad Usman. Kilometres to miles conversion — approximation of fibonacci series, 09 2019.

[33] Faisal Shah Khan and Ning Bao. Quantum prisoner's dilemma and high frequency trading on the quantum cloud. *Frontiers in Artificial Intelligence*, 4, 11 2021.

[34] Alexis R. Legón and Ernesto Medina. Dilemma breaking in quantum games by joint probabilities approach. *Scientific Reports*, 12, 08 2022.

[35] Brian Siegelwax. Quantum memory: Qram. what is it and why do we need it? making quantum algorithms thrive., 01 2022.

[36] Gabriel Landi. Density matrices and composite systems.

[37] V. Vijayakrishnan and S. Balakrishnan. Role of two-qubit entangling operators in the modified eisert–wilkens–lewenstein approach of quantization. *Quantum Information Processing*, 18, 03 2019.

[38] Real Python. Scientific python: Using scipy for optimization – real python.

[39] scipy optimize minimize scalar scipy $v1.12.0$ manual.

[40] Matt Davis. Optimization (scipy.optimize) — scipy v0.19.0 reference guide.

[41] scipy.optimize.minimize — scipy v1.6.0 reference guide.