



**UNIVERSIDAD
DE LA RIOJA**

Facultad de Ciencia y Tecnología

TRABAJO DE FIN DE GRADO

Grado en Matemáticas

**Fundamentos matemáticos de algoritmos de
búsqueda en computación cuántica.**

Realizado por:

Jesús Gil Jiménez

Tutelado por:

Jesús María Aransay Azofra

Eduardo Sáenz de Cabezón Irigaray

28 de octubre de 2022

Resumen

En este trabajo se pretende dar una introducción a la computación cuántica, un paradigma de computación del que cada vez se habla más y poco a poco va ganando protagonismo. Se presentarán las bases matemáticas de este paradigma hasta llegar a uno de los algoritmos de búsqueda más importantes de la computación cuántica: el algoritmo de Grover. Primero introduciremos los conceptos básicos necesarios del álgebra lineal. Una vez hecho esto, pasaremos a relacionar esos conceptos algebraicos con los fundamentos de la computación cuántica, pasando primero por la computación clásica. Partiendo desde el qubit, la unidad básica en computación cuántica, explicaremos cómo se representan los estados y qué tipo tienen y como se construyen las operaciones de este paradigma hasta llegar a los circuitos y algoritmos cuánticos. Una vez presentado el paradigma y toda la base matemática que hay detrás, analizaremos el algoritmo de búsqueda de Grover. Demostraremos matemáticamente su complejidad, explicaremos su funcionamiento y lo llevaremos a un simulador cuántico donde implementaremos diferentes casos en los que aplicar el algoritmo escribiendo pequeños programas en Python. Una vez aquí, concluiremos todo lo estudiado y veremos realmente la eficacia de la computación cuántica y de este algoritmo en concreto.

Abstract

In this project we pretend to give an introduction to quantum computing, a paradigm that nowadays is more and more talked about and is gaining protagonism. We'll present the mathematical basis for this paradigm until we get to one of the most important search algorithms in quantum computing: Grover's algorithm. First we'll introduce the necessary basic concepts of linear algebra. Once we've done this we'll relate these concepts with the quantum computing foundations, first presenting classic computing. Starting from the qubit, the basic quantum unit, we'll explain how states are represented and what type of quantum operations exist and how they are done until we get to quantum algorithms and circuits. Once we've presented the paradigm and all the mathematical basis behind it, we'll analyze the Grover's search algorithm. We will mathematically prove its complexity, we'll explain it and we'll take it to a quantum simulator where we'll implement different cases where we can apply the algorithm writing small programs written in Python. Once we get here we'll conclude everything we've studied and we'll really see the effectiveness of quantum computing and this algorithm in particular.

Índice general

1. Introducción a la computación cuántica	1
1.1. Fundamentos matemáticos de Álgebra Lineal	1
1.1.1. Espacios de Hilbert	1
1.1.2. Matrices	2
1.1.3. Espacios complejos y productos escalares	4
1.2. El modelo de computación cuántica	4
1.2.1. El qubit, el espacio y los estados	4
1.2.2. Las operaciones	7
1.2.3. Funciones booleanas	8
1.2.4. Funciones booleanas factibles	10
1.2.5. Representación cuántica de funciones booleanas	11
1.2.6. Operaciones cuánticas factibles	14
1.2.7. Circuitos cuánticos	16
1.2.8. Matrices especiales	17
2. Algoritmo de Grover	25
2.1. Introducción	25
2.2. ¿Cómo funciona el algoritmo?	26
2.2.1. Contexto previo	26
2.2.2. Inicialización de estados	27
2.2.3. Oráculo de Grover	27
2.2.4. El algoritmo	28
2.3. Interpretación geométrica	30
2.3.1. Número de iteraciones	32
3. Implementación del algoritmo	37
3.1. Implementación con 2 qubits	37
3.1.1. Implementación del algoritmo en Qiskit	40
3.2. Implementación con 3 qubits	41
3.2.1. Implementación del algoritmo en Qiskit	43

3.2.2. Otro ejemplo con más iteraciones	44
3.3. Conclusiones	46
4. Retrospectiva y conclusiones finales	47
A. Implementación en Qiskit con 2 qubits	51
B. Implementación en Qiskit con 3 qubits	55
C. Implementación en Qiskit con 3 qubits y más iteraciones	59

Capítulo 1

Introducción a la computación cuántica

En este capítulo introduciremos el modelo de computación cuántica. Para ello, antes de hablar de cuántica es necesario presentar algunos conceptos de Álgebra Lineal que constituyen los cimientos del modelo en cuestión.

1.1. Fundamentos matemáticos de Álgebra Lineal

Definición 1.1. *Un **vector** a de dimensión N es una lista ordenada de n valores. Denotaremos un vector a de dimensión N como sigue:*

$$\begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{N-1} \end{pmatrix}$$

En esta sección, y a lo largo de esta memoria, seguiremos la notación funcional propuesta en ‘Quantum Algorithms via Linear Algebra’[1]. De modo que denotaremos a_k como $a(k)$.

1.1.1. Espacios de Hilbert

Un espacio de Hilbert es una generalización de un espacio Euclídeo. Un espacio H^N es un espacio de Hilbert de dimensión N donde los elementos son vectores a de dimensión N . En este espacio tenemos definida una norma y los vectores se suman y se multiplican por escalares de la forma usual. Sean a, b dos vectores de dimensión N y c una constante real:

$$\begin{pmatrix} a(0) \\ \vdots \\ a(N-1) \end{pmatrix} + \begin{pmatrix} b(0) \\ \vdots \\ b(N-1) \end{pmatrix} = \begin{pmatrix} a(0) + b(0) \\ \vdots \\ a(N-1) + b(N-1) \end{pmatrix}$$

$$c \begin{pmatrix} a(0) \\ \vdots \\ a(N-1) \end{pmatrix} = \begin{pmatrix} ca(0) \\ \vdots \\ ca(N-1) \end{pmatrix}$$

Definición 1.2. La norma o longitud de un vector a es:

$$\|a\| = \left(\sum_{k=0}^{N-1} a(k)^2 \right)^{1/2}$$

Diremos que un vector es **unitario** si su norma es 1.

En los espacios de Hilbert consideramos el producto cartesiano ordinario y el producto tensorial.

Definición 1.3. El producto **cartesiano** ordinario de un espacio de Hilbert H_1 de dimensión m y un espacio de Hilbert H_2 de dimensión n es el espacio de Hilbert de dimensión $(m+n)$ obtenido concatenando los vectores del primer espacio con los del segundo, por tanto los vectores tienen la forma $a(i)$ con $i \in [0, m+n-1]$.

Definición 1.4. El producto **tensorial** $H_1 \otimes H_2$ contiene los vectores de la forma $a(k)$ donde $k \in [0, m*n-1]$. De hecho, cada k tiene una correspondencia 1-1 con cada par (i, j) donde $i \in [0, m-1]$ y $j \in [0, n-1]$.

El producto tensorial de dos vectores a y b es el vector $c = a \otimes b$ definido por

$$c(ij) = a(i)b(j)$$

Definición 1.5. Decimos que un vector es **separable** si es el producto tensorial de otros dos vectores. Por el contrario, si no podemos expresarlo como producto tensorial de dos vectores, decimos que está **entrelazado**.

Esta noción es muy importante para cuando más adelante hablemos de los diferentes estados que encontramos en el modelo de computación cuántica.

1.1.2. Matrices

Las matrices representan operadores lineales en los espacios de Hilbert. Una matriz $m*n$ representa un operador lineal de un espacio H_1 de dimensión m en un espacio H_2 de dimensión n . Podemos sumarlas (lo que equivale a sumar funciones), multiplicarlas si es posible (lo que equivale a la composición de funciones) y por supuesto usarlas para operar con vectores. Si U es una matriz $m*n$, utilizaremos la siguiente notación:

$$U = \begin{pmatrix} U[0,0] & U[0,1] & \vdots & U[0,n-1] \\ U[1,0] & U[1,1] & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ U[m-1,0] & U[m-1,1] & \vdots & U[m-1,n-1] \end{pmatrix}$$

Una de las propiedades clave de las matrices es que definen operaciones lineales:

Sean U una matriz $m \times n$, a, b vectores de dimension n :

$$U(a + b) = Ua + Ub$$

El hecho de que todas nuestras transformaciones sean lineales es lo que hace que los algoritmos cuánticos sean tan diferentes de los clásicos, pero esto lo veremos más adelante.

