QUANTUM ORACLES - COMO TRANSFORMAR PROBLEMAS CLÁSSICOS EM QUÂNTICOS

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

ABSTRACT

1 Introdução

Hoje, não é difícil ver alguém falando sobre computação quântica e como essas máquinas vão mudar o 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 mostrado alguns testes feitos usando o qiskit, um framework open source da IBM para computação quântica, além de alguns resultados obtidos após executar os algoritmos em simuladores e máquinas reais, assim como seus relativos em computação clássica. Algoritmos dos quais tomam proveito dos quantum oracles, modelos ideias de função que não ajudam a descrever o algoritmo matematicamente, também tomam proveito de alguns efeitos quânticos, como superposição e interferência, para se sobressair à algumas estratégias clássicas.
Com isso, o projeto foi desenvolvido em cima de cinco pequenos problemas, sendo eles: conversão de milhas para quilômetros, torre de Hanoi, explorador de arquivos, Buckshot Roulette e QRAM. Todas as implementações e materiais utilizados podem ser encontrados nesse repositório do GitHub.

2 Início do projeto

Para dar início a pesquisa, foi necessário entender quais os tipos de oracles existem e como eles podem ser usados.
Em computação clássica, temos as Oracle Machines, as quais são maquinas de Turing, das quais implementam alguma função em seu interior, e ao ser chamado/invocado o resultado correto é retornado em tempo constante $O(1)$, podendo ser vista como uma caixa preta, abstraindo completamente o seu funcionamento. Devido a essa definição, as OMs são ideias matemáticos, sendo assim usados apenas para formalismo matemático.
Contudo em computação quântica, podemos de fato implementar certos modelos de Oracles e adiciona-los a um circuito maior, executando certas funções como: encoding de dados, aplicação de $f(x)$, abstração de partes do circuito, etc.

2.1 Tipos de Oracles

2.1.1 Phase Oracle

Um dos primeiros tipos de Oracles usados para a criação de algoritmos como os de: Grover e Deutsch–Jozsa; é comummente conhecido como *Phase Oracle*.
Tal dispositivo, é usado para atribuir uma fase ao circuito, sendo muito usado para configurar valores, explorar a interferência ou se aproveitar de outros efeitos como o *Phase Kickback*. Matematicamente poderiamos descrever ele da seguinte forma: $|x\rangle |-\rangle \rightarrow (-1)^{f(x)} |x\rangle |-\rangle$, do qual $|x\rangle$ é a entrada do oracle e $|-\rangle$ é a ancilla que prove a fase.

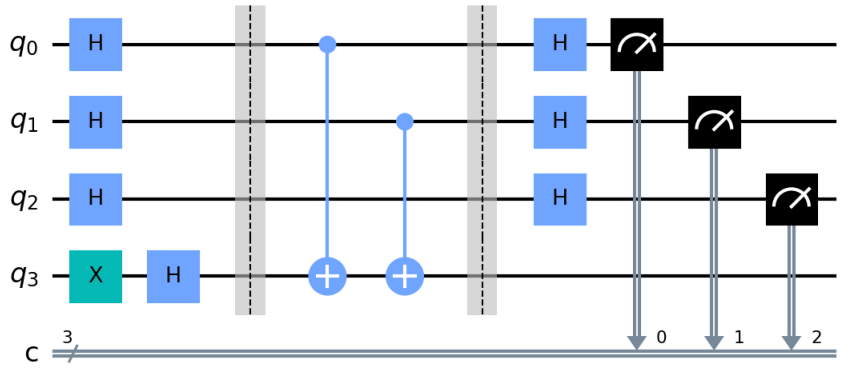


Figura 1: Exemplo de phase oracle usado para o algoritmo de Deutsch–Jozsa

No exemplo acima, utilizamos o *Phase Kickback* para adicionar uma fase nos qubits 0 e 1, transformando seus estados de $|+\rangle$ para $|-\rangle$, fazendo com que ao serem colapsados o resultado $|1\rangle$ apareça na saída.

É possível também criar um phase oracle removendo o qubit adicional (nesse exemplo o Q3), uma vez que podemos utilizar outros gates para introduzir a fase e manter ainda natureza unitária.

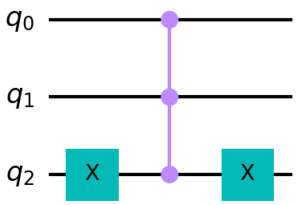
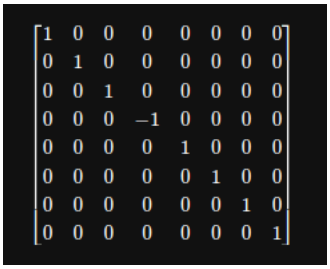


Figura 2: Exemplo fase Oracle sem a Ancilla

Dessa vez, utilizamos o MCP gate para adicionar uma fase global π e dois gates X para dizer quais qubits queremos q tenham o valor 0, codificando assim o valor 011 ou 3 na base decimal.



$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Figura 3: Matriz unitária do Phase oracle

É possível verificar então que ao criarmos esse circuito, matemos a matriz identidade e adicionamos a fase -1 no valor da coluna relativa ao 011.
Essa versão pode ser considerada como um minimal oracle, uma vez que a própria função interna se mantém unitária, sem a necessidade de ancilla.

2.2 Boolean Oracle

O Boolean oracle, por sua vez, representa uma função booleana, sem qualquer adição de fases.
Nesse caso, $|x\rangle$ representa a entrada do oracle e $|y\rangle$ represetam os qubits auxiliares que receberam a resposta, $|x\rangle\,|y\rangle \rightarrow |x\rangle\,|y \oplus f(x)\rangle$.

