Quantum Algorithms for the Leader Election Problem in Anonymous Networks

Leonardo Oleynik

Final Technical Report
Fourth bimester

Advisor: Prof. Luiz A. de P. Lima Jr

Curitiba, November of 2020

1 Abstract

Given the classical inexistence of an exact Leader Election algorithm (without errors and that ends in a limited time) for an anonymous network, this project aims to study and test the respective quantum solution. In this way, we will research and implement the algorithm presented in [1, 2], using the algebra of Quantum Mechanics. Moreover, we will develop a quantum circuit capable of testing it, in which we will use the Python language and the Qiskit library.

2 Resumo

Diante da inexistência clássica de um algoritmo de Eleição de Líder exato (sem erros e que termina em tempo limitado) para uma rede anônima, este projeto visa estudar e testar a respectiva solução quântica. Desse modo entenderemos e implementaremos o algoritmo apresentado em [1, 2], usando a álgebra da Mecânica Quântica, para isso; e desenvolveremos um simulador de circuitos quânticos capaz de testá-lo, no qual usaremos Python e Qiskit como linguagem e biblioteca, respectivamente.

3 Introdução

Os computadores usuais estão limitados pelo fato clássico que sistemas só podem estar em um estado de cada vez. Contudo, a Mecânica Quântica (MQ) tem nos ensinado o contrário — antes da medição o sistema está numa superposição de todos os estados possíveis. Um sistema quântico exibe, portanto efeitos de interferência e não-localidade, de modo que mais de uma operação pode ser realizada simultaneamente e em regiões distintas.

É nessa nova arena de possibilidades e explorando cada uma das propriedades quânticas mencionadas acima que reside as áreas de Computação e Informação quântica. E, dentre as inúmeras aplicações possíveis, podemos mencionar (para uma lista completa e atualizada, veja [3]):

- *Distribuição de chave quântica.* A vantagem quântica, em relação à clássica é evidente — pois, enquanto esta usa da dificuldade de certos problemas matemáticos como garantia da chave, aquela usa leis da MQ, que garantem, por exemplo, que um espião jamais conseguirá interceptar mensagens sem ser reconhecido.
- *Aleatoriedade quântica.* Ao contrário dos dispositivos clássicos determinísticos, os dispositivos quânticos podem gerar aleatoriedade intrínseca. Notavelmente, esse recurso pode ser explorado para expandir uma semente aleatória pequena para uma sequência muito maior de bits aleatoriamente certificados, mesmo que alguém desconfie do equipamento usado para esse fim. As aplicações disso são, por exemplo, para proteger protocolos de comunicação, amostragem estatística imparcial e simulações de Monte Carlo.
- *Metrologia quântica.* A tecnologia quântica tem vantagens para alguns tipos de detecção. Os sistemas quânticos podem detectar forças fracas com melhor sensibilidade e maior resolução espacial quando comparamos com as tecnologias usuais. O LIGO (*Laser Interferometer Gravitational-wave Observatory*) conta, por exemplo, com um dispositivo quântico de detecção.

O problema a ser tratado neste trabalho é o da *eleição de líder em uma rede anônima* (ELA). Esse, como afirma [1], é uma questão central na computação distribuída, pois, uma vez resolvida, permitiria-nos elucidar muitas outras — dentre os quais, podemos citar o protocolo de *Spanning Tree* (capaz de gerar topologias de rede livres de *loops*) e o problema de valor máximo (fundamental na construção de algoritmos mais complexos).

Assim, diante do difícil acesso à computadores quânticos e do aspecto irredutivelmente quântico da solução do problema acima proposto[1]; este trabalho tem dois grandes objetivos:

1. o uso de uma ferramenta capaz de preencher essas lacunas, *o simulador de computador quântico*;
2. e, por meio desse, a implementação, conforme encontrada na literatura, de um algoritmo quântico que eleja, de maneira unânime e em tempo limitado, um líder em uma rede anônima.

O restante do projeto estrutura-se da seguinte forma: Na seção 4, a fundação teórica sobre computação quântica é apresentada; em 5, a solução do problema apresentado anteriormente e seus aspectos tecnológicos são apresentados com detalhes; em 6 indicamos como os diversos módulos do projeto serão testados e validados estipulando, portanto, critérios de aceite; e, finalmente, na seção 7 este projeto é concluído e suas referências bibliográficas expostas.

[1] Como ficará claro na seção 5.

4 Estado da Arte

Dada a natureza do problema, esta seção conta com os seguintes tópicos: Mecânica Quântica, na qual apresentamos brevemente o formalismo de Dirac; Computação Quântica, na qual apresentamos as portas lógicas utilizadas na implementação do algoritmo; Simuladores Quânticos (SQ), dedicada a revisão dos principais simuladores universais; e Algoritmos de Eleição de Líder Quânticos (AELQ), na qual analisamos criticamente as soluções propostas em [1, 2].

4.1 Mecânica Quântica

A MQ afirma que para qualquer sistema físico é possível associar um *estado*. No formalismo de Dirac, esse é representado por um vetor [2], denotado por $|\psi\rangle$ que mora em um espaço de funções complexas, o espaço de Hilbert. Segundo as propriedades desse espaço podemos escrever o estado dessa forma:

$$|\psi\rangle = \sum_i \alpha_i |u_i\rangle ; \qquad \alpha_i := \langle u_i|\psi\rangle , \tag{1}$$

na qual $\{|u_i\rangle\}$ representa uma *base* do espaço e $\langle u_i|\psi\rangle$ o produto escalar entre $|\psi\rangle$ e $|u_i\rangle$.

Uma base é qualquer conjunto de vetores que satisfazem às seguintes propriedades:

1. Ortonormalide:$\langle u_i|u_j\rangle = \delta_{ij}$, em que δ_{ij} é a delta de Kronecher;
2. Completude: $\mathbb{I} = \sum_i |u_i\rangle\langle u_i|$.

(Perceba a semelhança da primeira com definição de base euclidiana.)

Vamos exemplificar os conceitos acima com um elétron livre. Das aulas de Química, sabemos que essa partícula possui um grau de liberdade intrínseco chamado *Spin* e que este está associado a rotação daquela em torno do próprio eixo, logo temos dois estados possíveis: o horário $|1\rangle$ e o anti-horário $|0\rangle$. Pela excludência das suas realizações — ou o elétron gira em um sentido ou no outro — esse estados formam uma base: a base *Computacional*. Portanto, uma possível representação do estado do elétron livre antes da medição é:

$$|\psi\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}, \tag{2}$$

caso específico de

$$|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle , \tag{3}$$

quando as probabilidades são as mesmas. Além disso, qualquer estado escrito nessa base são denominados *q-bits*.

Generalizando a ideia acima para n graus de liberdade (ou n elétrons livres) temos:

$$|\psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle , \tag{4}$$

em que i refere-se ao equivalente binário de todos os q-bit — *e.g.*, $|3\rangle = |1\rangle \otimes |1\rangle = |1\rangle^{\otimes 2} = |1\rangle |1\rangle = |1, 1\rangle$. Assim a sequência de n q-bits representa o *registrador quântico* de tamanho n.

[2] Mais conhecido por *ket*

4.1.1 Vetores consistentes e inconsistentes

Dizemos que um *string* $x = x_1x_2\ldots x_n$ de n bits é *consistente* sobre um conjunto de índices $S = \{1, 2, \ldots, n\}$ se $x_i = x_j$ para todo i e $j \in S$; caso contrário ela é *inconsistente*. O que essa definição está nos dizendo é que para x ser consistente, todos os seus bits devem ser iguais — *e.g.*, se n=3 a sequência 1 1 0 é inconsistente sobre $S = \{1, 2, 3\}$, mas a subsequência formada por seus dois primeiros bits 1 1 é consistente sobre $S' = \{1, 2\}$.