Definición 1.6. La **traspuesta** de una matriz $U = U[i, j]$ es la matriz denotada por U^T tal que:

$$U^T = U^T[i, j] = U[j, i]$$

Definición 1.7. Decimos que una matriz U de dimensión $n \times n$ es **unitaria** en \mathbb{R}^n si cumple $U^T U = I_n$, entendiendo por I_n la matriz identidad de dimensión n .

Podemos dar una caracterización de matriz unitaria basada en la noción de **ortogonalidad**.

Definición 1.8. Decimos que dos vectores a y b de dimensión m son **ortogonales** si su producto escalar es cero:

$$\langle a, b \rangle = \sum_{k=0}^{m-1} a(k)b(k) = 0$$

Lema 1.1. Una matriz U es **unitaria** si cada fila es un vector unitario y sus filas son ortogonales dos a dos.

Entre las matrices unitarias podemos encontrar las **matrices de permutación** (por ejemplo, I , X), que son matrices cuadradas cuyas filas y columnas tienen todo ceros excepto un único 1. Todas las matrices de permutación son unitarias.

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

Lema 1.2. Si U es una matriz unitaria y a es un vector, entonces $\|Ua\| = \|a\|$.

Visto lo anterior es importante hablar ahora de la **esfera unidad**. El modelo de computación cuántica está basado en matrices y vectores unitarios. Las matrices representarán las operaciones y los vectores los estados. Por esto, gracias a la propiedad anterior nos aseguramos de que los vectores con que trabajemos no se saldrán nunca de la esfera unidad, que es el conjunto S en el que todos los vectores del espacio de Hilbert que consideremos tienen norma uno, $S = \{a \in H^N : \|a\| = 1\}$.

1.1.3. Espacios complejos y productos escalares

Para describir ciertos algoritmos cuánticos necesitamos usar vectores y matrices de números complejos. En este caso, todas las definiciones son idénticas a las dadas anteriormente excepto por las nociones de matriz traspuesta y producto escalar. Para ambas nociones necesitamos la operación **conjugación**:

Definición 1.9. Sea $z = x + iy$ donde $x, y \in \mathbb{R}$ y $i = \sqrt{-1}$, el **conjugado** de z es $x - iy$, denotado por \bar{z} .

Definición 1.10. La matriz **adjunta** de una matriz U es la matriz U^* tal que:

$$U^* = U^*[i, j] = \overline{U[j, i]}$$

Donde \overline{U} denota la matriz en la que a cada entrada de U se le aplica el conjugado. Notemos que la matriz adjunta es la traspuesta de la conjugada (o la conjugada de la traspuesta).

$$U^* = \overline{U^T}$$

En el caso complejo, para definir una matriz unitaria, sustituimos la noción de matriz traspuesta por la de adjunta. Por tanto una matriz U de números complejos es **unitaria** si $UU^* = I$. También cambia la definición de producto escalar tal que el producto escalar de dos vectores a y b de dimensión m de números complejos está definido como:

$$\sum_{k=0}^{m-1} \overline{a(k)} b(k) = 0$$

Notar que como el conjugado de un número real es el propio número, estos conceptos son los mismos que los dados anteriormente para matrices y vectores reales. Utilizaremos la misma notación para el caso complejo que para el caso real, el contexto aclarará en qué caso estamos. Tras esta introducción de conceptos de Álgebra Lineal, introduciremos el modelo de computación cuántica.

1.2. El modelo de computación cuántica

1.2.1. El qubit, el espacio y los estados

El qubit, abreviación de ‘quantum bit’ es la unidad de información fundamental utilizada en computación cuántica. Podemos verlo como una generalización de un bit clásico en computación cuántica.

Notación algebraica

En primer lugar hay que entender que el estado de los sistemas cuánticos siempre será descrito por un vector unitario a sobre un espacio vectorial fijo de dimensión $N = 2^n$, siendo n el número de qubits. Luego el estado siempre es representado por un vector

$$a = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{N-1} \end{pmatrix}$$

donde cada entrada a_k es real o compleja dependiendo del espacio en el que nos encontremos, pues trabajaremos con espacios de Hilbert H^N . Cada entrada del vector se dice que es una **amplitud**.

En primer lugar nos encontramos con los **estados básicos**, los cuales forman una base compuesta por las diferentes configuraciones que podamos observar. En la **base estándar** los estados básicos son denotados e_k (son los vectores de la base canónica a la que estamos acostumbrados), cuyas entradas son todas 0 excepto 1 en la posición k , con k en $[0, N - 1]$.

En computación cuántica nos encontramos con estados en **superposición**, que son una combinación lineal de estados básicos, donde los cuadrados absolutos de los coeficientes (amplitudes) que multiplican a cada e_k suman 1. De ahí que el vector a sea unitario.

Por ejemplo, tomando $k = 2$ (por comodidad, es análogo para cualquier k natural), un estado en superposición será

$$a_0 \begin{pmatrix} 1 \\ 0 \end{pmatrix} + a_1 \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

con $|a_0|^2 + |a_1|^2 = 1$. Aquí vemos la diferencia más importante entre un qubit y un bit clásico. Mientras que un bit clásico o bien vale 0 o bien vale 1, un qubit puede *estar en varios estados a la vez* (en superposición de estados). En computación cuántica uno no puede saber cual es el estado de un qubit a no ser que lo **mida**. Cualquier medida de un qubit hace que se altere inevitablemente su estado, rompiendo la superposición y, por tanto, pasando a un estado básico. Las amplitudes que hemos mencionado, es decir las entradas del vector que representa el estado del qubit, nos indican la probabilidad con la que un estado en superposición colapsará a un estado básico. Al observar un estado en superposición, colapsará a un estado básico e_k con probabilidad $|a_k|^2$. En el caso anterior, colapsará a e_0 con probabilidad $|a_0|^2$ y a e_1 con probabilidad $|a_1|^2$.

Notación bra-ket

El estado de un qubit viene representado por:

$$|\phi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{1.1}$$

En la llamada *notación bra-ket*, $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ y $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$, son abreviaturas para representar los estados básicos. Decimos que este tipo de estados son estados *ket*. La ecuación 1.1 expresa realmente el vector bidimensional de números complejos $\begin{pmatrix} \alpha \\ \beta \end{pmatrix}$, siendo α y β las amplitudes de cada estado.

Dado un estado ket $|\phi\rangle$, el estado *bra* correspondiente, denotado por $\langle\phi|$ representa el vector traspuesto con entradas conjugadas, tal que si $|\phi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$, tenemos entonces $\langle\phi| = (\alpha^* \ \beta^*)$, donde $*$ denota el conjugado.

Producto escalar y producto exterior

Dados $|\phi\rangle = \alpha|0\rangle + \beta|1\rangle$ y $|\psi\rangle = \gamma|0\rangle + \delta|1\rangle$, el producto escalar entre estos estados cuánticos es denotado $\langle\psi|\phi\rangle$ y viene dado por:

$$\langle\psi|\phi\rangle = (\gamma^* \ \delta^*) \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \gamma^* \alpha + \delta^* \beta$$

Notar que $\langle\psi|\phi\rangle$ es lo mismo que $\langle\phi|\psi\rangle^*$.

De esta forma también podemos definir el producto exterior de dos estados cuánticos $|\phi\rangle$ y $|\psi\rangle$ como $|\phi\rangle\langle\psi|$ tal que:

$$|\phi\rangle\langle\psi| = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} * (\gamma^* \ \delta^*) = \begin{pmatrix} \alpha\gamma^* & \alpha\delta^* \\ \beta\gamma^* & \beta\delta^* \end{pmatrix}$$

[2]

Sistema de qubits

La estructura matemática de un qubit puede generalizarse a sistemas de mayor dimensión $n > 2$.

Un ordenador cuántico contiene muchos qubits, luego es necesario saber cómo construir el estado combinado de un sistema de qubits dados los estados de cada uno de los qubits. Para ello utilizamos el producto tensorial, introducido en la sección de espacios de Hilbert.

