QUANTUM ORACLES - COMO TRANSFORMAR PROBLEMAS CLÁSSICOS EM QUÂNTICOS

Alexandre Silva
Ciências da Computação
UNIVEM - Centro Universitário Eurípides de Marília

Luis Hilário Tobler Garcia
Ciências da Computação
UNIVEM - Centro Universitário Eurípides de Marília

Maúricio Duarte
Tecnologia da Informação
Fatec Garça – Deputado Julio Julinho Marcondes de Moura

26 de abril de 2024

ABSTRACT

A partir do uso de quantum Oracles e outros fatores quânticos, como a superposição, foram feitos *5 mini-projetos*. O objetivo desses projetos foi tentar responder se é possível transformar certos problemas em quânticos e se realmente tal transformação vale a pena. Após os testes foi possível ver que, há casos em que a versão quântica apresenta um aproveitamento igual ou um pouco superior, contudo ainda é necessário o uso de computadores clássicos para conseguir melhores resultados.

1 Introdução

Hoje, não é difícil ver alguém falando sobre computadores quânticos e como essas máquinas vão mudar o nosso futuro. Contudo, muitas dessas frases acabam se levando por extrapolações e/ou usos indevidos de ficção. Neste artigo, mostrarei que nem tudo é possível ser feito com um computador quântico atual, assim como existem pequenas áreas que se beneficiam ao máximo dessa nova tecnologia.
Para esse feito, serão mostrados *5 mini-projetos* usando o qiskit, framework open source da IBM, e os resultados obtidos após executar os algoritmos quânticos e seus relativos em computação clássica.
Tais mini-projetos foram os seguintes: Explorador de Arquivos 3.1, conversão de milhas para quilômetros 3.2, Torres de Hanoi 3.3, Buckshot Roulette 3.4 e QRAM 3.5. Todas as implementações podem ser encontradas nesse repositório do GitHub.
Para a criação desses projetos, foram usados os Quantum Oracles em conjunto com alguns algoritmos, explorando efeitos quânticos e algumas outras técnicas, clássicas e quânticas, para cada caso específico.

2 Oracles

Partindo da ideia das *Oracle Turing Machines* [1][2][3][4], os Oracles são modelos matemáticos ideais, usados para abstrair certas partes de um algoritmo principal, em formato de caixa preta, facilitando a análise do algoritmo, assim como sua descrição matemática. Tais máquinas podem ser vistas, também, como uma função, recebendo uma entrada x e retornando $f(x)$ em tempo $O(1)$. Em computação clássica, esse modelo não possui implementação real, sendo usado apenas descrições formais para problemas de decisão.
Contudo, em computação quântica, é possível implementar esses componentes e tomar proveito de sua estrutura e efeitos quânticos para conseguir um *Speed-up* em relação aos algoritmos clássicos, como mostrado pelo algoritmo de Deutsch–Jozsa [5]. Além disso, os Oracles possuem um papel importante ao demonstrar a complexidade de um circuito, alguns dos meios utilizados são: profundidade (*depth*), calculando o maior caminho que uma informação percorre no circuito, ou ainda, a quantidade de gates aplicados. No entanto, essas maneiras acabam se prejudicando ao *transpilar* o circuito para uma outra máquina, variando então a complexidade de acordo com a topologia e com os gates fisicamente

implementados. Para solucionar isso, outra maneira de calcular é inserir partes do circuito em um Oracle, e descrever sua complexidade a partir da quantidade de vezes que ele chamado, também conhecido como *query complexity* [6] [4].

2.1 Tipos de Oracles

A partir da definição dos Quantum Oracles, podemos classificá-los em relação a suas estruturas e maneiras de computar os dados.

2.1.1 Phase Oracle

O Phase Oracle, é o formato mais conhecido e usado em circuitos quânticos. Algoritmos como os de Deutsch–Jozsa, Grover, Simon e Bernstein–Vazirani, tomam proveito desse artifício para se sobressair em relação às soluções clássicas.

2.1.1.1 Funcionamento Padrão

Seu funcionamento, se baseia em atribuir uma fase global ao circuito, tomando proveito de fatores como *Phase Kickback* (fase passa do target do CNOT e é aplicado no qubit de controle), para conseguir modificar valores em superposição.

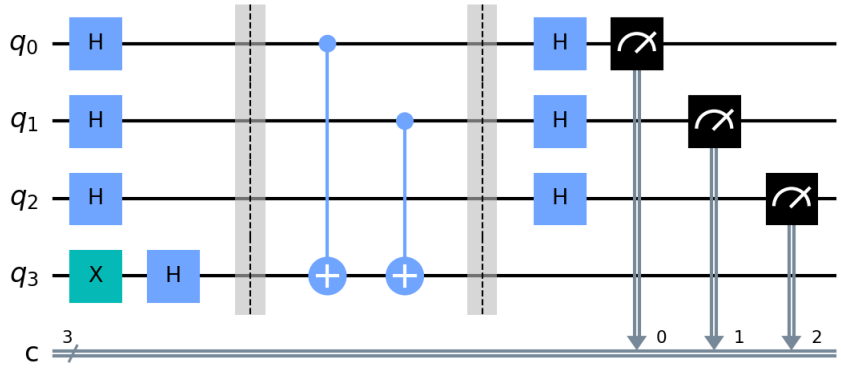

Figura 1: Exemplo - Phase Oracle

Na Imagem 1, foi introduzida uma fase π no qubit auxiliar ($q3$) através do estado $|-\rangle$. Essa fase será responsável por modificar os valores na matriz unitária final. Nessa configuração, os CNOTs agem de uma forma um tanto diferente do convencional, aqui, ao invés de apenas inverter o valor do qubit no target quando o qubit de controle for 1, devido a fase, ele também agirá como um gate Z sendo aplicado no estado do qubit de controle. Sendo assim, ao aplicar $CNOT\,|-\rangle\,|+\rangle$ (qubit menos significativo à direita), o estado se torna $\frac{1}{\sqrt{2}}(|0\rangle\,|-\rangle - |1\rangle\,|-\rangle)$, e ao remover a superposição com o H, a saída se torna: $\frac{1}{\sqrt{2}}(|+\rangle\,|1\rangle - |-\rangle\,|1\rangle)$. Dessa forma, o qubit que antes estava como controle do gate, sofre a ação do *Phase Kickback*, e seu estado padrão $|0\rangle$ é modificado pela fase e se torna $|1\rangle$. A partir disso, é possível encodar um certo valor binário dentro do Oracle e utilizá-lo para cálculos.

2.1.1.2 Versão Minimal Oracle

Além disso, esse não é o único formato possível de Phase Oracle. Por apenas aplicar uma fase em certas bit-strings, o qubit auxiliar pode ser removido, e a fase pode ser adicionada através de gates Zcontrolados (ou outro gate capaz de aplicar uma fase π para certa bit-string), mas ainda assim mantendo a natureza unitária, podendo ser visto também como um Minimal Oracle.

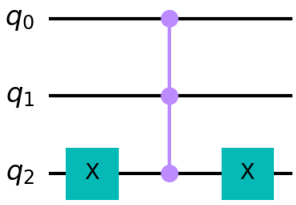

Figura 2: Exemplo Phase Oracle como um Minimal Oracle

No exemplo da imagem 2, foi adicionado um gate MCP com a fase global π e dois gates X para inverter os qubits queremos que tenham o valor 0, codificando assim o valor 011_2 ou 3_{10}.

