Projeto-1

Ricardo da Silva Correia

PG47607

Universidade do Minho
Computação Quântica

Algoritmo Quantum Counting

Professor/es:Renato Neves e Luís Barbosa

4/11/2021

1 Introdução:

No trabalho de casa acerca do *algoritmo de Grover* verificou-se que se conseguia determinar a/as solução/ões (caso houvesse) de problemas de satisfatibilidade, porém para saber se é satisfazível basta saber se existe ou não soluções.
Como tal, foi-nos proposto que realizássemos um trabalho acerca do algoritmo *Quantum Counting*, de forma a determinar o número de soluções de problemas de satisfatibilidade.
Neste trabalho irei inicialmente falar acerca dos seguintes algoritmos: *algoritmo de Grover , algoritmo de estimativa de fase, Quantum Fourier Transform*(QFT) e *algoritmo de estimativa dos valores próprios*, pois são peças fundamentais para se elaborar o algoritmo *Quantum Counting*. De seguida, irei determinar o número de soluções para o problema de satisfatibilidade proposto no enunciado, utilizando o algoritmo *Quantum Counting.*
Por fim, será realizada uma conclusão acerca deste trabalho.

2 O algoritmo de Grover:

O algoritmo de Grover é uma versão quântica do algoritmo de busca, que usa o fenómeno do paralelismo quântico para encontrar soluções para um dado problema. É utilizado em problemas do caminho mais curto, criptografia , problemas de satisfatibilidade, entre outros.
Relativamente ao algoritmo de busca clássico, o algoritmo de Grover apresenta uma melhoria quadrática, permitindo uma maior performance em situações mais complexas.

Para elaborar este algoritmo, precisamos de colocar as entradas $|0\rangle^{\otimes}$ em sobreposição, utilizando a gate de Hadamard:

$|\Phi\rangle = |0\rangle^{\otimes} |H\rangle^{\otimes} = \frac{1}{\sqrt{2^n}} \sum_{Y \in \{0,1\}^n}^{2^n-1} |X\rangle = \sqrt{\frac{M}{N}} |A\rangle + \sqrt{\frac{N-M}{N}} |R\rangle$

Na qual M é o número se soluções, N é o número de possibilidades e : $|A\rangle = \frac{1}{\sqrt{M}} \sum_{Xsolution} |X\rangle$ e $|R\rangle = \frac{1}{\sqrt{N-M}} \sum_{Xnosolution} |X\rangle$

Sabemos que precisamos de determinar a/s solução/ões, e para tal este algoritmo aplica um oráculo que nega a amplitude do/s estado/s correspondente/s a/s essa/s solução/ões :
$U_f = (-1)^{f(x)} |X\rangle |-\rangle$
Ou seja se $f(x) = 1$ (é solução) a amplitude do estado é negada e se $f(x) = 0$ (não é solução) a amplitude do estado permanece inalterada. O estado $|-\rangle$ é o estado do Qbit target.

U_f pode também ser representado pela sua forma matricial: $I - 2\left|A\right\rangle\left\langle A\right|$

Negar a/s solução/ões não aumenta a probabilidade de determinar a/s solução/ões, para conseguir aumentar essa/s probabilidade/s face ás restantes é necessário colocar um bloco amplificador.

O amplificador consiste na seguinte forma matricial: G= $2\left|\Phi\right\rangle\left\langle\Phi\right| - I$

De seguida é necessário saber quantas vezes se itera o oráculo+amplificador de forma às soluções terem a maior probabilidade possível face às não soluções, para tal utiliza-se a seguinte fórmula:

(1) Nº de iterações= $\left[\frac{\frac{\pi}{2} - sin^{-1}(\sqrt{\frac{M}{N}})}{2sin^{-1}(\sqrt{\frac{M}{N}})}\right]$ na qual os parênteses rectos indicam o inteiro mais próximo.

3 Algoritmo de estimativa de fase e Quantum Fourier Transform:

3.1 Algoritmo de estimativa de fase

Em muitos algoritmos quânticos a informação está 'codificada' na fase relativa do estado quântico.
Em geral essas fases são complexas, e como tal têm a seguinte expressão: $e^{2\pi iw}$, onde w está contido entre $[0, 1]$.