Posto isso, dizemos que um estado puro de n q-bits é consistente (inconsistente) sobre S se $\alpha_i \neq 0$ somente para i consistente (inconsistente) sobre S. Ou seja, para o estado ser consistente (inconsistente) ele só pode ter componentes nas direções consistentes (inconsistentes) — *e.g.*, se $n = 2$ os estados

$$|C\rangle = \alpha_0 |00\rangle + \alpha_3 |11\rangle \tag{5}$$

e

$$|I\rangle = \alpha_1 |01\rangle + \alpha_2 |10\rangle \tag{6}$$

são, respectivamente, consistentes e inconsistentes. Perceba, ainda, que os dois vetores formam uma base $\{|C\rangle, |I\rangle\}$, denominada de *consistente*.

4.2 Computação Quântica

Se em computação clássica podemos representar qualquer algoritmo como um circuito composto de *bits* e *portas lógicas*, em Computação Quântica não será (muito) diferente. Evidentemente, a generalização desses ingredientes para o caso quântico merece algumas ressalvas. A primeira delas diz respeito ao q-bit que de forma geral e antes da medição é uma superposição dos bits clássicos 0 e 1, de modo que a melhor previsão que alguém pode fazer será em termos de probabilidades (*e.g.*: metade das vezes o resultado será um). É justamente, por isso que toda computação quântica deve ser analisada a partir de um conjunto suficientemente grande de realizações afim de que as frequências relativas aproximem-se, pela *regra dos grandes números*, às distribuições probabilísticas. A segunda está relacionada às portas que, devido ao caráter reversível da mecânica quântica antes do colapso, devem ser todas unitárias.

A seguir descrevo as portas Hadamard, X e Controled-Not, recorrentemente usadas no algoritmo.

4.2.1 Hadamard

O circuito 1 representa a seguinte operação sobre o q-bit q, inicializado com $|0\rangle$ (por padrão todo q-bit é inicializado dessa forma):

$$H |0\rangle = (|0\rangle + |1\rangle)/\sqrt{2}. \tag{7}$$

Ou seja, ele gera uma *superposição equiprovável*.

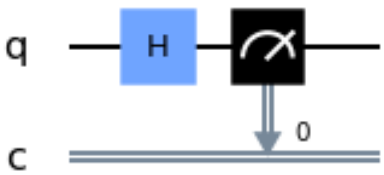

Figura 1: Hadamard

4.2.2 X

O circuito 2 representa a seguinte operação sobre o q-bit q, inicializado com $|0\rangle$ e $|1\rangle$ respectivamente:

$$\begin{aligned} X\,|0\rangle &= |1\rangle \\ X\,|1\rangle &= |0\rangle \end{aligned} \tag{8}$$

Ou seja, ele troca o estado clássico 0 por 1 e *vice-versa*.

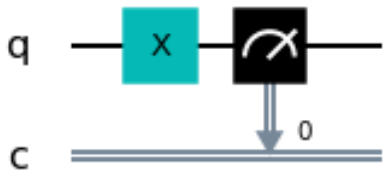

Figura 2: Porta X

4.2.3 Controled-NOT

O circuito 3 representa a seguinte operação sobre os q-bits inicializados com $|x\rangle$ e $|y\rangle$:

$$CNOT\,|x,y\rangle = |x, x \oplus y\rangle\,, \tag{9}$$

em que $\oplus$ representa soma módulo 2. Isto é, a rotação X só é aplicada sobre o segundo q-bit se o primeiro for $|1\rangle$.

Essa porta é especialmente valiosa quando $|x\rangle$ é uma superposição, pois o resultado (aplicando a relação acima) é o seguinte estado maximamente *emaranhado*:

$$\begin{aligned} CNOT\frac{(|0\rangle + |1\rangle)}{\sqrt{2}}\,|0\rangle = CNOT\frac{(|0,0\rangle + |1,0\rangle)}{\sqrt{2}} &= \frac{(CNOT\,|0,0\rangle + CNOT\,|1,0\rangle)}{\sqrt{2}} \\ &= \frac{(|0,0\rangle + |1,1\rangle)}{\sqrt{2}} \end{aligned} \tag{10}$$

também conhecido como estado GHZ_2. Sua generalização para n dimensões é:

$$|GHZ_n\rangle = \frac{|0\rangle^{\otimes n} + |1\rangle^{\otimes n}}{\sqrt{2}}, \tag{11}$$

e quando a normalização não é importante adotamos a seguinte notação

$$|ghz_n\rangle = |0\rangle^{\otimes n} + |1\rangle^{\otimes n}\,. \tag{12}$$

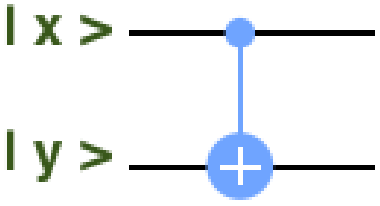

Figura 3: Controled-NOT

4.3 Simuladores

Segundo [4] existem duas categorias de simuladores: os *Universais*[3], que fornecem uma representação gráfica do circuito; e os *Simbólicos*, que focam na representação simbólica dos estados quânticos. Ou seja, enquanto aqueles possibilitam uma visão global do algoritmo implementado, mas restritos, em geral, à representações numéricas; estes são ótimas "calculadoras", pois reproduzem e simplificam expressões algébricas. Contudo, diante do problema proposto 5, nos restringiremos, neste estudo, à categoria dos Universais.

Posto isso, destacaremos algumas das principais particularidades de apenas sete dos inúmeros simuladores catalogados (vide [5]) de acordo com os seguintes "tipos":

- *Precursores*. São aqueles dignos de relato por motivos bibliográficos, i.e., por serem os pioneiros, contudo ultrapassados tecnologicamente;
- *Comerciais*. São aqueles disponibilizados por grandes empresas da computação (as quais, inclusive, oferecem acesso à *hardwares* quânticos) e, portanto, de qualidade profissional. Em poucas palavras, são os simuladores que os pesquisadores da área usam;
- *Simples*. São os que apresentam apenas as funcionalidades básicas (mas que, ao mesmo tempo, *provam o conceito*), sendo, desse modo, próximos da solução que pretende-se implementar neste trabalho.

4.3.1 Precursores

Considerado o primeiro simulador comercial, o *Senko's Quantum Computer* da *Senko Corporation* contava com um layout gráfico bastante simples, a construção gráfica dos circuitos era por manipulação direta, e era capaz de criar portas arbitrarias por edição de matriz. Além disso, dispunha da representação simbólica dos estados quando associado ao *Mathematica* — exigindo, portanto, que o usuário comprasse e dominasse a linguagem desse último, como destacado em [4].

Visando justamente sanar essa dificuldade e como forma de integrar duas categorias (Simbólica e Universal), até então disjuntas em um único simulador, Alexandre em [4] atualiza o simulador *Zeno* (cuja primeira versão foi desenvolvida por Gustavo E. M. Cabral em [6]) com um Sistema de Aritmética Computacional. Assim, a nova versão do Zeno, além de contar com as mesmas funcionalidades do *Senko*, oferecia gratuitamente ao usuário uma ferramenta com o *melhor dos dois mundos*.

Além desses vale destacar o *Quirk*, cuja última versão (2019) encontra-se em [7]. Considerado por Wagner, em [8], o "simulador universal mais completo disponível gratuitamente", o simulador, além de contar com as funcionalidades dos demais já citados, dispõem, também, de diversas maneiras de apresentar os estados (*e.g.*, ponto na esfera de Bloch); de uma caixa de ferramentas editáveis; de uma construção dinâmica do circuito, sem a necessidade de compilá-lo; entre outras.

