Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Iniciación a la Computación Cuántica. Algoritmo de Dijkstra Multiobjetivo Cuántico.

An introduction to Quantum Computation. Quantum Multiobjective Dijkstra Algorithm.

La Laguna, 8 de septiembre de 2021

D. **Antonio Alberto Sedeño Noda**, con N.I.F. 45.439.024-V Catedrático de Universidad adscrito al Departamento de Matemáticas, Estadística e Investigación Operativa de la Universidad de La Laguna, como tutor

C E R T I F I C A (N)

Que la presente memoria titulada:

"Iniciación a la Computación Cuántica. Algoritmo de Dijkstra Multiobjetivo Cuántico."

ha sido realizada bajo su dirección por D. **Himar Manuel Barquín Carrasco**,

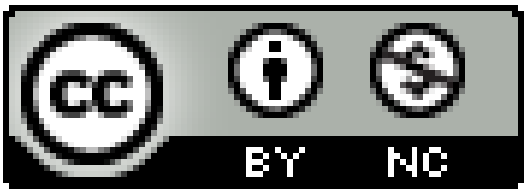
con N.I.F. 46.260.297-Y.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 8 de septiembre de 2021.

Agradecimientos

A mis padres, por apoyarme en todo momento en mis decisiones.
A Carlos, Danielys, Víctor y César por estar siempre ahí cuando lo he necesitado.
A mi tutor de TFG, Antonio, por haber sido paciente y comprensivo pese a todos los problemas.

Licencia



Resumen

En este Trabajo de Fin de Grado se estudia la posible disminución en el tiempo de computación que originaría el desarrollo de un algoritmo de búsqueda similar al dado por el algoritmo de Grover [1] para realizar el test de dominancia en los algoritmos de Optimización Combinatoria Multiobjetivo.

Para ello, en primer lugar, se hará una introducción tanto al estado actual de la Computación Cuántica como a los conceptos de este paradigma necesarios para comprender el algoritmo de Grover. Tras esta introducción, se procede a hablar del propio algoritmo de Grover, se realiza una demostración empírica de este algoritmo y se finaliza explicando posibles vías futuras para continuar este desarrollo.

Palabras clave: Computación cuántica, Algoritmo de Grover, cúbit, Qiskit, IBM Quantum Experience (IBMQ).

Abstract

This end of degree project aims to perform a similar Grover [1] search algorithm to accelerate the computational time of the Combinatorial Multiobjective Optimization dominance test.

To achieve this, an introduction to Quantum Computation actual state and concepts are shown first, followed by Grover algorithm explanation and an empirical demonstration of this algorithm. This document finishes with possible future ways to continue with this development.

Keywords: Quantum Computation, Grover algorithm, Qubit, Qiskit, IBM Quantum Experience (IBMQ).

Índice general

Índice de figuras

Capítulo 1 Introducción

1.1 La importancia de la Computación Cuántica

A principios de siglo XX, la comunidad científica enfrentaba problemas, como podían ser la radiación electromagnética de un cuerpo negro o la inestabilidad de los átomos, que no podían explicarse mediante la mecánica clásica. Es por ello que en la década de los años 20 surgió un nuevo modelo, que era capaz de explicar estos conceptos: la mecánica cuántica.

Este nuevo modelo permitió el surgimiento de la computación cuántica, un nuevo paradigma de programación basado un nuevo sistema de codificación que aprovecha las propiedades de la mecánica cuántica para no solo poder representar el estado 0 o 1 sino también ambos de forma simultánea.

Gracias a estas propiedades de naturaleza cuántica, se pueden resolver problemas que con la computación clásica se han considerado computacionalmente inviables. Para un algoritmo que en computación clásica podría tardar un millón de años, en computación cuántica puede tardar al orden de días o incluso horas. Ejemplos claros de esto son algoritmos como el de Shor [2], que consigue factorizar un número primo en tiempo polinomial, o el algoritmo de Grover [1], que reduce el tiempo de los algoritmos de búsqueda a $O(\sqrt{N})$, siendo N el tamaño del conjunto de búsqueda.

Tanto es así que se conoce como *supremacía cuántica* [3] a la idea de que los ordenadores cuánticos pueden ejecutar tareas computacionales exponencialmente más rápido que un ordenador clásico. El algoritmo de Shor [2] es un ejemplo de tal superioridad.

Sin embargo, actualmente no se dispone de la tecnología necesaria para que resulte eficiente trabajar con este nuevo paradigma de forma autónoma y resolver con él los problemas actuales. De hecho, se utilizan algoritmos híbridos, que combinan computación clásica con computación cuántica, para poder aprovecharnos de las cualidades de este último y reducir a su vez la complejidad subyacente de trabajar con propiedades de la mecánica cuántica.

1.2 Estado del arte

La computación cuántica surgió de la mente de Paul Benioff en el año 1981, imaginando un computador que fuera capaz de aprovechar algunos principios de la mecánica cuántica [4]. Tras él se unieron más científicos, como Richard Feynman o Dan Simon, que ayudaron a desarrollar esa idea sobre la posibilidad de crear un ordenador capaz de realizar cálculos computacionalmente más rápido gracias a la cuántica. Desde entonces ha habido muchos avances en este campo. Han surgido algoritmos de mucha relevancia, se han llegado a desarrollar computadores cuánticos reales y se ha puesto a prueba lo que este paradigma puede ofrecer al mundo.

Sin embargo, todavía estamos lejos de poder utilizar un ordenador cuántico para resolver problemas de la vida real. Los computadores que existen actualmente, además de ser escasos en los cúbits que tienen disponibles, se enfrentan a otros problemas inherentes a la mecánica cuántica. Algunos de estos problemas son la decoherencia cuántica [5], el ruido, la calidad de los

cúbits, la complejidad del hardware o el teorema de no clonado [6].

Actualmente existe empresas como IBM o AMD que tienen ramas dedicadas al desarrollo e investigación de nuevas tecnologías y algoritmos para la computación cuántica. Ejemplo de estos trabajos son el ordenador *Quantum System One*, considerado el más potente de Europa, que utiliza el procesador *Quantum Falcon* de 27 cúbits, o la patente registrada por AMD, que pretende usar módulos de teleportación cuántica para desarrollar un procesador capaz de realizar ejecuciones fuera de orden [7].

1.3 Objetivos

El objetivo de este trabajo es estudiar la posible disminución en el tiempo de computación que originaría el desarrollo de un algoritmo de búsqueda similar al dado por el algoritmo de Grover [1] para realizar el test de dominancia en los algoritmos de Optimización Combinatoria Multiobjetivo.

Para lograr este objetivo se debe partir del estudio de los conceptos más básicos de la computación cuántica. Esto es, qué es un cúbit y cómo interactuar con ellos, las puertas cuánticas más utilizadas y el algoritmo de Grover y su funcionamiento.

Posteriormente se realizan pruebas empíricas con diferentes detectores para comprobar la efectividad de Grover tanto en una simulación como en un ordenador cuántico real. Se expondrán los resultados y se aprovecharán estos circuitos para ejemplificar algunos problemas inherentes al algoritmo cuántico de búsqueda.

Para la realización de las pruebas utilizaremos el *Software Development Kit* (SDK) para Python llamado Qiskit [8], además de las herramientas que proporciona *IBM Quantum Experience* [9] como son *IBM Quantum Composer* y *IBM Quantum Lab*.

Finalmente, expondremos la idea de la utilización de algoritmos similares al de Grover para realizar el test de dominancia sobre un conjunto vectorial de N elementos.

Capítulo 2 Conceptos

2.1 Cúbits

2.1.1 Definición de cúbit

En computación clásica, los bits representan la unidad mínima de información. Son capaces de representar el estado cero y el estado uno, y es lo que hasta ahora nos ha permitido avanzar tecnológicamente de forma significativa. Para codificar esta información, se puede utilizar diversas herramientas, siendo la más extendida el uso de la electricidad para reflejar el uno, como traspaso de corriente, y el cero como ausencia de esta.

En computación cuántica, un cúbit [10] representa la unidad mínima de codificación, similar a lo que un bit representa en la computación clásica. No obstante, es capaz no solo de asumir el estado cero o uno (denominados *estados básicos*) si no también una combinación de estos.

Es decir, no solo se puede tener un cúbit en el estado cero, sino también un cúbit que está conformado parte por el estado cero y parte por el estado uno. Esto es lo que se conoce como *estado de superposición*. Sin embargo, no se podrá observar esa combinación. Si se intenta medir, el cúbit colapsará a uno de los dos *estados básicos*.

La verdadera potencia de este paradigma radica justo en el concepto de estado de superposición. Gracias al uso de las *puertas cuánticas* (véase apartado 2.2) se pueden manipular estos estados sin necesidad de medición, lo que permite aprovecharse de esta característica.

Por ejemplo, se pueden tener los siguientes cúbits:

$$q1 = |0\rangle$$
$$q2 = |1\rangle$$
$$q3 = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$$

El primer y el segundo cúbit son estados básicos, por lo que no hay duda qué estados representan. Por otro lado, el cúbit $q3$ está en estado de superposición. Como se puede observar, ambos estados básicos tienen un valor, denominado *amplitud*, que nos permite calcular la probabilidad de que $q3$ colapse a cero o a uno. La fórmula para calcular dicha probabilidad es la siguiente:

$$Si\ |\phi\rangle = c_1|0\rangle + c_2|1\rangle, entonces$$
$$p(|0\rangle) = c_1^2$$
$$p(|1\rangle) = c_2^2$$

Aplicando dicha fórmula sabemos que la probabilidad de que colapse a cero es $\left(\frac{1}{\sqrt{2}}\right)^2 = 0.5$. El cúbit $q3$ tiene una probabilidad del 50% de colapsar al estado básico cero y un 50% de colapsar al estado básico uno.

Como es evidente, la suma de las probabilidades de los estados básicos debe ser uno:

$$\sum_{i\in\{1,2\}} c_i^2 = 1$$

2.1.2 Representación de los cúbits

En el campo de las matemáticas, los cúbits son vectores formados por una combinación lineal de los componentes de una base en el espacio de Hilbert [11].

$$c_1|\alpha_1\rangle + c_2|\alpha_2\rangle + \cdots + c_n|\alpha_n\rangle$$

Dicho de otro modo, los estados básicos $\alpha_1, \alpha_2, \ldots, \alpha_n$ forman una base en el espacio de Hilbert, y una combinación de estos forman todos los posibles estados con n cúbits.

Para representar un cúbit tenemos varias formas. Se pueden representar como matrices. Por ejemplo, $\binom{0}{1}$ es un cúbit que siempre valdrá el estado básico uno. Para el caso del estado cero, se escribirá como $\binom{1}{0}$. El estado

$$\begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} = \frac{1}{\sqrt{2}}\binom{1}{1} = \frac{1}{\sqrt{2}}\binom{1}{0} + \frac{1}{\sqrt{2}}\binom{0}{1}$$

es el cúbit $q3$ del apartado anterior en estado de superposición.

Otras formas, más típicas para representar cúbits, es la notación *Bra-Ket* o notación de Dirac y la esfera de Bloch [12].

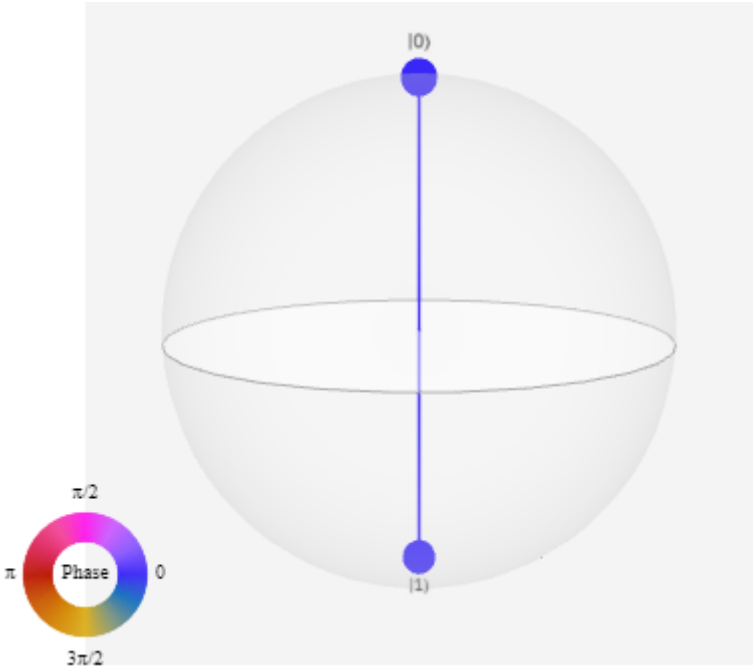


Figura 1. Esfera de Bloch.

Un ejemplo de una esfera de Bloch es la mostrada en la *Figura 1*. En dicha figura se puede observar el mismo estado en superposición $q3$.

La notación de Dirac ya ha sido usada en el apartado 2.1.1 . Se representa el cúbit de la forma

$$|\phi\rangle = c_1|0\rangle + c_2|1\rangle$$

donde c_1 y c_2 son las amplitudes de los estados básicos $|0\rangle$ y $|1\rangle$.

También podemos representar varios cúbits de forma conjunta, en caso de necesitar más de uno. Por ejemplo, podemos representar hasta cuatro estados básicos si se usan dos cúbits. El estado

$$|\phi^+\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

que en forma matricial se describe como

$$\begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \frac{1}{\sqrt{2}}\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

es un estado en superposición con un 50% de probabilidad de colapsar en el estado básico $|00\rangle$ y un 50% de colapsar en $|11\rangle$. Este cúbit es imposible que colapse en $|01\rangle$ o en $|10\rangle$, ya que la probabilidad para ambos casos es 0% por lo dicho anteriormente: la suma de las probabilidades de cada estado básico de un cúbit debe ser siempre uno. Este estado de superposición se conoce como uno de los pares EPR o estados de Bell *[13]*, e introduce uno de los principios de la cuántica más importantes: el entrelazamiento cuántico *[14]*.

Este fenómeno se describe como la característica que poseen algunos cúbits en estado de superposición para, midiendo solo uno de ellos, colapsar a la vez al otro. En el caso del ket $|\phi^+\rangle$, se puede observar que, si cogemos el primer cúbit y lo medimos, también estamos definiendo al otro cúbit. En otras palabras, si medimos el primer cúbit y nos da como resultado $|0\rangle$, sabemos que el otro cúbit ha colapsado a $|0\rangle$, mientras que si da $|1\rangle$, sabemos que el otro cúbit ha colapsado a $|1\rangle$. Esto ocurre debido a que, según nuestro estado de superposición, solo pueden aparecer los estados $|00\rangle$ o $|11\rangle$. Cuando se da este caso, se dice que los cúbits implicados están *entrelazados*.