Para um estado quântico geral temos a seguinte expressão: $\frac{1}{\sqrt{2^n}} \sum_{y \in 2^n} e^{2\pi iwy} \left|y\right\rangle$, na qual $w = 0.x_1x_2...$, sendo que w está escrito na base 2.

Exemplo: '$w = 0.x_1x_2$ para um estado de 2 qubits'

$\frac{1}{\sqrt{2^2}} \sum_{y \in 2^2} e^{2\pi i(0.x_1x_2)y} \left|y\right\rangle = (\frac{\left|0\right\rangle + e^{2\pi i 0.x_2}\left|1\right\rangle}{\sqrt{2}}) \otimes (\frac{\left|0\right\rangle + e^{2\pi i 0.x_1x_2}\left|1\right\rangle}{\sqrt{2}})$

Para se chegar ao resultado anterior, usa-se a seguinte estratégia:

$e^{2\pi iw2^k} = e^{2\pi ix_1,...,x_k.x_{k+1}x_{k+2},...}$
$= e^{2\pi ix_1,...,x_k} e^{0.x_{k+1}x_{k+2},...}$
$= 1 \times e^{0.x_{k+1}x_{k+2},...}$ (pois o primeiro termo dá sempre 1, seja qual for o inteiro)
$= e^{0.x_{k+1}x_{k+2},...}$

Onde o 2^k desta estratégia é o y no expoente do exemplo. Por exemplo, se o y for o estado $|10\rangle$ então $2^k = 2$, o que leva a um shift para a esquerda e utilizando a estratégia acima obtém-se a fase relativa do estado $|10\rangle$.

Para se estimar o valor aproximado de w, que é o objectivo do algoritmo de estimativa de fase, então tem de se aplicar uma gate de Hadamart no primeiro qubit e já se sabe o valor x_2. Para saber o valor do x_1 utiliza-se o seguinte operador (designado operador *phase rotator* inverso com $k = 2$) aplicado ao segundo qubit, caso o x_2 seja 1:

$$R_2^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{-2\pi i}{4}} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & e^{-2\pi i(0.01)} \end{bmatrix}$$

Pois:

$$R_2^{-1}(\tfrac{|0\rangle + e^{2\pi i 0.x_1 1}|1\rangle}{\sqrt{2}}) = (\tfrac{|0\rangle + e^{2\pi i 0.x_1}|1\rangle}{\sqrt{2}})$$

Podendo assim obter o valor x_1 aplicando uma gate de Hadamart no segundo qubit.

O circuito completo para este exemplo é seguinte:

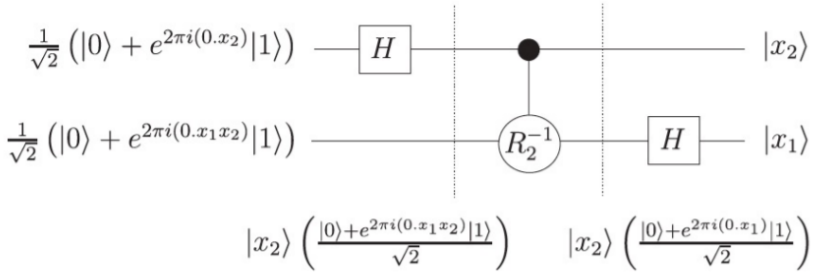

Figura 1: Cicuito para extrair o w da fase para 2 qubits

Aplicando um raciocínio análogo, verifica-se que para determinar o $w = 0.x_1x_2x_3$ para um circuito de 3 qubits temos o seguinte circuito:

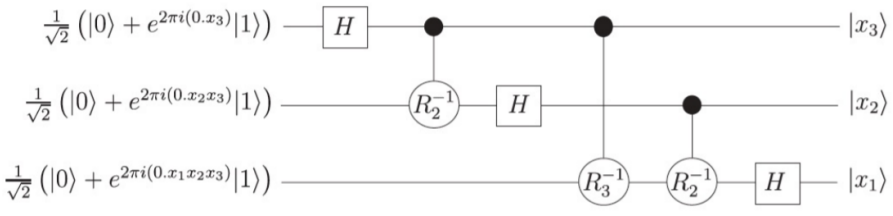

Figura 2: Cicuito para extrair o w da fase para 3 qubits

Onde o *phase rotator* inverso para k geral tem a seguinte forma matricial: $R_k^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{-2\pi i}{2^k}} \end{bmatrix}$

Sendo assim, o *algoritmo de estimativa de fase* recebe como input: $\frac{1}{\sqrt{2^n}} \sum_{y \in 2^n} e^{2\pi i (0.x_1,...,x_n)y} |y\rangle$
E dá como output: $|x_n, ..., x_1\rangle$ (os qubits ficam na ordem inversa).