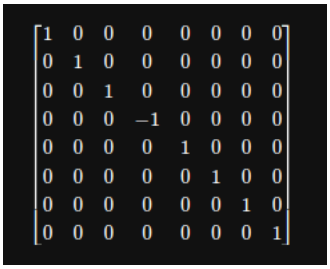

Figura 3: Matriz unitária do Phase Oracle da imagem 2

Assim, é possível verificar que ao criar esse circuito, é mantida a matriz identidade mas com a fase (-1) no valor 1 na coluna relativa à 011_2 (figura 3).

2.1.2 Boolean Oracle

O Boolean Oracle, por sua vez, apresenta um funcionamento semelhante ao do Phase Oracle. Contudo, neste não é provida uma fase. Dessa forma, o Oracle age como uma função Booleana convencional, mapeando as entradas para valores de saída.

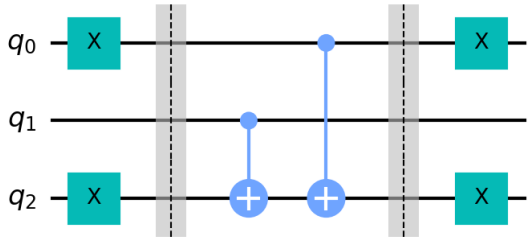

Figura 4: Exemplo de Oracle Booleano

O Oracle implementado na figura 4, pode ser reutilizado para o algoritmo de Deutsch-Jozsa, bastando apenas introduzir uma fase, e o Boolean Oracle se comportará como um Phase Oracle.

2.1.3 Minimal Oracle

Como já citado anteriormente, o Minimal Oracle possui uma função que, em sua essência, é unitária, não requerendo qubits adicionais. Sendo assim, este pode ser tanto Booleano como um Phase Oracle, dependendo de sua implementação.

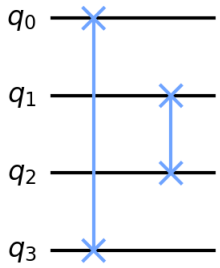

Figura 5: Exemplo de Minimal Oracle

No exemplo da figura 5, foram utilizados dois $SWAP$ para inverter a ordem dos valores. Com isso, a matriz final ainda se mantém unitária, com apenas valores invertidos em certas posições.

2.1.4 QFT(Quantum Fourier Transform)

O QFT, em suma, é um algoritmo quântico usado para projetar os valores da base computacional para a base X (ou também conhecido como base de Fourier). Esse algoritmo, toma como base a transformada discreta de Fourier e aplica essa transformação em estados quânticos.
Mesmo sendo um algoritmo por si só, sua aplicação em circuitos se dá seguindo o formato de Oracles.

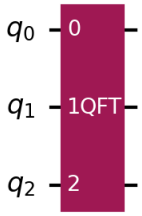

Figura 6: Exemplo do Oracle de QFT

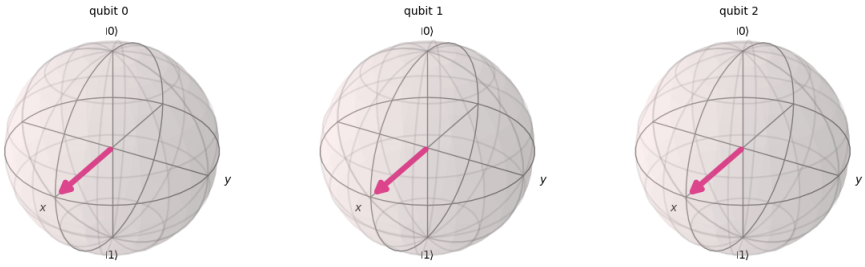

Figura 7: Valores mapeados na base de Fourier

2.1.5 Outros Oracles

Além dos Oracles citados, é possível encontrar na literatura citações descrevendo o Oracle de Simon, o de Deutsch-Jozsa, etc. No entanto, esses são implementações de Oracles já mostrados e além disso, para o desenvolvimento deste projeto, os Oracles mais relevantes são o Phase e Boolean. Portanto, não há a necessidade de profundas investigações dessas subcategorias de Oracles.

3 Desenvolvimento

3.1 Explorador de Arquivos

Imagine um computador quântico, com um sistema operacional quântico, capaz de interagir não apenas com a parte quântica, mas também com uma porção clássica a qualquer momento que for necessário. Essa máquina, possui todas as capacidades de um computador pessoal mais as capacidades de um computador quântico atual.
Pensando nas partes desse sistema, como seria possível pegar arquivos da memória?
Com essa ideia, foram testados alguns modelos de implementação de um circuito para essa finalidade.

3.1.1 Algoritmos usados

3.1.1.1 Grover

O algoritmo evidente para esse problema é o algoritmo de Grover. Este realiza buscas em "bancos de dados"(bit strings) desorganizados em tempo $O(\sqrt{2^n})$, do qual n é o número de qubits usados. Nele, usamos um circuito do qual amplificam-se as probabilidades de encontrar os valores marcados no Oracle.

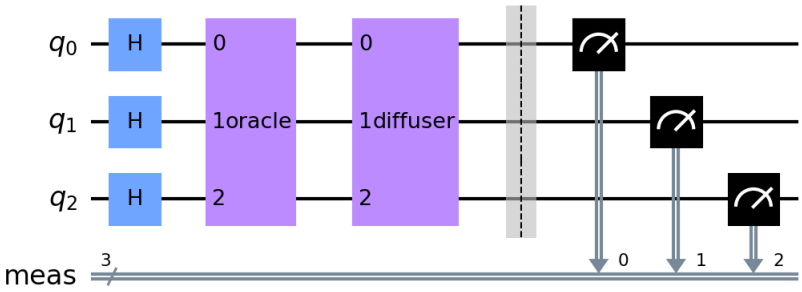

Figura 8: Exemplo algoritmo de Grover com 3 qubits

Para esse circuito, é preciso adicionar o conjunto $Oracle + Diffuser$ k vezes, sendo $k \approx \frac{\pi}{4\sqrt{\frac{a}{2^n}}} - \frac{1}{2}$, do qual a representa o número de valores marcados pelo Oracle. Como nesse projeto, visamos encontrar apenas 1 arquivo encodado, não há necessidade de usar tal relação, sendo necessário apenas uma aplicação do conjunto para alcançar bons resultados.
Contudo, mesmo sendo o melhor algoritmo, conhecido, para buscas em computação quântica, foi testado também hipóteses para possíveis maneiras de melhor os valores das distribuições finais. Para isso, foram testadas inúmeras combinações de rotações RY e foram comparadas com o convencional usado pelo algoritmo (H).

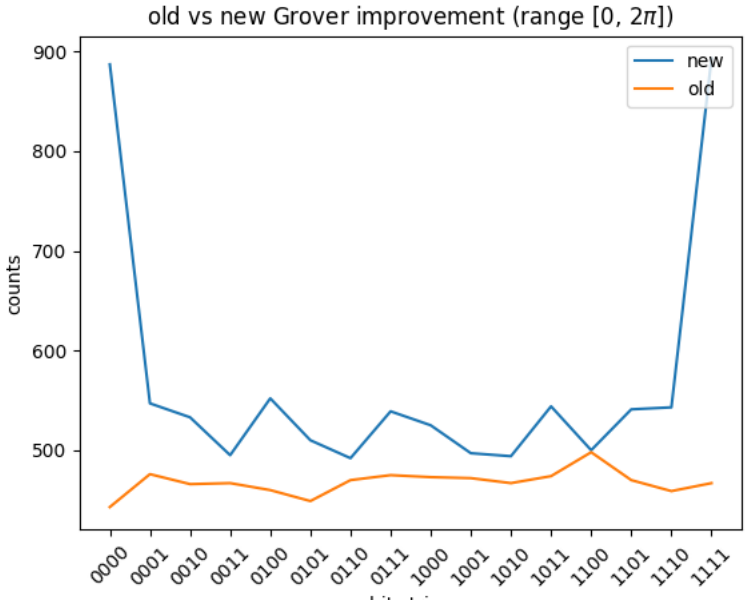

Figura 9: Comparação usando o algoritmo de Grover convencional e o algoritmo modificado com o melhor ângulo (entre $[0, 2\pi]$) para cada bit-string de quatro bits

Ao utilizar as rotações específicas para cada bit-string, é possível conseguir melhores resultados ao medir os valores na saída, se sobressaindo em relação a rotação padrão.