Los n cúbits de un sistema pueden escribirse de forma matricial en una matriz columna, siendo cada fila un estado básico.

$$\left.\begin{matrix} 0..00 \rightarrow \\ 0..01 \rightarrow \\ 0..10 \rightarrow \\ \vdots \\ 0..11 \rightarrow \end{matrix}\begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}\right\} 2^n$$

El número de filas viene determinado 2^n estados básicos, siendo n el número de cúbits. Los estados básicos forman una matriz columna en la cual hay un uno en la fila que le corresponda (según qué estado básico represente) y ceros en el resto de las $2^n - 1$ filas.

2.2 Puertas cuánticas

La mayor problemática que presenta trabajar con los cúbits es el concepto de colapsamiento. Se debe trabajar con los cúbits sin medirlos, para poder aprovechar el potencial que presenta la superposición. Para ello, se pueden utilizar las denominadas *puertas cuánticas* para procesar estos cúbits de la forma que se necesite. En este apartado se mostrarán algunas de ellas.

Podemos distinguir dos tipos principales de puertas: **puertas unarias y puertas binarias**. Las puertas unarias son aquellas que toman como entrada un único cúbit, ejemplos de estas son *Hadamard* (véase 2.2.2 , X o T). Las puertas binarias toman como entrada dos cúbits. Ejemplos de estas son las puertas de control.

Asimismo, se debe tener en cuenta que una puerta cuántica es siempre reversible, por lo que **toda puerta cuántica debe tener el mismo número de entradas que salidas**.

2.2.1 Cómo aplicar puertas cuánticas

Toda puerta cuántica tiene una representación matricial. Esto es porque para aplicar sus efectos se debe multiplicar el cúbit al que se aplica con la matriz de dicha puerta.

Por ejemplo, si se dispone del cúbit en estado básico $|1\rangle$ y se quiere aplicar la puerta X (para más información véase apartado 2.2.3) se debe realizar la siguiente operación:

$$\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}\cdot\begin{pmatrix}0\\ 1\end{pmatrix}=\begin{pmatrix}1\\ 0\end{pmatrix}=|0\rangle$$

Se obtiene como resultado el *ket* $|0\rangle$.

2.2.2 Puerta Hadamard

Esta puerta es la más práctica, y la que se suele usar primero en casi todos los circuitos cuánticos. Permite poner un cúbit en estado de superposición. Su representación matricial es la siguiente:

$$H=\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}$$

Si aplicamos esta puerta al cúbit $|0\rangle$ se obtiene

$$H|0\rangle=\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}1\\ 0\end{pmatrix}=\frac{1}{\sqrt{2}}\begin{pmatrix}1\\ 1\end{pmatrix}=\frac{1}{\sqrt{2}}|0\rangle+\frac{1}{\sqrt{2}}|1\rangle=|+\rangle$$

Para el caso del cúbit $|1\rangle$ obtenemos

$$H|1\rangle=\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}0\\ 1\end{pmatrix}=\frac{1}{\sqrt{2}}\begin{pmatrix}1\\ -1\end{pmatrix}=\frac{1}{\sqrt{2}}|0\rangle-\frac{1}{\sqrt{2}}|1\rangle=|-\rangle$$

Como se puede observar, estos estados equiprobables de un cúbit también se pueden representar como el estado $|+\rangle$ y el estado $|-\rangle$.

A pesar de que la fase del estado básico $|1\rangle$ cambie de signo, aplicar la puerta H a ambos estados básicos da como resultado una equiprobabilidad de colapsar tanto en el estado $|0\rangle$ como en el estado $|1\rangle$.

Aplicar la puerta Hadamard a un estado ya en superposición, reproduce el efecto contrario.

$$H\left(\frac{1}{\sqrt{2}}|0\rangle+\frac{1}{\sqrt{2}}|1\rangle\right)=\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\cdot\frac{1}{\sqrt{2}}\begin{pmatrix}1\\ 1\end{pmatrix}=\begin{pmatrix}1\\ 0\end{pmatrix}=|0\rangle$$

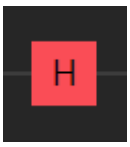


Figura 2. Imagen de una puerta Hadamard.

2.2.3 Puerta X

También conocida como la puerta NOT. La puerta X convierte el estado básico $|0\rangle$ en $|1\rangle$ y viceversa. Su representación matricial es

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Aplicar al estado $|0\rangle$ resulta en

$$X|0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Con un efecto similar, aplicar la puerta X a $|1\rangle$ resulta

$$X|1\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Esta puerta a veces se representa con un símbolo de '+' en los circuitos cuánticos (véase Figura 3).

Figura 3. Imagen de la puerta NOT.

2.2.4 Puerta Z

Esta puerta invierte el signo de la amplitud del estado básico $|1\rangle$. La fórmula que mejor describe esta puerta es la siguiente:

$$Z|j\rangle = (-1)^j|j\rangle$$

Es decir, al estado $|0\rangle$ no le afecta, mientras que el estado $|1\rangle$ se convierte en $-|1\rangle$.

Su representación matricial es la siguiente:

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Aplicando la puerta Z a $|0\rangle$

$$Z|0\rangle = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle$$

Aplicando la puerta Z a $|1\rangle$

$$Z|1\rangle = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} = (-1) \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} = -|1\rangle$$

Para determinados casos, es conveniente conocer la relación de esta puerta con la puerta X y la puerta H:

$$Z = HXH$$

Figura 4. Imagen de la puerta Z.

2.2.5 Puertas de control

Existen puertas binarias que nos permiten aplicar determinada puerta cuántica sobre un cúbit, llamado objetivo, cuando otro, el cúbit de control, está a uno. En este apartado, ilustraremos algunas de ellas.

Puerta Control X (CX/CNOT)

La puerta control X, control NOT o $CNOT$ hace lo siguiente: si el cúbit de control vale uno, se aplica la puerta X al cúbit objetivo.

Figura 5. Imagen de la puerta CNOT o CX.

Ejemplos de este comportamiento son los mostrados en la siguiente figura.

Figura 6. Aplicando CNOT cuando el control está a 1.

Siguiendo el circuito reflejado en la *Figura 6*, se obtiene lo siguiente:

$$|q1q0\rangle \Rightarrow |00\rangle \xrightarrow{X} |01\rangle \xrightarrow{CNOT} |11\rangle$$

Por el contrario, si no se aplica esa puerta X al primer cúbit, el resultado sería:

Figura 7. Aplicando CNOT cuando el control está a 0.

Esto se traduce en

$$|q1q0\rangle \Rightarrow |00\rangle \xrightarrow{CNOT} |00\rangle$$

Es decir, como el cúbit de control está a cero, no se aplica al cúbit objetivo.

Si se quiere aplicar una puerta $CNOT$, usando como cúbit de control uno en estado de superposición, el resultado sería como a continuación.

Figura 8. Aplicando CNOT usando como control un cúbit en superposición.

El resultado de este circuito sería el siguiente:

$$|q1q0\rangle \Rightarrow |00\rangle \rightarrow \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|01\rangle \rightarrow \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$$

Como se puede observar, la puerta $CNOT$ aplica a ambos, pero el cúbit objetivo solo cambia en aquel estado básico cuyo cúbit de control está a uno.

Puerta de Toffoli (CCX)

Esta puerta es una de las más famosas de la Computación Cuántica. Se trata de una puerta de funcionamiento idéntico a la $CNOT$, con la diferencia de que esta tiene **dos cúbits de control** y un cúbit objetivo. Se trata de una **puerta ternaria**.

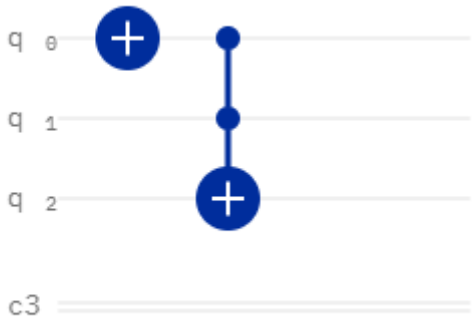


Figura 9. Circuito que aplica la puerta de Toffoli.

En el circuito que se ilustra arriba se puede observar cómo se aplica una primero una puerta X al cúbit cero y aplicamos una puerta de Toffoli teniendo como cúbits de control a $q0$ y $q1$ y como cúbit objetivo $q2$.

$$|q2q1q0\rangle \Rightarrow |000\rangle \xrightarrow{X} |001\rangle \xrightarrow{CCZ} |001\rangle$$

Como se puede observar, el valor de $q2$ no ha cambiado, precisamente porque el cúbit de control $q1$ no esta en estado $|1\rangle$.

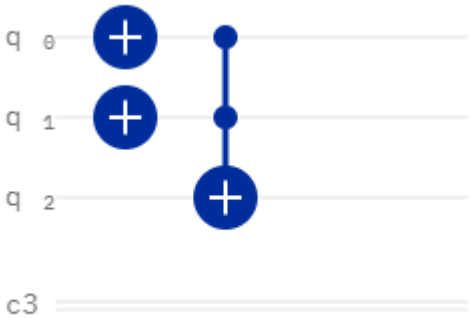


Figura 10. Circuito que aplica la puerta Toffoli y los cúbits de control están en estado uno.

En este circuito se puede observar cómo ahora los dos cúbits de control están en estado $|1\rangle$, lo que permite aplicarse la puerta X al cúbit $q2$.

$$|q2q1q0\rangle \Rightarrow |000\rangle \xrightarrow{X(q0)} |001\rangle \xrightarrow{X(q1)} |011\rangle \xrightarrow{CCZ} |111\rangle$$

Puerta Control Z (CZ)

Esta puerta es muy similar a la puerta $CNOT$. Aplica la puerta Z al cúbit objetivo solo si el cúbit de control está uno.

Debido a que en algunos simuladores no se encuentra disponible directamente esta puerta, se

puede aprovechar la relación comentada en el apartado 2.2.4

$$Z = HXH$$

y adaptarla, cambiando la puerta X con una puerta CX.

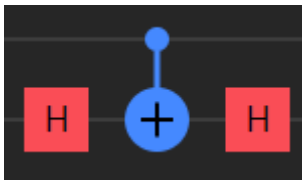


Figura 11. Imagen de la puerta Control Z expresada a través de la relación Z=HXH

A continuación, se muestra un ejemplo del funcionamiento de esta puerta.

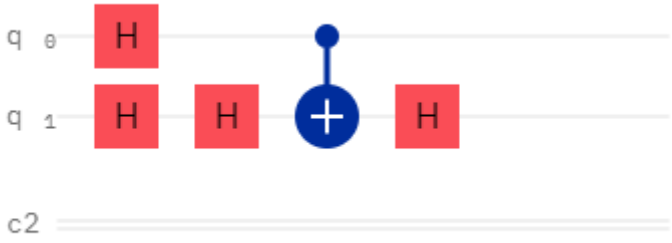


Figura 12. Ejemplo de aplicación de la puerta Control Z.

Este ejemplo resulta un poco más complejo, pero es capaz de mostrar todas las posibilidades de aplicar CZ a dos cúbits (véase *Figura 13*, donde el color rojo indica que la amplitud de ese cúbit se ha invertido).

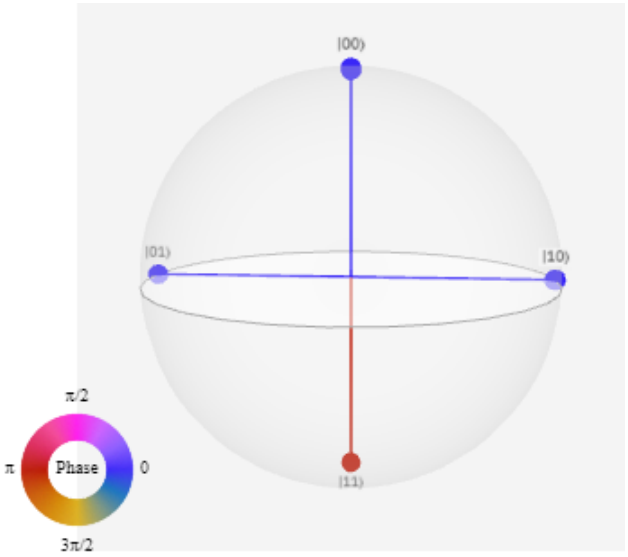


Figura 13. Resultado de aplicar CZ.

Lo que está ocurriendo es lo siguiente:

$$|q1q0\rangle \Rightarrow$$

$$|00\rangle \xrightarrow{H} \frac{1}{\sqrt{4}}(|00\rangle + |01\rangle + |10\rangle + |11\rangle) \xrightarrow{CZ} \frac{1}{\sqrt{4}}(|00\rangle + |01\rangle + |10\rangle - |11\rangle)$$

Puerta Control Control Z (CCZ)

Esta puerta funciona de forma idéntica a la puerta Control Z, con la excepción de que esta tiene dos cúbits de control y un cúbit objetivo.

Debido a que en algunas herramientas de simulación no se dispone directamente de esta puerta, se puede optar por aprovechar la propiedad descrita en el apartado sobre la puerta X (véase 2.2.3). Entonces aplicar la puerta CCZ se puede hacer como se refleja en la Figura 14.

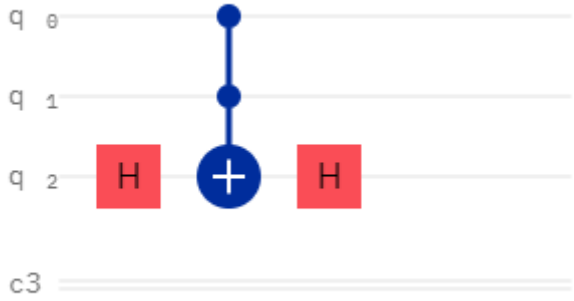


Figura 14. Puerta CCZ expresada con la relación Z=HXH.

Para el caso de la puerta CZ utilizamos una puerta $CNOT$, mientras que para este caso nos hace falta aplicar la puerta de Toffoli.