Dados $|\phi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$ y $|\phi'\rangle = \begin{pmatrix} \alpha' \\ \beta' \end{pmatrix}$, el estado completo de un sistema compuesto por estos dos qubits viene dado por:

$$|\phi\rangle \otimes |\phi'\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \otimes \begin{pmatrix} \alpha' \\ \beta' \end{pmatrix} = \begin{pmatrix} \alpha\alpha' \\ \alpha\beta' \\ \beta\alpha' \\ \beta\beta' \end{pmatrix}$$

Normalmente obviaremos el símbolo \otimes , simplificando, por ejemplo, $|\phi\rangle \otimes |\phi'\rangle$ como $|\phi\phi'\rangle$, y $|0\rangle \otimes |0\rangle \otimes |0\rangle$ como $|000\rangle$. Para sistemas más grandes, con esta notación y apoyándonos en la propiedad distributiva del producto de Kroenecker, tendremos, por ejemplo, para un sistema de 3 qubits con cada qubit en el estado $|\gamma_j\rangle = \alpha_j|0\rangle + \beta_j|1\rangle$, para $j = 1, 2, 3$, el estado del sistema es:

$$\begin{aligned} |\gamma_1\gamma_2\gamma_3\rangle &= |\gamma_1\rangle \otimes |\gamma_2\rangle \otimes |\gamma_3\rangle \\ &= \alpha_1\alpha_2\alpha_3|000\rangle + \alpha_1\alpha_2\beta_3|001\rangle + \alpha_1\beta_2\alpha_3|010\rangle + \alpha_1\beta_2\beta_3|011\rangle \\ &\quad + \beta_1\alpha_2\alpha_3|100\rangle + \beta_1\alpha_2\beta_3|101\rangle + \beta_1\beta_2\alpha_3|110\rangle + \beta_1\beta_2\beta_3|111\rangle \end{aligned}$$

Un sistema de qubits puede estar en estado **producto** si su estado puede representarse como producto tensorial de los estados de sus componentes, es decir, si el estado del sistema viene dado por un vector *separable*, concepto que intrujimos en la sección 1.1.1. En caso contrario, se dice que está en un estado de **entrelazamiento**, o sea que su vector está *entrelazado*. [3]

Ejemplo 1.1. *Un estado en superposición:*

$$\frac{1}{2}|00\rangle - \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle - \frac{1}{2}|11\rangle = \left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right) \otimes \left(\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle\right)$$

Sin embargo, el siguiente es un estado de entrelazamiento:

$$\frac{1}{2}|00\rangle + \frac{1}{2}|11\rangle$$

Para probarlo se procede por reducción al absurdo. Suponemos que es un estado producto:

$$\frac{1}{2}|00\rangle + \frac{1}{2}|11\rangle = (\alpha_1|0\rangle + \alpha_2|1\rangle) \otimes (\alpha_3|0\rangle + \alpha_4|1\rangle), \alpha_1^2 + \alpha_2^2 = 1 = \alpha_3^2 + \alpha_4^2$$

Desarrollamos:

$$\frac{1}{2}|00\rangle + \frac{1}{2}|11\rangle = \alpha_1\alpha_3|00\rangle + \alpha_1\alpha_4|01\rangle + \alpha_2\alpha_3|10\rangle + \alpha_2\alpha_4|11\rangle$$

Luego, agrupando coeficientes nos queda:

$$\alpha_1\alpha_3 = \frac{1}{2} = \alpha_2\alpha_4$$

$$\alpha_1\alpha_4 = 0 = \alpha_2\alpha_3$$

Lo cual es contradictorio, por lo que el estado inicial está en entrelazamiento.

1.2.2. Las operaciones

El que para un estado a el vector sea unitario, significa que cada estado es un punto sobre la esfera unidad N -dimensional. Las operaciones que trabajaremos siempre mandarán un punto de la esfera unidad a otro punto sobre la esfera unidad. Las operaciones son lineales

sobre toda la esfera unidad, de lo que sigue que deben mandar el espacio de Hilbert H^N a todo H^N . Esto significa que deben ser operaciones **invertibles** lo cual es una de las principales diferencias con el modelo de computación clásica que le dan tanta potencia al modelo cuántico.

Estas operaciones las representaremos con **matrices unitarias**. De esta forma, conservamos la norma (no nos saldremos de la esfera unidad), podremos invertir las operaciones, y serán lineales.

En definitiva, programas basados en este modelo no serán más que composiciones de matrices unitarias, pero claro, en muchos casos, de tamaños desorbitados. Por ello, más adelante definiremos qué operaciones son **factibles** y cuales no lo son.[1]

1.2.3. Funciones booleanas

Una **función booleana** es una aplicación de $\{0, 1\}^n$ a $\{0, 1\}^m$ para n, m naturales. Cuando definimos una función booleana $f(x_1, \dots, x_n) = (y_1, \dots, y_m)$, pensamos en x_i como bits de entrada e y_i como bits de salida.

Las funciones booleanas más básicas tienen $n = 1$, o 2 y $m = 1$, como pueden ser la operación unaria NOT(\neg) (con $n, m = 1$), o las binarias AND(\wedge), OR(\vee ó $+$) y XOR(\oplus) (con $n = 2, m = 1$). Estas funciones binarias también podemos interpretarlas para n mayor que 2:

- AND: Es la función $f(x_1, \dots, x_n)$ igual a 1 si y solo si cada x_i en x_1, \dots, x_n es igual a 1. Por ejemplo:

$$f(1, 1, 1, 1) = 1, f(1, 0, 0, 1) = 0$$

- OR: Es la función $f(x_1, \dots, x_n)$ igual a 1 si por lo menos un x_i en x_1, \dots, x_n es igual a 1. Por ejemplo:

$$f(1, 0, 0) = 1, f(0, 0, 0, 0) = 0$$

- XOR: Es la función $f(x_1, \dots, x_n)$ igual a 1 si el número de unos en x_1, \dots, x_n es impar. Por ejemplo:

$$f(1, 1, 1, 0) = 1, f(1, 1, 0, 0) = 0$$

Además, las funciones booleanas se pueden definir de una forma más visual por sus **tablas de verdad**. Veamos las tablas de verdad de las operaciones que acabamos de comentar:

Ejemplo 1.2. Para $n, m = 1$:

x	$\neg x$
0	1
1	0

Para $n = 2, m = 1$

x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x \vee y$
0	0	0
0	1	1
1	0	1
1	1	1

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Las operaciones binarias pueden aplicarse a parejas de cadenas de bits. Si tenemos $x = (x_1, \dots, x_n)$ e $y = (y_1, \dots, y_n)$, $x + y$ (siendo $+$ la operación OR) es igual a

$$z = (x_1 + y_1, \dots, x_n + y_n)$$

Lo que hacemos es aplicar la operación binaria OR n veces para cada par de bits x_i, y_i . En este caso tendríamos un circuito de n funciones booleanas que ejecutan la función $f : \{0, 1\}^r \rightarrow \{0, 1\}^n$ con $r = 2n$ tal que $f(x, y) = x + y$. Deberemos especificar si x e y vienen dados secuencialmente por $(x_1, \dots, x_n, y_1, \dots, y_n)$ o $(x_1, y_1, \dots, x_n, y_n)$. En cualquier caso, representan la misma función.

El número de funciones empleadas en un circuito o algoritmo será proporcional a la complejidad del mismo (más adelante haremos hincapié en la complejidad de los algoritmos), lo que finalmente repercute en el tiempo que será necesario para ejecutarlo. Es crucial saber que solamente se utilizan operaciones básicas.

Tras dar algunos ejemplos sobre funciones booleanas clásicas, a continuación mostramos dos operaciones que no debemos contemplar como básicas:

- **PRIME:** Es la función $f(x_1, \dots, x_n)$ igual a 1 si $x = (x_1, \dots, x_n)$ representa un número primo en forma de cadena binaria.

- **FACTOR:** Es la función $f(x_1, \dots, x_n, w_1, \dots, w_m)$ donde $x = (x_1, \dots, x_n)$, $w = (w_1, \dots, w_m)$ representan enteros en forma de cadenas binarias. Es igual a 1 si y sólo si el mayor divisor de x aparte de él mismo es w .

Se tiene que $\text{PRIME}(x) = \text{FACTOR}(x, 1)$ para cualquier x . Por tanto un circuito para la función FACTOR da inmediatamente uno para PRIME, pues podemos fijar los w_i de entrada rellenándolos como el número uno. Esto no significa que se dé la otra conversión, es decir, que podamos construir la función FACTOR a partir de PRIME. Estas funciones llevan siendo estudiadas miles de años, pero fue hace poco más de diez años que se probó que PRIME es *factible* en el sentido que veremos a continuación, mientras que muchos piensan que FACTOR no es factible. A continuación veremos el concepto de funciones factibles.[1]

1.2.4. Funciones booleanas factibles

Hay funciones booleanas más complejas que otras, es por eso que no todas se construyen de la misma forma. La función OR n -aria parece la más fácil: si solo vemos ceros en la entrada, la salida será claramente 0, si vemos algún uno, entonces la salida es uno. La función AND n -aria es lo contrario, pero ya la función XOR n -aria implica contar el número de bits de entrada, y aún más complicada es la función PRIME, pues no hay ningún algoritmo conocido que podamos usar para ver si una cadena binaria de bits corresponde a un primo en decimal (más allá de reconstruir el número decimal a partir de la cadena binaria y comprobar si es primo).