2.3 Minimal oracle

Como já citado anteriormente, o minimal oracle possui uma função que em sua essência é unitária, não requerendo qubits adicionais $|x\rangle \rightarrow |f(x)\rangle$.

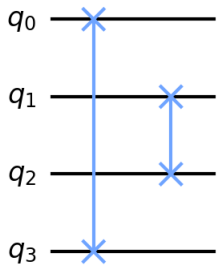


Figura 4: Exemplo de minimal oracle

Lembrando que este pode também adicionar fases ao circuito.

2.4 Simon's Oracle

O Oracle de Simon, é uma instância do Boolean Oracle. Neste visamos encontrar os períodos da função implementada, ou seja, dado x e $f(x) = y$ existe ao menos uma função em que $f(y) = x$?

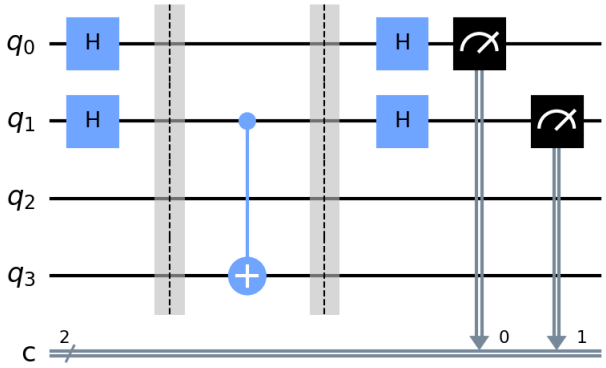


Figura 5: Exemplo do algoritmo de Simon

Nesse algoritmo, configuramos uma chave s dentro do oracle, e ao executar o algoritmo temos os possíveis períodos da função, sendo necessário rotinas de pós processamento para identificar o valor correto.

2.5 QFT(Quantum Fourier Transformation) Oracle

Por fim, o Oracle QFT aplica a versão quântica da transformada de Fourier, projetando os valores de entrada na base X (também conhecido como base de Fourier).

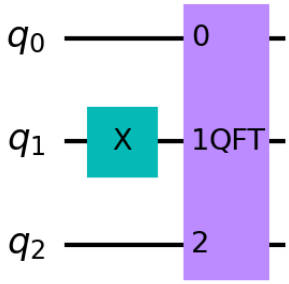


Figura 6: Exemplo do algoritmo de QFT

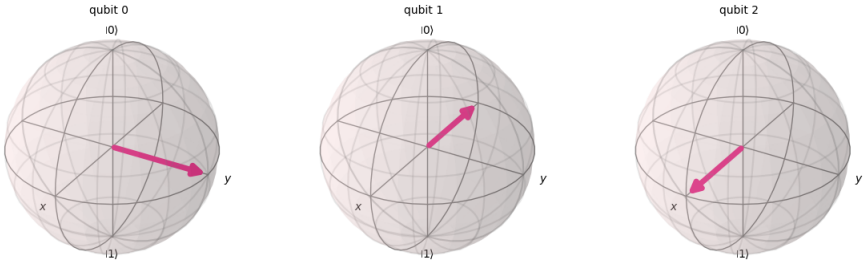


Figura 7: Valores mapeados na base de Fourier

3 Desenvolvimento

3.1 File Explorer

Imagine um computador quântico com um sistema operacional quântico (semelhante aos computadores convencionais, mas dessa vez seguindo as leis da mecânica quântica). Pensando nas partes desse sistema operacional, como seria possível pegar arquivos da memória usando a computação quântica?

3.1.1 Algoritmos usados

3.1.2 Grover

Um dos algoritmos mais comuns para a área é o algoritmo de Grover. Esse algoritmo realiza buscas em "bancos de dados"(bit strings) desorganizados em tempo $O(\sqrt{2^n})$ onde n é o número de qubits usados. Nele, usamos um circuito do qual amplifica-se a probabilidade de encontrar os valores marcados no Oracle na saída. Tal algoritmo segue o seguinte padrão:

1. Configuramos todas as possíveis bit strings, ou seja aplicamos uma superposição uniforme $H^{\otimes}_{n}$
2. aplicamos o phase oracle Uf do qual implementa uma função que marca os valores que desejamos encontrar
3. aplicamos o operador de Grover Diffuser, $\mathbb{I} - 2\left|s\right\rangle\left\langle s\right|$, sendo s o estado com o valor que queremos

Convertendo para um circuito temos algo semelhante a:

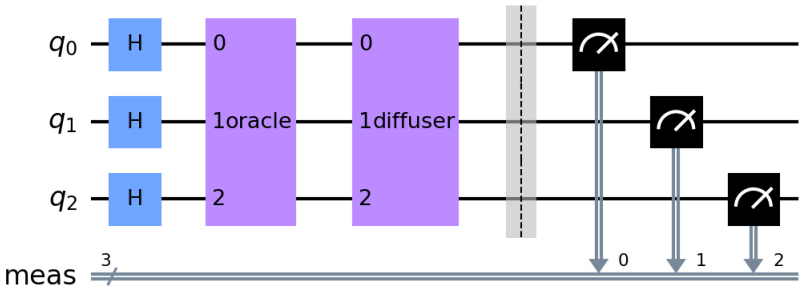


Figura 8: Exemplo algoritmo de Grover

Para 1 ou 2 valores, podemos usar a configuração acima. Mas maiores quantidade de valores, precisamos adcionar o conjunto Oracle + Diffuser k vezes, sendo $k \approx \frac{\pi}{4\sqrt{\frac{a}{2^n}}} - \frac{1}{2}$, sendo a o numero de valores marcados.

3.1.3 Diferença de conjuntos

Utilizando os phase oracles, podemos criar dois Oracles distintis com ranges de valores diferentes e sobrepor seus valores, realizando a operação de diferença de conjuntos.