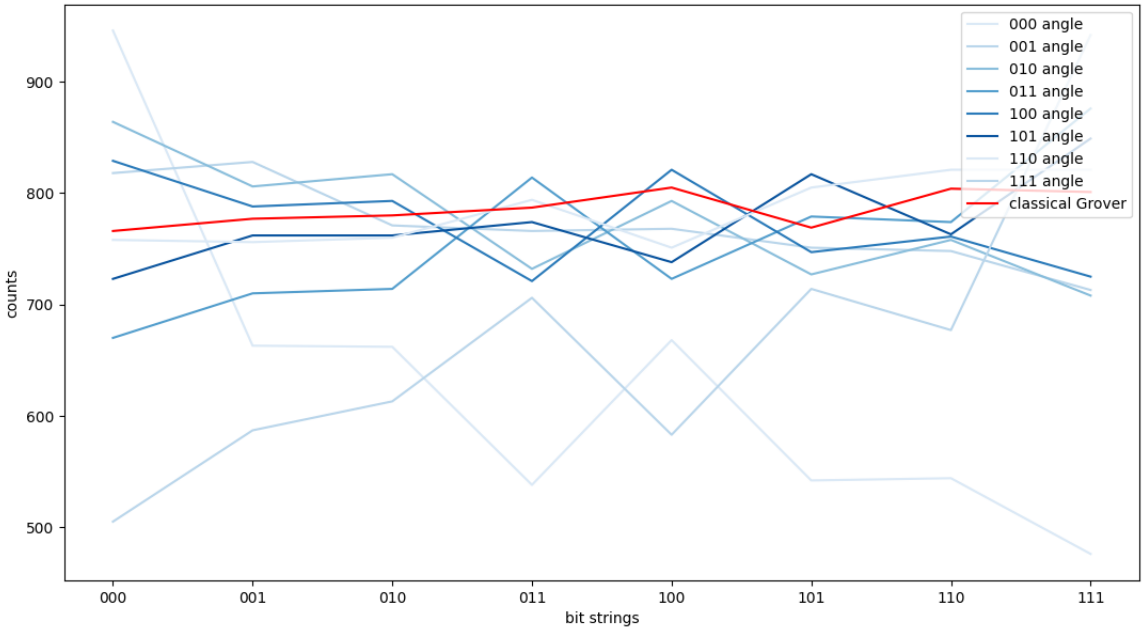

Figura 10: Teste utilizando os melhores ângulos de cada bit-string em bit-strings diferentes

No entanto, ao utilizar esses valores com outras bit-strings, os resultados não conseguem alcançar tal limiar, além modificar as outras probabilidades de forma irregular. Sendo assim, a rotação convencional é a melhor na maioria das vezes.
Além disso, para bit-strings de dois bits, utilizar a superposição dada por H se mostra a melhor alternativa.

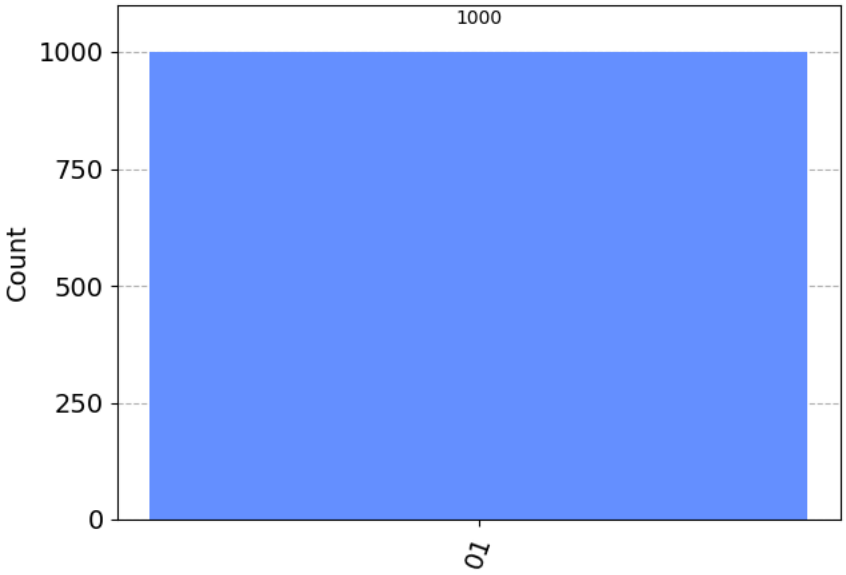

Figura 11: Resultado Grover padrão encodado uma bit-string de 2 bits

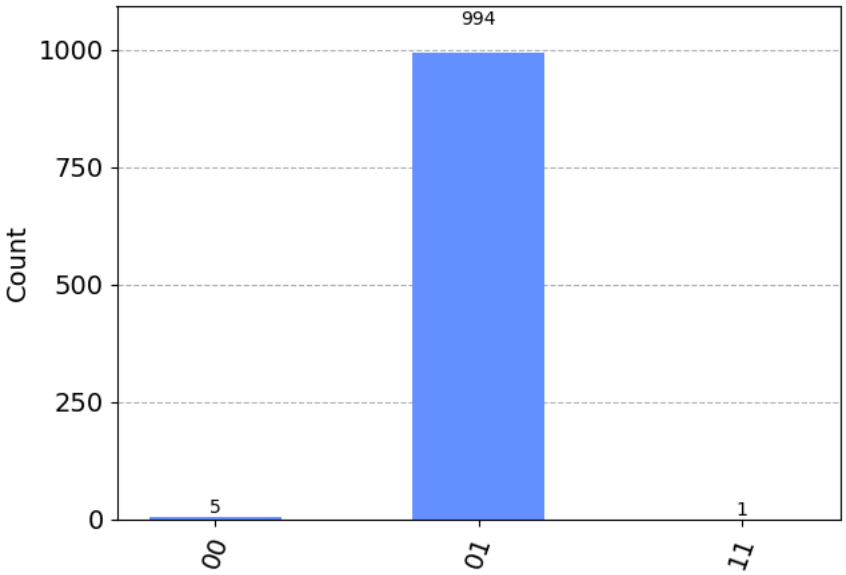

Figura 12: Resultado Grover modificado encodado uma bit-string de 2 bits

Com isso, para ter o melhor dos dois mundos, foi usado uma versão híbrida do algoritmo. Assim, para criar o circuito, é passado o valor a ser encodado por uma Hash-Table com os ângulos otimizados. Dessa forma, é possível maximizar as probabilidades de encontrar, nesse caso, o arquivo que está sendo procurado.

3.1.1.2 Diferença de conjuntos

Sobrepondo dois Phase Oracles distintos, com ranges de valores diferentes, é realizada a operação de diferença entre conjuntos [22].

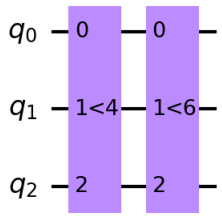

Figura 13: Exemplo - diferença de conjuntos

Nesse exemplo 13 foi encodado no primeiro Oracle o set $\{000, 001, 010, 0110\}$ e no segundo $\{000, 001, 010, 011, 100, 101\}$.