4.3.2 Comerciais

Desenvolvido pela *Rigetti* o *Quantum Virtual Machine* (QVM) é uma plataforma *online* que, segundo [9], oferece ao usuário simulação com até 30 q-bits (mas com tempo de execução limitado) e teste real com até oito q-bits; repositório de exemplos e tutoriais (o Grove); fóruns de discussão; simulação de ruído sofisticada (fundamental no desenvolvimento de algoritmos de curta profundidade, a realidade dos computadores quânticos à curto prazo); entre outras funcionalidades. Além disso, sua linguagem host é o Python, a favorita dos principiantes.

[3] Vale salientar que o termo "Universal", aqui, em nada se refere àquele usado em computação

Em pé de igualdade com esse temos o *Quiskit* da *IBM* (cuja plataforma, *IBM-Q experience*, igualmente oferece um repositório de tutorias e fórum), que também oferece até 30 q-bits de simulação, mas teste real com até 20 q-bits. Além disso, ele possui inúmeros simuladores, cada um especializado em dada tarefa. Suas linguagens *host* são o Python, JavaScript e Swift.

4.3.3 Simples

Criado por Davy Wybiral o *Quantum Computer Simulator* QCS — cuja documentação e applet estão disponíveis, respectivamente, em [10] e [11] — é um simulador bastante simples, mas com as portas básicas necessárias e exemplos dos principais algoritmos quânticos.

Inspirado no anterior, Igor V. Blackmann desenvolveu, conforme descrito em [12], um simulador com o mesmo nome e as mesmas funcionalidades, mas capaz de simular o tempo de execução de cada porta. Diferentemente do primeiro, nesse os cálculos quânticos foram feitos através da biblioteca Qutip.

4.4 Algoritmos Quânticos de Eleição de Líder

Nesta seção revisaremos os trabalhos de Tani et. al [1, 13] e do Ellie D'Hondt e Prakash Panangaden [2]. Ambos desenvolvem soluções para eleição de líder em redes quânticas, distribuídas, anônimas e síncronas, mas, enquanto aquele busca entender as particularidades dos recursos quânticos envolvidos à nível mais fundamental, este é voltado à aplicação; pois, além de avaliar os impactos de cada algoritmo (*e.g.*, o tempo de execução), ele também explica como gerar os recursos quânticos necessários. Para um revisão mais detalhada consulte [14]

Em relação ao artigo [1], esse apresenta dois algoritmos quânticos que, dado o número de partes n, resolvem *exatamente* o problema de eleição de líder, em qualquer topologia, com complexidade tempo-comunicacional polinomial (com relação à n) quando as partes estão conectadas através de *links* de comunicação quânticos. O primeiro oferece um tempo e complexidade comunicativa bastante inferior, quando comparado com o segundo. Contudo, este é mais geral, pois funciona satisfatoriamente em qualquer topologia, enquanto que aquele apenas naquelas contendo grafos indiretos.

Quanto à [2], sua grande contribuição consiste na demonstração do poder computacional especial dos estados W e GHZ. Ele a faz provando que tais estados são os únicos que resolvem de forma *exata* os problemas de eleição de líder e distribuição de consenso, respectivamente. No âmago de suas provas residem argumentos de simetria.

Comparar e afirmar qual dos algoritmos é o melhor é difícil, pois cada um foi concebido com uma finalidade distinta. A solução do segundo é simples e bastante didática, sendo, assim, um ótimo exemplo das vantagens quânticas na resolução do problema; enquanto que a compreensão do primeiro fica restrita aos especialistas da área.

5 Detalhamento do Projeto

5.1 O pseudo-código do algoritmo

O algoritmo consiste em repetir o mesmo procedimento $n-1$ vezes, as quais chamaremos de *fase*. Na i-ésima fase o conjunto dos índices das partes é $S_i \subseteq \{1, 2, \ldots, n\}$ (*e.g.*, $S_1 = \{1, 2, \ldots, n\}$, pois todas as partes são elegíveis na primeira fase), logo o objetivo do procedimento é garantir que $S_{i+1} \subseteq S_i$. Antes de seguirmos com sua descrição pormenorizada, cabe definir o parâmetro $k := n - i + 1$; a variável booleana `status`, cujos valores 1 e 0 denotam, respectivamente, elegível e inelegível; e o vetor `Status`, de tamanho n, cujas componentes são os `status` de cada parte.

Essencialmente, o objetivo das etapas 1 à 3 é preparar um estado inconsistente em R_0, de modo que ao medi-lo, em 4, ao menos uma das partes (idealmente apenas uma) obtém o valor 1 enquanto as demais zero. Nesse ponto a simetria do problema é quebrada, pois cada parte possui uma variável z que, em geral, é diferente das demais. Resta, agora, comunicar os resultados encontrados, computar o máximo sobre as partes elegíveis S_i e então definir que aquelas com z diferente do valor máximo são inelegíveis, isso é feito na etapa em 5 e 6

Todas as operações descritas a seguir (medições, leitura de registrador, etc) são *locais* (considerando cada parte individualmente e seus registradores de 1 q-bit), com excessão daquelas mencionadas explicitamente como *globais* (considerando todas as partes). Vale ressaltar, também, a seguinte notação: Registradores locais (que guardam apenas 1 q-bit) são denotados por letras minúsculas (r_0, r_1 e s); e os registradores globais (que guardam n q-bits) são denotados por letras maiúsculas (R_0, R_1 e S). Sendo estes compostos por aqueles. Por fim, os parênteses que aparecem no pseudo código a seguir denotam que a tal variável é necessária caso o número de partes seja ímpar.

Entrada:Variável n

Saída:Variável `Status`

- De $k = n$ até 2:
- Se `status` = 1:
    1. Prepara-se três registradores R_0, (R_1) e Sm com k e 1 q-bit, respectivamente.
    2. Executa-se a Sub-Rotina A com R_0, (R_1) e S
    3. Mede-se S, globalmente. Se o resultado é zero, executa-se a Sub-Rotina B com R_0 e (R_1).
    4. Mede-se r_0 e (r_1) e guarda-se o resultado em z.
    5. Comunica-se o resultado para todas as partes, encontrado o maior valor de z, z_m.
    6. Cada parte compara seu z com o z_m. Se $z \neq z_m$, então `status` = 0. Caso contrário, `status` =1.
    7. Atualiza-se `Status`.
- retorna `Status`.

A sub-rotina A essencialmente verifica a consistência de R_0 com S sobre o conjunto S_i. Fisicamente falando, os registradores interagem entre si, de modo que o estado resultante (que descreve o conteúdo dos dois) pode ser escrito como

$$|\psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i \big(|i\rangle \otimes |s_i\rangle^{\otimes n}\big), \tag{13}$$

tal que s_i é consistente se i for, e inconsistente caso contrário. É aqui que gera-se emaranhamento entre as partes. Perceba que $|\psi\rangle$ é um estado produto (tanto entre as partes como entre os registradores R0 e S) por definição (*c.f.* (13)), contudo seus ramos, associados as medições de S, são fatoráveis (apenas entre os registradores, mas não entre as partes)[4]. Ou seja, tendo correlacionado S com R_0, ao medir aquele, criamos emaranhamento entre as partes deste.

Separando o primeiro e último termo e explicitando os termos consistentes (primeiro e último) e inconsistentes (segundo) em S_i, (13) pode ser reescrito assim:

$$|\psi\rangle = \alpha_0\big(|0\rangle \otimes |s_i\rangle^{\otimes n}\big) + \sum_{i=1}^{2^n} \alpha_i\big(|i\rangle \otimes |s_i\rangle^{\otimes n}\big) + \alpha_{2^n-1}\big(|2^n-1\rangle \otimes |s_i\rangle^{\otimes n}\big). \tag{14}$$

Como exemplo ilustrativo, considere $n = 2$ e $S_i = \{1, 2\}$, então (14) fica

$$|\psi\rangle = \alpha_0(|00\rangle \otimes |C\rangle) + \alpha_1(|01\rangle \otimes |I\rangle) + \alpha_2(|10\rangle \otimes |I\rangle) + \alpha_3(|11\rangle \otimes |C\rangle). \tag{15}$$

Logo, no etapa 4, antes de medirmos globalmente S podemos afirmar que $|\alpha_0|^2 + |\alpha_3|^2$ da vezes o resultado da medida é consistente e $|\alpha_1|^2 + |\alpha_2|^2$ é inconsistente.