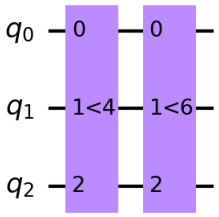


Figura 9: Exemplo de diferença de conjuntos

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

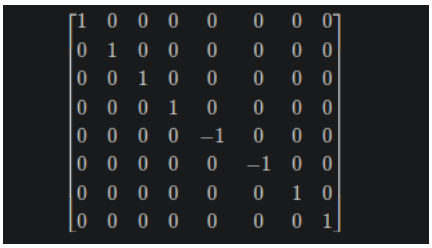


Figura 10: Resultado da diferença de conjuntos

Como pode ser visto, apenas os valores $\{100, 101\}$ permaneceram com a fase, representando então a sobreposição delas.

3.1.4 Solução para o problema

Para a solução do problema, podemos criar em uma hash function $C(v)$ da qual recebe o path de um arquivo e retorna uma bit string respectiva. Com essa função em mãos, podemos utilizar o conjunto dos valores retornados e encoda-los em um phase oracle, agindo como uma especie de Look Up Table para os arquivos existentes na máquina.

No entando, para ter sucesso na pesquisa, é necessário utilizar um segundo Oracle encondando $S = P - s$, sendo s o conjunto de arquivos do qual estamos procurando e S os arquivos restantes, aproveitando-se então da diferença de conjuntos para encontrar então apenas os valores desejados para a pesquisa.

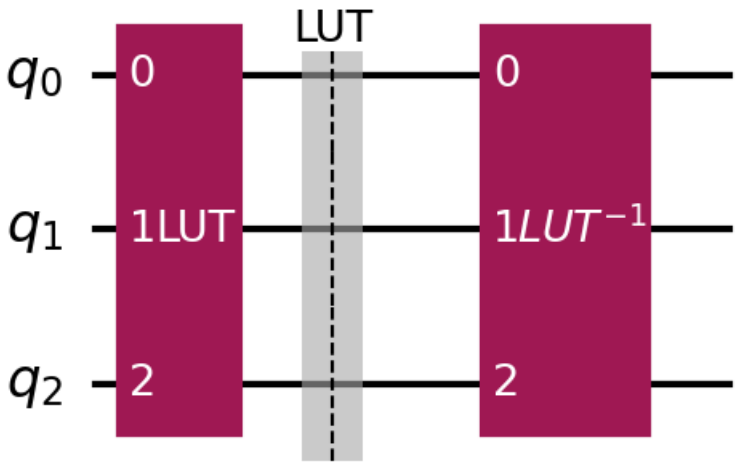


Figura 11: Diferença de conjuntos com as Look Up Tables

Sendo assim, o primeiro oracle age como o HD da marquina, marcando todos os arquivos existentes, e o segundo age como o mediador da pesquisa.

Com isso, adicionamos a Look Up Table final ao algoritmo de Grover.

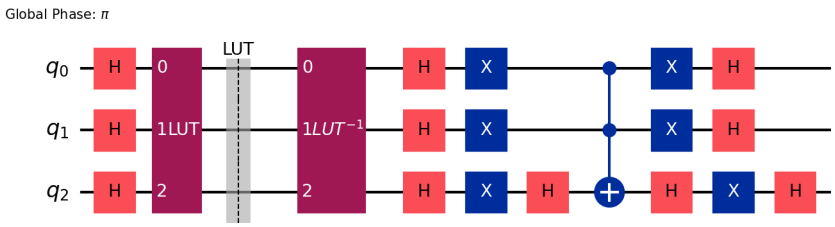


Figura 12: File explorer circuito

3.1.5 Resultados

Para um caso hipotetico de sistema completamente quântico, certamente esse é um das maneiras de encontrar arquivos em meio a todos os outros.

Contudo, como o objetivo dessa pesquisa sugere, para utilizar esse modelo em um sistema clássico tomando proveito da computação quântica, não se mostra como a melhor opção.

Para sistemas convencionais, dos quais utilizam métodos baseados em árvores, não é possivel tirar qualquer proveito aqui, sendo O polinomial contra O quadrático.

Mesmo perante os testes clássicos feitos durante o desenvolvimento, esse protocolo só se mantém util quando a pesquisa era feita de forma linear.

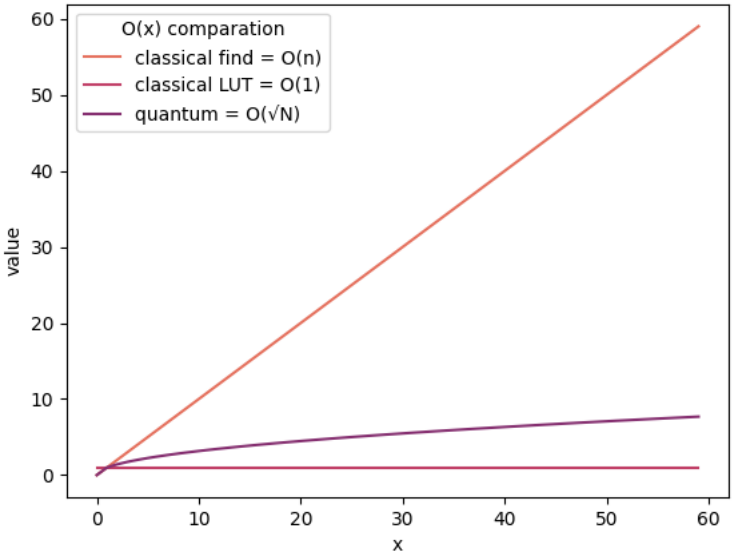


Figura 13: Comparação algoritmos usados na pesquisa

Sendo assim, o algoritmo de Grover deve ser pensando para casos do qual a versão clássica possui complexidade $\geq O(n)$.