2.3 Medición

Todo lo que se ha visto hasta ahora trabaja con los cúbits sin comprobar su estado. Sin embargo, surge la necesidad de colapsar estos cúbits para poder trabajar con los resultados de los algoritmos cuánticos. Para ello existe la puerta de medida, normalmente ilustrada como un medidor de gas (véase Figura 15). Esta puerta da como resultado lo equivalente a un bit clásico, por lo que a partir de aquí se pierden las propiedades cuánticas. Es por esto que esta puerta se suele poner al final de cualquier algoritmo.

Figura 15. Imagen de la puerta de medida.

Para medir, simplemente debemos conectar esta puerta al cúbit que deseamos medir y "proyectarla" sobre el bit donde se quiera aplicar el resultado. Hay que considerar que el cúbit colapsará en uno de los estados básicos por lo que, en caso de un cúbit en superposición, este dará como resultado uno de los estados básicos que tenga amplitud distinta de cero.

En la *Figura 16* se muestra un circuito en donde se mide. En este circuito se puede observar cómo los cúbits $q1$ y $q0$ se encuentra en el estado de superposición

$$\frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

Al medir, habrá un 50% de probabilidades de que colapse al estado básico $|00\rangle$ y un 50% de probabilidades de que colapse al estado básico $|11\rangle$.

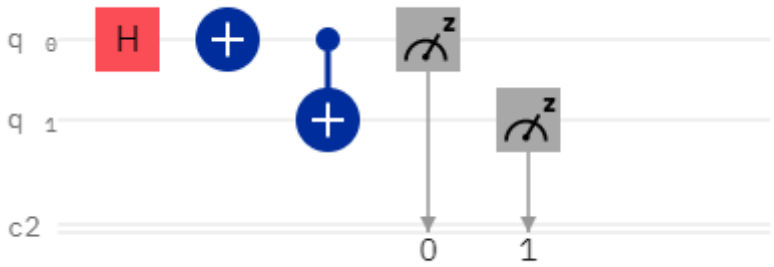


Figura 16. Circuito en donde realizamos una medida.

Capítulo 3 Algoritmo de Grover

3.1 Introducción al algoritmo de Grover

Supongamos una secuencia no ordenada de $N = 2^n$ elementos, representada por n cúbits y conformada por los valores $X_1, X_2, \dots, X_N$. Buscamos un elemento X_v que cumpla

$$C(X) = \begin{cases} 1, & X = X_v \\ 0, & X \neq X_v \end{cases}$$

Este estado X_v, que satisface la función arriba especificada, debe existir en el conjunto de valores indicado como entrada.

En computación clásica, existen algoritmos como la búsqueda secuencial [15] o la búsqueda mediante tabla de dispersión [16] que permiten dar solución a este problema en un tiempo de $O(N)$. Sin embargo, en 1996 Lov K. Grover planteó un algoritmo cuántico de tipo probabilístico para dar solución a este problema. Este algoritmo, denominado Algoritmo de Grover, resulta de mucho interés, pues es capaz de resolverlo en un tiempo de $O(\sqrt{N})$, siendo más rápido que los algoritmos clásicos desarrollados hasta la fecha. Y posteriormente, en 1999 Goong Chen, Stephen A. Fulling y Marlan Scully presentaron un paper que ampliaba la idea de Grover, generalizando el algoritmo para casos en los que más de un objeto satisface $C(X)$ [17].

En este capítulo se procederá a comentar este algoritmo, explicando su estructura y su funcionamiento.

3.2 Estructura y funcionamiento del algoritmo

3.2.1 Funcionamiento de Grover

Grover está compuesto de tres partes:

- Inicialización de los estados básicos a la forma

$$|s\rangle = \frac{1}{\sqrt{N}} \sum_x |x\rangle$$

  Es decir, inicializar el sistema para conseguir una equiprobabilidad de todos los estados básicos. Para conseguir esto, aplicamos una puerta *Hadamard* a cada cúbit del sistema.

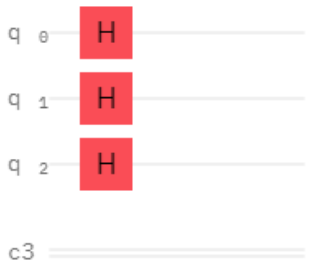


Figura 17. Aplicamos una puerta H a cada cúbit del sistema.

Lo reflejado en la *Figura 17* nos dará como resultado una equiprobabilidad de todos los estados básicos que pueden formar tres cúbits. En la *Figura 18* se puede observar el resultado de esta fase del algoritmo.

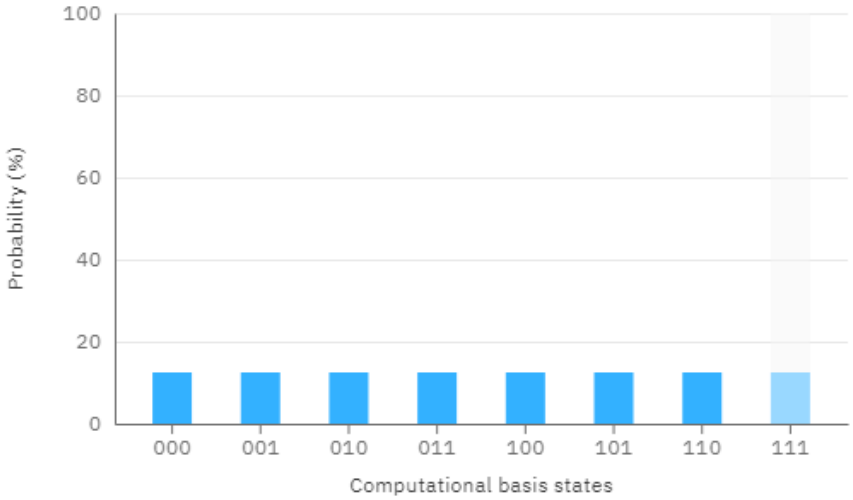


Figura 18. Resultado de la ejecución de una puerta H a cada cúbit.

Se puede plantear la necesidad de realizar la búsqueda sobre un subconjunto de elementos. La realidad es que los resultados de sustituir esta fase por algún circuito que elimine del espacio de búsqueda determinados estados no serán los que esperamos. Tal y como se deduce en el apartado 4 de *[1]*, es necesario una equiprobabilidad de todos los estados del sistema. Cuando hablamos de reducir el espacio a un subconjunto, realmente estamos expresando que la amplitud de aquellos estados que no pertenecen al subconjunto es cero. Es decir, contrario a lo que parece, seguimos teniendo todos los estados básicos como espacio de búsqueda. Esto afectará a la media, puesto que seguirán participando aquellos estados que tengan amplitud cero. En consecuencia, la reflexión sobre la media que realiza la tercera fase del algoritmo no funcionará correctamente, provocando que Grover falle.

No obstante, en el Capítulo 5 se comentan alternativas y líneas futuras para investigar este problema.

- Realizar, con r iteraciones, la siguiente traza:
    - Aplicar el operador U_w, también conocido como oráculo o detector.
    - Aplicar el operador $U_s = 2|s\rangle\langle s| - I$, también llamado amplificador, difusor o puerta de Grover.

  Cuando tratamos con tres o más cúbits, y dependiendo del valor de N, se pueden reducir las posibilidades de éxito del algoritmo con una sola iteración. Es por esto por lo que a veces nos hará falta más de una.

  Podemos calcular el número de iteraciones necesarias con la fórmula *[18]*

$$r = \left\lfloor \frac{\pi\sqrt{N/k}}{4} \right\rfloor$$

  Donde N es el número de posibilidades del espacio de búsqueda y k es el número de elementos que se están buscando, es decir, para los que la función a evaluar devuelve 1.

  Tras cada iteración de esta fase, la amplitud del estado que buscamos aumenta al orden $O\left(\frac{1}{\sqrt{N}}\right)$. Ejecutando estas iteraciones en $O\left(\sqrt{N}\right)$ veces, la probabilidad del estado deseado alcanza 1. Una explicación más detallada de estas conclusiones puede encontrarse en *[1]*.

  A la hora de implementar este algoritmo, el valor indicado por r resulta ser la mejor opción para maximizar la probabilidad de éxito de Grover, tal y como es descrito en *[19]*.

  Por ejemplo, si estamos aplicando Grover con tres cúbits y buscamos un solo estado, el número de iteraciones necesario es $r = 2$, mientras que para cuatro cúbits y $k = 4$ el número es $r = 1$.

  La probabilidad de fallo se encuentra acotada en $\frac{1}{N}$.

- Medición del estado de los cúbits. Teniendo en cuenta el comportamiento probabilístico del algoritmo, nos debería dar una de las soluciones que buscamos con el detector.

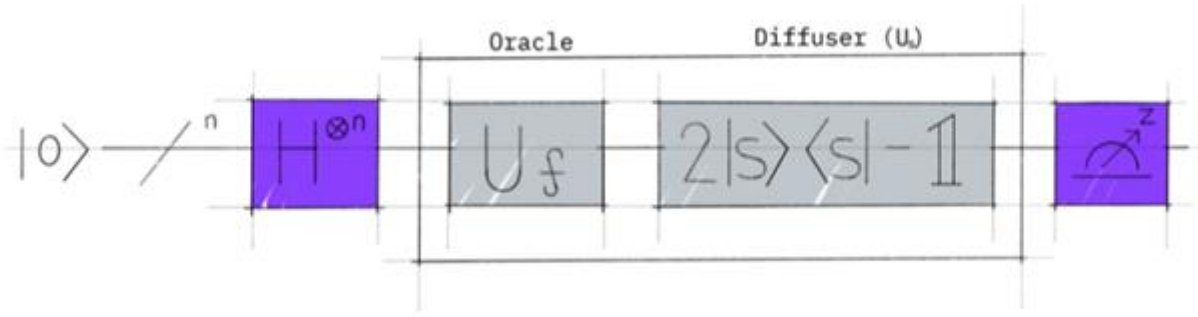


Figura 19. Estructura del algoritmo de Grover.

Además del número de iteraciones para el detector y la puerta de Grover, hay que prestar atención a la relación entre el número de elementos del espacio de búsqueda N y el número de elementos que se buscan k. Es conveniente procurar que

$$k < N/2$$

Casos en los que $k \geq N/2$ reducirá considerablemente la probabilidad de éxito del algoritmo como se discute en *[20]*. Cuando ocurra esto, una forma de evitarlo es ampliar el número del espacio de búsqueda sin alterar el número de elementos detectados por el oráculo, por ejemplo, ampliando en uno el número de cúbits del sistema.

3.2.2 Oráculo

El oráculo se suele representar como una puerta U_f (véase Figura 19), haciendo referencia a que es una *caja negra* en donde nosotros pondremos nuestro circuito, que hará de detector para *marcar* aquellos estados que deseamos que Grover encuentre.

Es decir, U_f debe ser una matriz que aplicada a un estado cuántico resulte en

$$U_w|x\rangle = \begin{cases} |x\rangle & si\ x \neq w \\ -|x\rangle & si\ x = w \end{cases}$$

donde w es el estado que estamos buscando, aquel que $C(w) = 1$.

Para marcar, debemos **rotar la fase π radianes** del estado o estados en cuestión. Esto se hace aplicando una puerta Z a alguno de los cúbits del sistema, de forma que la amplitud del estado final sea negativa. Por ejemplo, si queremos detectar el ket $|010\rangle$, podríamos utilizar el circuito de la Figura 20, que cambia de signo la amplitud de dicho estado rotando π radianes la fase de este.

Nótese como se aplica inicialmente una puerta *Hadamard* a cada cúbit del sistema. Esto es un proceso de inicialización para superponer todos los estados y posteriormente detectar el que deseamos. El oráculo es a partir de estas puertas.

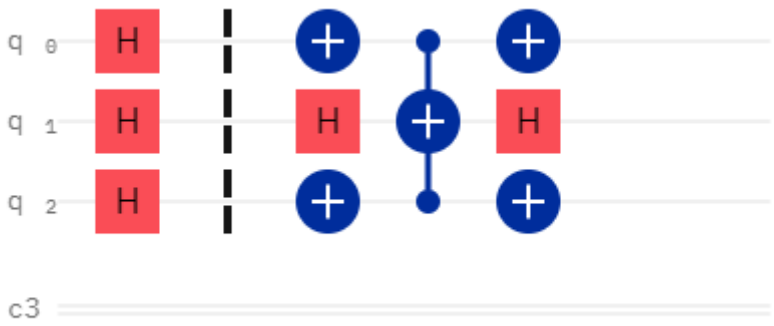


Figura 20. Circuito que cambia de signo la amplitud del estado 010.

Las amplitudes resultado de estas operaciones se reflejan en la Figura 21.

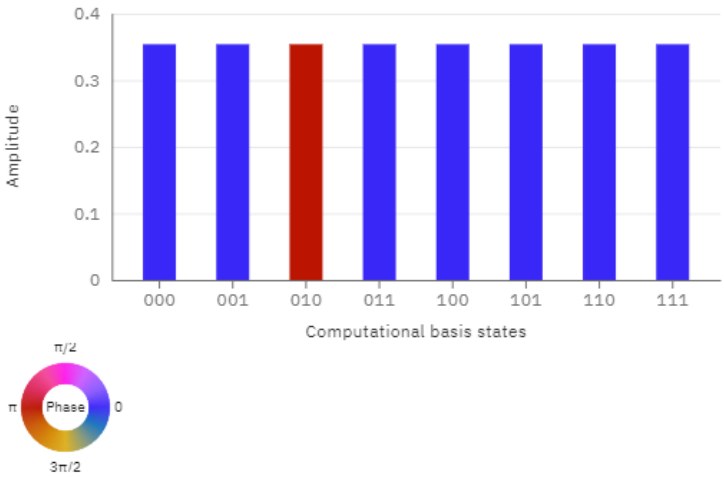


Figura 21. Resultados de aplicar el cambio de signo a la amplitud del estado 010.

También podríamos hacer un oráculo que destaque más de un estado. Por ejemplo, podríamos querer detectar los estados $|010\rangle$ y $|011\rangle$. El circuito que tendríamos que desarrollar se muestra en la Figura 22.