A sub-rotina B tem como objetivo tornar inconsistente o estado em R_0: caso k seja par, basta evoluir r_0 de cada parte segundo a matriz U_k,

$$U_k = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & e^{-i\frac{\pi}{k}} \\ -e^{i\frac{\pi}{k}} & 1 \end{pmatrix}, \tag{16}$$

responsável por anular as componentes consistentes; e caso k seja ímpar, é necessário copiar o conteúdo de R_0 em R_1 e evoluir r_0 e r_1 de cada parte segundo V_k,

$$V_k = \frac{1}{\sqrt{R_k+1}} \begin{pmatrix} 1/\sqrt{2} & 0 & \sqrt{R_k} & e^{i\frac{\pi}{k}}/\sqrt{2} \\ 1/\sqrt{2} & 0 & -\sqrt{R_k}e^{-i\frac{\pi}{k}} & e^{-i\frac{\pi}{k}}/\sqrt{2} \\ \sqrt{R_k} & 0 & \frac{e^{-i\frac{\pi}{2k}}I_k}{i\sqrt{2}R_{2k}} & -\sqrt{R_k} \\ 0 & \sqrt{R_k+1} & 0 & 0 \end{pmatrix}, \tag{17}$$

tal que R_k e I_k são as partes reais e imaginárias de $e^{i\frac{\pi}{k}}$, respectivamente; e cujo efeito é análogo à U_k. Essa operação é global no sentido de que deve (e só pode) ser feita sobre o estado global, pois a composição (produto tensorial) dos estados reduzidos não totalizam o estado global e suas correlações — i.e., existem correlações que não estão em nenhuma das partes o que é típico em estados emaranhados.

Em suma, o algoritmo é composto pelas seguintes operações responsáveis por:

- *Inicialização.* Inicializar os registradores R_0 e R_1 de acordo com o número de partes elegíveis;
- *Sub-Rotina A.* Emaranhar R_0 com S de maneira apropriada (conforme descrito em (13));
- *Sub-Rotina B*: Transformar o estado de R_0 (e R_1, caso o número de partes ilegíveis seja ímpar) em *inconsistente* (em pelo menos um dos registradores). Tal rotina só é chamada quando o resultado da medição de S é *zero*, i.e., quando o estado preparado na rotina anterior colapsa para um ramo *consistente*;
- *Maximização.* Encontrar o maior valor de z entre as partes em cada rodada.

Em conjunto tais operações reduzem o número de partes elegíveis a um, no pior dos cenários; e a uma parte elegível, no melhor dos cenários. Além disso todas as partes tem a mesma chance de *ganhar a eleição*, assim seus respectivos estados são *equiprováveis*.

[4]Inclusive os estados no registador R0, após a medida, são *estados de Bell*, isto é, são maximamente emaranhados.

5.2 Circuito lógico do Algoritmo

Posto a descrição geral do algoritmo dada na seção anterior, apresentaremos agora sua implementação em Python para 2 e 3 partes (respectivamente, o caso par e ímpar mais simples), bem como para 4, para tanto usaremos os circuitos lógicos gerados a partir do código fonte — nessa linguagem, as linhas e caixas representam, respectivamente, registradores e operações. Além disso, destacaremos porque não foi possível generalizar para $n > 4$.

5.2.1 Algoritmo para 2 partes

Apresentaremos a seguir os trechos mais relevantes da função `alg_par` do código fonte 8, analisando em detalhe os efeitos de cada sub-rotina e operação.

Primeiramente é necessário definir o tamanho do circuito de acordo com o número de partes elegíveis k e sua paridade,

```
s=[0]; r0=list(range(1,k+1));  r1=list(range(k+1,2*k+1))
qr=QuantumRegister(k+1,'q')
cr=ClassicalRegister(k+1,'c')
qc=QuantumCircuit(qr,cr)
```

Listing 1: Definição do circuito

Em seguida, inicializamos o registradores quânticos conforme prevê a sub-rotina A e medimos S

```
qc.initialize(psi(k),list(range(k+1)))
qc.measure(s,s)
```

Listing 2: Sub-Rotina A

usando, para tanto, da função `psi(k)`:

```
d=int(pow(2,n)); a=float(sqrt(1/pow(2,n)))
v=[0]*2*d; v[0]=a; v[int(2*(d-1))]=a;

for i in list(range(1,d-1)):
    v[2*i+1]=a
```

Listing 3: Função psi(k): Expressão matemática geral do estado $|\psi\rangle$

Então adicionamos a unitária `uk` sobre R_0,condicionando-a ao valor do registrador clássico `cr` e medindo R_0 ao final:

```
uk=UnitaryGate(u(k))
qc.append(uk,[r0]).c_if(cr,0)
qc.measure(r0,r0)
```

Listing 4: Sub-Rotina B para k par

Finalmente, a partir dos trechos acima, geramos a figura 4 pelo método `qc.draw( )`, a qual servirá de referência para as descrições subsequentes.

Quanto a sub-rotina A, conforme a figura 4, $|\psi_1\rangle$, $|\psi_2\rangle$ e $|\psi_3\rangle$ são os estados produto dos três registradores (ou seja, $|r0_0, r0_1, s_0\rangle \equiv |r0_0\rangle \otimes |r0_1\rangle \otimes |s_0\rangle \equiv |r0_0\rangle\,|r0_1\rangle\,|s_0\rangle$) nos seguintes instantes: após as Hadamard, após a primeira CNOT e final. E, suas expressões analíticas são (aplicando

(9) duas vezes):

$$
\begin{aligned}
|\psi_1\rangle &= |r0_0, r0_1, s_0\rangle = \frac{(|0\rangle + |1\rangle)}{\sqrt{2}} \frac{(|0\rangle + |1\rangle)}{\sqrt{2}} |0\rangle = \frac{|00\rangle + |01\rangle + |10\rangle + |11\rangle}{2} |0\rangle \\
|\psi_2\rangle &= |r0_0, r0_1, r0_0 \oplus s_0\rangle = |r0_0, r0_1, r0_0\rangle \\
|\psi_3\rangle &= |r0_0, r0_1, r0_1 \oplus (r0_0 \oplus s_0)\rangle = |r0_0, r0_1, r0_1 \oplus r0_0\rangle = \frac{|00\rangle + |11\rangle}{2} |0\rangle + \frac{|01\rangle + |10\rangle}{2} |1\rangle \\
&\quad .
\end{aligned}
\tag{18}
$$

Perceba que $|\psi_3\rangle$, a *saída* da Sub-Rotina A, corresponde exatamente à (15) quando definimos $|C\rangle$ e $|I\rangle$ como $|0\rangle$ e $|1\rangle$, respectivamente.