Podemos así definir la **complejidad** clásica de funciones booleanas. Se ve claramente que las funciones AND, XOR, OR, tienen complejidad lineal, que denotamos por $O(n)$. Los circuitos conocidos para la función PRIME requieren más pasos, pero su complejidad sigue siendo polinómica, $n^{O(1)}$. Pero hay funciones booleanas que requieren un tiempo exponencial $O(e^n)$.

Para ver esto, debemos tener en cuenta que cada función booleana se puede definir por su **tabla de verdad**. Recordemos el ejemplo 1.2 donde vimos la tabla de verdad correspondiente a AND:

x	y	$AND(x, y)$
0	0	0
0	1	0
1	0	0
1	1	1

Cada fila indica cuál es el resultado de la función para las entradas de esa fila. En general una función booleana $f(x_1, \dots, x_n)$ es definida por una tabla de verdad de 2^n filas. La dificultad se encuentra en que cuando aumenta el número de entradas, el número de filas de la tabla de verdad crece exponencialmente. Luego siempre es posible representar una función booleana por su tabla de verdad, pero no siempre es **factible**.

Ahora, como en la realidad los circuitos booleanos se complican, necesitamos el concepto de familia de funciones booleanas $[f_n]$. Una familia $[f_n]$, no es mas que una función g sobre cadenas de bits de cualquier longitud, por lo que escribimos $g : \{0, 1\}^* \rightarrow \{0, 1\}^*$, función con una o más salidas.

Ejemplo 1.3. Una familia $g = [f_n]$ podría ser $g = [\text{NOT}, \text{XOR}] : \{0, 1\}^3 \rightarrow \{0, 1\}^2$. Esta función recibe una cadena de tres bits a la que devuelve el resultado de aplicar NOT en el primer bit concatenado con XOR en el segundo y el tercer bit.

Definición 1.11. Una función booleana $g = [f_n]$ es **factible** si cada una de las funciones f_n es de complejidad polinómica $n^{O(1)}$.

A continuación explicaremos cómo llevar al paradigma de la computación cuántica las operaciones del modelo de computación clásico.[1]

1.2.5. Representación cuántica de funciones booleanas

Como ya indicamos anteriormente al comienzo de la sección 1.2.2, las operaciones del modelo de computación cuántica son representadas con matrices. Sea $N = 2^n$, cada coordenada de un espacio de Hilbert de dimensión N se corresponde con una cadena binaria de longitud n . Cada índice (por filas, columnas) $j \in [0, \dots, N - 1]$ se asigna con la cadena de n bits correspondiente a j en binario, con ceros delante si es necesario. Por ejemplo, con $n = 2$, se tiene que $N = 4$ y $j \in [0, 1, 2, 3]$, que se corresponden en forma de cadena binaria de 2 bits con $[00, 01, 10, 11]$ respectivamente:

	00	01	10	11
00	1	0	0	0
01	0	1	0	0
10	0	0	0	1
11	0	0	1	0

Esta tabla de verdad se correspondería con una función f tal que $f(00) = 00, f(01) = 01, f(10) = 11, f(11) = 10$. La función está definida de la forma: $f(x_1, x_2) = (x_1, x_1 \oplus x_2)$. El operador efectúa XOR en el segundo bit, dejando el primero igual. Dicho de otra forma, niega el segundo bit si y solo si el primero es un uno. La negación en sí sobre un bit está representada por la siguiente matriz 2×2 , que sería el operador cuántico equivalente a NOT en el modelo clásico visto en la sección anterior:

	0	1
0	0	1
1	1	0

Esta negación controlada por el primer bit explica que este operador se llame CNOT, abreviatura de ‘Controlled-not’.

Para representar una función booleana $y = f(x_1, \dots, x_n)$, necesitamos $n+1$ coordenadas booleanas, por tanto obtendremos una matriz de orden $2^{n+1} = 2N$. Lo que realmente construimos es la función

$$F(x_1, \dots, x_n, z) = (x_1, \dots, x_n, z \oplus f(x_1, \dots, x_n))$$

F es una función booleana con salidas en $\{0, 1\}^{(n+1)}$. La propiedad principal, básica en computación cuántica (como indicamos en la sección 1.2.2), es que es invertible, de hecho F es su propia inversa:

$$F(F(x_1, \dots, x_n, z)) = F(x_1, \dots, x_n, z \oplus y) = (x_1, \dots, x_n, (z \oplus y) \oplus y) = (x_1, \dots, x_n, z)$$

A simple vista puede parecer que estamos construyendo una función diferente, pero claro, hemos de tener en cuenta que $0 \oplus x = x$, $x \in \{0, 1\}$. De este modo, veremos la función binaria clásica cuando en la tabla de verdad observemos únicamente las filas que tienen la entrada auxiliar z a cero.

Veamos un ejemplo importante para ilustrar lo anterior:

Ejemplo 1.4. *Tomando como función booleana binaria $f = \text{AND}$, tenemos que $y = f(x_1, x_2) = x_1 \wedge x_2$, entonces la función F inducida por f es:*

$$F(x_1, x_2, z) = (x_1, x_2, z \oplus (x_1 \wedge x_2))$$

*Esta función tiene nombre propio, **Toffoli**, e induce una muy importante puerta cuántica con el mismo nombre. Esta función es importante ya que es universal, lo que significa que cualquier circuito booleano puede construirse a partir de puertas Toffoli únicamente. ¿Por qué? En el modelo de computación clásica se puede construir cualquier función a partir de AND y NOT, que son universales y pueden obtenerse a partir a la función Toffoli de la siguiente manera:*

- $\text{NOT}(a) = \text{TOF}(1, 1, a)$
- $\text{AND}(a, b) = \text{TOF}(a, b, 0)$

La siguiente propiedad de las funciones booleanas que nos interesa presentar es que para cada función f , podemos construir una matriz de permutación (con esto nos referimos a una matriz formada únicamente por unos y ceros, que cuando multiplica a otra matriz permuta sus filas/columnas o ambas) que llamaremos P_f . La matriz P_f será de orden $2^{n+1} \times 2^{n+1}$ donde el 1 en cada fila $x_1 x_2 \dots x_n z$ estará en la columna $x_1 x_2 \dots x_n b$, con $b = z \oplus f(x_1, \dots, x_n)$.

Por ejemplo, veamos cómo construir la matriz P_f correspondiente a la función Toffoli.

Finalmente, la matriz de permutación inducida por la función Toffoli f es:

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

Ahora, puede darse el caso en el que f es una función booleana con m salidas (y_1, \dots, y_m) en lugar de solo una. La idea para construir la matriz en este caso es una simple extensión de la idea anterior, tendremos:

$$F(x_1, \dots, x_n, z_1, \dots, z_m) = (x_1, \dots, x_n, z_1 \oplus y_1, \dots, z_m \oplus y_m)$$

En este caso la matriz de permutación asociada P_f es de dimensión $2^{n+m} * 2^{n+m}$.

En caso de necesitar un número h de bits de ayuda (también llamados bits ancilla) para efectuar la función original f , podemos tratarlos como salidas extra de la función, utilizando variables fijas para que se mantenga la invertibilidad.

Siguiendo este esquema, finalmente representaremos toda la información que tengamos en $2^{n'}$ filas, con $n' = n + m + h$, filas a las que llamaremos ‘qubit lines’ puesto que cada fila representará un qubit. Más adelante veremos que se corresponden con los ‘cables’ del circuito cuántico asociado.

Parece más cómodo pensar los operadores P_f como partes de código para crear circuitos, siendo las qubit lines coordenadas de cadenas binarias que representan índices de estas ‘partes de código’. Estas cadenas son de tamaño n' y sus índices $1, \dots, n'$ son lo que llamaremos **coordenadas cuánticas**. Lo que estudiaremos en la siguiente sección es qué operaciones cuánticas son factibles.[1]

1.2.6. Operaciones cuánticas factibles

Los algoritmos cuánticos aplican una serie de matrices unitarias a un vector inicial, pero claro, no podemos aplicar cualquier matriz unitaria que queramos.