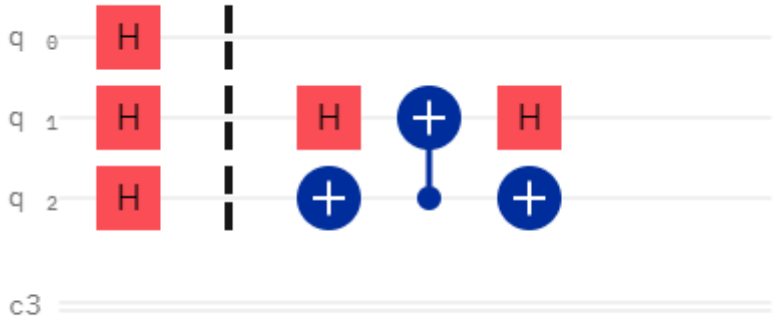


Figura 22. Circuito que detecta los estados 010 y 011.

Las amplitudes de los estados tras aplicar esta puerta se reflejan en la Figura 23.

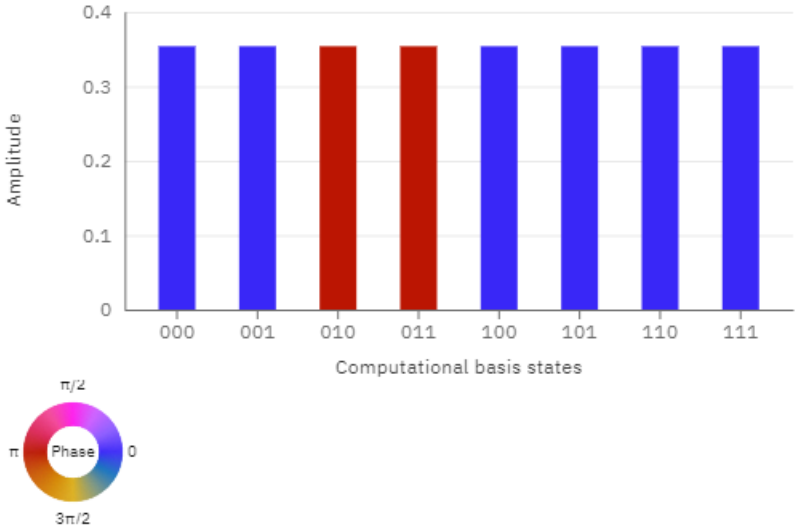


Figura 23. Resultado de aplicar el circuito de la figura anterior.

Es importante tener en cuenta que cuando se desarrolla un oráculo que busca más de un estado debemos asegurarnos de que la relación $k < N/2$ se cumple, tal y como se indica al final del apartado 3.2.1 . En este caso, como se tiene $k = 2$ y $N/2 = 4$, no hay ningún problema.

3.2.3 Puerta de Grover

El amplificador, difusor, o puerta de Grover, es un circuito encargado de realizar la **reflexión sobre la media** de todos los estados del espacio de búsqueda. Su representación matricial se puede describir como

$$G = \begin{cases} \frac{2}{N} & i \neq j \\ -1 + \frac{2}{N} & i = j \end{cases}$$

donde i es la fila y j es la columna de la matriz de esta puerta.

Consideremos un detector como el expresado en la Figura 22. Inicialmente, antes de aplicar el oráculo, realizaremos una inicialización de los estados, aplicando una puerta *Hadamard* a cada cúbit del sistema (véase Figura 18). Tras esto, y tras aplicar el oráculo, podemos ver como resultado una gráfica como la de la Figura 24.

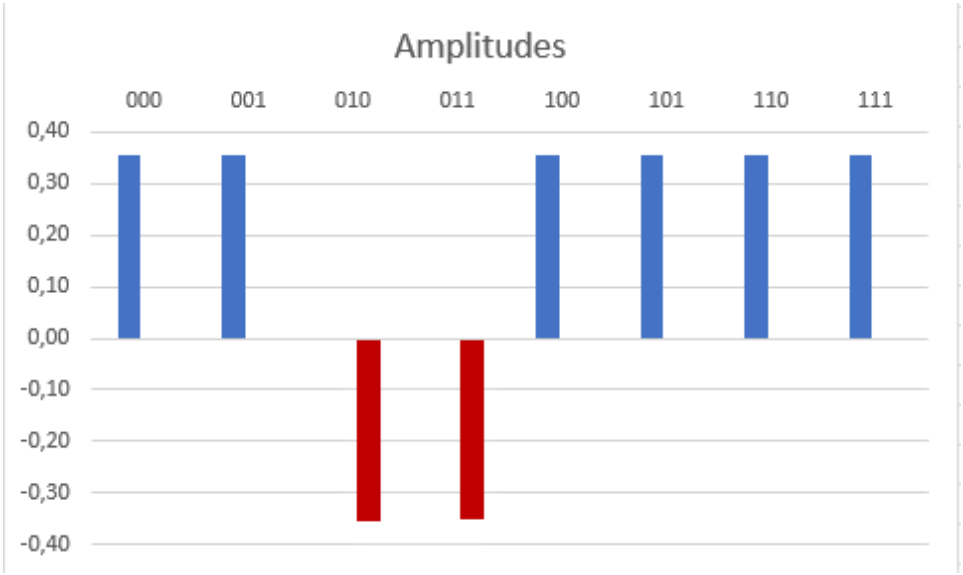


Figura 24. Resultado de aplicar el oráculo al sistema.

La puerta de Grover recibe al sistema de esta manera, y con esta información obtiene la media de las amplitudes (véase Figura 25). Esta media, como se puede suponer, inicialmente era igual al valor de las amplitudes de todos los estados. Al rotar π radianes la fase de los estados que buscamos, esta media disminuye.

Al realizar la reflexión de estas sobre dicha media [21] (véase Figura 26), se puede observar cómo los estados disminuyen en probabilidad hasta alcanzar el cero, mientras que los estados que deseamos se encuentren logran aumentar su amplitud notoriamente. Si medimos ahora, encontramos uno de esos dos estados, cumpliendo con éxito el objetivo del algoritmo.

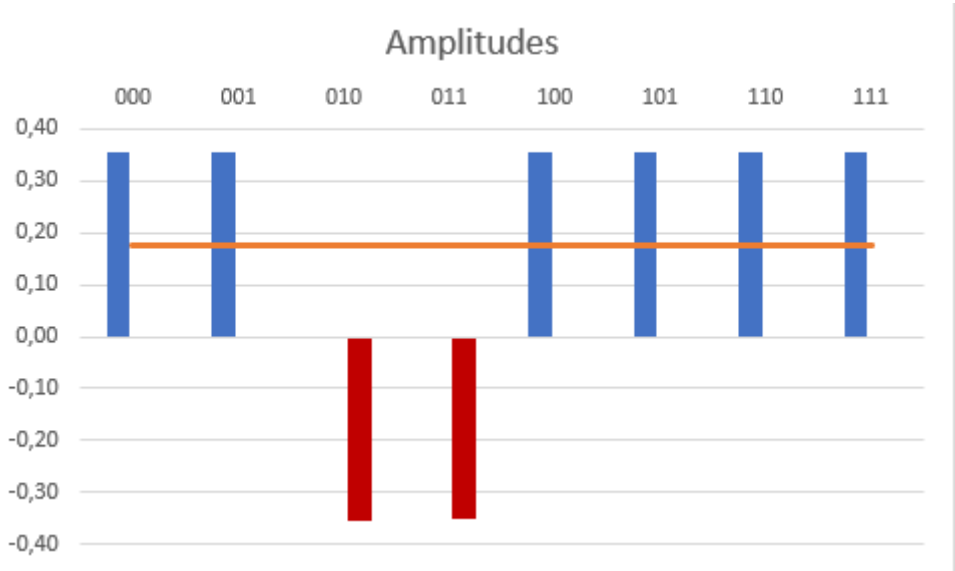


Figura 25. Media de las amplitudes de los estados del sistema.

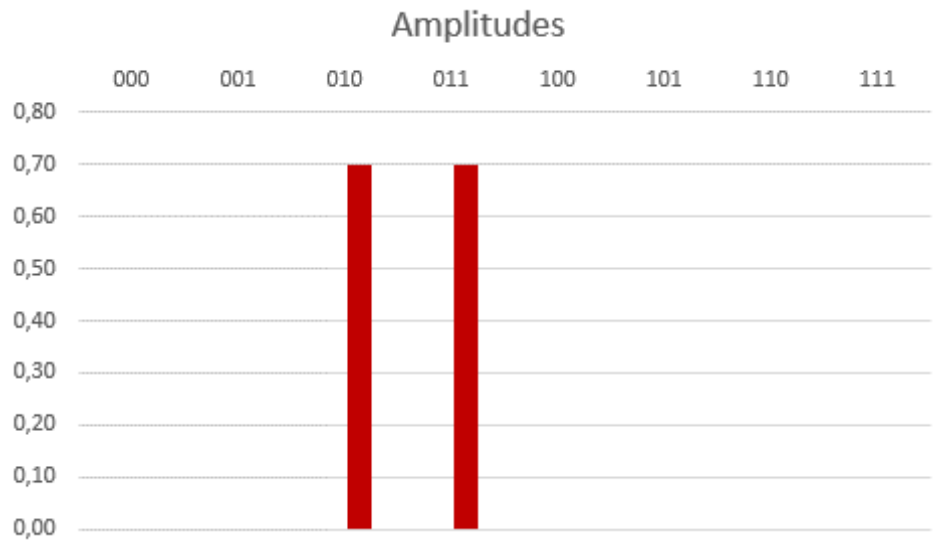


Figura 26.Resultados tras la reflexión sobre la media.

En este ejemplo se tiene $N = 8$ y $k = 2$, necesitando una única iteración de oráculo y amplificador para obtener un 100% de éxito. Sin embargo, variando alguno de los datos, podremos ilustrar la necesidad, por un lado, de realizar más de una repetición y, por otro, de tener en cuenta la recomendación de que $k < N/2$ (véase Capítulo 4).

Cuanto más grande es el espacio de búsqueda, más alta será la media de las amplitudes, lo que provoca que tanto el estado en fase π como los demás estados tengan una probabilidad de aparecer relativamente alta y similar. Evidentemente, siempre el estado con amplitud negativa tendrá más probabilidad que los demás, pero al ser tan similar, hay un porcentaje demasiado alto de que Grover no encuentre el estado correcto. Cuantas más iteraciones se realicen sobre estos estados, siempre que se repita $O(\sqrt{N})$ veces, más crecerá la amplitud del estado que buscamos, haciendo evidente la necesidad de repetir la detección y la amplificación el mismo número de veces. Sin embargo, la amplificación del estado detectado crecerá hasta que lleguemos al número de iteraciones determinado por r. A partir de ese número de iteraciones, la probabilidad de éxito se reduce considerablemente.

Por otro lado, cuantos más estados estemos buscando sobre un espacio de búsqueda determinado, podemos llegar a encontrar un problema similar. La regla que indica que $k < N/2$ se debe cumplir para asegurar un buen porcentaje de éxito radica en el problema de la media de las amplitudes. Y es que, si el número de elementos que buscamos supera la mitad del espacio de búsqueda, la media quedará demasiado baja, provocando un incremento demasiado leve de las amplitudes de los estados con fase negativa, haciendo inútil la reflexión sobre la media.

Cuando implementemos el algoritmo de búsqueda probabilístico Grover, debemos tener en cuenta ambas premisas, y mantener un equilibrio entre el número de elementos a buscar y el número de estados del espacio de búsqueda.

El circuito del amplificador puede variar, dependiendo del número de cúbits del sistema. Para el caso de dos cúbits, el amplificador es bastante sencillo (véase Figura 27). Para el caso de tres cúbits, la puerta de Grover se complica un poco más, teniendo como resultado lo reflejado en la Figura 28.

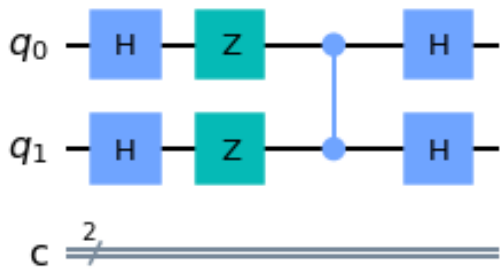


Figura 27. Amplificador para la búsqueda en un sistema de dos cúbits.

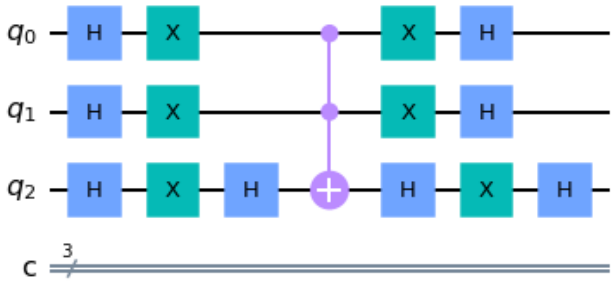


Figura 28. Amplificador para la búsqueda en un sistema de tres cúbits.

A partir de cuatro cúbits, se puede realizar un amplificador generalizando el circuito de la Figura 28, añadiendo una puerta *Hadamard* y una puerta *NOT* a cada nuevo cúbit y utilizando la puerta de Toffoli con los primeros cúbits como control y el último cúbit como objetivo.

Capítulo 4 Implementación del algoritmo

Para la implementación del algoritmo hemos escogido tres ejemplos, cada uno aportando una visión de las consideraciones explicadas en el Capítulo 3 .

El primero de ellos buscará un elemento en un espacio de búsqueda de $N = 4$ (dos cúbits), el segundo utilizará un espacio de $N = 8$ (tres cúbits) y buscará dos elementos.

El tercero de nuestros ejemplos ilustrará la necesidad de ejecutar más de una iteración, siempre acercándose al valor indicado por la fórmula de r expuesta en el apartado 3.2.1 . Finalmente, el último de nuestros ejemplos reflejará el problema de que $k \geq N/2$.

Para la ejecución de Grover se ha utilizado el lenguaje de programación Python, con el SDK Qiskit, y el entorno de desarrollo de *IBM Quantum Experience*: *Quantum Lab* y *Quantum Composer*. En todos los circuitos utilizaremos primeramente el simulador *qasm_simulator* [22], y finalizaremos con una prueba empírica sobre un ordenador cuántico real, el *ibmq_burlington* [23].

Nótese que, en todas las ejecuciones que se ilustran en este capítulo, tanto las ejecuciones del simulador como las del computador real se han realizado 2.048 veces, para mostrar la probabilidad de medición de cada uno de los estados del sistema.