Acerca da sub-rotina B, na figura 4 as linhas verticais que terminam nas unitárias (caixas roxas) — matricialmente representadas como

$$
U_2 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix}, \tag{19}
$$

i.e., (16) com $k = 2$ — condicionam a operação ao resultado da medida de S_0, armazenado no registrador clássico cS. No caso, essa só é acionada quando o resultado for *zero*, o qual corresponde ao ramo esquerdo de $|\psi_3\rangle$; i.e., $(|00\rangle + |11\rangle)/2$, um estado *consistente*. Por fim, a ação dessas unitárias em cada q-bit resulta em um estado *inconsistente*, pois

$$
\begin{aligned}
U_2 \otimes U_2 \, |\psi_3\rangle &= \frac{1}{2}[U_2 \, |0\rangle \, U_2 \, |0\rangle + U_2 \, |1\rangle \, U_2 \, |1\rangle] \\
&= \frac{1}{4}[(|0\rangle - i|1\rangle)(|0\rangle - i|1\rangle) + (-i|0\rangle + |1\rangle)(-i|0\rangle + |1\rangle)] \\
&= \frac{1}{4}\big[|00\rangle - i|01\rangle - i|10\rangle + i^2|11\rangle + i^2|00\rangle - i|01\rangle - i|10\rangle + |11\rangle\big] \\
&= \frac{1}{2}[-i|01\rangle - i|10\rangle]
\end{aligned}
\tag{20}
$$

equivale à (6) com $\alpha_1 = \alpha_2 = i/2$. Assim, a saída da sub-Rotina B é sempre *inconsistente* como estipulado na seção anterior.

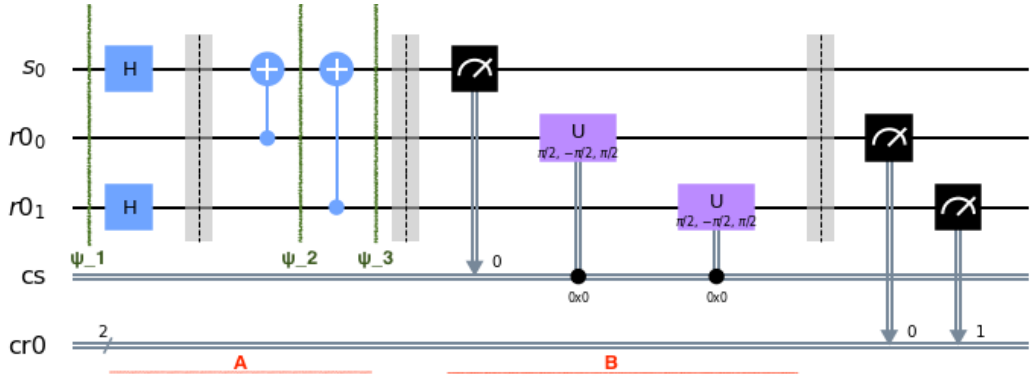

Figura 4: Circuito lógico do Algoritmo para 2 partes

5.2.2 Algoritmo para 3 partes

O código para um número ímpar de partes (função `alg_impar(k)` do código principal 8) é análogo ao apresentado acima, assim o papel das sub-rotinas, quando temos uma rede com 3 partes elegíveis, é idêntico àqueles descritos na seção anterior. A grande diferença está na forma de fazê-los. Se com 2 partes bastava evoluir cada q-bit em R_0 de acordo com (19), quando o resultado de S fosse *zero*, afim de obter *inconsistência*; aqui, pelo fato de termos um número ímpar, precisamos duplicar a dimensão e trabalhar tanto com R_0 como R_1. Logo o código é:

```
vk = UnitaryGate(V(k))
for i in range(k):
     subB.cx(i,k+i);
     subB.append(vk, [i,k+i])
qc.append(subB(k),list(range(1,2*k+1))).c_if(cr,0)
```

Listing 5: Sub-rotina B para k ímpar

Repare que — apesar de v_k ter o dobro da dimensão de u_k (*c.f.* (17) e (16), respectivamente) — o código é muito semelhante à 6, exceto pelo laço *for*, cuja função é *copiar* o conteúdo de R_0 em R_1.

Por fim, apresentamos o estado resultante $|\phi\rangle$ dessa sub-rotina:

$$\begin{aligned}|\phi\rangle^T =&[0, \frac{1+\sqrt[3]{-1}}{3\sqrt{6}}, -\frac{(-1)^{2/3}-1}{3\sqrt{6}}, 0, \frac{1+\sqrt[3]{-1}}{3\sqrt{6}}, -\frac{(-1)^{2/3}-1}{3\sqrt{6}}, 0, 0, -\frac{(-1)^{2/3}-1}{3\sqrt{6}}, 0, \frac{1+\sqrt[3]{-1}}{3\sqrt{6}},\\ &0,0,0,0,0, \frac{1+\sqrt[3]{-1}}{3\sqrt{6}}, -\frac{(-1)^{2/3}-1}{3\sqrt{6}}, 0, 0, -\frac{(-1)^{2/3}-1}{3\sqrt{6}}, 0, \frac{1+\sqrt[3]{-1}}{3\sqrt{6}}, 0, 0, \frac{1+\sqrt[3]{-1}}{3\sqrt{6}}, -\frac{(-1)^{2/3}-1}{3\sqrt{6}},\\ &0,0,0,0,0, -\frac{(-1)^{2/3}-1}{3\sqrt{6}}, 0, \frac{1+\sqrt[3]{-1}}{3\sqrt{6}}, 0, 0, \frac{1+\sqrt[3]{-1}}{3\sqrt{6}}, -\frac{(-1)^{2/3}-1}{3\sqrt{6}}, 0, \frac{1+\sqrt[3]{-1}}{3\sqrt{6}}, -\frac{(-1)^{2/3}-1}{3\sqrt{6}},\\ &0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]\\ &.\end{aligned} \tag{21}$$

Evidentemente é um estado um tanto quanto esquizofrênico e grande (64 componentes), contudo o que nos interessa nele são os seguintes pontos:

1. Ele é equiprovável. Ou seja, todas as suas componentes (não nulas) tem a mesma chance de serem medidas, pois o módulo ao quadrado de qualquer uma das amplitudes (que equivale a probabilidade) é igual $1/18$;

2. O estado só tem componentes *inconsistentes* ao menos em um dos registradores. Perceba que $|0\rangle = |7\rangle = |56\rangle = |63\rangle = 0$

Pontos esses em pleno acordo com a saída de uma sub-rotina B geral, como explicado anteriormente em 5.1.

5.2.3 Algoritmo para 4 partes

Posto o funcionamento do algoritmo no caso par, $n = 2$, e ímpar $n = 3$, o caminho para generalização, n arbitrário, consiste em conectá-los apropriadamente. Ou seja, é necessário:

- *Atualizar o Status continuamente*, de modo que apenas as partes elegíveis executem o algoritmo;
- *Controlar a Paridade da fase*, afim de chamar `alg_par(k)` e `alg_impar(k)` no momento certo.

o que é feito, respectivamente, pela função `status3` e pelo `If k%2` em:

```
def alg(n):
    status=[1]*n

    while status.count(1)>1:
        k=status.count(1)

        if k%2==0 :
            y=alg_par(k)
            while y.count(1)==0:
                y=alg_par(k)
            status=status3(status,y)

        else:
            y=maximo(alg_impar(k))
            while y.count(1)==0:
                y=maximo(alg_impar(k))
            status=status3(status,y)

    if status.count(1)==0:
        return -1
    if status.count(1)==1:
        return status.index(1)
```

Listing 6: Laço principal

A princípio temos todos os elementos necessários para eleger o líder de uma rede anônima de tamanho arbitrário (e maior que 4), contudo sua implementação real não foi possível devido a imprecisão da representação numérica da matriz v_k. Isto é, apesar de (17) ser exatamente unitária para todo valor de k, quando calculamos seu valor numérico (o que é exigido pela sintaxe da biblioteca Qiskit) ganhamos uma incerteza que para $k > 3$ é grande demais — grande a ponto da função `UnitaryGate()`, responsável por transformar uma matriz unitária em porta lógica, não aceitar o input.