Observando las matrices P_f de la sección anterior, en primer lugar vemos que para construir estas matrices no tenemos en cuenta la complejidad de la función f sino que creamos una permutación sobre su tabla de verdad extendida exponencialmente. Incluso para funciones simples como $\text{AND}(x_1, \dots, x_n)$ la matriz se hace muy grande, de dimensión $2^{n+1} * 2^{n+1}$. Surge pues el dilema, ¿cómo distinguimos las operaciones básicas factibles?

Si mantenemos el número de argumentos k para cualquier operación constante, entonces 2^k se mantiene constante. Por tanto utilizaremos matrices $2^k * 2^k$ con k el número de coordenadas cuánticas (las cuales denotamos por x_1, x_2, \dots, x_n , con n el número de entradas de la función), extendiendo la notación para salidas y para los bits de ayuda o ‘bits ancilla’ que sean necesarios.

Entendido esto, la noción de factibilidad para matrices unitarias no es más que la extensión de la misma para circuitos booleanos clásicos. Cualquier matriz unitaria B de dimensión 2^k donde k es constante es factible (en particular, para $k \leq 3$, que será con lo que trabajaremos en casi todo momento). Podemos operar con tal matriz en cualquier subconjunto de k coordenadas cuánticas, garantizando dejar las otras $n' - h$ coordenadas solas.

Definición 1.12. Sea B cualquier matriz unitaria de dimensión 2^k donde k es constante. Llamaremos **matriz básica** al producto tensorial de B con matrices identidad sobre las otras coordenadas cuánticas.

Ejemplo 1.6. Dada la matriz unitaria

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

y la matriz identidad de orden 3:

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

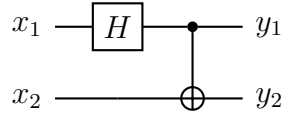
Su producto tensorial nos da una matriz básica:

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

Ahora supongamos que U es una matriz unitaria cualquiera de dimensión N . Diremos que es factible garantizando que existe una manera de construirla fácilmente a partir de matrices básicas. Esto es lo mismo que decir que U pertenece a una familia infinita $[U_n]$ con cada U_n construible sencillamente a partir de $n^{O(1)}$ matrices básicas. Más concretamente, podremos expresar U mediante un **circuito cuántico** de puertas cuánticas asociadas a matrices básicas.[1]

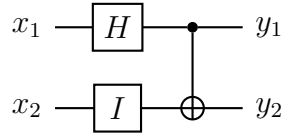
1.2.7. Circuitos cuánticos

Los circuitos cuánticos emplean líneas que van horizontalmente como pentagramas musicales y sitúan las puertas cuánticas sobre las líneas (que ya mencionamos que íbamos a llamar qubit lines) como notas musicales y acordes. Las primeras n líneas corresponden a las entradas x_1, \dots, x_n mientras que el resto de ‘qubit lines’, hasta $n + m + h$, se inicializan a 0. Los ‘cables’ que se cruzan con las líneas horizontales principales nos indican que son parte de puertas cuánticas que requieren varias entradas. Estas puertas pueden aparecer explícitamente sobre el circuito o pueden ser invisibles. Vamos a ilustrar esto con un ejemplo.



En este caso tenemos un circuito compuesto por la llamada **puerta de Hadamard** (la cual introduciremos más adelante), en la qubit line 1, seguido por una operación CNOT con control en la primera línea y objetivo en la segunda.

Debajo de la puerta Hadamard hay implícita (invisible) una puerta identidad, que indica que en el primer paso el segundo qubit no cambia. Podríamos dibujarla para que aparezca explícitamente (no se suele hacer):



Para ver el circuito matricialmente, consideramos que cada vez que aparecen dos puertas verticalmente una sobre otra hay un producto tensorial implicado. La forma matricial del circuito sería la siguiente composición $V_2 * V_1$, donde V_2 denota la operación CNOT y V_1 la puerta de Hadamard.

$$V_2 = \text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, V_1 = H \otimes I = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix}$$

Luego la composición $U = V_2 * V_1$, actuando sobre el vector de entrada $x = (1, 0, 0, 0)^T$ da:

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

El vector de entrada $x = (1, 0, 0, 0)^T$ se corresponde con la cadena binaria 00, es decir, las entradas son $x_1 = 0, x_2 = 0$. La salida no corresponde a un estado básico, es una suma con pesos iguales de los vectores asociados a 00 y 11:

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

Está por tanto asociada a un estado en superposición de los estados básicos $e_0 = (1, 0, 0, 0)^T$ y $e_3 = (0, 0, 0, 1)^T$ con amplitudes $1/\sqrt{2}$. El poder de los algoritmos cuánticos está en estas salidas, con las que será necesario interactuar *mediéndolas* (explicaremos esto más adelante) incluso repetidas veces hasta llegar a una salida final booleana.

Visto lo anterior damos el siguiente resultado:

Definición 1.13. Una operación cuántica C sobre s qubits es **factible** si y solo si

$$C = U_t U_{t-1} \dots U_1$$

donde cada U_i es una operación factible y s, t están acotados por un polinomio en el número n de qubits de entrada.

En la siguiente sección introduciremos algunas puertas/operaciones que son consideradas factibles.[1]

1.2.8. Matrices especiales

Ya hemos visto que los algoritmos cuánticos son en definitiva el resultado de aplicar transformaciones lineales unitarias a vectores unitarios, por lo que estamos en la obligación de presentar diferentes matrices unitarias que son comúnmente utilizadas en estos algoritmos.

Matrices de Hadamard

La primera familia que estudiaremos son las matrices de Hadamard. Notar que al igual que asociamos los vectores de la base canónica con los estados básicos, identificaremos las transformaciones con sus matrices. Seguimos asumiendo que $N = 2^n$ para un cierto n .

Definición 1.14. La matriz de Hadamard de orden N , H_N está definida recursivamente para $N = 2^n \geq 4$ por :

$$H_1 = (1); H_2 = H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

$$H_N = H_{N/2} \otimes H = \frac{1}{\sqrt{2}} \begin{pmatrix} H_{N/2} & H_{N/2} \\ H_{N/2} & -H_{N/2} \end{pmatrix}$$

La matriz también se puede denotar $H^{\otimes n}$.

Lema 1.3. Para cualquier fila r y columna c ,

$$H_N[r, c] = \frac{1}{\sqrt{N}}(-1)^{r_b \bullet c_b}$$

donde $r_b \bullet c_b$ es el producto escalar de r, c tratados como cadenas booleanas.

Demostración. Procedamos por inducción. El caso inicial es para $H_N = H_{2^n} = H_2$. Veamos que

$$H_2[r, c] = \frac{1}{\sqrt{2}}(-1)^{r_b \bullet c_b}$$

Comprobemos que se cumple para cada una de las cuatro componentes:

$$H_2[0, 0] = \frac{1}{\sqrt{2}}(-1)^{0_b \bullet 0_b} = \frac{1}{\sqrt{2}}$$

$$H_2[0, 1] = \frac{1}{\sqrt{2}}(-1)^{0_b \bullet 1_b} = \frac{1}{\sqrt{2}}$$

$$H_2[1, 0] = \frac{1}{\sqrt{2}}(-1)^{1_b \bullet 0_b} = \frac{1}{\sqrt{2}}$$

$$H_2[1, 1] = \frac{1}{\sqrt{2}}(-1)^{1_b \bullet 1_b} = -\frac{1}{\sqrt{2}}$$

$$H_2 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

Supongamos ahora que el lema se cumple para H_N con $N = 2^n$. Veamos si se cumple para H_{2N} con $2N = 2^n + 1$. Apoyándonos en la definición 1.14, sabemos que:

$$H_N = H_{N/2} \otimes H = \frac{1}{\sqrt{2}} \begin{pmatrix} H_{N/2} & H_{N/2} \\ H_{N/2} & -H_{N/2} \end{pmatrix}$$

Por tanto, se tiene que:

$$H_{2N} = H_N \otimes H = \frac{1}{\sqrt{2}} \begin{pmatrix} H_N & H_N \\ H_N & -H_N \end{pmatrix}$$

Esto sugiere poner las entradas de H_{2N} en función de las entradas de H_N . Si nos fijamos estamos construyendo matrices de Hadamard de mayor orden por medio de cuatro bloques cada vez. Por lo tanto podemos considerar los siguientes casos:

- Si $r < N$ y $c < N$ tenemos que $H_{2N}[r, c] = \frac{1}{\sqrt{2}}H_N[r, c]$
- Si $r < N$ y $c \geq N$ tenemos que $H_{2N}[r, c] = \frac{1}{\sqrt{2}}H_N[r, c - N]$
- Si $r \geq N$ y $c < N$ tenemos que $H_{2N}[r, c] = \frac{1}{\sqrt{2}}H_N[r - N, c]$
- Si $r \geq N$ y $c \geq N$ tenemos que $H_{2N}[r, c] = -\frac{1}{\sqrt{2}}H_N[r - N, c - N]$

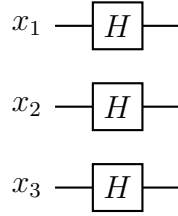
Por tanto ya hemos encontrado una forma de poner las entradas de H_2N en función de las de H_N y como hemos supuesto que el lema se cumple para cada componente de H_N , queda demostrado el resultado. \square

Entonces, para cualquier vector a , el vector $b = H_N a$ en su componente x viene dado por:

$$b(x) = \frac{1}{\sqrt{N}} \sum_{t=0}^{N-1} (-1)^{x_b \bullet t_b} a(t)$$

donde $x_b \bullet t_b$ es el producto escalar de x, t tratados como cadenas booleanas.

En un circuito cuántico con n qubit lines, H_N se ve como una columna de n puertas Hadamard de un solo qubit. A continuación podemos ver cómo se representaría la puerta H_N para $N = 2^n = 2^3$.



Una propiedad importante de esta operación es que cuando la aplicamos a un qubit en un estado básico, este cambia a un estado en superposición de estados básicos. Por ejemplo, tomando $H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ y el estado $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, el estado de ese qubit pasa a ser $\begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$. De forma similar, si la aplicamos al estado $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$, obtenemos el estado $\begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix}$. Como se puede observar, la amplitud de cada estado es la misma, pero en el segundo caso la amplitud del estado básico $|1\rangle$ queda con signo negativo. Si consideramos el caso en el que tenemos n qubits en estados básicos, el resultado de aplicar la matriz de Hadamard correspondiente a todos los qubits sería tener cada uno de ellos en un estado de superposición donde todas las amplitudes serán igual a $\frac{1}{\sqrt{N}}$ variando el signo. Por tanto cada qubit colapsará a cualquier estado básico con la misma probabilidad. Esto es muy útil en los algoritmos cuánticos y se usa en muchos de ellos. Más adelante, cuando veamos el **algoritmo de Grover**, veremos la utilidad que tiene para ese caso concreto.

Computación reversible y matrices de permutación

Recuperemos las matrices de permutación P_f dadas en la sección 1.2.5. Cualquier matriz de permutación de dimensión $N * N$ es unitaria. Sin embargo en términos de n , con $N = 2^n$ existe un número doblemente exponencial de matrices de permutación. No todas serán factibles. De hecho, la mayoría no lo son, luego, ¿qué matrices de permutación son factibles?

Teorema 1.4. *Todas las funciones booleanas clásicas factibles en el sentido clásico tienen un operador cuántico factible asociado en la forma de una matriz de permutación P_f*

Para probar este teorema debemos echar mano de una función que construimos en la sección anterior, la función **Toffoli**. Recordemos como se definía:

$$TOF(x_1, x_2, x_3) = (x_1, x_2, x_3 \oplus (x_1 \wedge x_2))$$

Demostración. Como AND y NOT funcionan como puertas lógicas universales, cualquier función factible en el sentido clásico se podrá representar por un circuito booleano formado por r puertas NOT y s puertas AND. Las puertas NOT podemos obviarlas dado que ya hemos visto la matriz 2×2 correspondiente en la sección 1.2.5. Por tanto solo tenemos que considerar las s puertas AND. Podemos simularlas mediante s puertas Toffoli, cada una de ellas con una línea ancilla establecida a 0 para la entrada. El problema es que puede darse la posibilidad de necesitar múltiples copias de la salida c de una AND sobre a, b ($a \wedge b = c$), una por cada cable que queda fuera de la puerta.

Aquí es donde actúa la puerta Toffoli. Para cada cable de salida w , colocamos una línea ancilla z y ponemos una puerta Toffoli con control en a, b y objetivo en z . Esto ejecuta automáticamente $z \oplus (a \wedge b)$, que con z inicializada con todo ceros es justo lo que queremos:

a	b	z	$a \wedge b$	$z \oplus (a \wedge b)$
0	0	0	0	0
0	1	0	0	0
1	0	0	0	0
1	1	0	1	1

Múltiples puertas Toffoli con los mismos controles no se afectan unas a otras. Por tanto la sobrecarga está limitada por el número de cables que tenga el circuito, que es polinómico, y las únicas líneas ancilla que necesitamos se inicializan a cero. \square

Por tanto, dada una función booleana clásica factible, su matriz de permutación será factible.

Matrices diagonales factibles

Cualquier matriz diagonal cuyas entradas no nulas tienen como valor absoluto 1 es unitaria. Por tanto, puede ser un operador cuántico. Pero, ¿qué operaciones de este estilo son factibles?

Por supuesto que si el tamaño de la matriz es un número fijo pequeño, la matriz es básica y por tanto factible. ¿Qué ocurre cuando las matrices son $N \times N$?

Incluso si limitamos las entradas a tomar los valores 1 y -1, tenemos, por cada subconjunto $S \subseteq \{0, 1\}^n$, una matriz diagonal U_S de la forma:

$$U_S[x, x] = \begin{cases} -1 & \text{si } x \in S \\ 1 & \text{en otro caso} \end{cases}$$

Como el número de subconjuntos S es doblemente exponencial, así lo es también el número de matrices U_S , luego algunas no serán factibles. ¿Cuáles sí lo son? Cuando S coincide con el conjunto de argumentos que hacen cierta una función booleana f , escribimos U_f en lugar de U_S y a esta matriz U_f la llamamos **oráculo de Grover** para f .

Ejemplo 1.7. *Por ejemplo, construyamos el oráculo de Grover para la función $f = \text{XOR}$ con $n = 2$. Como sabemos, la función XOR para $n = 2$ es igual a 1 para 1,2, ya que en binario se corresponden con 01, 10, y se tiene que $\text{XOR}(01) = 1 = \text{XOR}(10)$. Por tanto en este caso el conjunto de argumentos que hacen cierta f es $S = \{2, 3\}$, por lo que el oráculo de Grover para f es:*

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

Teorema 1.5. *Si f es una función booleana factible, entonces su oráculo de Grover U_f es factible.*

Esta demostración se escapa del alcance de este trabajo. Se puede encontrar en el libro ‘Quantum Algorithms Via Linear Algebra’[1]

Con este último teorema ya tenemos un buen repertorio de operaciones factibles. A continuación daremos una familia más de operaciones.

Reflejos

Dado cualquier vector unitario a , podemos crear un operador unitario Ref_a que *refleja* sobre la circunferencia unidad cualquier otro vector unitario b con el mismo origen en torno a a . Geométricamente, dibujamos una línea recta desde el extremo de b que incide sobre a formando un ángulo recto y continúa durante la misma distancia (la distancia entre el punto donde coincide con el vector a y el origen de la línea) hasta un punto b' . La operación que lleva de b a b' mantiene la imagen del vector en la esfera unidad y de igual forma su inversa, luego es unitaria. Otra forma de ver esta operación, es como un giro: dados los dos vectores unitarios a, b el reflejo Ref_a de b , sería, si a, b forman un ángulo α , el vector que obtenemos girando el vector a una amplitud de α grados, o de igual forma, el que obtenemos girando b una amplitud de $2 * \alpha$.