4.1 Ejecución de un circuito simple con dos cúbits

Vamos a ejecutar un circuito sencillo con dos cúbits, para ilustrar los pasos que debemos realizar para este algoritmo. En primer lugar, debemos saber cuántas iteraciones se deben ejecutar:

$$r = \left\lfloor \frac{\pi\sqrt{4/1}}{4} \right\rfloor = 1$$

Perfecto, una sola iteración. También debemos asegurarnos de que cumplimos la premisa que relaciona k con $N/2$.

$$k < \frac{N}{2} \Rightarrow 1 < \frac{4}{2} \Rightarrow 1 < 2$$

Efectivamente, podemos proceder con el espacio de búsqueda actual.

Por otro lado, debemos decidir qué estado vamos a buscar. Para este ejemplo, buscaremos el estado $|11\rangle$. Asimismo, debemos desarrollar el oráculo que se encargue de invertir el signo de la amplitud de este estado. El oráculo desarrollado se refleja en la Figura 29.

Con el oráculo desarrollado y, utilizando la puerta de Grover adaptada para dos cúbits (véase Figura 27), procedemos a realizar el circuito (véase Figura 30).

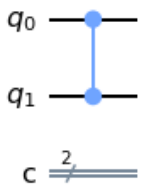


Figura 29. Oráculo para la detección del estado 11.

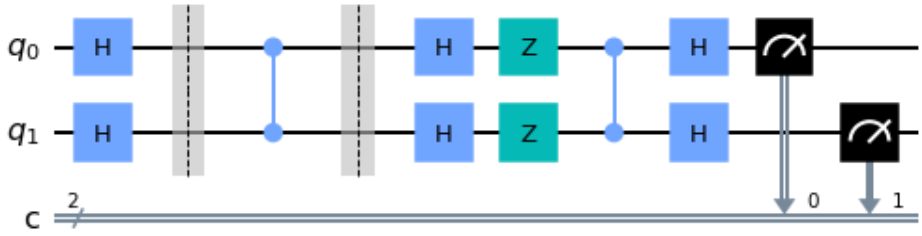


Figura 30. Circuito cuántico para buscar el estado 11 en un sistema con dos cúbits.

4.1.1 Ejecución en un simulador

Una vez desarrollado, en *Quantum Lab* procedemos a ejecutarla en el simulador. El resultado se refleja en la Figura 31.

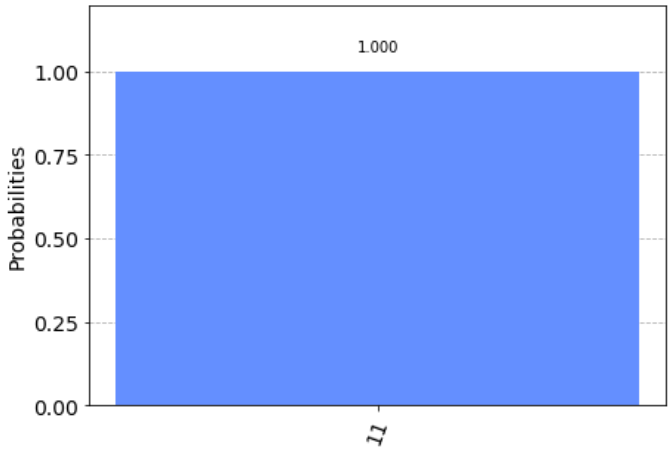


Figura 31. Resultado de la ejecución en un simulador de la búsqueda del estado 11.

4.1.2 Ejecución en un computador cuántico

Cuando ejecutamos este circuito en un ordenador cuántico real, veremos que principalmente obtenemos el estado que queremos, pero el ruido existente provoca la aparición, en mucha menor medida, de otros estados del sistema (véase Figura 32).

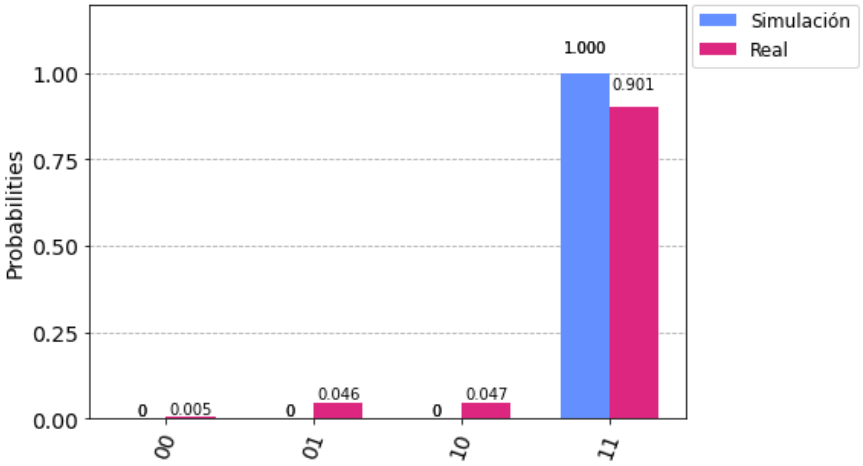


Figura 32. Ejecución del circuito de búsqueda del estado 11 en un ordenador cuántico real.

4.2 Ejecución de un circuito con tres cúbits y una iteración

A continuación, vamos a mostrar un ejemplo de ejecución con tres cúbits y dos elementos a buscar. Con $N = 8$ y $k = 2$ el número de iteraciones necesarias se reduce a

$$r = \left\lfloor \frac{\pi\sqrt{8/2}}{4} \right\rfloor = 1$$

Por lo que solo requerimos una iteración.

Se van a buscar los estados $|101\rangle$ y $|111\rangle$, como hemos visto antes, debemos desarrollar el oráculo para invertir el signo de la fase de estos estados. El oráculo para este caso se muestra en la Figura 33.

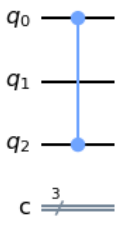


Figura 33. Oráculo para la detección de los estados 101 y 111.

Para el amplificador, usaremos el que se ha descrito en la Figura 28. Teniendo tanto el oráculo como el amplificador, se puede desarrollar el circuito (véase Figura 34).

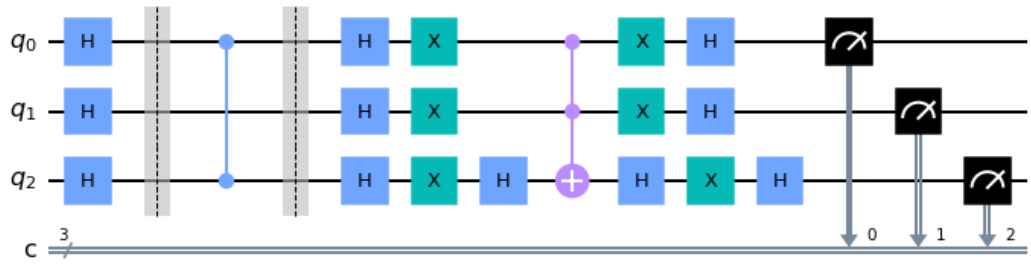


Figura 34. Circuito para la búsqueda de los estados 101 y 111.

4.2.1 Ejecución en un simulador

Con este desarrollo y *Quantum Lab* en mano, nos disponemos a ejecutar el código en el simulador. Los resultados se reflejan en la Figura 35.

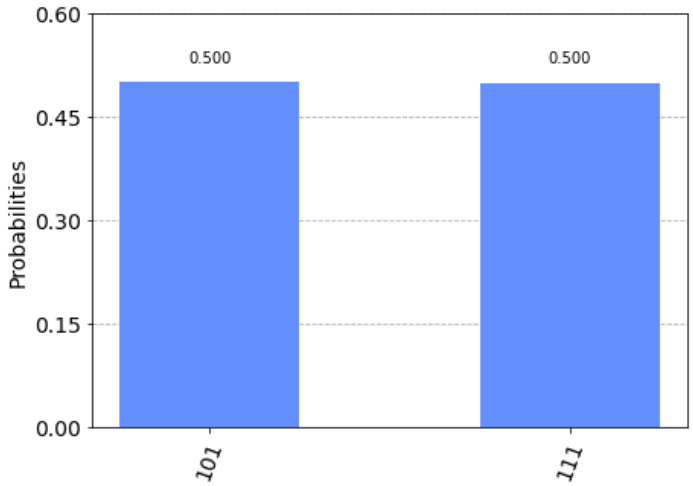


Figura 35. Resultado de la simulación del circuito de búsqueda de los estados 101 y 111.

Como se puede observar, el circuito funciona correctamente, y nos devuelve los estados que deseamos.

4.2.2 Ejecución en un computador cuántico

Ejecutando este algoritmo en un computador cuántico real, observaremos mucho ruido, pero los estados que buscamos tienen mayor probabilidad que el resto de aparecer (véase Figura 36).

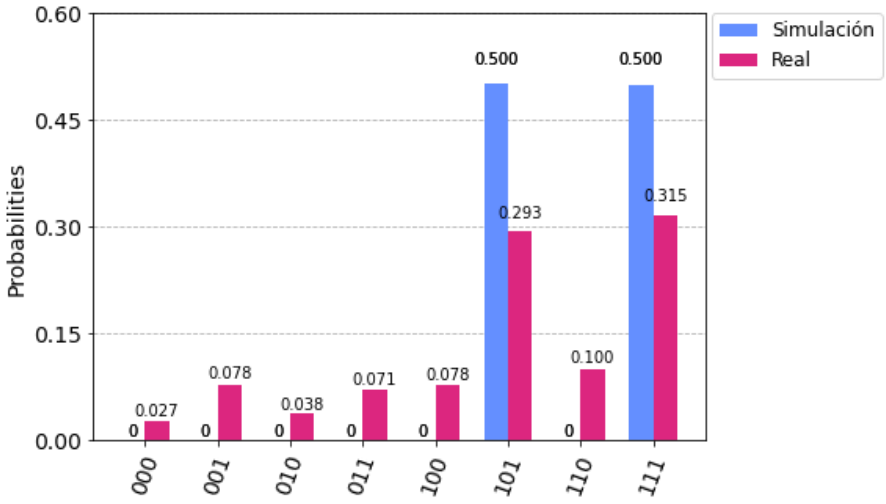


Figura 36. Comparativa de los resultados reales y simulados del circuito de búsqueda de los estados 101 y 111.

4.3 Ejecución de un circuito que requiere dos iteraciones

Con esta ejecución se pretende ilustrar la necesidad de realizar más de una iteración en determinados casos, dependiendo del número de estados del espacio de búsqueda y del número de elementos a buscar.

Para este ejemplo usaremos el detector de la Figura 20. Como se puede observar en la Figura 21, este oráculo cambiará el signo de la amplitud del estado $|010\rangle$. Según la fórmula presentada en el apartado 3.2.1 , el número de iteraciones r necesarias para maximizar las probabilidades de éxito es

$$r = \left\lfloor \frac{\pi\sqrt{8/1}}{4} \right\rfloor = 2$$

No obstante, vamos a ejecutar una primera aproximación utilizando una sola iteración. El circuito que estaremos aplicando se ilustra en la Figura 37.

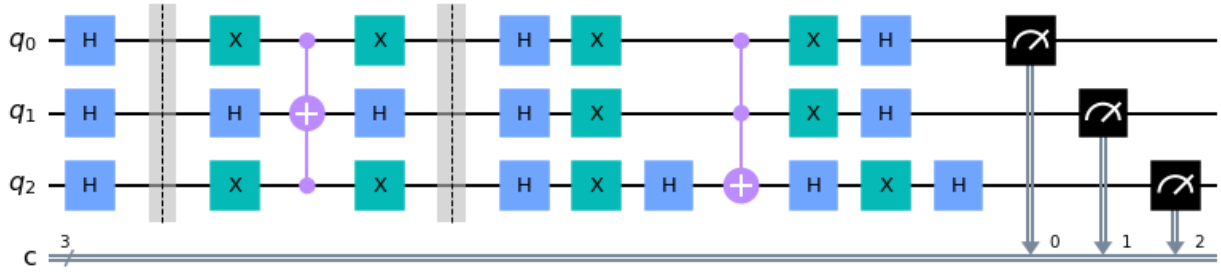


Figura 37. Circuito para buscar el estado 010 con una sola iteración.

4.3.1 Ejecución en un simulador

Si ejecutamos este circuito, veremos como la probabilidad de cada estado no es del todo precisa. Buscamos el estado $|010\rangle$ y hemos encontrado *principalmente* el estado $|010\rangle$, pero con pequeñas apariciones de los otros siete (véase Figura 38).

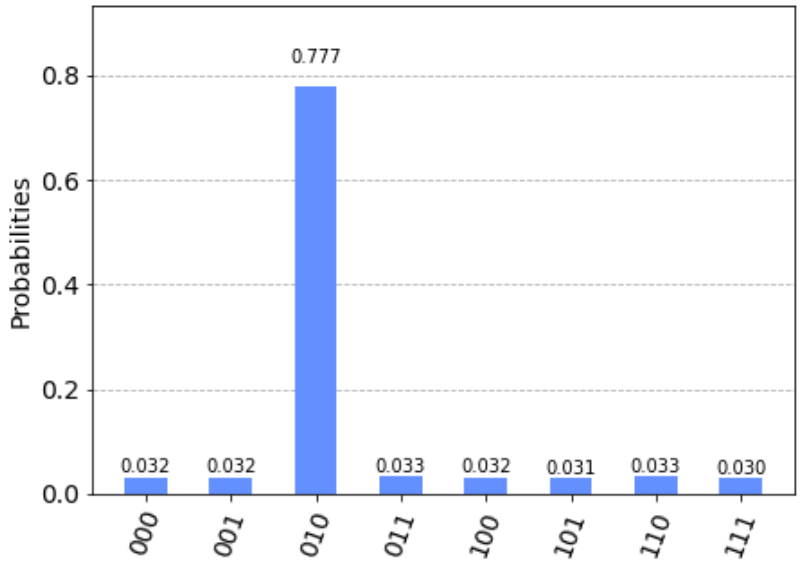


Figura 38. Resultado de buscar el estado 010 con una iteración.

En este caso las probabilidades de aparición son tan bajas que podría servirnos así. No obstante, se puede aproximar mucho más aplicando una segunda iteración del detector más puerta de Grover. Modificamos el circuito como se refleja en la Figura 39.