6 Testes e Resultados

Para avaliar o *desempenho* do AELQ devemos computar o número de médio de *fases* para execução em função do número de partes n. Gerando, então, curvas denominadas por $f(n)$ que serão comparadas com os resultados teóricos. Assim, o desempenho será inversamente proporcional ao crescimento de f, *e.g.*, n^3 é ruim e n^2 é ótimo.

Quanto ao intervalo médio, definido e escolhido acima como métrica de desempenho, cabem alguns esclarecimentos. Perceba que o algoritmo (descrito em 5.1) executa exatamente $n-1$ vezes, como define o laço na etapa 2; mas o número de *passos* envolvidos em cada repetição varia de forma imprevisível — pois, dependendo do valor da variável `status`, procedimentos mais simples (ou mais complexos) serão executados, os quais correspondem a menos (mais) passos. Precisamente por isso que a análise estatística, por meio do número *médio* de fases, se faz necessária e pertinente.

Posto isso, surge imediatamente a pergunta: *porque não usar o tempo total, aquele medido com um cronômetro?* A resposta curta é que isso iria mascarar os resultados. Pois, ao adotar tal métrica, estaríamos computando o tempo gasto com as simulações dos efeitos quânticos, necessárias em nosso caso, mas muito maiores que aquele gasto com as demais operações que caracterizam o algoritmo em questão. Mais precisamente, enquanto o desempenho daquela cai exponencialmente com n, o desta cai polinomialmente. Em outras palavras, nosso sensor seria grosseiro demais.

Quanto ao teste *funcional* do algoritmo, o processo é mais simples. O protocolo deve eleger um líder de maneira exata, isto é, conforme definido por Ellie D'Hondt e Prakash Panangaden, em tempo limitado e sem erro. Em outras palavras, o algoritmo não pode ficar executando para sempre e a eleição deve ser unânime em 100% das vezes e, no caso do algoritmo descrito em 5.1, é imprescindível que seja imparcial, todas as partes tem a mesma chance de vencer.

6.1 Algoritmo para 2 partes

Como os estados não são *observáveis* (não se pode medi-los no laboratório), justificaremos as saídas previstas de acordo com suas respectivas probabilidades — na verdade, sendo mais preciso, de acordo com as frequências relativas em 1000 realizações do algoritmo. No eixo y do histograma temos as frequências relativas e no x os resultados das medidas dos registradores.

Perceba que todas as probabilidades em 5 são aproximadamente 1/4 de modo que a soma aos pares resulta em 1/2, conforme previsto na primeira linha de 1. Além disso, para toda medida de S que resulte em *zero*, o valor de R_0 será 00 ou 11 exatamente como prevê a correlação de um estado *GHZ*.

Sub-Rotina	Entrada $\|\psi_i\rangle$	Saída $\|\psi_o\rangle$	Probabilidades
A	$\|R_0, S\rangle = \|00\rangle \|0\rangle$	$\frac{\|ghz_2\rangle\|0\rangle + \|w_2\rangle\|1\rangle}{2}$	$P(0\|\psi_o) = P(1\|\psi_o) = 1/2$
B	$\|R_0\rangle = \|ghz_2\rangle$	$\|w_2\rangle = \frac{\|01\rangle + \|10\rangle}{2}$	$P(01\|\psi_o) = P(10\|\psi_o) = 1/2$

Tabela 1: Entrada e saída desejada das sub-rotinas para 2 partes.

Perceba que todas as probabilidades em 6 são aproximadamente 1/4 de modo que a soma aos pares resulta em 1/2, conforme anteriormente, contudo as correlações são distintas. Agora,

independentemente de S, obtemos, com probabilidade $1/2$, tanto 01 como 10. Isso significa que as duas partes têm a mesma chance de vencer a eleição.

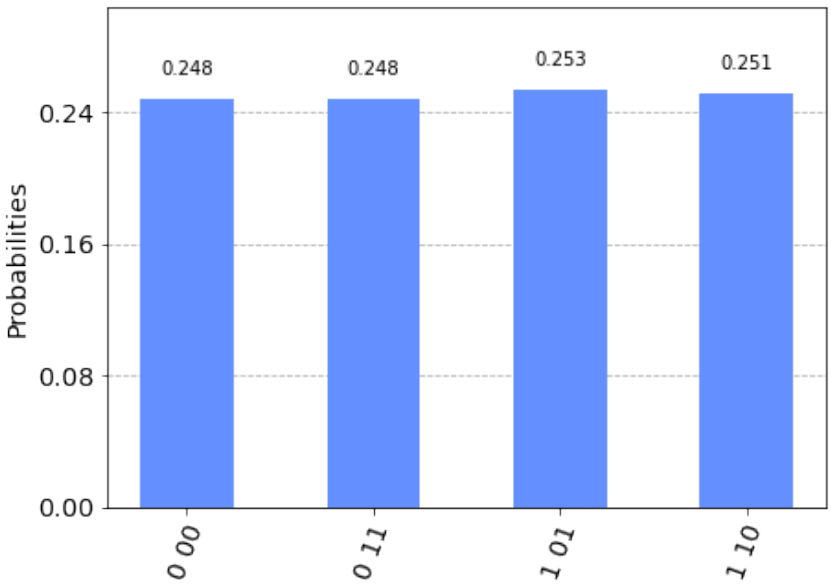

Figura 5: Histograma da Sub-Rotina A para 2 partes. Registradores (de baixo para cima): S e R_0

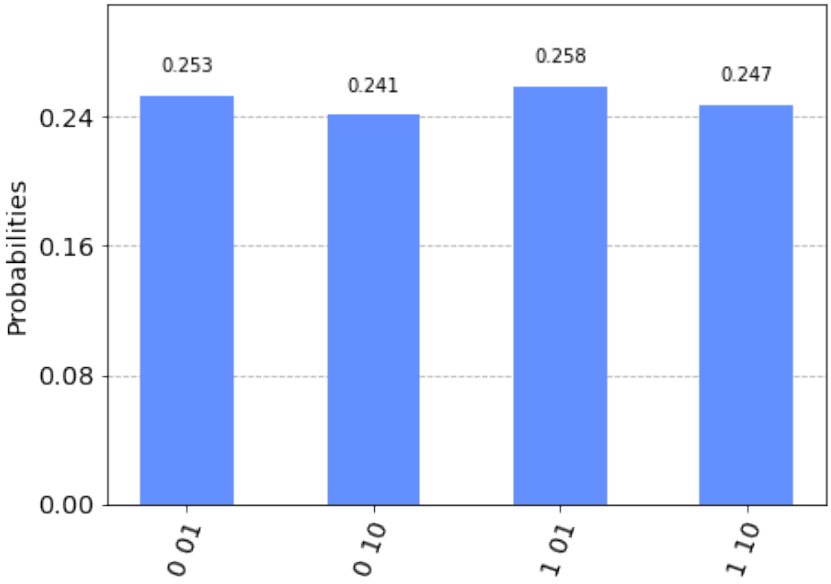

Figura 6: Histograma da Sub-Rotina B para 2 partes. Registradores (de baixo para cima): S e R_0

6.2 Algoritmo para 3 partes

Aqui a análise é análoga à anterior, tendo em mente 2, mas com 3 q-bits e com o registrador R_1 a mais.

Sub-Rotina	Entrada $\|\psi_i\rangle$	Saída $\|\psi_o\rangle$	Probabilidades
A	$\|R_0, S\rangle = \|000\rangle \|0\rangle$	$\frac{\|ghz_3\rangle\|0\rangle + \sum_{i=1}^{6}\|i\rangle\|1\rangle}{2}$	$P(0\|\psi_o) = 2/8$ e $P(1\|\psi_o) = 6/8$
B	$\|R_0, R_1\rangle = \|ghz_3\rangle \|000\rangle$	$\|\phi\rangle$	$P(i\|\psi_o) = 1/18 \times 2/8 = 0,0139$

Tabela 2: Entrada e saída desejada das sub-rotinas para 3 partes.