3.2 Milhas para quilômetros

O segundo problema testado foi a conversão de milhas para Km. Essa idéia se deu a após a descoberta de um algoritmo capaz de calcular fibonacci com circuitos quânticos.

3.2.1 algoritmo de Fibonacci

A versão quântica usada para calcular fibonacci apresentada em [15], demonstra que utilizando um circuito que coloca em superposição todas as bit strings com n qubits que não possuem valores uns consecutivos, é possivel encontrar o valor n de fibonacci.

Sua implementação se dá da seguinte maneira:

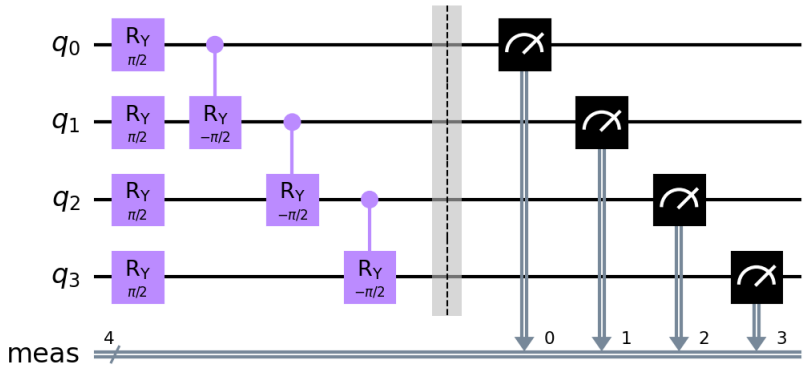


Figura 14: Exemplo Algoritmo de Fibonacci

Tendo como resultado:

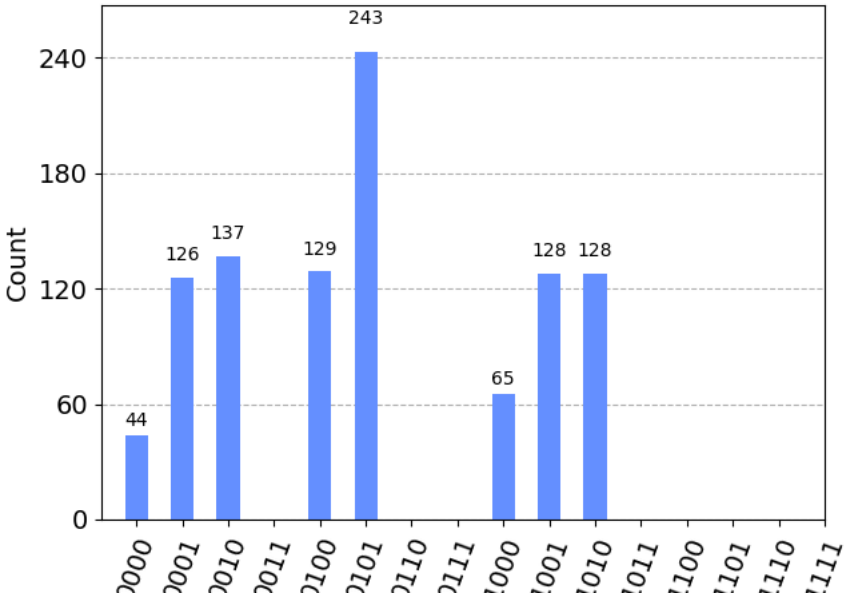


Figura 15: Resultado exemplo Fibonacci - F(4)

Essa implementação inicia a cadeia partir do valor 2, sendo assim: $2, 3, 5, 8, 13, 21, ...$ Repare que na figura 15, a quantidade de número de valores encontrados (nesse caso 8) é o mesmo do que o 4º valor da sequência. Ou seja, para esse circuito, o valor de qubits n representa o valor n de fibonacci que queremos encontrar. Sendo assim, podemos pensar em um oracle implementado $F(n)$ do qual a entrada n é igual a $|0\rangle^{\otimes n}$.

3.2.2 Implementação

Para aproximar o valor de milhas para quilômetros, podemos utilizar a relação de fibonacci da seguinte forma:

$F_{km} = F_{milhas}(n + 1)$, sendo assim, para conseguir o valor relativo em quilômetros, basta pegar o próximo valor na sequência.

milhas	km
1	2
2	3
3	5
5	8

Tabela 1: valores aproximados de milhas para Km

No entanto, dessa forma, temos um problema, a quantidade necessária de qubits. Para entradas n tendendo ao infinito, precisaremos de infinitos qubits. Para resolver isso, foi criado um algoritmo clássico simples para quebrar numero desejado em partes menores que podem ser calculadas em um circuito com menos qubits, $f : (n) \rightarrow ((n_1, i), (n_2, i), ...)$, mapeando então a entrada n para uma lista de tuplas contendo o número base e a quantidade de aplicações necessária i.