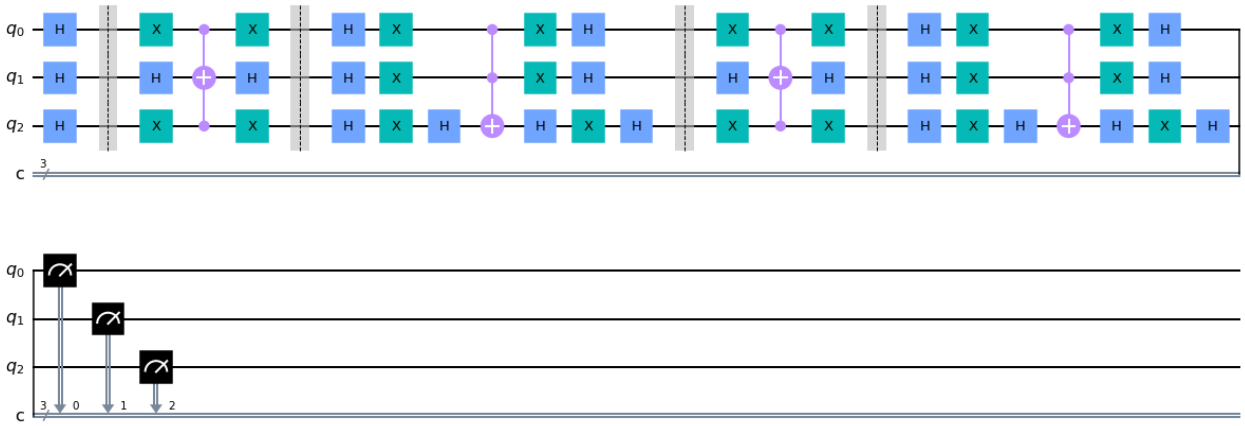


Figura 39. Circuito para buscar el estado 010 con dos iteraciones.

A continuación, ejecutamos el circuito en el simulador y observamos los resultados (véase Figura 40).

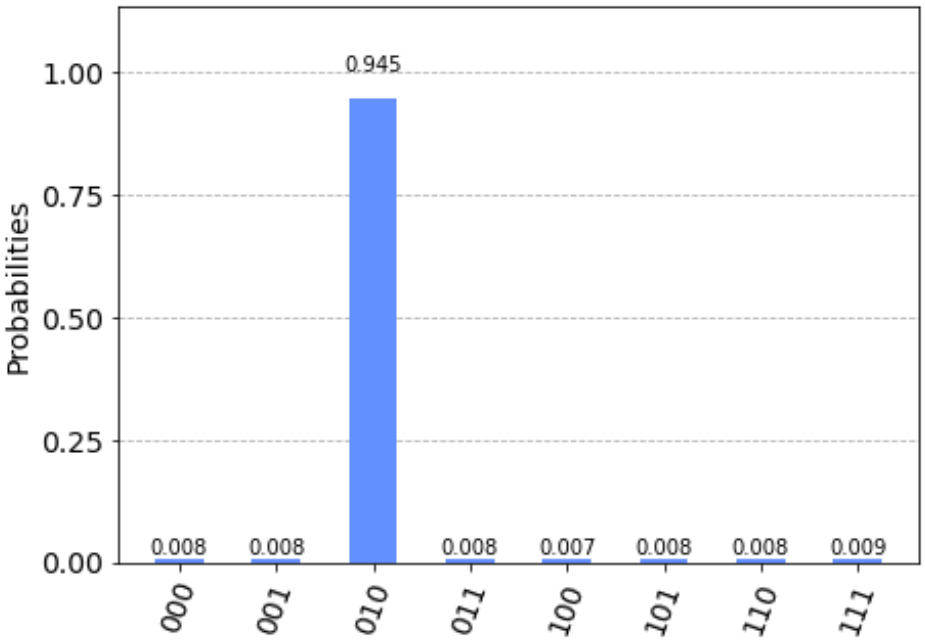


Figura 40. Resultados de buscar el estado 010 con dos iteraciones.

Como se puede ver, las probabilidades de que aparezcan los otros estados han pasado de 3.2% a un 0.8%, siendo bastante poco frecuente su aparición.

Este ejemplo es una versión con solo tres cúbits. Si quisiéramos trabajar con más de tres la necesidad de realizar las iteraciones necesarias para maximizar el éxito se vuelve considerablemente más crucial.

Para ilustrar la necesidad de ajustarse al resultado de la fórmula de r, podemos probar a ejecutar tres iteraciones. El resultado será una pérdida considerable del porcentaje de éxito (véase Figura 41).

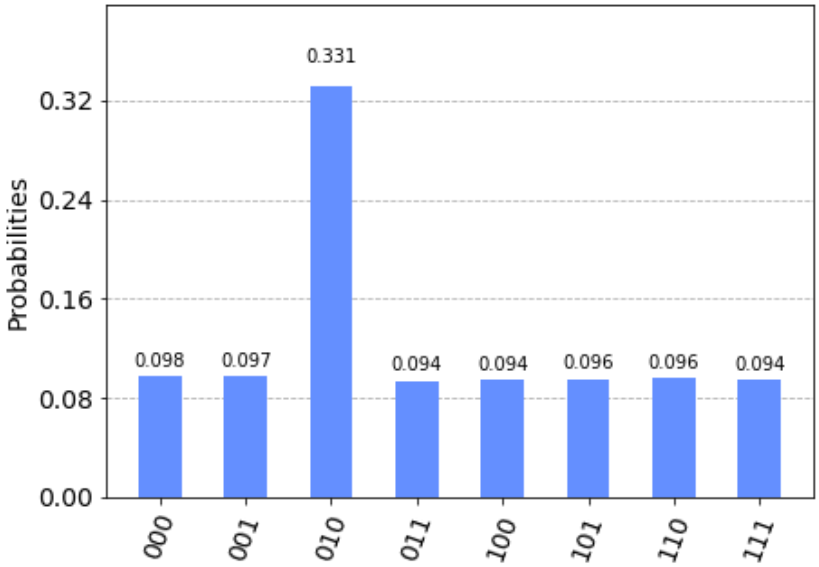


Figura 41. Resultado de buscar el estado 010 con tres iteraciones.

4.3.2 Ejecución en un computador cuántico

Si ejecutamos este circuito en un computador cuántico real, con dos iteraciones (las indicadas por r), obtenemos unos resultados con mucho ruido. En la Figura 42 se puede observar una comparativa de la ejecución del algoritmo en la simulación con respecto a la ejecución en un computador cuántico. Como se puede observar, el ruido presente en los computadores cuánticos perjudica al éxito del algoritmo. Si a esto le sumáramos la probabilidad de fallo provocada por un incorrecto número de iteraciones, no se podría utilizar este circuito para obtener el estado $|010\rangle$.

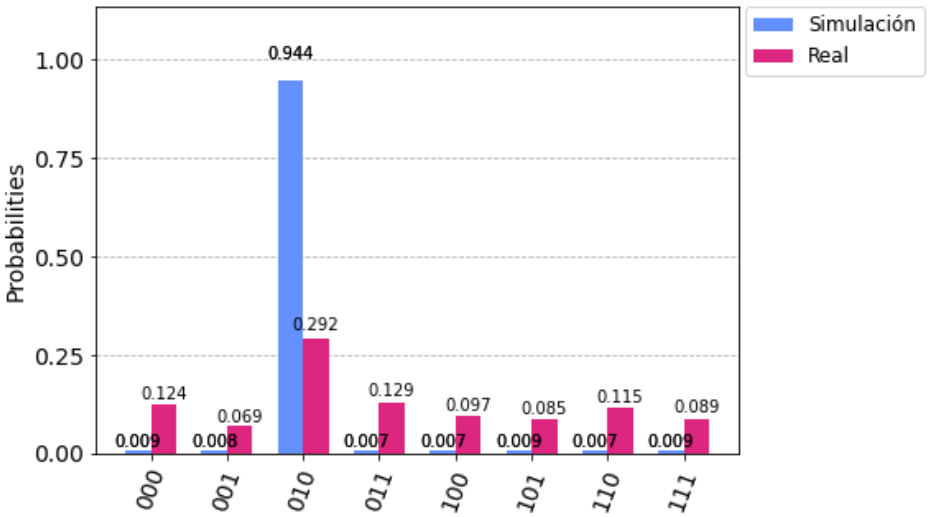


Figura 42. Comparación de simulación vs. real sobre la ejecución del segundo circuito.

4.4 Ejecución de un circuito con k mayor a N/2

En este apartado, vamos a realizar la búsqueda de aquellos números entre el cero y el siete cuyo número de letras sea impar. Es decir, buscaríamos los números uno, dos, cinco y siete, codificados como los estados $|001\rangle$, $|010\rangle$, $|101\rangle$ y $|111\rangle$.

Aprovecharemos este problema para ilustrar la necesidad de que el número de estados a buscar debe ser inferior a la mitad del espacio de búsqueda. Según la fórmula aprendida en el apartado 3.2.1 , para este caso con tres cúbits no nos será posible obtener las soluciones que buscamos, puesto que $4 \geq 8/2 \Rightarrow 4 \geq 4$, y deberíamos utilizar un cúbit adicional, para cumplir esta premisa. Sin embargo, vamos a comprobar empíricamente que esto es cierto.

4.4.1 Ejecución utilizando tres cúbits

Primeramente, realizaremos un intento de resolver este problema con tres cúbits. Por un lado, requerimos de un oráculo que rote las fases de estos cuatro estados (véase Figura 43).

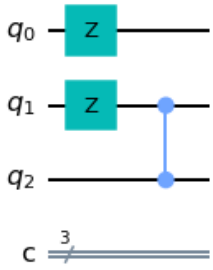


Figura 43. Oráculo para la detección de los estados 001, 010, 101 y 111.

Una vez definido el oráculo, y utilizando un amplificador como el descrito en la Figura 28, disponemos circuito mostrado a continuación:

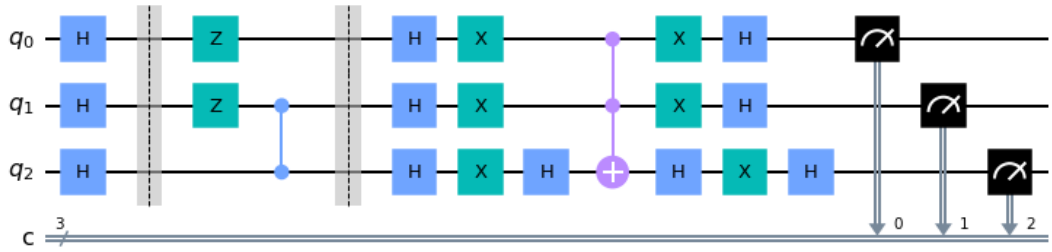


Figura 44. Circuito para buscar los estados 001, 010, 101 y 111.

Si ejecutamos este circuito en un simulador, observaremos que hay un problema (véase Figura 45).

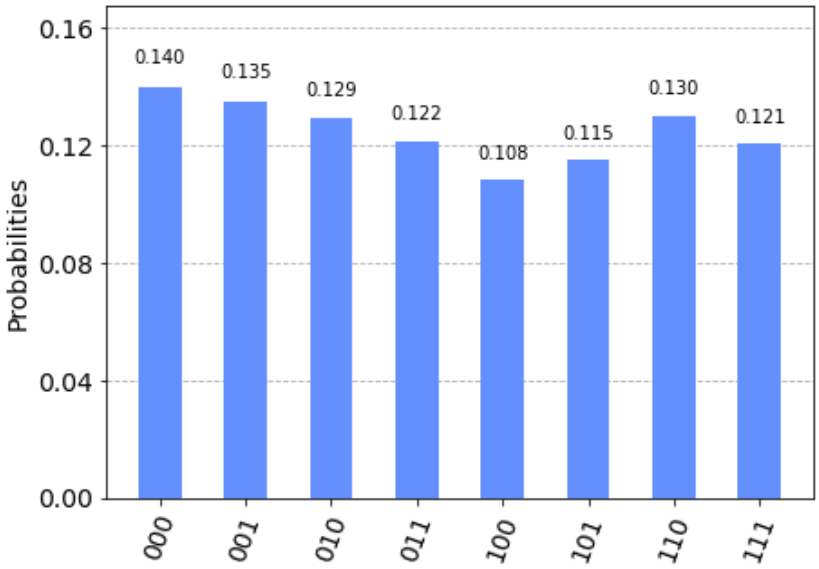


Figura 45. Resultado de ejecutar el algoritmo de Grover para 4 estados y 3 cúbits.

Esto ocurre precisamente por la premisa de $k < N/2$ explicada en varias ocasiones. En nuestro problema, disponemos de cuatro estados a buscar y de ocho estados en el espacio de búsqueda. A efectos de la reflexión sobre la media, podemos darnos cuenta de que la media de las amplitudes de un sistema en el todos tienen el mismo valor de amplitud, pero una mitad es negativa y la otra positiva, es 0. Esto provocará que se inviertan las amplitudes, resultando en la misma probabilidad que al inicio.

Para evitar esto, debemos ampliar el espacio de búsqueda añadiendo un cúbit más, y modificar el oráculo para que dicho cúbit adicional no afecte a la búsqueda.

4.4.2 Ejecución utilizando cuatro cúbits

Al ampliar el espacio de búsqueda, observaremos que el oráculo ahora detecta más elementos (véase Figura 46).

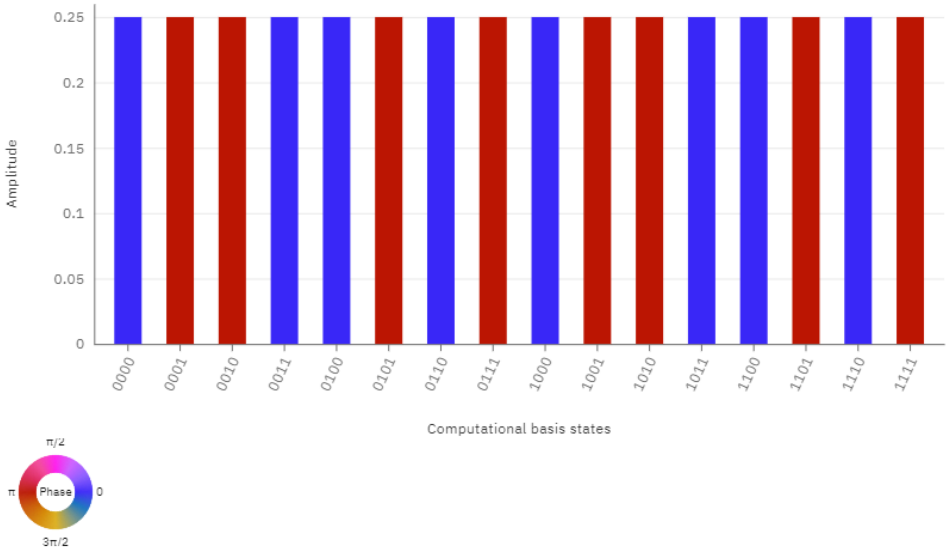


Figura 46. Resultado del oráculo al ampliar el espacio de búsqueda a 16 elementos.