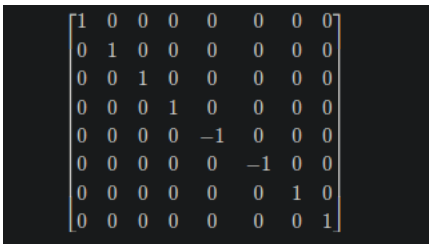

Figura 14: Resultado da diferença de conjuntos

Ao sobrepô-los 14, apenas os valores $\{100, 101\}$ permaneceram com a fase, representando então a sobreposição delas.

3.1.2 Solução

Para a solução do problema, foi criado uma hash function $C : v \to c$, da qual v é o path de um arquivo e c sua bit-string respectiva. Com essa função em mãos, podemos utilizar o conjunto dos valores retornados e encodá-los em um Phase Oracle, criando então uma Look-Up-Table para os arquivos existentes na máquina (agindo como a memória).
Além disso, é necessário utilizar um segundo Oracle para a pesquisa, encodando todos os valores existentes, menos os que foram requisitados. Assim, ao realizar a diferença entre conjuntos, apenas os valores procurados se manterão marcados.

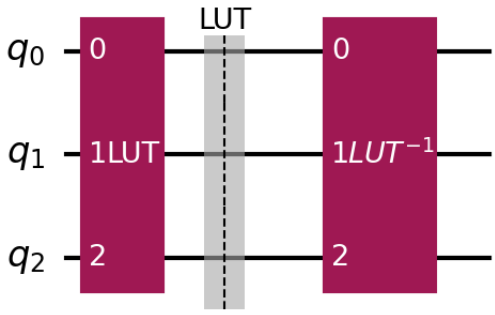

Figura 15: Diferença de conjuntos com as Look-Up-Tables

Por fim, é usado o aprimoramento dos ângulos para conseguir melhores probabilidades.

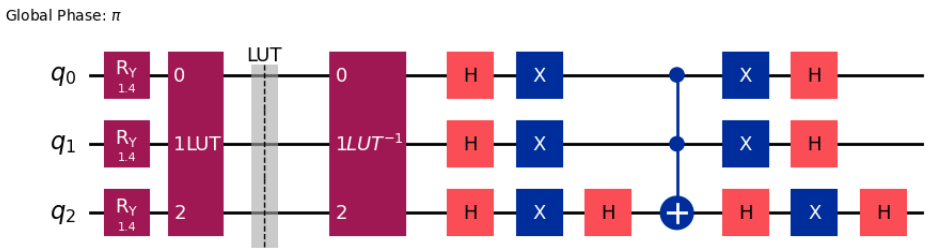

Figura 16: Explorador de arquivos implementação

Dessa forma, o arquivo procurado tem sua probabilidade maximizada pelo circuito, sendo apresentada a distribuição após n medições.

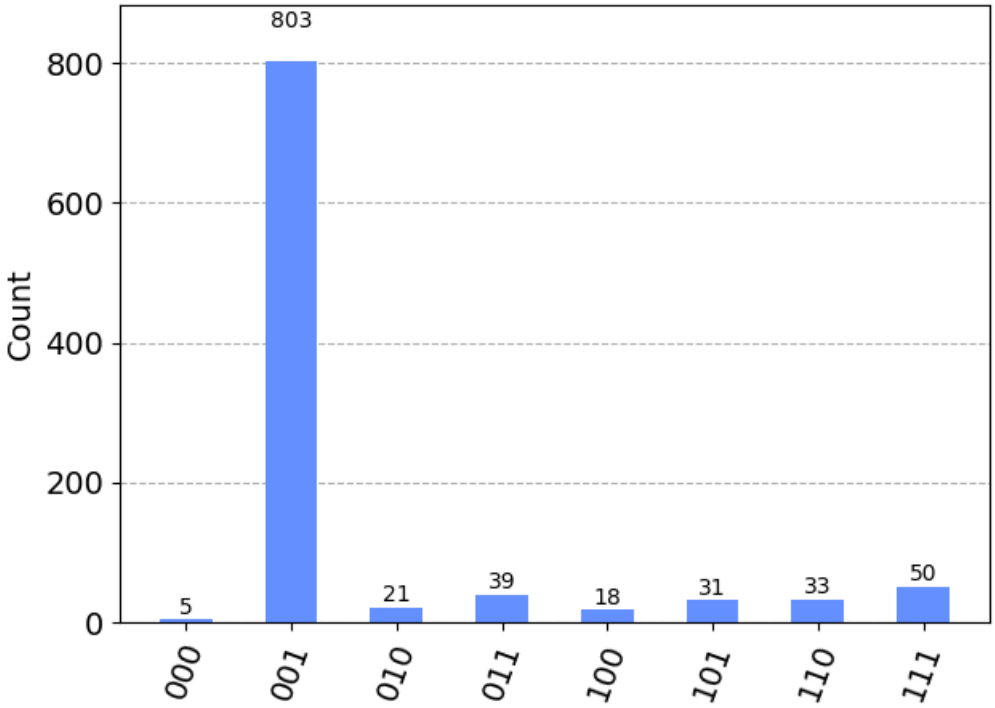

Figura 17: Explorador de arquivos resultados - Qiskit AER ($shots = 1000$)

3.1.3 Resultados

Para esse caso hipotético, certamente essa é um das melhores maneiras para fazer buscas dentre todos os arquivos armazenados.
Contudo, ao projetar esse modelo para um sistema clássico, tentando tomar proveito da computação quântica, essa não se mostra como a melhor opção. Isso acontece pois, guardar uma Look-Up-Table para os arquivos, e outra para cada ângulo de cada bit-string dentre as 2^n combinações, pode ser custoso e lento, além de requerer uma hash function com pouca probabilidade de colisão. Para diminuir esse overhead, poderia ser utilizado, simplesmente, o algoritmo de Grover sem maiores alterações, mas ainda assim seria necessário ter mapeado todos os arquivos em disco para a tabela. Assim, tomando como referência sistemas que utilizam o mapeamento de arquivos baseado em árvores ($O(log(n))$), esse método não apresenta ganho algum, além de possuir a probabilidade de não encontrar, ou retornar o arquivo errado.

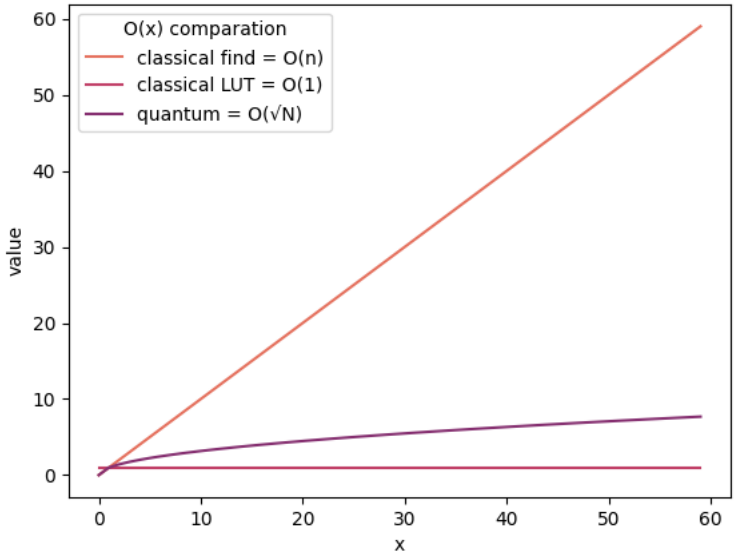

Figura 18: Comparação algoritmos usados na pesquisa

Sendo assim, os algoritmos apresentados, são as melhores alternativas para serem utilizadas em um sistema que é, principalmente, quântico. Mas para casos de otimização clássica, deve ser utilizado apenas para complexidades $\geq O(n)$.