Com isso, o circuito final é apresentado da seguinte forma:

Algorithm 1 Algoritmo quântico para a conversão

for cada número base n **do**
    Aplique o oracle F(n)
    Faça as medições nos qubits
    reset os qubits usados
end for
verifique o resultado de cada bit string
multplique cada resultado com o valor i correspondente

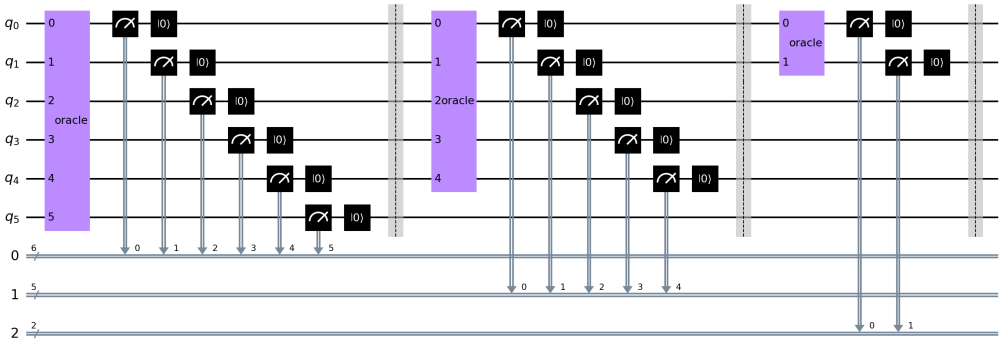


Figura 16: Circuito de conversão implementado

3.2.3 Resultados

Usando esse método é possível alcançar os valores esperados. Contudo existem alguns pontos que torna esse método pior do que a versão clássica:

1. Quantidade necessária de mediçoes e tempo de execução
   Para cada vez que medirmos o circuito, precisamos de uma quantidade alta de shots para alcançar um resultado melhor, aumentando também o tempo necessário para executar desse versão.

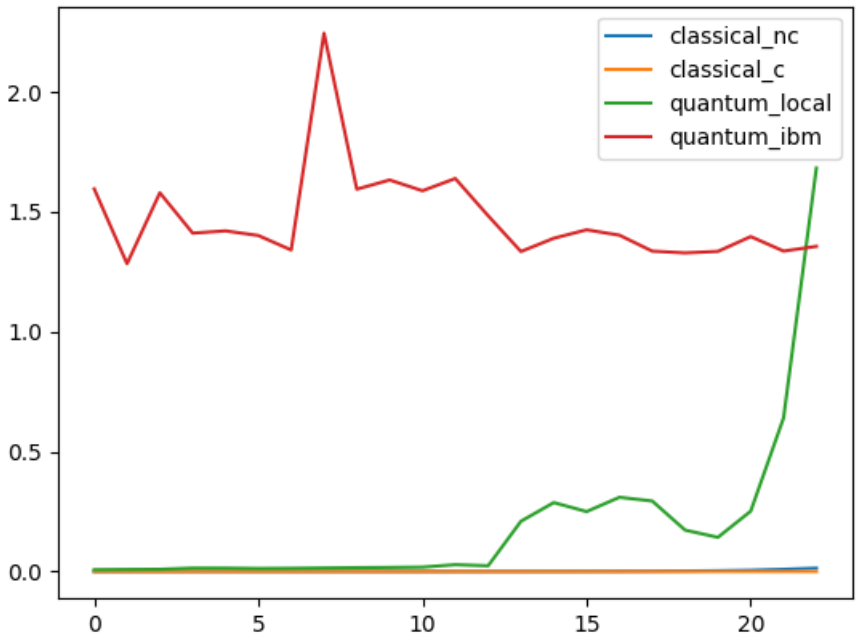


Figura 17: Comparação tempos de execução

Como é possivel ver em 17, o tempo das versões clássicas com e sem memoization possuem tempo praticamente constante de execução em comparação com as versões quânticas em simuladores e hardwares reais.

2. Erros
   Como a maioria dos algoritmos quanticos da era NISQ(noisy intermediate-scale quantum), os erros também estão presentes, e por utilizarmos muitos gates de dois qubits, esses erros podem se intensificar dependendo do hardware usado.

3. Imprecisão conforme a entrada cresce ao infinito

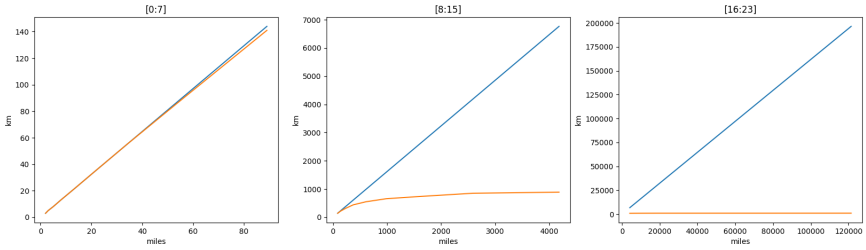


Figura 18: Comparação resultados versão clássica e quântica

Como é possível ver em 18, valores pequenos possuem uma boa precisão com os números esperados(em azul), mas apartir de certo ponto eles começam a se distanciar.

4. necessidade de intervenção clássica

   Por fim, esse algoritmo requer que seja primeiro verificado quais são os valores de fibonacci necessários para cada parte, além de ser necessário pós processamento após as medições.

   Para amenizar tal problema, é necessário criar outros algoritmos capazes de encodar o valor requerdio diretamente no circuito sem maiores intervenções clásssicas, necessitando apenas de pós processamento (em certos casos).

3.3 Torres de Hanoi Hanoi

Para o terceiro teste, foi implementado uma versão das torres de Hanoi usando os Oracles como meio de encoding.

3.3.1 Implementação

Para implementar as torres de hanoi em um oracle, são necessários $(\lfloor \log_2 x \rfloor + 1) * 3$ qubits, sendo x o número de discos com a seguinte 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 t a ultima torre, a a torre do meio e s a primeira torre, e $n = \frac{nqubits}{3}$.

Com essa configuração, os numeros de 1 à x são codificados em um phase oracle, e em seguida são realizadas operaçoes de $swap$ para modificar a posição dos valores binários e mover os valores do n qubits menos significativos, para os n qubits mais significativos.