Por tanto, debemos modificarlo para que siga detectando solo los estados que queremos. El resultado de la modificación se refleja en la Figura 47.

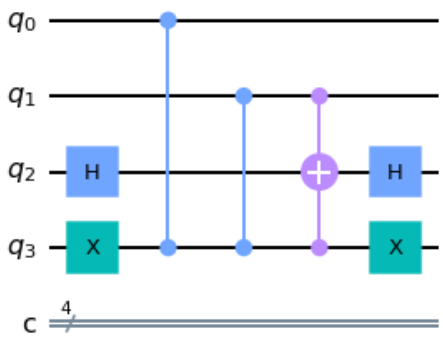


Figura 47. Oráculo para la detección de los estados 001, 010, 101 y 111 con 4 cúbits.

Con el oráculo modificado, y el circuito listo para probar (véase Figura 48), ejecutamos el simulador y veremos que efectivamente los resultados que deseamos ahora sí aparecen (véase Figura 49).

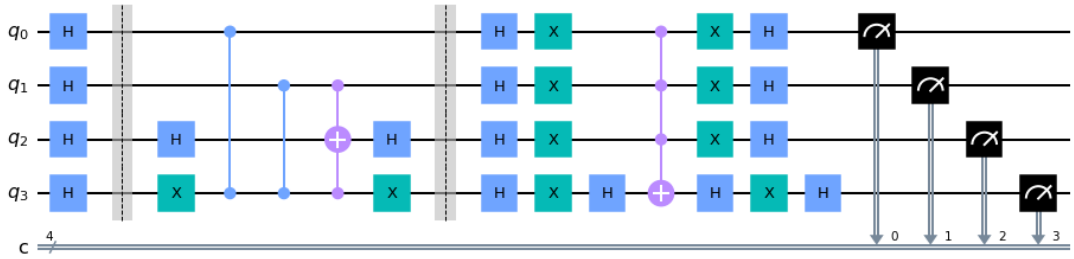


Figura 48. Circuito para buscar los estados 001, 010, 101 y 111 con cuatro cúbits.

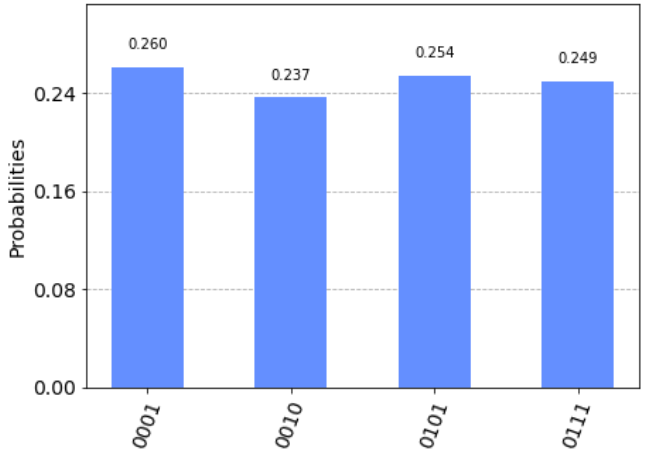


Figura 49. Resultado de ejecutar el algoritmo de Grover para 4 estados y 4 cúbits.

Ahora nuestra solución al problema tiene un 100% de probabilidad de éxito, debido a que usamos el número de iteraciones correcto, especificado por

$$r = \left\lfloor \frac{\pi\sqrt{16/4}}{4} \right\rfloor = 1$$

y mantenemos que

$$4 \leq \frac{16}{2} \Rightarrow 4 \leq 8$$

lo cual nos garantiza una alta probabilidad de éxito.

Si intentamos ejecutar este algoritmo en un computador cuántico real, veremos que las soluciones están alejadas de ser correctas (véase Figura 50). Se puede observar un ligero incremento de la probabilidad de aquellos estados que buscamos, pero no son suficientes para que haya un alto porcentaje de éxito. Esto es porque al computador cuántico le afecta principalmente el ruido que se proporciona durante los cálculos, y es necesario un código reductor de ruido para aumentar las probabilidades de éxito en estos casos.

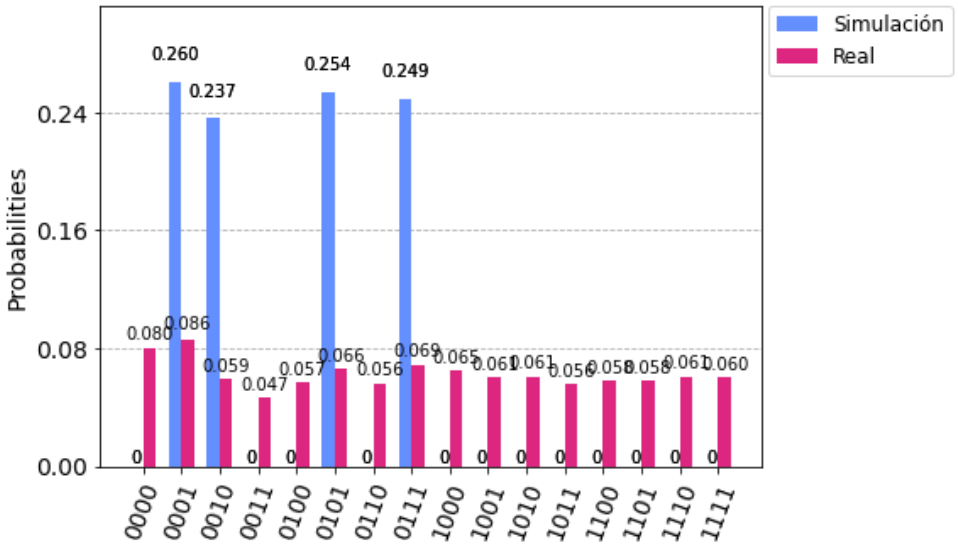


Figura 50. Comparativa de los resultados de ejecutar la búsqueda en el simulador y en el computador cuántico.

Capítulo 5 Conclusiones y líneas futuras

El objetivo principal de este Trabajo de Fin de Grado ha sido realizar una introducción a la computación cuántica, para adquirir una base que después nos permitió entender y trabajar con el algoritmo de búsqueda cuántico llamado Grover. Así, estudiamos sus características, y realizamos una serie de implementaciones que dejan al descubierto algunas de las problemáticas que se deben tener en cuenta.

Es evidente que la computación cuántica supondrá un punto de inflexión tanto en la informática como en la tecnología del futuro. No obstante, todavía nos queda camino por recorrer para que los ordenadores cuánticos dejen atrás al mundo de la computación clásica.

El algoritmo estudiado en este trabajo resulta bastante potente, si bien está muy limitado por el número de cúbits, el espacio de búsqueda y el número de elementos que se deseen encontrar. Esto, sumado a los problemas que poseen los ordenadores cuánticos actuales, hace inviable su uso en problemas reales. En el apartado 4.4.2 se puede observar cómo, a pesar de que tanto el oráculo como la puerta de Grover son correctos, en un ordenador real el ruido evita que aumenten como se espera las probabilidades de las soluciones que se buscan.

En cuanto a líneas de trabajo futuras, este TFG se inició con la intención de comenzar la investigación de un algoritmo cuántico de búsqueda, similar a Grover, que permita realizar el test de dominancia en los algoritmos de Optimización Combinatoria Multiobjetivo. Este se define de la manera siguiente:

Dado un conjunto L de puntos en $\mathbb{R}^p$ y $a \in \mathbb{R}^p$, ¿existe un $b \in L$ tal que $b \leq a \Leftrightarrow b_i \leq a_i, \forall i \in \{1, \dots, p\}$?

Enfocados en eso, los siguientes pasos deberían ser, por un lado, resolver la problemática de realizar una búsqueda sobre un subconjunto de elementos, es decir, codificar la entrada de $|L|$ vectores de tamaño p almacenando enteros que sean equiprobables y cuyas amplitudes al cuadrado sumen 1. Posteriormente, para un vector a de p enteros, el oráculo debería cambiar la fase para los elementos de L tales que $b \leq a$. Se aplicaría el amplificador o puerta de Grover. Se debería finalmente, ser capaz de detectar cuando Grover no ha podido encontrar una solución viable, es decir, cuando la respuesta es negativa (no hay elemento que domina a a).

Para el subconjunto de elementos como entrada para Grover, se puede plantear el concepto de una *control oracle*, que solo aplicaría el oráculo a aquellos estados que pertenecen al conjunto. Sin embargo, seguiríamos teniendo que afrontar el problema de que el oráculo debe ser diseñado conociendo previamente tanto el tamaño del subconjunto como los estados que se buscan.

Particularmente, para el caso de posible respuesta negativa, una posible implementación podría ser añadir un cúbit exclusivo, que inicialmente está en estado $|0\rangle$ (no se le aplica puerta Hadamard porque no pertenece al flujo principal del algoritmo) y que si, tras ejecutar el oráculo, detectamos que todos los estados participantes del subconjunto tienen fase cero, o no están rotados π radianes, aplicar una puerta X a dicho cúbit para pasarlo al estado $|1\rangle$. Se debe intentar que este estado acabe con una amplitud lo más cercana a uno posible para que sea siempre este el que se muestre cuando se mida el resultado.

Capítulo 6 Summary and Conclusions

The main purpose of this Final Degree Project has been to make an introduction to quantum computation, to finally reach an elementary understanding of this paradigm to work with this quantum search algorithm called Grover. Thereby, we studied its features, and developed a set of implementations to test its advantages and disadvantages.

It is obvious that quantum computation will make an inflection point both in computer science and technology. However, there is a long way to go to let quantum computers get away with classical computing.

The algorithm studied in this project is quite powerful, but it is limited to the number of qubits, the search space and the number of elements we desire to find. Thus, and beside quantum computers problems they have, makes Grover non-viable in a real environment. In section 4.4.2 we can see, despite both oracle and diffuser works properly, in a real quantum computer noise makes this implementation useless.

As possible future studies, this TFG was created to begin research about a quantum search algorithm, similar to Grover, who completes the Combinatorial Multiobjective Optimization dominance test. This is defined on this way:

Let L to be a set of points in $\mathbb{R}^p$ and let $a \in \mathbb{R}^p$. Does exists a $b \in L$ such that $b \leq a \Leftrightarrow b_i \leq a_i, \forall i \in \{1, \dots, p\}$?

Focused on that, next steps should be, on the one hand, find a way to make a search using a subset of elements, that is to say, encode the input of $|L|$ vectors of size p storing integers who are equiprobable and such the sum of squared amplitudes equals 1. Subsequently, for a p integers vector a, the oracle should change the phase for elements of L such that $b \leq a$. Then, we apply the amplifier. Finally, we should find a way to detect whether there is a viable solution or not. In other way, when exists a negative response (there are no elements who master a).

Related to the elements subset as input for Grover, we can theorize of a *control oracle*. This gate only applies the Oracle circuito in those states who belongs to the subset. Nevertheless, we will follow having to struggle with the problem that the oracle must be designed knowing both the subset size and the states we are looking for.

In particular, for a possible negative response case, a possible implementation for finding the no solution output, is to add an additional qubit to the algorithm. This qubit will no passed by a Hadamard gate because it is not part of the main flow. Instead, it can start at $|0\rangle$ state and, if after oracle there are no one rotated π radians, apply an X gate to this qubit to change it to $|1\rangle$. We should try to increment the amplitude of this qubits to make sure this qubit appears as $|1\rangle$ when it is necessary.

Capítulo 7 Apéndice con la ejecución de las pruebas de Grover

7.1 Algoritmo para la ejecución de Grover del apartado 4.1

```
/*****************************************************************************
*
* Autor: HIMAR MANUEL BARQUÍN CARRASCO
* Fecha: 07/09/2021
*
* Descripción:
* Código en Python con Qiskit realizado en IBM Quantum Lab para la ejecución del circuito
* ilustrado en el apartado 4.1 .
*
*****************************************************************************/
from qiskit import *
from qiskit.visualization import plot_histogram
from qiskit.tools.monitor import job_monitor

# Puerta de Grover para 2 cubits
def Grover2q ():
    g = QuantumCircuit(2, 2)

    g.h(range(2))

    g.z(range(2))
    g.cz(0,1)

    g.h(range(2))

    return g

# Puerta de Grover
def Grover (qubits):
    g = QuantumCircuit(qubits, qubits)
```

```
    g.h(range(qubits))
    g.x(range(qubits))

    g.h(qubits - 1)
    g.mct(list(range(qubits - 1)), qubits - 1, 0)
    g.h(qubits - 1)

    g.x(range(qubits))
    g.h(range(qubits))

    return g

# Mostramos la puerta de Grover para 2 Qubits
Grover2q().draw()

# Función para la creación del circuito de Grover
def grover_algorithm (detector, qubits, repetitions = 1):
    circ = QuantumCircuit(qubits,qubits)
    circ.h(range(qubits))

    for i in range(repetitions):
        circ.barrier(range(qubits))
        circ = circ + detector
        circ.barrier(range(qubits))
        circ = circ + (Grover(qubits) if qubits != 2 else Grover2q())

    circ.measure(range(qubits), range(qubits))
    return circ

# Este detector pone en fase pi (invierte el signo de la amplitud) del cúbit 11
Detector = QuantumCircuit(2,2)

Detector.cz(0,1)

# Mostramos el detector
Detector.draw()

# Creamos el circuito del algoritmo de Grover
circ = grover_algorithm(Detector, 2)
circ.draw()
```

```
# Realizamos 2048 ejecuciones del algoritmo en un simulador y mostramos el resultado
backend = Aer.get_backend("qasm_simulator")
job = execute(circ, backend, shots = 2048)
result = job.result().get_counts(circ)
plot_histogram(result)

# Realizamos 2048 ejecuciones del algoritmo de un computador cuántico real y mostramos una
# comparativa de los resultados del simulador con los proporcionados por el computador cuántico
# real
IBMQ.load_account()
provider = IBMQ.get_provider(hub='ibm-q')
device = provider.get_backend('ibmq_manila')
```