A inversa do *algoritmo de estimativa de fase* recebe como input:$|x_n, ..., x_1\rangle$
E dá como output: $\frac{1}{\sqrt{2^n}} \sum_{y \in 2^n} e^{2\pi i (0.x_1,...,x_n)y} |y\rangle$

3.2 Algoritmo Quantum Fourier Transform

Relativamente ao *algoritmo Quantum Fourier Transform*, ele é praticamente idêntico á inversa do *algoritmo de estimativa de fase*:

$QFT_k(|x\rangle) = \frac{1}{\sqrt{k}} \sum_0^{k-1} e^{2\pi i \frac{x}{k} y}$

Sendo o circuito:

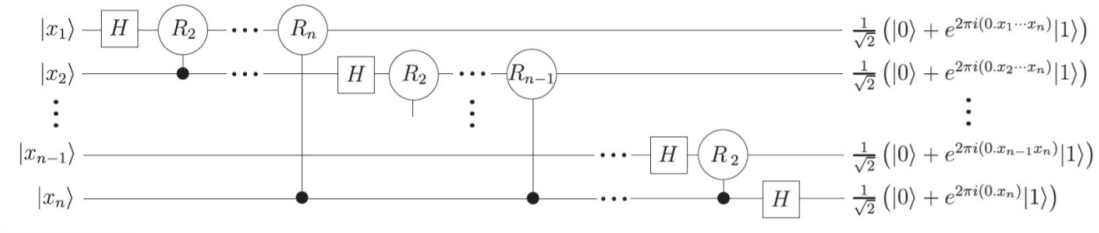

Figura 3: Circuito do algoritmo QFT

Quanto á inversa do *algoritmo Quantum Fourier Transform*, a construção do seu circuito para 2 e 3 qubits, por exemplo, é idêntica á da Figura 1 e Figura 2, respetivamente. Ou seja, este algoritmo é semelhante ao *algoritmo de estimativa de fase*.

Este algoritmo necessita do seguinte número de gates: $n+(n-1)+(n-2)+...+1+3*\frac{n}{2} = \frac{n^2+2n}{2} \approx O(n^2)$

Sendo que a 'melhor' versão clássica necessita de aproximadamente $O(n2^n)$.
Ou seja, esta versão quântica requer exponencialmente menos operações do que a transformada de Fourier clássica mais rápida.

4 Algoritmo de estimativa dos valores próprios:

Suponha que um operador U tem um vetor próprio $|u\rangle$ e o respetivo valor próprio $e^{2\pi i\varphi}$, o objetivo do *algoritmo de estimativa dos valores próprios* é determinar o valor φ.

Para tal é necessário preparar 2 registos. O primeiro registo é composto por t qubits iniciados no estado $|0\rangle$, onde a escolha de t depende do número de bits de precisão (n) que pretendemos ter na estimativa de φ e da probabilidade $(1-\epsilon)$ do *algoritmo de estimativa de valores próprios* ter êxito: $t = n + \lceil log(2 + \frac{1}{2\epsilon}) \rceil$.
O segundo registo é composto pelo estado $|u\rangle$, e contém o número de qubits necessário para criar o $|u\rangle$.