3.2 Milhas para Quilômetros

O segundo problema testado, foi a conversão de milhas para quilômetros. Essa ideia se deu após a descoberta de um algoritmo capaz de calcular a sequência de Fibonacci usando circuitos quânticos, algoritmo essencial para esse projeto.

3.2.1 Algoritmos usados

3.2.1.1 Algoritmo Quântico de Fibonacci

A versão quântica usada para calcular Fibonacci foi apresentada em [7] e demonstra que, utilizando um circuito do qual coloca em superposição todas as bit-strings com n qubits, e então realizando operações para remover valores que possuem 1s consecutivos, é possível encontrar o valor n na sequência.

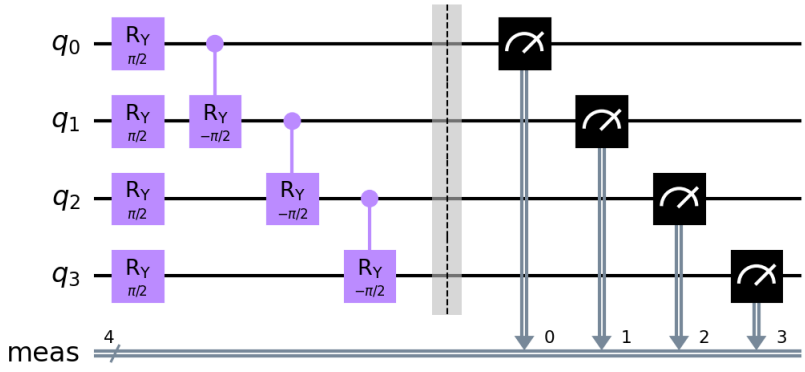

Figura 19: Exemplo Algoritmo Quântico de Fibonacci

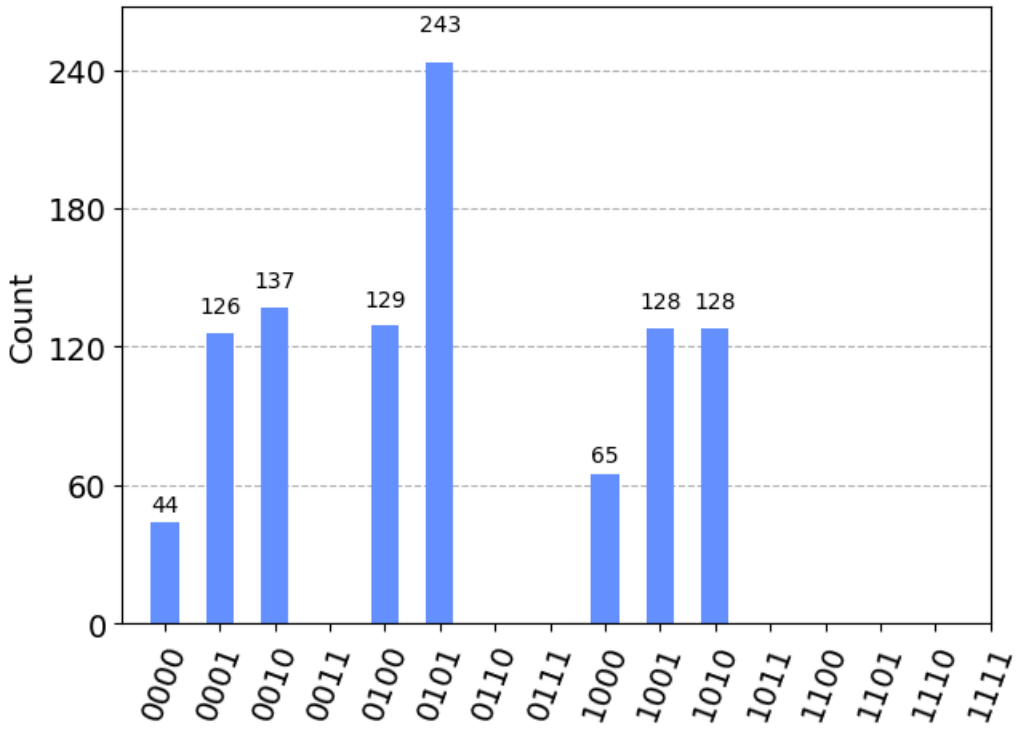

Figura 20: Resultado Fibonacci - F(4)

Após executar o circuito, é necessário verificar a quantidade de bit-strings únicas que apareceram durante os experimentos. No exemplo em 20, foram usados 4 qubits para calcular $F(4)$. Assim, ao contar as bit-strings, temos $F(4) = 8$, retornando então o quarto valor da sequência (nesse caso, a sequência começa do valor 2, seguindo dessa forma: $F(1) = 2, F(2) = 3, F(3) = 5, F(4) = 8, F(5) = 13, F(6) = 21, ...$).
Com isso, é possível usar esse circuito para computações de $F(n)$ utilizando n qubits para encontrar o valor requisitado nessa mesma posição n.

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

Tabela 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.

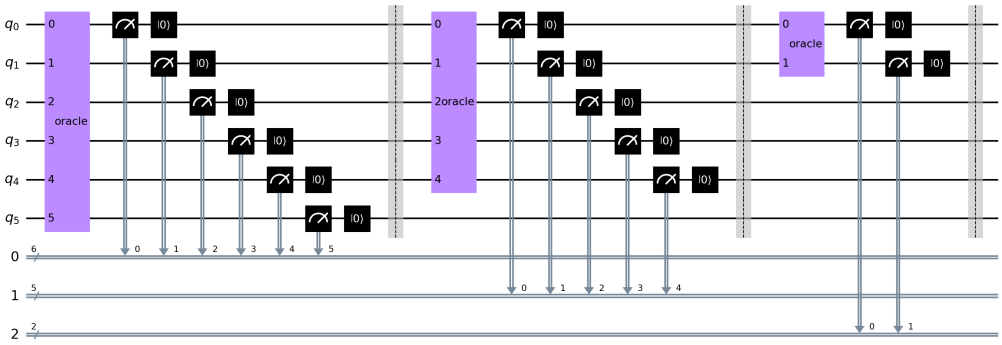

Figura 21: Circuito de conversão

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.