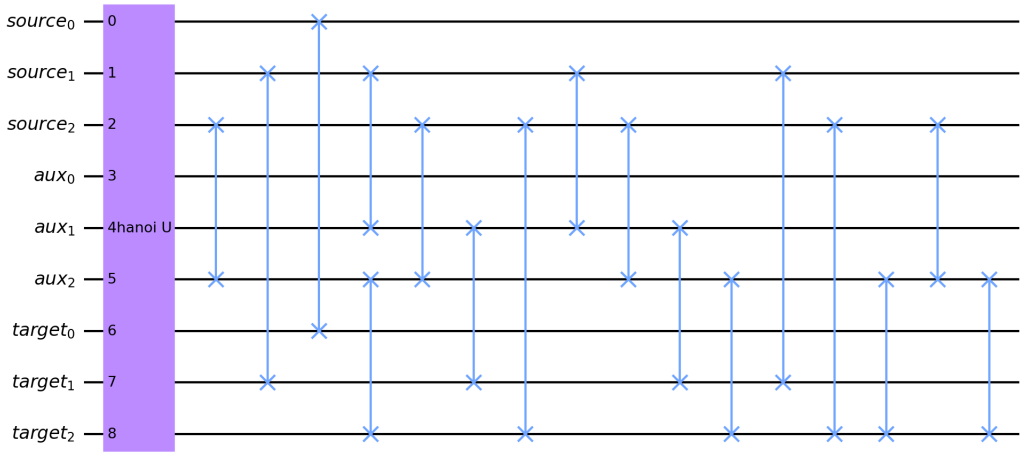


Figura 19: Torre de hanoi com 4 discos

Nesse formato, é possível utilizar o unitário gerado pelo oracle, assim como o algoritmo de Grover para verificar o resultado.

3.3.2 Resultados

Nessa implementação, o algoritmo segue o mesmo padrão da versão clássica, sendo o melhor ou pior desempenho dependendo estritamente do hardware usado.

3.4 Buckshot Roulette

Buckshot Roulette é um jogo feito pelo desenvolvedor Mike Klubnika para computador que toma como base a premissa de reinventar a infame roleta russa. No jogo, você é desafiado por um demônio (dealer), e caso você ganhe você ganha uma recompensa, mas caso contrário o jogo reinicia.

Para esse projeto, tomamos como base a primeira rodada do jogo buscando encontrar a melhor estrategia para maximizar os ganhos.

Nessa primeira rodada temos 3 vidas, 2 balas falsas e 1 bala verdadeira.

A dinamica funciona da seguinte forma: você começa jogando,tendo duas possíveis escolhas, atirar em você mesmo ou no dealer. Caso você atire em você mesmo e a bala for real, você perde uma vida, mas você joga novamente na próxima rodada, caso a bala seja falsa você joga novamente. Agora, caso a escolha seja atirar no dealer, se a bala for real você ganha, se não o dealer joga.

3.4.1 Versão clássica

Antes de crair o circuito, foi feita a modelagem do jogo usando a estrutura em árvore.

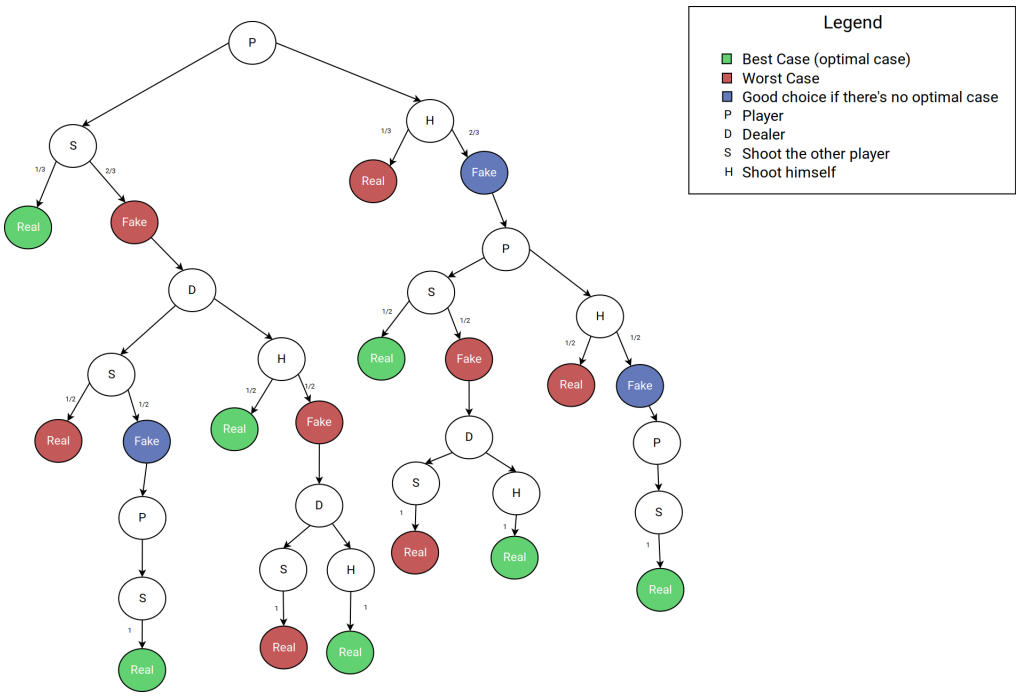


Figura 20: Buckshot Roulette diagrama

Note que nessa estrutura, o dealer na ultima bala, ainda pode escolher atirar ou não nele mesmo, isso acontence pois o jogo implementa ações randomicas para o dealer.

Seguindo essa estrutura, podemos tentar encontrar o melhor caminho dentre a àrvore e simular os resultados da partida.