De seguida é necessário aplicar uma gate de Hadamart nos qubits do primeiro registo, aplicando posteriormente uma gate controlada $cU^{2^{t-j}}$ no segundo registo, na qual j é a posição do qubit pertencente a t. O circuito correspondente a esta etapa é o seguinte:

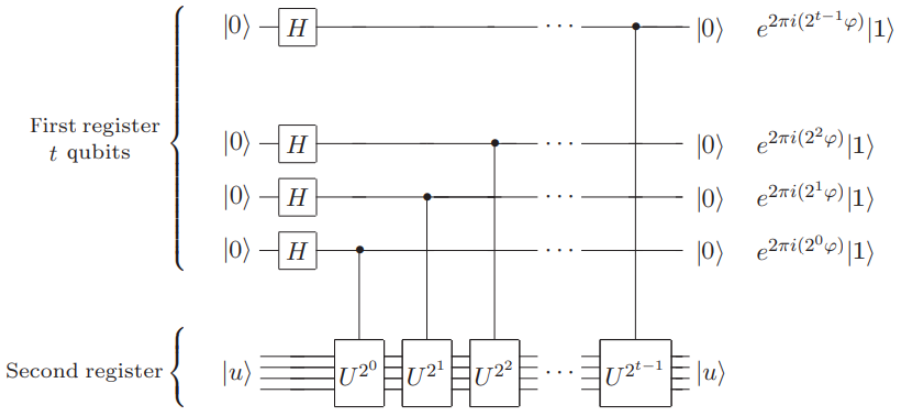

Figura 4: Circuito da primeira etapa do algoritmo de estimativa aos valores próprios. No lado direito foi omitido um sinal '+' entre os estados e ainda o fator de normalização $\frac{1}{\sqrt{2}}$

Podemos ver que aplicar a gate cU ao estado $|+\rangle|u\rangle$ dá o seguinte resultado:
$cU(|+\rangle|u\rangle) = \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i\varphi}|1\rangle)|u\rangle$ (de notar que $cU(|0\rangle|u\rangle) = |0\rangle|u\rangle$ e $cU(|1\rangle|u\rangle) = |1\rangle U|u\rangle = e^{2\pi i\varphi}|1\rangle|u\rangle$)

E por consequência, aplicar $cU^{2^{t-j}}$ ao estado $|+\rangle|u\rangle$ dá o resultado:
$cU^{2^{t-j}}(|+\rangle|u\rangle) = \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i 2^{t-j}\varphi}|1\rangle)|u\rangle$, estando por isso concordante com a saída do circuito da Figura 4.

A segunda etapa consiste em aplicar a QFT inversa no primeiro registo e a terceira e última etapa consiste em medir os qubits do primeiro registo na base computacional,

tendo assim um valor aproximado de φ*2^t, pois da QFT obtém-se o φ deslocado t casas para a esquerda.

Nota: Se quisessemos saber o valor de φ em vez de φ*2^t, bastava dividir o resultado à saída do circuito por 2^t.

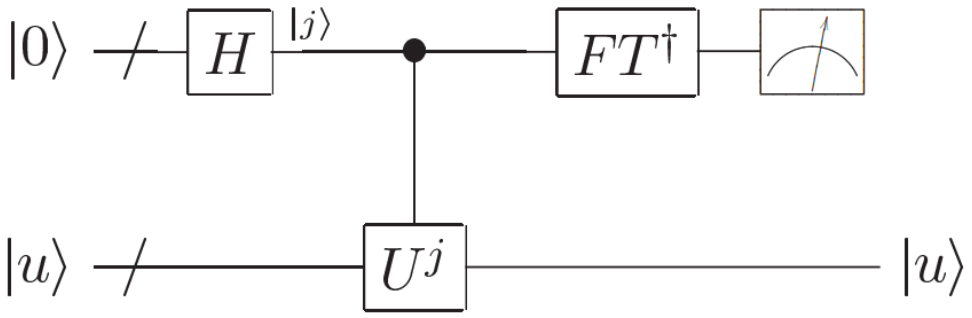

Figura 5: Circuito completo do algoritmo de estimativa dos valores próprios

Este algoritmo tem algumas considerações importantes a ter em conta.

Uma delas é que ele permite obter um valor aproximado (ou exato) de φ como tenho vindo a referir ao longo deste trabalho, sendo que digo isto pois φ pode ser totalmente definido em t bits binários, o que leva a este algoritmo a obter o valor exato de φ. Porém, φ pode não ser definido por t bits binários, sendo que devido a isso podemos ter resultados aproximados ou bastante fracos relativamente a estimativa desse valor. Para evitar resultados fracos é necessário escolher um número t de qubits adequados de forma ao resultado final ser exato ou pelo menos aproximado, como tal tem de se verificar os bits de precisão que se pretende obter á saída do circuito (n) de forma ao resultado ser aproximado, e também tem de se estabelecer uma probabilidade de êxito elevada do algoritmo que se pretende (1-ϵ), calculando assim o número de t necessário: $t = n + \lceil log(2 + \frac{1}{2\epsilon}) \rceil$.