Geoméricamente hablando, cada ‘punto’ sobre el vector a es la **proyección** de b sobre a , y esta viene dada por $a' = a\langle a, b \rangle$. Por tanto:

$$b' = b - 2(b - a\langle a, b \rangle) = (2P_a - I)b$$

Donde P_a es el operador que realiza la proyección: para todo b :

$$P_a b = a\langle a, b \rangle$$

Veamos un ejemplo:

Ejemplo 1.8. Sea j el vector unitario con todas las entradas igual a $\frac{1}{\sqrt{N}}$. Entonces el operador que realiza la proyección es la matriz $N \times N$ cuyas entradas son todas $\frac{1}{N}$, que denotamos por J (notar que $P_j b = j\langle j, b \rangle = j * (j^T * b) = J * b$, por lo que $P_j = J$). Finalmente el operador reflejo es:

$$V = 2J - I = \begin{pmatrix} \frac{2}{N} - 1 & \frac{2}{N} & \dots \\ \frac{2}{N} & \frac{2}{N} - 1 & \dots \\ \frac{2}{N} & \frac{2}{N} & \dots \end{pmatrix}$$

Definición 1.15. Decimos que a es el vector característico de un conjunto no vacío S si se cumple lo siguiente:

$$a(x) = \begin{cases} \frac{1}{\sqrt{|S|}} & \text{si } x \in S \\ 0 & \text{en otro caso} \end{cases}$$

Supongamos que a es el vector característico de un conjunto no vacío S y aplicamos Ref_a a un vector b tal que $b(x) = e$ para $x \in S$. Sea $k = |S|$. Entonces tenemos que $\langle a, b \rangle = ke/\sqrt{k} = e\sqrt{k}$. Si ahora tomamos la proyección $a' = P_a b$ tenemos:

$$a'(x) = \begin{cases} e & \text{si } x \in S \\ 0 & \text{en otro caso} \end{cases}$$

El reflejo $b' = 2a' - b$ entonces cumple:

$$b'(x) = \begin{cases} b(x) & \text{si } x \in S \\ -b(x) & \text{en otro caso} \end{cases}$$

Pues en caso de que $x \in S$, $b'(x) = 2e - b(x) = 2e - e = e = b(x)$. Esto es lo mismo que multiplicar por la matriz diagonal que tiene componentes igual a -1 para las coordenadas que no están en S , esto es, por el oráculo de Grover para el complementario de S . Como la negación (NOT) de una función booleana factible sigue siendo factible esto junto con el caso anterior demuestra el siguiente enunciado:

Teorema 1.6. Para toda función booleana factible f , si nos restringimos al subespacio vectorial de vectores cuyas entradas indexadas por el conjunto S_f de argumentos que hacen f cierta ($f = 1$) son iguales, entonces el reflejo por el vector característico de S_f es un operador cuántico factible.

Además, el conjunto de estos vectores forma un subespacio vectorial y siempre contiene al vector j que presentamos antes, el cual se utilizará como vector de salida en diferentes algoritmos. Los reflejos por a, b , cuando son aplicados a vectores pertenecientes al subespacio generado por a, b siguen perteneciendo a dicho subespacio. Esto lo utilizaremos en el algoritmo de Grover, que introduciremos a continuación.[1]

Capítulo 2

Algoritmo de Grover

Tras haber presentado algunas bases de la computación cuántica y los principales resultados que hacen esta herramienta tan potente, pasamos a poner el foco en un algoritmo muy importante que nos ha dado esta tecnología, el **algoritmo de Grover**.

2.1. Introducción

El algoritmo de Grover fue creado por Lov Kumar Grover, un informático teórico indio-estadounidense nacido en India en 1961. Grover se licenció por el Instituto Indio de Tecnología, en Delhi en 1981 y es doctor en ingeniería eléctrica por la Universidad de Stanford en 1985. Grover publicó el algoritmo que lleva su nombre en mayo de 1996, en el artículo ‘A fast quantum mechanical algorithm for database search’[4]. El algoritmo se aplica a un problema clásico y cotidiano: se dispone de una lista que contiene N elementos desordenados, de donde uno de ellos satisface una determinada condición y queremos hallar precisamente ese. Este problema se ha estudiado durante muchísimos años y sigue siendo objeto de estudio a día de hoy. El algoritmo clásico más eficiente conocido para resolver este problema es el de examinar todos los elementos de la lista uno a uno: si un elemento cumple la condición requerida paramos, si no, examinamos el siguiente elemento y así sucesivamente. En el mejor de los casos el elemento buscado estará el primero o al principio y lo obtendremos en pocos pasos; en el peor, el elemento buscado estará el último o al final. Por tanto si ejecutamos el algoritmo múltiples veces, podemos afirmar que de media habrá que observar $\frac{N}{2}$ elementos para encontrar el elemento buscado. Este problema no puede ser resuelto en menos de $O(N)$ observaciones. Por otro lado, el algoritmo de Grover es capaz de reducir la complejidad del algoritmo clásico a un orden de $O\sqrt{N}$.

2.2. ¿Cómo funciona el algoritmo?

El algoritmo de Grover consiste en tres pasos bien diferenciados: **inicialización** de los estados, **aplicación** del operador de Grover (formado por el *oráculo de Grover* y la *inversión sobre la media*) y por último la **medición** de los qubits. El algoritmo de Grover es probabilístico, lo que significa que devolverá la respuesta correcta con una probabilidad de error determinada. Esto genera la siguiente pregunta: ¿entonces no siempre obtendremos la respuesta correcta? Como veremos, la probabilidad de error podremos reducirla tanto como queramos.

A grandes rasgos, lo que hace el algoritmo es lo siguiente: Disponemos de un sistema de n qubits y $N = 2^n$ estados básicos posibles diferentes. Buscamos uno (o varios estados) en concreto. La idea del algoritmo es aumentar la probabilidad de que cuando midamos, el sistema colapse al estado buscado. Para ello:

1. En primer lugar el sistema de qubits estará en el estado $|0 \dots 0\rangle$ y el algoritmo comenzará poniendo el sistema en un estado de superposición, con las amplitudes de cada estado básico iguales, luego con igual probabilidad de colapsar a cualquiera de esos estados (Sección 2.2.2).
2. Aplicaremos el **operador de Grover**, una operación compuesta por otras dos operaciones:
 - Una operación dependerá del problema y de la condición que deba cumplir el elemento (o elementos) que queramos encontrar (**Oráculo de Grover**, Sección 2.2.3); esta operación se encargará de cambiar de signo la amplitud del estado (o estados) que estemos buscando.
 - Otra operación que reflejará todas las amplitudes respecto de la amplitud media, por lo que la amplitud (o amplitudes) del estado (o estados) que queramos encontrar aumentará mucho (ya que anteriormente la habremos invertido y por tanto será la que diste más de la media) mientras que el resto variarán poco. De esta forma la probabilidad de que el sistema colapse al estado que busquemos aumentará.
3. Por último mediremos el sistema y obtendremos el resultado.

Ahora procedamos a explicar el algoritmo en profundidad.

2.2.1. Contexto previo

Queremos buscar en una lista de N elementos. Por conveniencia asumimos $N = 2^n$, así podemos guardar los índices de cada estado en n bits. El algoritmo empleará un sistema

de n qubits con $N = 2^n$ estados básicos posibles que denotaremos S_1, S_2, \dots, S_N . Si bien el algoritmo original buscaba un elemento entre N , podemos asumir sin pérdida de generalidad que el problema tiene M soluciones con $1 \leq M \leq N$.

2.2.2. Inicialización de estados

Inicializamos el sistema con n qubits en el estado $|0\rangle$ y en primer lugar aplicamos una puerta Hadamard a todos ellos, dejando el sistema en un estado de superposición, con las amplitudes de todos los estados igual a $\frac{1}{\sqrt{N}}$. Así conseguimos que la probabilidad de que el sistema colapse a cualquier estado básico sea la misma. Esto solo se realiza una vez.

2.2.3. Oráculo de Grover

En primer lugar se debe dar la noción de qué es un oráculo en el contexto de computación cuántica. Un oráculo es un operador cuántico que actúa como una especie de *caja negra*, es decir, realiza diferentes operaciones en un algoritmo pero su implementación no es relevante a la hora de centrarnos en el funcionamiento del propio algoritmo. En el caso que nos ocupa, el oráculo de Grover (que ya introdujimos en la sección 1.2.8) tiene la capacidad de **reconocer** soluciones para el problema de búsqueda. Hacemos énfasis en que el oráculo reconoce soluciones y en que el oráculo no encuentra soluciones. Por ejemplo, podríamos tener una función que haga de oráculo que identifique si un número es un factor de otro dado. Sin embargo, otra cosa diferente y que puede llegar a ser muy difícil es encontrar los factores de un número.

Podemos representar el problema de la búsqueda en la lista con una función f , que tiene como entrada un entero $x \in \{0, N-1\}$ tal que está definida por $f(x) = 1$ si x corresponde a una solución y $f(x) = 0$ si no corresponde a ninguna solución, es decir, una función booleana que actúa como función característica del conjunto de soluciones.