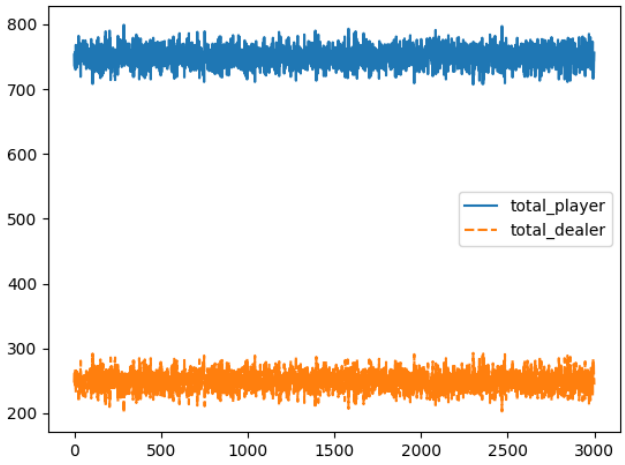


Figura 21: Buckshot Roulette clássico - melhor estratégia

Após testar os possíveis caminhso, o melhor resultado obtido foi esse apresentado acima em 21. Com um pouco de investigação, foi possível entender que a melhor estratégia encontrada foi o player começar atirando no dealer. Isso pois ao seguir tal caminho, ele tem uma chance a menos de perder a rodada, já que a chance de perder logo no começo é eliminada.

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

3.4.2 Versão quântica

A partir dessa ideia, um circuito quântico foi modelado imitando o primeiro round, e um oracle foi usado para cada jogador implementando sua estratégia em seu interior.

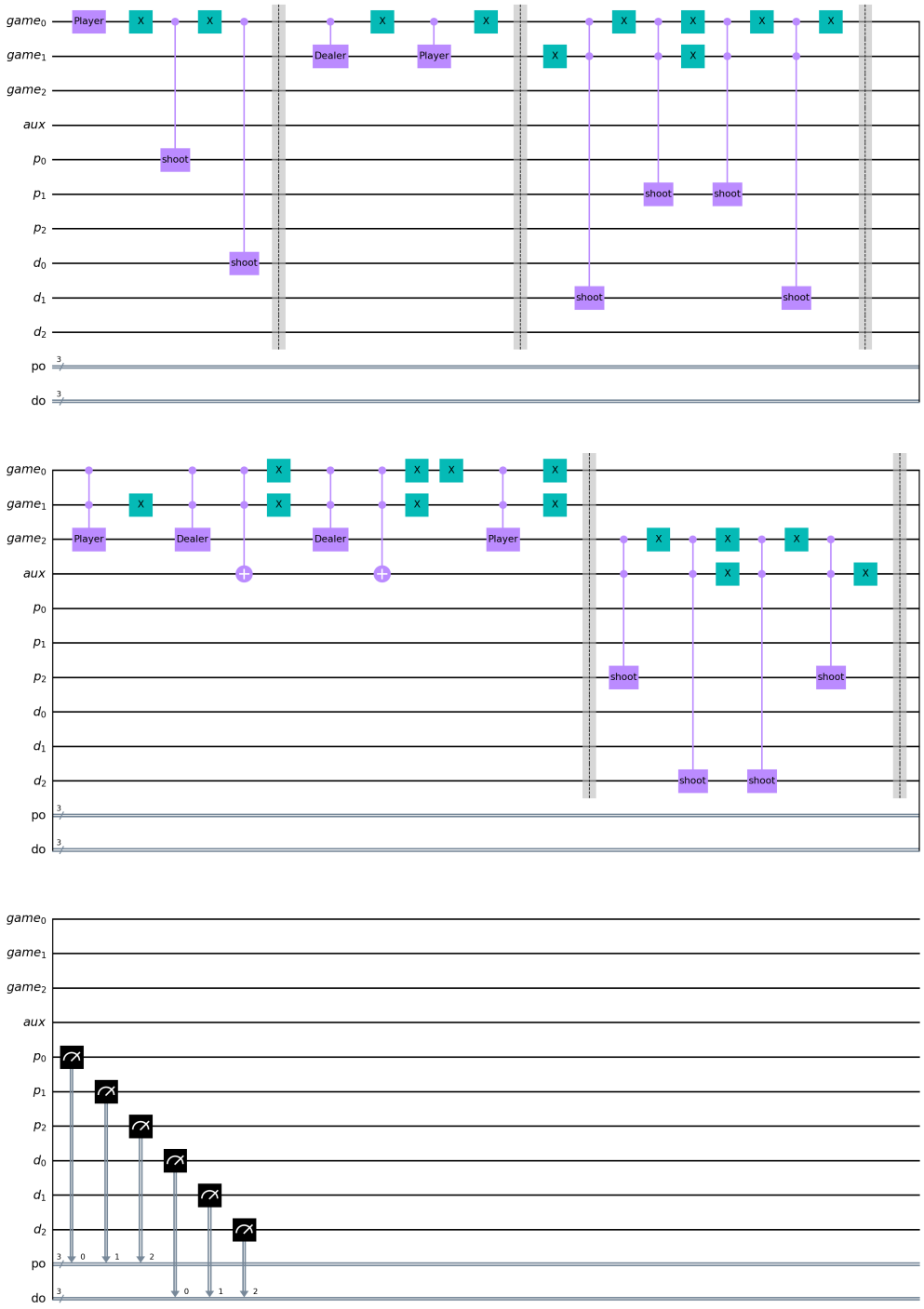


Figura 22: Circuito para o buckshot roulette

Além disso, para encontrarmos a melhor estratégia para o player, foi inserido dois parâmetros dentro do oracle, sendo possível inserir qualquer valor θ e ϕ para encontrar a melhor rotação na bloch sphere.