Outra consideração importante é que para preparar este algoritmo é necessário que os estados do segundo registo sejam vetores próprios do operador U, mas e se não soubermos como preparar esse vetores próprios? Então basta preparar o segundo registo como uma expansão de $|u\rangle$, ou seja, $|\phi\rangle = \sum_u c_u |u\rangle$.
Assim, o resultado deste algoritmo é $\sum_u |\varphi_{aprox}\rangle\, c_u |u\rangle$, sendo que á saída vai-se obter o valor aproximado de φ com uma probabilidade $|c_u|^2$.
Este procedimento permite-nos preparar os vetores próprios de U á qual desconhecemos, com o custo de se introduzir alguma aleatoriedade adicional no algoritmo.

5 Algoritmo Quantum Counting:

Este algoritmo permite saber o número de soluções M em N combinações num problema de busca, sendo que este algoritmo quântico é bem mais rápido que a sua versão clássica.
Além disso, este algoritmo tem interesse quando aplicado em conjunto com algoritmos de busca, como por exemplo o Grover, pois uma das dificuldades de se implementar o Grover é saber o número de soluções M, e com a aplicação deste algoritmo podemos determinar esse valor e assim já se sabe quantas iterações oráculo + amplificador se deve fazer para achar as soluções.

Para se elaborar este algoritmo, precisamos apenas de combinar o *algoritmo de Grover* com o *algoritmo de estimativa dos valores próprios* que abordamos atrás com algum detalhe. No fundo, o operador U que se falou no *algoritmo de estimativa dos valores próprios* é o iterador de Grover (iterações do oráculo + amplificador necessárias para um dado problema) á qual chamaremos G, assim o que se obtem á saída do circuito do *algoritmo Quantum Counting* é uma estimativa dos valores próprios de G, á qual depois se pode realizar alguns cálculos para determinar o número de soluções do problema.

Para se perceber melhor porque os valores próprios de G podem nos dar o número de soluções, faremos a seguinte análise:
O que o iterador de Grover faz é uma rotação do estado $|\psi\rangle$ por um ângulo θ na base $|\beta\rangle, |\alpha\rangle$, onde a percentagem de soluções afeta a distância entre $|\psi\rangle$ e $|\alpha\rangle$, ou seja, se existe muitas soluções o θ é grande, mas se a percentagem for pequena o θ é pequeno.

$$G = \begin{pmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{pmatrix}$$

Acontece que os valores próprios de G são $e^{i\theta}$ e $e^{i(2\pi-\theta)}$ e se extrairmos o valor de θ conseguimos determinar o valor de M, pois como vimos eles estão relacionados.

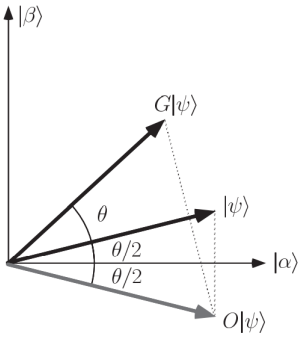

Figura 6: Visualização geométrica do que o iterador de Grover faz

Para se elaborar este circuito, é necessário criar os dois registos, tal como se fez no *algoritmo de estimativa dos valores próprios*, onde o primeiro registo tem de ter um número adequado de qubits, de forma a ter uma precisão satisfatória à saída do circuito (n), com uma probabilidade de êxito alta ($1-\epsilon$), ou seja o número de qubits t tem de satisfazer a seguinte fórmula: $t = n + \lceil log(2 + \frac{1}{2\epsilon}) \rceil$.

Quanto ao segundo registo, tem de se colocar esses qubits tal como se fez no trabalho de casa do *algoritmo de Grover*, ou seja temos de criar os qubits de entrada á qual pretendemos estimar as soluções, o qubit target e as ancilhas que forem precisas.

O resto do circuito é exatamente igual ao da Figura 5, apenas U passa a ser G.