Veja que todas as probabilidades em 7 são aproximadamente $1/8$, assim a soma do primeiro com o último ($P(0|\psi_o)$) resulta em $2/8$, e a dos seis intermediários ($P(1|\psi_o)$) em $6/8$. Ou seja, as colunas do meio representam $\frac{|ghz_3\rangle|0\rangle}{2}$, enquanto a primeira e última $\frac{\sum_{i=1}^{6}|i\rangle|1\rangle}{2}$.

A leitura de 8 é a seguinte. As 18 primeiras colunas (todas com S resultando em *zero*), correspondem aos casos em que a sub-rotina B foi aplicada; e as demais (que correspondem aos estados intermediários de 8) àqueles em que o estado já era inconsistente (e, portanto, sem a necessidade de aplicá-la). Repare que, para o primeiro conjunto, pelo menos um dos registradores (R_0 ou R_1) é inconsistente e todos com probabilidade são aproximadamente $0,0139\%$. Já para o segundo, R_1 é sempre *zero*, afinal ele não é necessário para decidir a eleição, e suas probabilidades são aproximadamente $1/8$, conforme previsto em 2

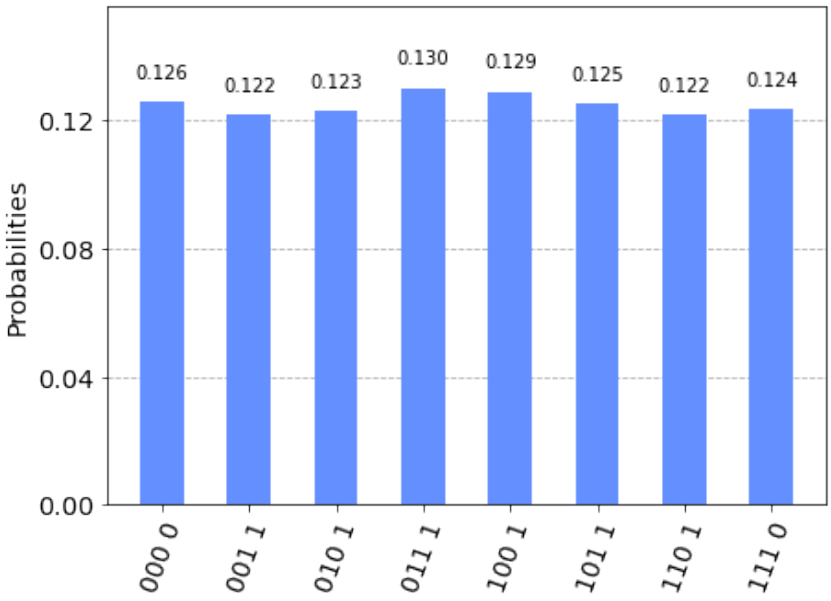

Figura 7: Histograma da Sub-Rotina A para 3 partes.Registradores (de baixo para cima): R_0 e S

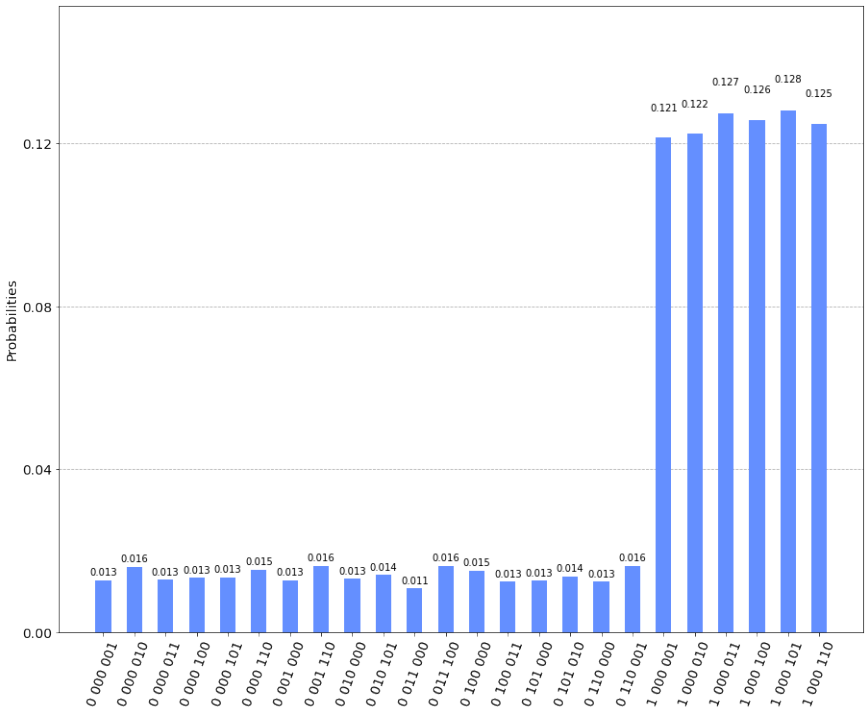

Figura 8: Histograma da Sub-Rotina B para 3 partes.Registradores (de baixo para cima): S,R_1 e R_0

6.3 Algoritmo para 4 partes

Desejamos avaliar agora a *funcionalidade* e *desempenho* do algoritmo como um todo. No que se refere àquele esperamos que a eleição seja *imparcial* — i.e., que todas as partes envolvidas na eleição tenham a mesma chance de vencer — e acerca deste, esperamos que o processo ocorra com o menor número de fases. Para tanto, executamos o algoritmo para uma rede com 2, 3 e 4 partes em dois simuladores — o primeiro exato e o segundo sujeito a erros idênticos àqueles presentes nos computadores quânticos da IBM —, comparando-os com os resultados teóricos, calculados a partir da álgebra dos estados quânticos. Como todos os resultados são médias, executamos as simulações 10000 vezes.

Quanto a funcionalidade, os resultados agrupados em 3 comprovam a imparcialidade da eleição, visto que os resultados simulados estão muito próximos do teórico. Além disso, nota-se o efeito do ruído nos resultados quando comparamos as duas simulações: um leve distúrbio na distribuição de probabilidades.

Parte	Teórico	Simulação	Simulação com ruído
A	1/4	0,2474	0,2712
B	1/4	0,2534	0,2467
C	1/4	0,2439	0,2417
D	1/4	0,2553	0,2404

Tabela 3: Probabilidade de cada parte vencer a eleição de acordo com cada método.

Acerca do desempenho, os resultados tabelados em 4 comprovam a eficiência do algoritmo, pois estão muito próximos do teórico. Além disso, percebe-se a resiliência do algoritmo frente ao ruído o que custa apenas uma leve perda de desempenho.

n	Teórico	Simulação	Simulação com ruído	Clássico
2	1	1,0000	1,1668	1,5
3	3/2	1,5018	1,5644	2,331
4	$46/25 \approx 1,840$	1,8403	2,0097	2,669

Tabela 4: Número médio de fases necessários para concluir a eleição em função de n de acordo com cada método.

7 Conclusão

Este projeto é composto por dois objetivos — (1) implementar o AELQ e (2) desenvolver um simulador de circuitos quânticos para testá-lo — e duas soluções complementares: a primeira, mais abstrata, reside em entender, conceber e se apropriar da lógica e linguagem por trás do algoritmo de [1], usando MQ, para tanto; e a segunda, mais pragmática, consiste na construção do simulador, na qual usamos Python e Quiskit como linguagem e biblioteca, respectivamente. Acerca da validação e teste daquela, compararemos a unanimidade da eleição e o tempo médio de execução do nosso protocolo com os resultados teóricos.

Referências

[1] Seiichiro Tani, Hirotada Kobayashi, and Keiji Matsumoto. Exact quantum algorithms for the leader election problem. *ACM Transactions on Computation Theory (TOCT)*, 4(1):1–24, 2012.