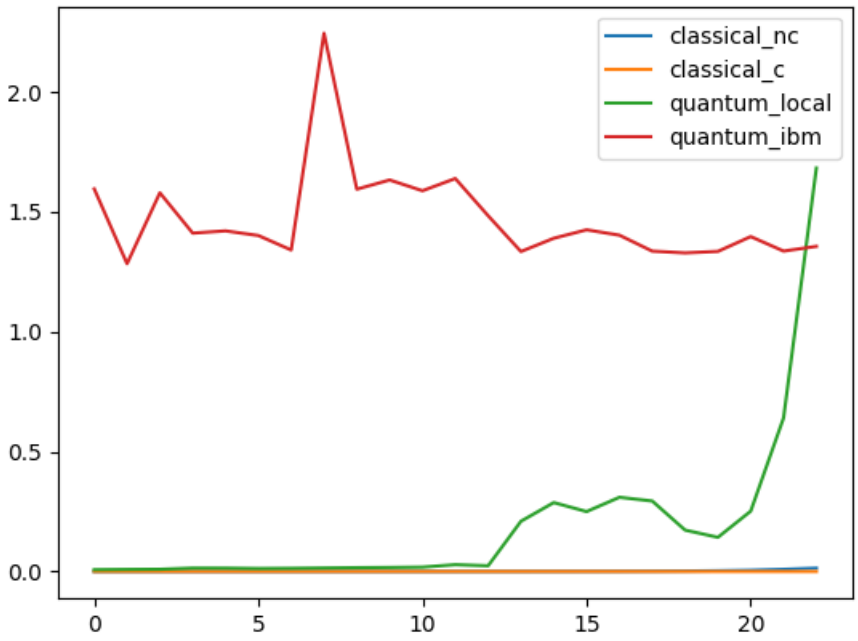

Figura 22: Comparação tempos de execução

Como mostrado em 22, 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**

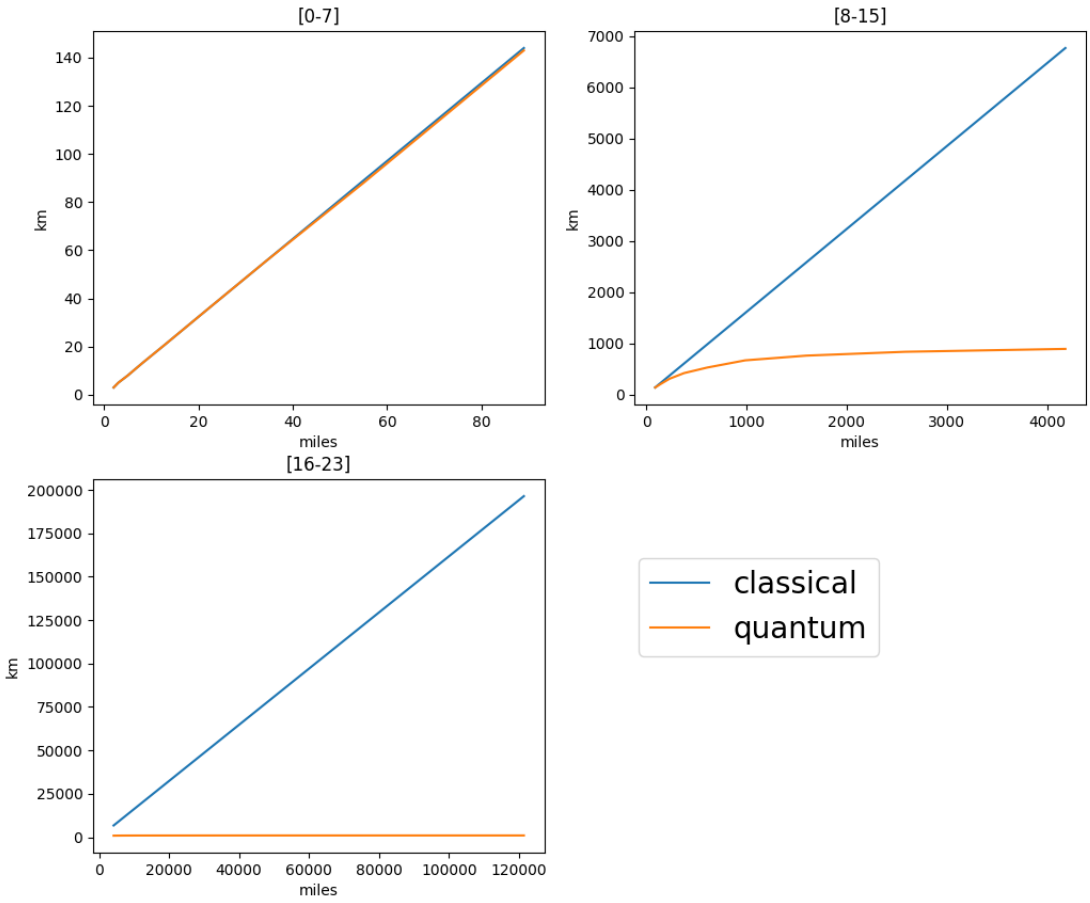

Figura 23: Comparação resultados versão clássica e quântica

Como mostrado em 23, 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 [8] [9] [10] [11]. Dessa forma, essa versão quântica age como um jogador com uma lista de passos a serem seguidos, executando-os um-a-um.

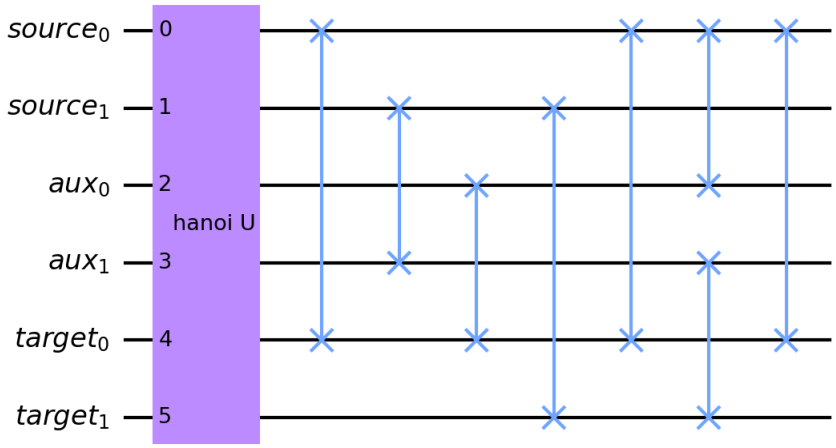

Figura 24: Torre de Hanoi com 3 discos

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

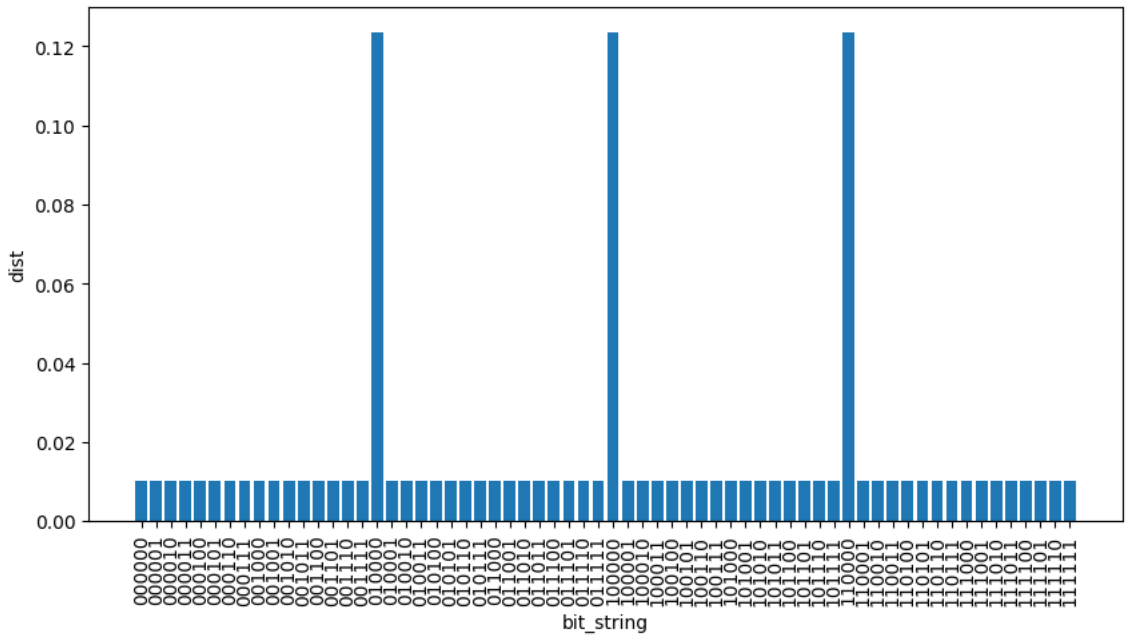

Figura 25: Resultado usando Grover - Torre de Hanoi com 3 discos

Em 25, 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.