Como output vamos obter o $\varphi * 2^t$ que vimos no capítulo 4, porém o que interessa é determinar o θ. Sendo assim, $\theta = \frac{\varphi}{2^t} * 2\pi$, pois os valores próprios são: $e^{i\theta}$ e $e^{i(2\pi-\theta)}$.

Agora para determinar o M temos de realizar o seguinte procedimento:

Na Figura 6, verifica-se que se faz um ângulo $\theta/2$ entre o estado $|\psi\rangle$ e o estado $|\alpha\rangle$.
Como vimos no capítulo 2, o estado $|\psi\rangle$ é uma sobreposição dos estados de base de G: $|\psi\rangle = \sqrt{\frac{M'}{N}}\,|\beta\rangle + \sqrt{\frac{M'-N}{N}}\,|\alpha\rangle$.
E o produto interno entre $|\psi\rangle$ e $|\alpha\rangle$ é igual: $\langle\alpha|\psi\rangle = cos(\theta/2) = \sqrt{\frac{M'-N}{N}}$.
Manipulando a expressão anterior com alguma álgebra e alguma trigonometria obtêm-se: $Nsin^2(\theta/2) = M'$.
Porém este resultado M' não é o número de soluções, mas sim o número de não soluções. Pode parecer estranho, porém o amplificador no *algoritmo de Grover* introduz uma fase de global de π, o que no *algoritmo de Grover* não faz diferença, porém neste algoritmo faz e leva-nos a obter o resultado das não soluções ($e^{i\theta}$ é um valor próprio das soluções, mas $e^{i(\theta+\pi)}$ é um valor próprio das não soluções e devido á fase do amplificador temos à saída o valor que nos permite calcular $\theta + \pi$ e não só θ).
No entanto, resolver este 'problema' de o M' ser as não soluções é extremamente simples, basta fazer: $M = N - M'$ e temos o número de soluções como pretendido.

6 Algoritmo Quantum Counting em problemas de satisfatibilidade:

Como abordado na introdução, este trabalho exigia que se aplicasse o algoritmo de Quantum Counting para determinar o número de soluções de problemas de satisfatibilidade.

Como tal irei tentar aplicar este algoritmo ao seguinte problema de satisfatibilidade: A $\wedge$ ($\neg$B $\vee$ C) $\wedge$ D, onde o D não altera o número de soluções, apenas aumenta o número de possibilidades de forma a funcionar melhor no *algoritmo de Grover*, tal como se viu nesse TPC. Esta fórmula booleana tem 3 soluções em 16 resultados possíveis e determinaram-se as soluções através do seguinte circuito (o circuito apresentado é o que utilizei no TPC anterior, porém agora tive em atenção a simetria do oráculo):

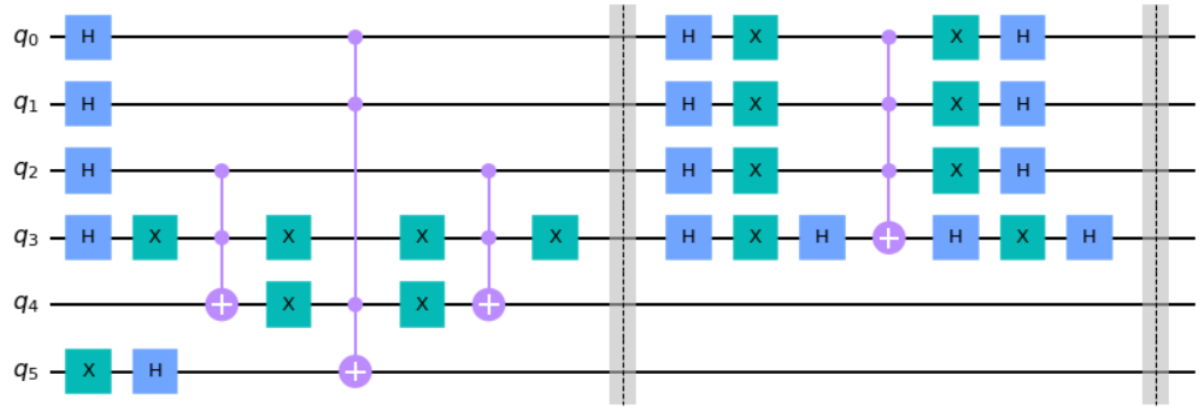

Figura 7: Circuito do algoritmo de Grover para a fórmula booleana em questão

Como se referiu no Capítulo 5, precisamos então do iterador de Grover, que neste caso é simplesmente o oráculo e o amplificador da figura acima.