Após verificar uma grande variedade de valores possíveis, a rotação que entregou o melhor resultado foi $\theta = 3.0853981633974477, \phi = 5.685398163397447$. Usando essa estratégia, os resultados foram semelhantes a versão clássica usando o simulador Aer:

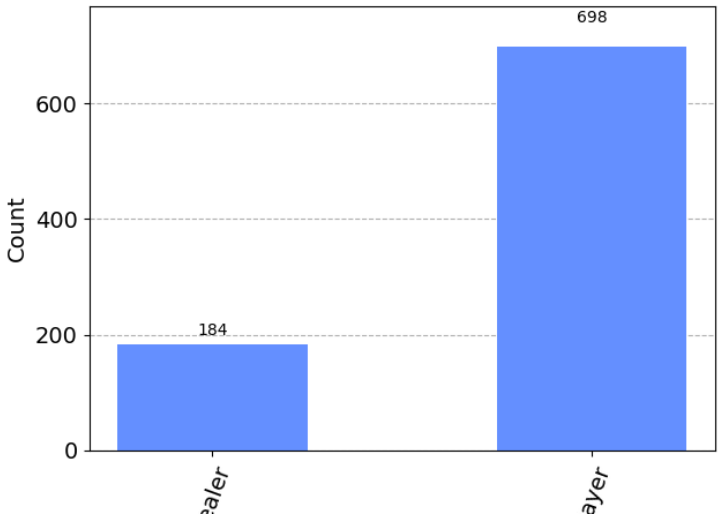


Figura 23: Resultado Buckshot Roulette quântico

Observando a bloch sphere do estado gerado por essa rotação, é possível ver também que a estratégia de fato se assemelha a versão classica, com o player preferindo atirar no dealer a maior parte do tempo (o valor 1 aqui representa atirar no outro jogador e 0 atirar em si mesmo).

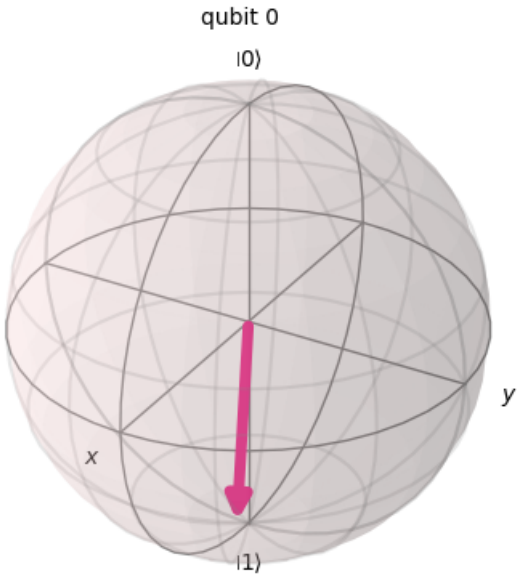


Figura 24: Melhor estratégia Buckshot Roulette quântico - Bloch Sphere

3.4.3 Conclusões

Para esse problema, não há uma competição certa entre as duas versões, uma vez que uma é diretamente inspirada na outra.

Além disso, a versão quântica possui ainda a possibilidade de explorar mais valores do que a versão clássica, deixando o player mais aberto a escolha de estratégias.

Em relação a erros providos pelo hardware não afeta diretamente os resultados, uma vez que mesmo com os erros a proporção se mantém.

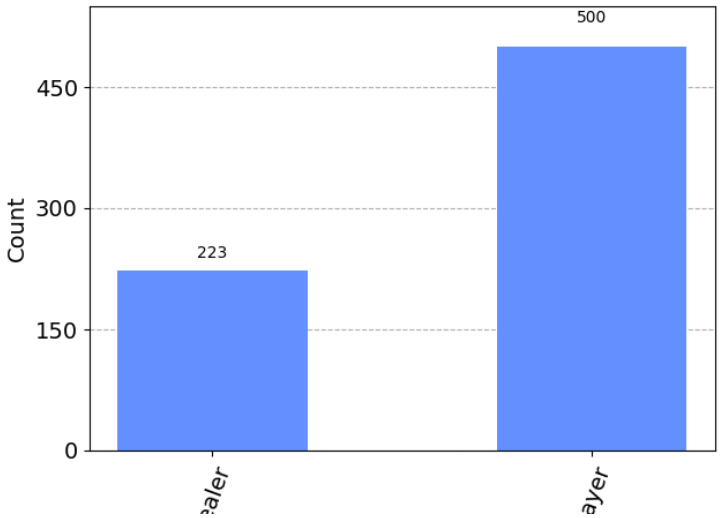


Figura 25: Resultados Buckshot Roulette usando o fake backend Melbourne da IBM

Note também que o total de partidas ganhas não chega ao total jogado, 1000 partidas no total, isso pois, pelo design do circuito, não é possível verificar a jogada do player anterior, sendo possível continuar jogando mesmo que um dos players já tenham perdido, o que foi necessário correções usando pós processamento após a simulação.

Em suma, ambos as simulações atingiram o mesmo resultado e foi demonstrado que é possível usar aqui 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 para encodar os valores desejados. Nessa versão, foi testado maneiras de criar QROMs (com dados estaticos dentro), e uma possível maneira de utilizar uma QRAM habil para escrita.

Neste, foi levado em consideração o armazenamento de estados quânticos, e não de bitstrings clássicas.

3.5.1 QROM

Primeiro, foi feita uma versão de QROM, do qual utiliza n qubits para endereços e m qubits para a criação dos estados. Não necessáriamente os valores precisam estar correlacionados, podemos ter $n = 3, m = 10$. Isso pois, nesse formato, podemos mapear diversas superposições diferentes e aplicálas quando certo endereço for chamado. Sendo assim, o algoritmo aqui mostrado 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: $|0\rangle^{\otimes m} |a_{n-1}a_{n-2}...a_0\rangle$

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

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

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

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

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

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

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

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

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

[11] 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.

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

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

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

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

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

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

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

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

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

[21] 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.

[22] 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.

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

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

[25] 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`.

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

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

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

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

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

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

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

[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.