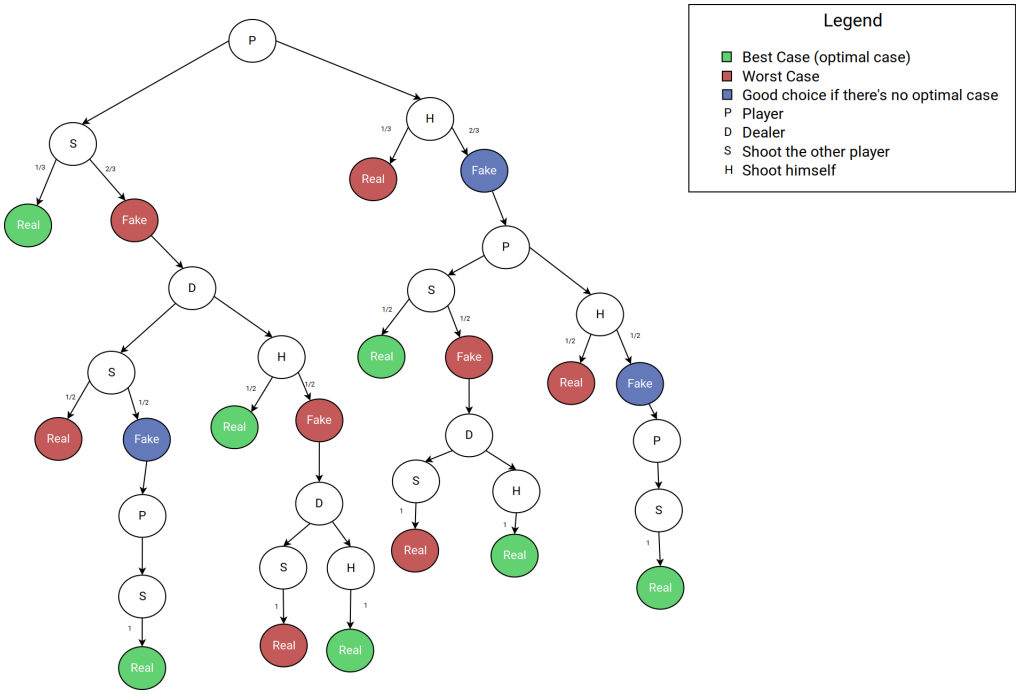

Figura 26: Buckshot Roulette diagrama de árvore

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.

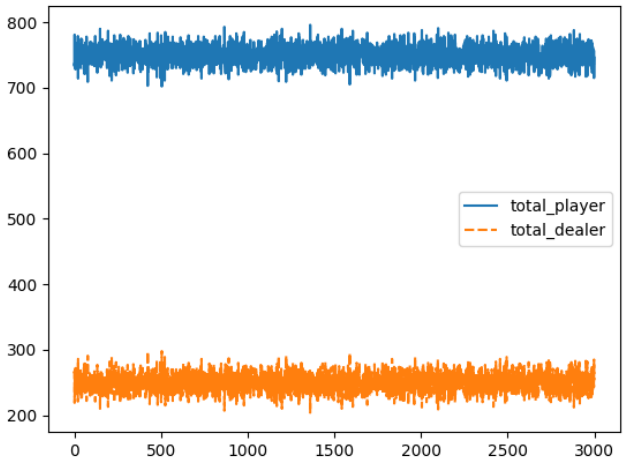

Figura 27: Buckshot Roulette clássico - melhor estratégia

Após testar os caminhos possíveis, o melhor resultado obtido foi esse apresentado acima em 27. 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

Tabela 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.

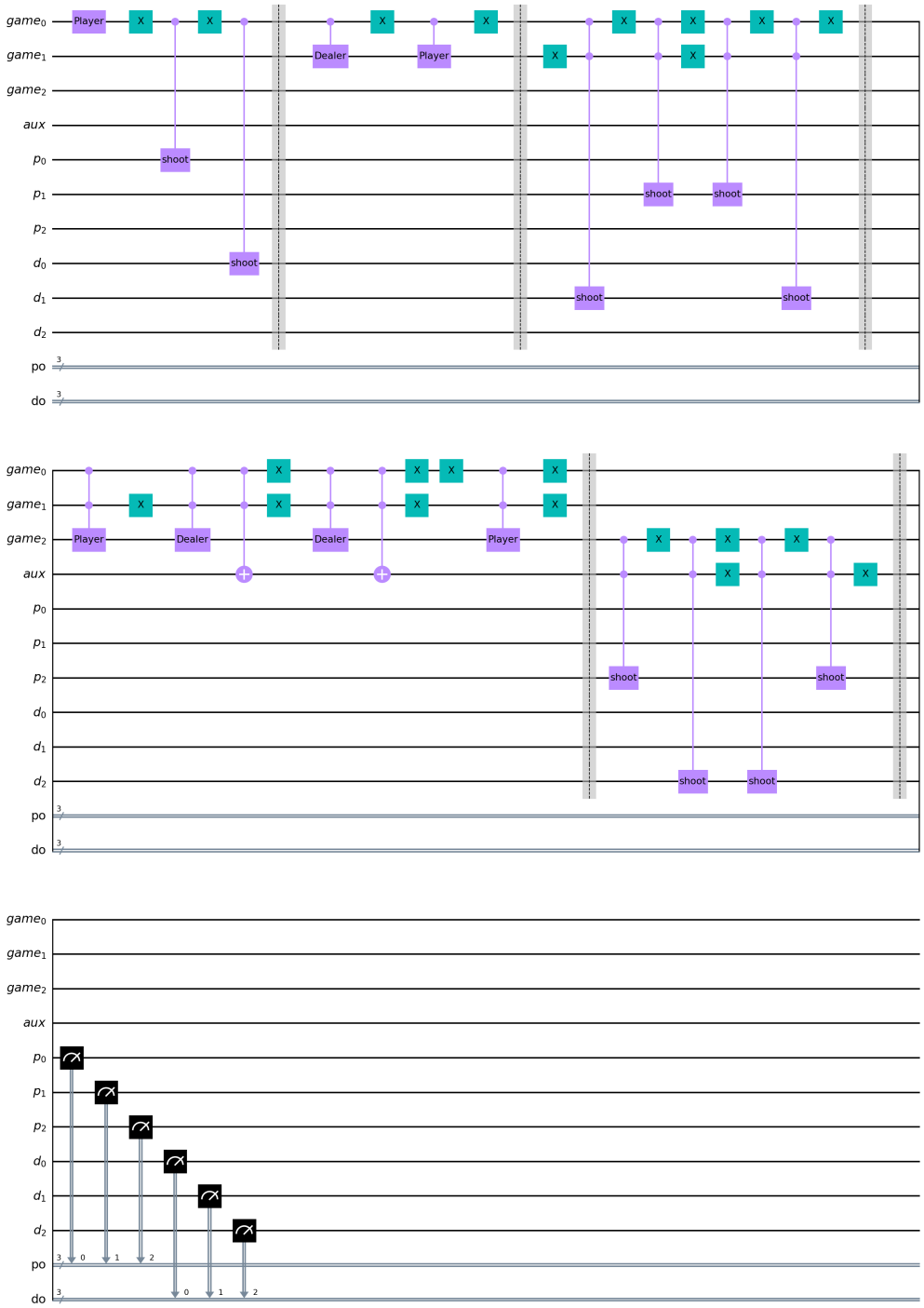

Figura 28: Circuito para o Buckshot Roulette

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 $\theta \approx 3.0853981633974477, \phi \approx 3.7853981633974474$. Usando essa estratégia, os resultados foram semelhantes a versão clássica.