Tendo o G criado, é necessário convertê-lo para um cG, ou seja uma gate controlada. Para tal criou-se uma função designada 'grover_circuit', que retornava o iterador de Grover G.
De seguida, converteu-se essa função para uma gate controlada (ativa quando o estado é $|1\rangle$), através das seguintes instruções em Qiskit:

```
grover = grover_circuit().to_gate()
grover.label = "Grover"
cgrover = grover.control(1)
```

Depois criou-se uma função capaz de retornar o circuito da QFT inversa, esta função recebia como argumento o número de qubits do primeiro registo:

```
def qft_dagger(n):



    circuit = QuantumCircuit(n)
```

```
    def qft_dagger_rotations(circuit,n):

        #circuit.h(0)


        for j in range(0,n,1):

            for i in range(0,j,1):
                circuit.cp(-(np.pi)/2**(j-i),i, j)
            circuit.h(j)


    qft_dagger_rotations(circuit,n)

    return circuit
```

De seguida, criou-se uma gate através da função 'qft_dagger':

```
qft_dagger=qft_dagger(4).to_gate()
qft_dagger.label = "QFT†"
```

Desta forma, podemos utilizar estas duas gates na montagem do circuito do *algoritmo Quantum Counting*, ficando com um aspeto melhor e mais intuitivo.

Antes de se começar a montar o circuito, precisamos de saber o número qubits (t) que o primeiro registo irá ter, para tal verificou-se o número de bits de precisão que iria ter à saída (n), com uma dada probabilidade de êxito ($1-\epsilon$).
Imaginemos então que se decidia utilizar 4 qubits no primeiro registo, $t = 4$, e se esperava obter 90% de probabilidade de êxito, fazendo as contas:
$t = n + \lceil log(2 + \frac{1}{2\epsilon}) \rceil$
$\Leftrightarrow n = t - \lceil log(2 + \frac{1}{2\epsilon}) \rceil$
Substituindo $t = 4$ e $\epsilon = 0.1$:
$\Leftrightarrow n = 4 - \lceil log(2 + \frac{1}{2*0.1}) \rceil = 1$
Ou seja, tenho 90% de chance de conseguir ter 1 bits de precisão à saída do circuito, o que é uma precisão bastante baixa.

Mas e se $t = 10$?
Então $n = 7$, sendo esta precisão muito elevada, e no fundo não era preciso assim tão elevada. Como tal, viu-se quantos bits de precisão se tem para $t = 10$ e $\epsilon = 0.01$:
$n = 10 - \lceil log(2 + \frac{1}{2*0.01}) \rceil = 4$
Ou seja, 99% de chance de obter uma precisão de 4 bits à saída do circuito para $t = 10$, o que já é bastante bom, e por isso utilizar 10 qubits neste registo é suficiente. Também

se pode verificar que para um dado t, existem várias combinações de n e ϵ, sendo que quanto maior for a probabilidade sucesso, menor vai ser o número de bits de precisão à saída do circuito. Também se verifica que quanto maior a complexidade do circuito, ou seja quanto maior for o número de qubits do primeiro registo (t) melhor será a relação entre a probabilidade de sucesso e o número de bits de precisão à saída do circuito.

Tendo já definido o número de bits do primeiro registo, criado a gate controlada do iterador de Grover, e a gate do QFT inverso, foi só implementar um algoritmo em Qiskit idêntico ao da Figura 5, onde U é o iterador de Grover desta fórmula booleana e $|u\rangle$ são os estados q_0, q_1, q_2, q_3, q_4 e q_5 da Figura 7 antes de aplicar o oráculo e o amplificador.