[2] Ellie D'Hondt and Prakash Panangaden. Leader election and distributed consensus with quantum resources. Technical report, 2004.

[3] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018.

[4] Alexandre de Andrade BARBOSA et al. Um simulador simbólico de circuitos quânticos. 2007.

[5] List of qc simulators. `https://quantiki.org/wiki/list-qc-simulators`, 2020.

[6] Gustavo Elálio Cabral. *Uma ferramenta para simulação de circuitos quânticos*. PhD thesis, Universidade Federal de Campina Grande (UFCG), 2004.

[7] Craig Gidney. My quantum circuit simulator. `https://github.com/Strilanc/Quirk`, 2020.

[8] Wagner Jorcuvich Nunes da Silva. *Uma introdução à Computação Quântica*. PhD thesis, Universidade de São Paulo, 2018.

[9] Ryan LaRose. Overview and comparison of gate level quantum software platforms. *Quantum*, 3:130, 2019.

[10] Davy Wybiral. Quantum circuit simulator. `https://github.com/qcsimulator/qcsimulator.github.io`, 2020.

[11] Davy Wybiral. Quantum circuit simulator. `https://qcsimulator.github.io/`, 2020.

[12] Igor Valente Blackman. Simulando algoritmos quânticos em um computador clássico.

[13] Hirotada Kobayashi, Keiji Matsumoto, and Seiichiro Tani. Computing on anonymous quantum network. *arXiv preprint arXiv:1001.5307*, 2010.

[14] Vasil S Denchev and Gopal Pandurangan. Distributed quantum computing: A new frontier in distributed systems or science fiction? *ACM SIGACT News*, 39(3):77–95, 2008.

8 Apêndice

Aqui apresentamos o código completo:

```
from qiskit import *

from qiskit.extensions import *
from qiskit.tools.visualization import *
from qiskit.tools.monitor import job_monitor
from qiskit.providers.aer import noise

from math import *
from sympy import*
import numpy as np
import cmath
import matplotlib.pyplot as plt


#UNIT RIAS
def u(k):
    return np.array( [[1,  cmath.exp(-pi*1j/k)],
                      [-cmath.exp(pi*1j/k), 1]] )/sqrt(2)

def V(k):
    def i(k):
        return im(cmath.exp(pi*1j/k))
    def r(k):
        return re(cmath.exp(pi*1j/k))
    m=np.array( [[1/sqrt(2), 0, sqrt(r(k)) , cmath.exp(pi*1j/k)/sqrt(2)],
                  [1/sqrt(2), 0,-cmath.exp(-pi*1j/3)*sqrt(r(k)),cmath.exp(-pi*1j
    /k)/sqrt(2)],
                 [sqrt(r(k)), 0,cmath.exp(-pi*1j/(2*k))*i(k)/(1j*sqrt(2)*r(2*k))
    ,-sqrt(r(k))],
                 [0,sqrt(r(k)+1),0,0]])

    return m/sqrt(r(k)+1)

def psi(n):
    d=int(pow(2,n)); a=float(sqrt(1/pow(2,n)))
    v=[0]*2*d; v[0]=a; v[int(2*(d-1))]=a;

    for i in list(range(1,d-1)):
        v[2*i+1]=a

    return v


def maximo(x):
    R0 = x[:len(x)//2]
    R1 = x[len(x)//2:]
    z=[0]*len(R1)
    s=[0]*len(R1)

    for i in range(len(R1)):
        if R1[i]==0 and R0[i]==0: z[i]=0
        if R1[i]==0 and R0[i]==1: z[i]=1
        if R1[i]==1 and R0[i]==0: z[i]=2
        if R1[i]==1 and R0[i]==1: z[i]=3

    for i in range(len(z)):
        if z[i]==max(z): s[i]=1
        else: s[i]=0
```

```
    return s

def subB(k): #ok
    subB = QuantumCircuit(2*k,name="SubB")
    vk = UnitaryGate(V(k))
    for i in range(k):
        subB.cx(i,k+i);
        subB.append(vk, [i,k+i])
    #qc.append(subB(k),list(range(1,2*k+1))).c_if(cr,0)
    subB=subB.to_instruction()

    return subB

def alg_impar(k):

    #Definição do tamanho do circuito
    s=[0]; r0=list(range(1,k+1));  r1=list(range(k+1,2*k+1))
    qr =QuantumRegister(2*k+1,'q');cr =ClassicalRegister(2*k+1,'c')
    qc=QuantumCircuit(qr,cr)

    #Sub-Rotina A: inicialização dos registradores com psi(k)
    qc.initialize(psi(k),list(range(k+1)))

    #Medição do Resgistrador S em cS
    qc.measure(s,s)
    #Adição da Sub-Rotina B sobre R0 e R1, se os registradores clássicos
    estiverem setados
    qc.append(subB(k),list(range(1,2*k+1))).c_if(cr,0)

    #Medição dos Registradores R0 e R1, respectivamente
    qc.measure(r0,r0); qc.measure(r1,r1)

    #Compilação e execução do circuito acima construído
    job=qiskit.execute(qc,backend=simu,noise_model=noise_model,shots=1)
    result=job.result()
    counts=result.get_counts(qc)

    #Tratamento do resultado
    lAnswer = [(p[::-1],v) for p,v in counts.items()]
    lAnswer.sort(key = lambda x: x[1], reverse=True)
    Y = []
    for p, v in lAnswer: Y.append( [ int(z) for z in p ] )
    del Y[0][0:1]

    return Y[0]

def alg_par(k):
    s=[0]; r0=list(range(1,k+1));  r1=list(range(k+1,2*k+1))
    qr =QuantumRegister(k+1,'q')
    cr =ClassicalRegister(k+1,'c')
    qc=QuantumCircuit(qr,cr)

    qc.initialize(psi(k),list(range(k+1)))

    qc.measure(s,s)
    uk=UnitaryGate(u(k));
    qc.append(uk,[r0]).c_if(cr,0)

    qc.measure(r0,r0)

    job=qiskit.execute(qc,backend=simu,noise_model=noise_model,shots=1)
    result=job.result()
    counts=result.get_counts(qc)
```

```
    lAnswer = [(p[::-1],v) for p,v in counts.items()]
    lAnswer.sort(key = lambda x: x[1], reverse=True)
    Y = []
    for p, v in lAnswer: Y.append( [ int(z) for z in p ] )

    del Y[0][0:1]
    return Y[0]

def status3(status,y):
    j=0
    for i in range(len(status)):
        if status[i]==1:
            status[i]=y[j]; j=j+1
    return status


def alg(n):
    status=[1]*n

    while status.count(1)>1:
        k=status.count(1)

        if k%2==0 :
            y=alg_par(k)
            while y.count(1)==0:
                y=alg_par(k)
            status=status3(status,y)

        else:
            y=maximo(alg_impar(k))
            while y.count(1)==0:
                y=maximo(alg_impar(k))
            status=status3(status,y)

    if status.count(1)==0:
        return -1
    if status.count(1)==1:
        return status.index(1)

#Para avaliar a performance
def algp(n):
    status=[1]*n
    e=o=0

    while status.count(1)>1:
        k=status.count(1)

        if k%2==0 :
            y=alg_par(k)
            e=e+1
            while y.count(1)==0:
                y=alg_par(k)
                e=e+1
            status=status3(status,y)

        else:
            y=maximo(alg_impar(k))
            o=o+1
            while y.count(1)==0:
                y=maximo(alg_impar(k))
                o=o+1
            status=status3(status,y)
```

```
    if status.count(1)==0:
        return 999
    if status.count(1)==1:
        return o+e
```

Listing 7: Código do algoritmo de eleição para n partes