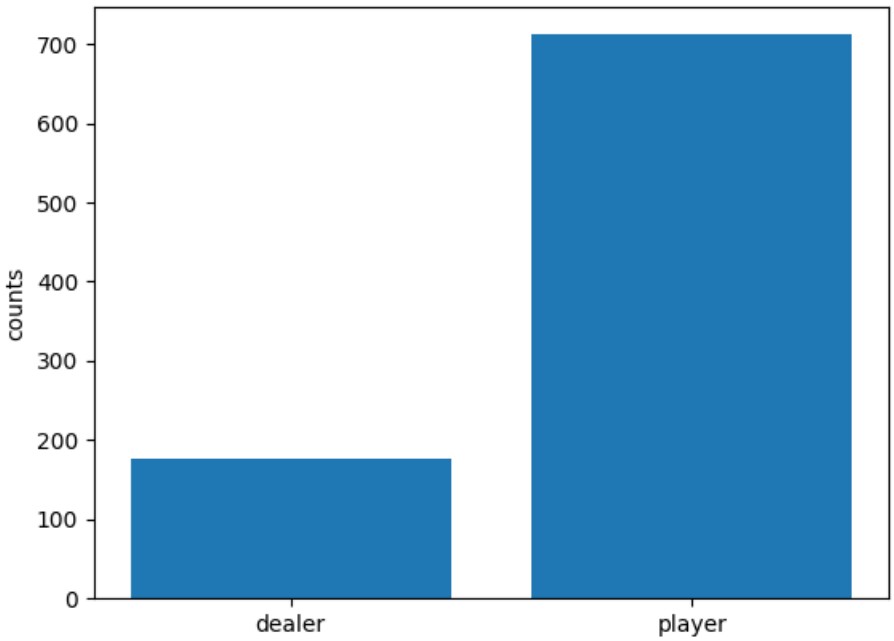

Figura 29: Resultado Buckshot Roulette quântico - Qiskit AER

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).

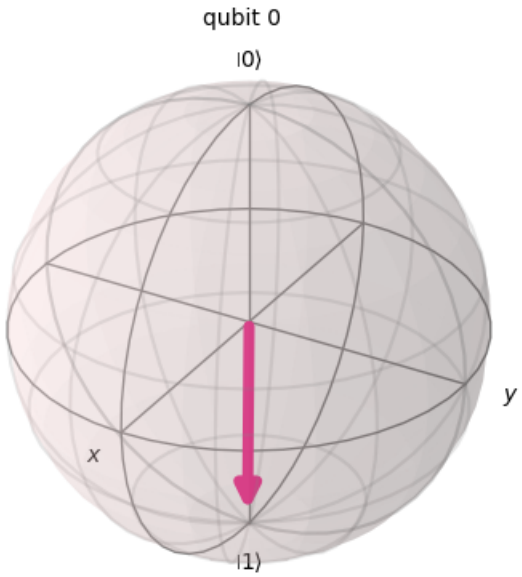

Figura 30: Melhor estratégia Buckshot Roulette quântico - Bloch Sphere

Como uma última nota sobre o circuito, no exemplo 29, 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 $n = 3, m = 10$. 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.

A entrada do circuito segue o seguinte formato: $|0\rangle^{\otimes m}|a_{n-1}a_{n-2}...a_0\rangle$

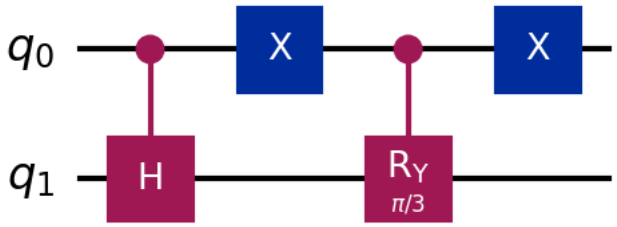

Figura 31: Exemplo circuito - QROM

Em 31, 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 para isso.

A partir desse circuito, basta abstrai-lo para um Oracle e utilizar em um circuito maior, chamando-o novamente sempre que for necessário um certo estado ou ainda colocar os endereços em superposição e ter uma mistura de estados na saída.

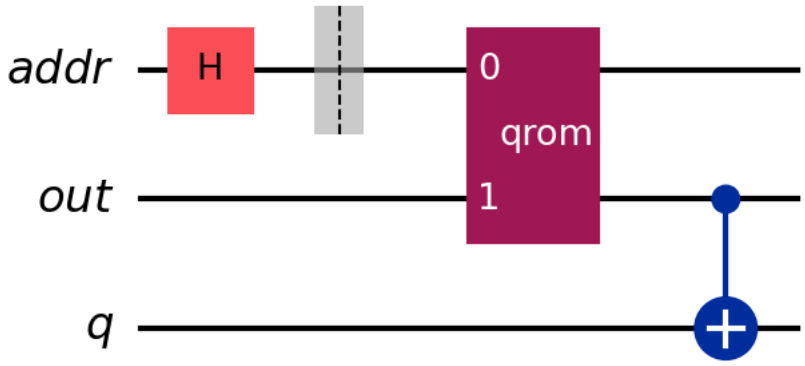

Figura 32: Exemplo circuito usando a QROM

Nesse exemplo 32, os estados em superposição serão colocados no qubit denominado *out*, sendo possível se aproveitar dela 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, então, dessa forma, não é possível ter dois qubits com o mesmo estado a partir do armazenado. Uma opção para isso, é utilizar do teleporte quântico para destruir o estado interno do Oracle e mover para outros qubits desejados, contudo isso influenciária nos estados já aplicados no circuito.

3.5.2 QRAM

Para criar uma QRAM com a habilidade de escrita, o teleporte quântico já citado antes 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.

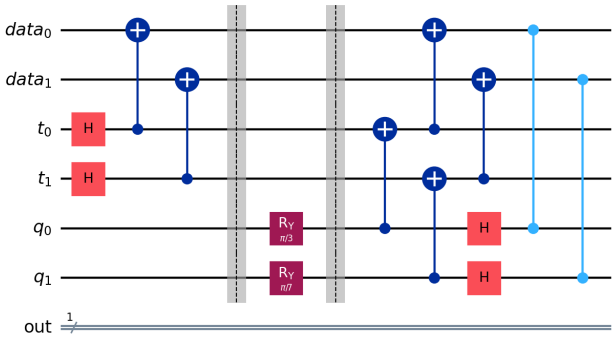

Figura 33: Exemplo circuito - QRAM

Aqui, são necessários n qubits para n endereços (*data*) também agindo como qubits de data, e n qubits para o teleporte (t), crescendo então de forma linear à capacidade de armazenamento de dados deste. Também 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 [12][13], é possível entender que criar versões quânticas de memória é uma tarefa desafiadora, e ainda não é possível tomar proveito do 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.

Assim como mostrado na literatura, o melhor approach para a sua implementação, é a utilização de um hardware especifico para isso, sem a intervenção de circuitos quânticos, mantendo depth do algoritmo implementado constante ajudando assim a manter a coerência do circuito.

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 no futuro 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. Assim 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, com o tempo, e o aperfeiçoamento das técnicas e do hardware em si darão um novo caminho para novos usos.

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 que em sua forma clássica seria impossível ou não usuais.

Referências

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[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.