Não se colocou a imagem do circuito pois para $t = 10$ e para 6 qubits no segundo registo, o circuito era muito grande e o Qiskit nem sequer permitia fazer o 'draw()' dele.
Porém, os resultados das medidas foram os seguintes:

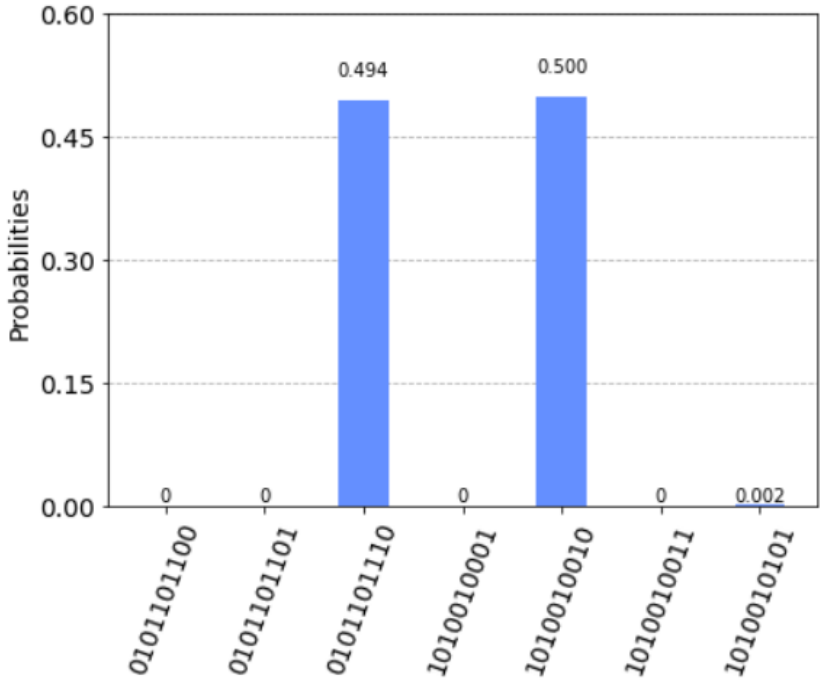

Figura 8: Resultados das medições para o circuito Quantum Counting da fórmula booleana em questão

Nota: O resultado da fase à saída do circuito é invertido tal como se viu no capítulo 3.1, porém a forma como coloco os medidores faz com que inverta novamente, dando por isso o resultado esperado. Pode-se ver o circuito implementado no notebook que irei disponibilizar no envio deste trabalho.

Verifica-se na figura 8 que temos praticamente 50% de probabilidade de obter o valor 0101101110 e 1010010010, que em decimal são 366 e 658, respectivamente. Um destes

valores está associado ao θ e o outro ao $2\pi - \theta$.

Agora é só realizar os cálculos que se realizou no capítulo 5 (irei utilizar o 658 pois tem maior probabilidade de acordo com a figura 8):
$\theta = \frac{658}{2^{10}} 2 * \pi \approx 4.037$
$M' = sin^2(\theta/2) * N = sin^2(4.037/2) * 16 \approx 13$
$M = N - M' = 16 - 13 = 3$

Ou seja o número de soluções é 3, que é o resultado que se esperava para a fórmula booleana pedida no trabalho.

Caso se mudasse para outra fórmula booleana, era seguir os mesmos procedimentos que fizemos para esta fórmula, ou seja, tem de se criar um iterador de Grover adaptado á fórmula boolena em questão, calcular o número de qubits no primeiro registo, criar a gate do QFT inverso, criar a gate controlada do iterador de Grover, montar tudo, medir e realizar os cálculos anteriores.

7 Conclusão:

Com a realização deste trabalho, fiquei a compreender melhor estes 5 algoritmos tão frequentemente usados na computação quântica, que são nomeadamente o *algoritmo de Grover*,o *algoritmo de estimativa de fase*, o *algoritmo QFT*, o *algoritmo de estimativa dos valores próprios* e o *algoritmo Quantum Counting.*
Também compreendi os procedimentos que se utiliza para utilizar o *algoritmo Quantum Counting*, no que toca a resolver problemas de satisfatibilidade.

8 Bibliografia:

[1] Michael A. Nielsen & Isaac L. Chuang. "Quantum Computation and Quantum Information".

[2] Slides das aulas

[3] Quantum Counting-Qiskit. `https://qiskit.org/textbook/ch-algorithms/quantum-counting.html`