7.2 Algoritmo para la ejecución de Grover del apartado 4.2

```
/*******************************************************************************
*
* Autor: HIMAR MANUEL BARQUÍN CARRASCO
* Fecha: 07/09/2021
*
* Descripción:
* Código en Python con Qiskit realizado en IBM Quantum Lab para la ejecución del circuito
* ilustrado en el apartado 4.2 .
*
*******************************************************************************/
from qiskit import *
from qiskit.visualization import plot_histogram
from qiskit.tools.monitor import job_monitor

# Puerta de Grover para 2 cubits
def Grover2q ():
    g = QuantumCircuit(2, 2)

    g.h(range(2))

    g.z(range(2))
    g.cz(0,1)

    g.h(range(2))

    return g

# Puerta de Grover
```

```
def Grover (qubits):
    g = QuantumCircuit(qubits, qubits)

    g.h(range(qubits))
    g.x(range(qubits))

    g.h(qubits - 1)
    g.mct(list(range(qubits - 1)), qubits - 1, 0)
    g.h(qubits - 1)

    g.x(range(qubits))
    g.h(range(qubits))

    return g

# Mostramos la puerta de Grover para 3 Qubits
Grover(3).draw()

# Función para la creación del circuito de Grover
def grover_algorithm (detector, qubits, repetitions = 1):
    circ = QuantumCircuit(qubits,qubits)
    circ.h(range(qubits))

    for i in range(repetitions):
        circ.barrier(range(qubits))
        circ = circ + detector
        circ.barrier(range(qubits))
        circ = circ + (Grover(qubits) if qubits != 2 else Grover2q())

    circ.measure(range(qubits), range(qubits))
    return circ

# Este detector pone en fase pi (invierte el signo de la amplitud) de los cúbits 101 y 111
Detector = QuantumCircuit(3,3)

Detector.cz(0,2)

# Mostramos el detector
Detector.draw()

# Creamos el circuito del algoritmo de Grover
circ = grover_algorithm(Detector, 3)
```

```
circ.draw()

# Realizamos 2048 ejecuciones del algoritmo en un simulador y mostramos el resultado
backend = Aer.get_backend("qasm_simulator")
job = execute(circ, backend, shots = 2048)
result = job.result().get_counts(circ)
plot_histogram(result)

# Realizamos 2048 ejecuciones del algoritmo de un computador cuántico real y mostramos una
# comparativa de los resultados del simulador con los proporcionados por el computador cuántico
# real
IBMQ.load_account()
provider = IBMQ.get_provider(hub='ibm-q')
device = provider.get_backend('ibmq_manila')
```

7.3 Algoritmo para la ejecución de Grover del apartado 4.3

```
/*******************************************************************************
*
* Autor: HIMAR MANUEL BARQUÍN CARRASCO
* Fecha: 07/09/2021
*
* Descripción:
* Código en Python con Qiskit realizado en IBM Quantum Lab para la ejecución del circuito
* ilustrado en el apartado 4.3 .
*
*******************************************************************************/
from qiskit import *
from qiskit.visualization import plot_histogram
from qiskit.tools.monitor import job_monitor

# Puerta de Grover para 2 cubits
def Grover2q ():
   g = QuantumCircuit(2, 2)

   g.h(range(2))

   g.z(range(2))
   g.cz(0,1)

   g.h(range(2))

   return g
```

```
# Puerta de Grover
def Grover (qubits):
    g = QuantumCircuit(qubits, qubits)

    g.h(range(qubits))
    g.x(range(qubits))

    g.h(qubits - 1)
    g.mct(list(range(qubits - 1)), qubits - 1, 0)
    g.h(qubits - 1)

    g.x(range(qubits))
    g.h(range(qubits))

    return g

# Mostramos la puerta de Grover para 3 Qubits
Grover(3).draw()

# Función para la creación del circuito de Grover
def grover_algorithm (detector, qubits, repetitions = 1):
    circ = QuantumCircuit(qubits,qubits)
    circ.h(range(qubits))

    for i in range(repetitions):
        circ.barrier(range(qubits))
        circ = circ + detector
        circ.barrier(range(qubits))
        circ = circ + (Grover(qubits) if qubits != 2 else Grover2q())

    circ.measure(range(qubits), range(qubits))
    return circ

# Este detector pone en fase pi (invierte el signo de la amplitud) del cúbit 010
Detector = QuantumCircuit(3,3)


Detector.x(0)
Detector.x(2)


Detector.h(1)
```

```
Detector.mct([0,2], 1, 0)
Detector.h(1)


Detector.x(0)
Detector.x(2)


# Mostramos el detector
Detector.draw()


# Creamos el circuito de Grover, con dos repeticiones
# de la fase dos del algoritmo
circ = grover_algorithm(Detector, 3, 2)
circ.draw()


# Realizamos 2048 ejecuciones del algoritmo en un simulador y mostramos el resultado
backend = Aer.get_backend("qasm_simulator")
job = execute(circ, backend, shots = 2048)
result = job.result().get_counts(circ)
plot_histogram(result)

# Realizamos 2048 ejecuciones del algoritmo de un computador cuántico real y mostramos una
# comparativa de los resultados del simulador con los proporcionados por el computador cuántico
# real
IBMQ.load_account()
provider = IBMQ.get_provider(hub='ibm-q')
device = provider.get_backend('ibmq_manila')
```

7.4 Algoritmo para la ejecución de Grover del apartado 4.4

7.4.1 Utilizando tres cúbits

```
/*******************************************************************************
*
* Autor: HIMAR MANUEL BARQUÍN CARRASCO
* Fecha: 07/09/2021
*
* Descripción:
* Código en Python con Qiskit realizado en IBM Quantum Lab para la ejecución del circuito
* ilustrado en el apartado 4.4.1 .
*
*******************************************************************************/
```

```
from qiskit import *
from qiskit.visualization import plot_histogram
from qiskit.tools.monitor import job_monitor

# Puerta de Grover para 2 cubits
def Grover2q ():
    g = QuantumCircuit(2, 2)

    g.h(range(2))

    g.z(range(2))
    g.cz(0,1)

    g.h(range(2))

    return g

# Puerta de Grover
def Grover (qubits):
    g = QuantumCircuit(qubits, qubits)

    g.h(range(qubits))
    g.x(range(qubits))

    g.h(qubits - 1)
    g.mct(list(range(qubits - 1)), qubits - 1, 0)
    g.h(qubits - 1)

    g.x(range(qubits))
    g.h(range(qubits))

    return g

# Mostramos la puerta de Grover para 3 Qubits
Grover(3).draw()

# Función para la creación del circuito de Grover
def grover_algorithm (detector, qubits, repetitions = 1):
    circ = QuantumCircuit(qubits,qubits)
    circ.h(range(qubits))

    for i in range(repetitions):
```

```
        circ.barrier(range(qubits))
        circ = circ + detector
        circ.barrier(range(qubits))
        circ = circ + (Grover(qubits) if qubits != 2 else Grover2q())

    circ.measure(range(qubits), range(qubits))
    return circ

# Este detector pone en fase pi (invierte el signo de la amplitud) de los cúbits
# 001, 010, 101 y 111
Detector = QuantumCircuit(3,3)


Detector.z(0)

Detector.z(1)


Detector.cz(1,2)


# Mostramos el detector
Detector.draw()


# Creamos el circuito del algoritmo de Grover
circ = grover_algorithm(Detector, 3)

circ.draw()


# Realizamos 2048 ejecuciones del algoritmo en un simulador y mostramos el resultado
backend = Aer.get_backend("qasm_simulator")
job = execute(circ, backend, shots = 2048)
result = job.result().get_counts(circ)
plot_histogram(result)

# Realizamos 2048 ejecuciones del algoritmo de un computador cuántico real y mostramos una
# comparativa de los resultados del simulador con los proporcionados por el computador cuántico
# real
IBMQ.load_account()
provider = IBMQ.get_provider(hub='ibm-q')
device = provider.get_backend('ibmq_manila')
```

7.4.2 Utilizando cuatro cúbits

```
/*****************************************************************************
*
* Autor: HIMAR MANUEL BARQUÍN CARRASCO
* Fecha: 07/09/2021
*
* Descripción:
* Código en Python con Qiskit realizado en IBM Quantum Lab para la ejecución del circuito
* ilustrado en el apartado 4.4.2 .
*
*****************************************************************************/
from qiskit import *
from qiskit.visualization import plot_histogram
from qiskit.tools.monitor import job_monitor

# Puerta de Grover para 2 cubits
def Grover2q ():
   g = QuantumCircuit(2, 2)

   g.h(range(2))

   g.z(range(2))
   g.cz(0,1)

   g.h(range(2))

   return g

# Puerta de Grover
def Grover (qubits):
   g = QuantumCircuit(qubits, qubits)

   g.h(range(qubits))
   g.x(range(qubits))

   g.h(qubits - 1)
   g.mct(list(range(qubits - 1)), qubits - 1, 0)
   g.h(qubits - 1)

   g.x(range(qubits))
   g.h(range(qubits))
```

```
    return g

# Mostramos la puerta de Grover para 3 Qubits
Grover(4).draw()

# Función para la creación del circuito de Grover
def grover_algorithm (detector, qubits, repetitions = 1):
    circ = QuantumCircuit(qubits,qubits)
    circ.h(range(qubits))

    for i in range(repetitions):
        circ.barrier(range(qubits))
        circ = circ + detector
        circ.barrier(range(qubits))
        circ = circ + (Grover(qubits) if qubits != 2 else Grover2q())

    circ.measure(range(qubits), range(qubits))
    return circ

# Este detector pone en fase pi (invierte el signo de la amplitud) de los cúbits
# 001, 010, 101 y 111
Detector = QuantumCircuit(4,4)


Detector.x(3)


Detector.cz(3, 0)
Detector.cz(3, 1)


Detector.h(2)
Detector.mct([3,1], 2, 0)
Detector.h(2)


Detector.x(3)


# Mostramos el detector
Detector.draw()


# Creamos el circuito del algoritmo de Grover
circ = grover_algorithm(Detector, 4)
```

```
circ.draw()


# Realizamos 2048 ejecuciones del algoritmo en un simulador y mostramos el resultado
backend = Aer.get_backend("qasm_simulator")
job = execute(circ, backend, shots = 2048)
result = job.result().get_counts(circ)
plot_histogram(result)

# Realizamos 2048 ejecuciones del algoritmo de un computador cuántico real y mostramos una
# comparativa de los resultados del simulador con los proporcionados por el computador cuántico
# real
IBMQ.load_account()
provider = IBMQ.get_provider(hub='ibm-q')
device = provider.get_backend('ibmq_manila')
```

Bibliografía

[1] L. K. Grover, «Quantum Mechanics helps in searching for a needle in a haystack,» *Physical Review Letters,* vol. 79, nº 2, pp. 325-328, 1997.

[2] P. W. Shor, «Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer,» *SIAM J.Sci.Statist.Comput. 26 (1997) 1484,* p. 28, 1994.

[3] F. Arute, K. Arya y R. Babbush, «Quantum supremacy using a programmable superconducting processor,» *Nature,* pp. 505-510, 23 October 2019.

[4] P. Benioff, «The computer as a physical system: A microscopic quantum,» *Journal of Statistical Physics,* vol. 22, nº 5, pp. 563-591, 1980.

[5] P. W. Shor, «Scheme for reducing decoherence in quantum computer memory,» *The American Physical Society,* vol. 52, nº 4, 17 May 1995.

[6] W. K. Wootters y W. H. Zurek, «The no-cloning theorem,» *Physics Today,* February 2009.

[7] K. Wickens, «AMD files teleportation patent to supercharge quantum computing,» *PCGamer,* 3 September 2021.

[8] «Qiskit,» [En línea]. Available: https://qiskit.org/. [Último acceso: 02 September 2021].

[9] «IBM Quantum Experience,» [En línea]. Available: https://quantum-computing.ibm.com/. [Último acceso: 2 September 2021].

[10] Wikipedia, «Cúbit,» [En línea]. Available: https://es.wikipedia.org/wiki/C%C3%BAbit. [Último acceso: 1 September 2021].

[11] A. M. Téllez, «La Mecánica Cuántica,» 11 August 2009. [En línea]. Available: http://la-mecanica-cuantica.blogspot.com/2009/08/el-espacio-de-hilbert.html. [Último acceso: 2 September 2021].

[12] Wikipedia, «Esfera de Bloch,» [En línea]. Available: https://es.wikipedia.org/wiki/Esfera_de_Bloch [Último acceso: 1 September 2021].

[13] Wikipedia, «Base de Bell,» [En línea]. Available: https://es.wikipedia.org/wiki/Base_de_Bell. [Último acceso: 2 September 2021].

[14] S. Campillo, «Entrelazamiento, así funcionan la computación y la teleportación cuántica,» *Hipertextual,* 1 September 2015.

[15] Wikipedia, «Búsqueda Lineal,» [En línea]. Available: https://es.wikipedia.org/wiki/B%C3%BAsqueda_lineal. [Último acceso: 7 Septembe 2021].

[16] Wikipedia, «Tabla hash,» [En línea]. Available: https://es.wikipedia.org/wiki/Tabla_hash. [Último acceso: 7 September 2021].

[17] G. Chen, S. A. Fulling y M. Scully, «Grover's Algorithm for Multiobject Search in Quantum Computing,» October 1999.

[18] Quantiki, «Grover's search algorithm,» *Quantiki,* 3 November 2015.

[19] Arriopolis, «Quantum Computing, Stack Exchange,» 30 April 2018. [En línea]. Available: https://quantumcomputing.stackexchange.com/questions/1939/in-grovers-algorithm-why-does-the-optimal-number-of-iterations-involve-a-floor. [Último acceso: 7 September 2021].

[20] cnada y P_Gate, «Quantum Computing, Stack Exchange,» 27 May 2019. [En línea]. Available: https://quantumcomputing.stackexchange.com/questions/6238/success probability-in-grovers-algorithm-where-there-are-multiple-targets. [Último acceso: 7 September 2021].

[21] E. Borbely, «Grover search algorithm,» Budapest, Hungary, Technological Universit Budapest, pp. 4-11.

[22] Qiskit, «qiskit.providers.aer.QasmSimulator,» [En línea]. Available: https://qiskit.org/documentation/stubs/qiskit.providers.aer.QasmSimulator.html. [Último acceso: 3 September 2021].

[23] Qiskit, «qiskit.providers.ibmq.IBMQBackend,» [En línea]. Available: https://qiskit.org/documentation/stubs/qiskit.providers.ibmq.IBMQBackend.html. [Último acceso: 3 September 2021].