Partimos de que el oráculo O es un operador unitario que definimos como:

$$O : |x\rangle|q\rangle \mapsto |x\rangle|q \oplus f(x)\rangle$$

Donde $|x\rangle$ es un estado y $|q\rangle$ es un qubit ancilla de registro cuyo estado se invertirá si $f(x) = 1$ y se mantiene tal cual en otro caso.

En nuestro algoritmo de búsqueda es útil inicializar el qubit de registro q al estado $\frac{|0\rangle - |1\rangle}{\sqrt{2}}$. Así, al aplicar el oráculo al estado $|x\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}}$, si x no es solución el estado no cambiará. Sin embargo, si x lo es:

$$|x\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}} \mapsto |x\rangle \frac{|0 \oplus 1\rangle - |1 \oplus 1\rangle}{\sqrt{2}} = |x\rangle \frac{|1\rangle - |0\rangle}{\sqrt{2}}$$

Es decir, $|0\rangle$ y $|1\rangle$ se intercambiarán, por tanto, proporcionando el estado final $-|x\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}}$. Luego lo que hace el oráculo O es lo siguiente, siendo f su función característica:

$$|x\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}} \mapsto (-1)^{f(x)} |x\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

Notemos que el estado del qubit de registro q no cambia, lo que sugiere que no necesitamos este qubit extra, pudiendo omitirlo. Por tanto, nuestro oráculo se simplifica sin ser necesario utilizar un qubit de registro, quedando definido por:

$$|x\rangle \mapsto (-1)^{f(x)}|x\rangle$$

2.2.4. El algoritmo

El primer paso (1) es inicializar el sistema a través de una puerta Hadamard, como hemos explicado antes. El siguiente paso del algoritmo (2) consiste en repetir las siguientes dos operaciones $O(\sqrt{N/M})$ veces. Más adelante explicaremos por qué solo son necesarias este número de iteraciones.

- 1 Aplicamos el Oráculo de Grover al sistema de qubits, lo cual invierte la amplitud del estado o estados correspondientes a la solución del problema.
- 2 Realizamos la llamada *inversión respecto de la media*, o según Grover, **la transformación de difusión** D (así aparece en el artículo original). El propósito de esta operación es realizar un reflejo (recordar cuando hablamos de reflejos en la sección 1.2.8) de todos los estados respecto de la media de las amplitudes. La operación D se define como: $D = HRH$ siendo H la matriz de Hadamard para n qubits y R la llamada **matriz de rotación** definida por:

- $R_{ij} = 0$ si $i \neq j$
- $R_{ii} = 1$ si $i = 0$; $R_{ii} = -1$ si $i \neq 0$

Estas dos operaciones juntas forman el **Operador de Grover**. El último paso (3) es realizar la medición del sistema que ha resultado de las anteriores operaciones. En el artículo original donde se presenta el algoritmo, Grover indica que en caso de que $C(S_v) = 1$, interpretando C como la condición que tiene que cumplir el estado buscado (según nuestra notación, esto sería equivalente a $f(v) = 1$, con $v \in \{0, \dots, N-1\}$), entonces hay un único estado¹ S_v tal que el estado final es S_v con una probabilidad de por lo menos $\frac{1}{2}$.

Aclaraciones y resultados importantes para la correcta comprensión del algoritmo

El bucle del paso (2) (Operador de Grover) que hemos introducido antes es la parte central del algoritmo. En cada iteración este bucle incrementa la amplitud del estado deseado por $O(\frac{1}{\sqrt{N}})$. Como resultado de iterar $O(\sqrt{N/M})$ veces, la amplitud y por tanto la probabilidad de que el sistema colapse al estado deseado alcanzan $O(1)$. Para ver que la amplitud aumenta en

¹Esto puede causar confusión ya que anteriormente hemos dicho que puede haber solución múltiple. Esto está indicado de esta manera ya que en el artículo original de Grover el algoritmo se presenta para una única solución, pero el algoritmo funciona para más de una solución tal y como se explica en este trabajo.

$O(\frac{1}{\sqrt{N}})$, primero mostraremos cómo la *transformación de difusión* D puede ser interpretada como una **inversión sobre la media** de todas las amplitudes. Comencemos.

En primer lugar puede surgir la siguiente pregunta acerca de la **transformación de difusión** D : Si hemos dicho que es un reflejo, ¿cómo podemos relacionar la expresión que hemos dado con lo que explicamos acerca de reflejos en la sección 1.2.8? Veamos:

Haciendo las operaciones pertinentes, $D = HRH$ toma la siguiente forma:

$$\blacksquare D_{ij} = \frac{2}{N} \text{ si } i \neq j; D_{ii} = -1 + \frac{2}{N}$$

Luego podemos representar D como $D = 2P - I$, donde I es la matriz identidad y P es el **proyector**, que es el mismo operador J que realizaba la proyección cuando introdujimos los reflejos en el ejemplo 1.8 de la sección 1.2.8.

$$J = \begin{pmatrix} \frac{1}{N} & \frac{1}{N} & \cdots & \frac{1}{N} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{1}{N} & \frac{1}{N} & \cdots & \frac{1}{N} \\ \frac{1}{N} & \frac{1}{N} & \cdots & \frac{1}{N} \end{pmatrix} = P$$

La matriz P actúa sobre cualquier vector b proporcionando un vector con todas sus componentes igual a la media de las componentes de b .

Veamos que efectivamente el operador D realiza la **inversión sobre la media** de las amplitudes de todos los estados. Expresando D como $2P - I$ se tiene que, para un vector b :

$$Db = (2P - I)b = 2Pb - b$$

Teniendo en cuenta lo que hemos dicho en el anterior párrafo, Pb nos da el vector que tiene en cada componente la media de las componentes de b , a la cual llamaremos A . Entonces la i -ésima componente del vector Db viene dada por $(2A - b(i))$ o escrito de otra forma $(A + (A - b(i)))$ que es precisamente la *inversión sobre la media*. Este operador se suele representar como: $D = 2|\psi\rangle\langle\psi| - I$, donde $|\psi\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |x_i\rangle$ es la superposición de todos los estados básicos. El operador D junto con el oráculo de Grover O forman el operador de Grover $G = (2|\psi\rangle\langle\psi| - I)O$.

Pensemos en el caso en que exista una única solución del problema. Veamos lo que ocurre cuando la inversión sobre la media se aplica a un vector cuyas componentes, excepto una, son todas iguales a un valor C que es aproximadamente $\frac{1}{\sqrt{N}}$ y la componente que es diferente es negativa, porque ya habremos aplicado el oráculo de Grover. Podemos ver una explicación visual en la figura 2.2.

La media A es aproximadamente igual a C . Como cada una de las $(N-1)$ componentes que son iguales son aproximadamente iguales a la media (cuantas más soluciones, más diferirá el valor de cada componente respecto de la media), el valor de estas no cambia significativamente tras aplicar la inversión sobre la media. Por el contrario la componente que era negativa, ahora se vuelve positiva y se incrementa aproximadamente en $2C$, que es aproximadamente $\frac{2}{\sqrt{N}}$. Podemos ver una explicación visual en la figura 2.3.[4]

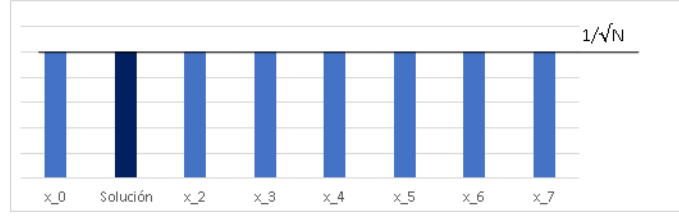


Figura 2.1: Inicialización de estados

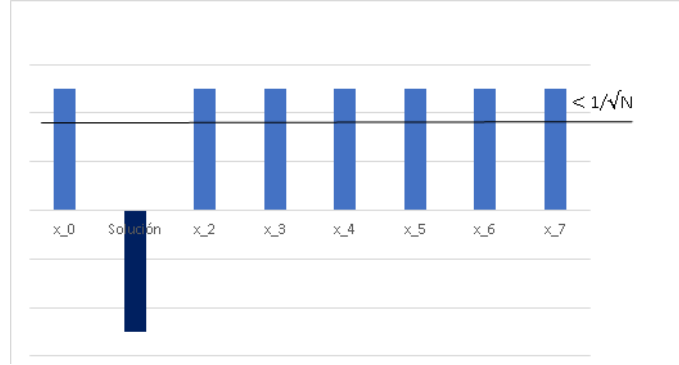


Figura 2.2: Aplicación del oráculo

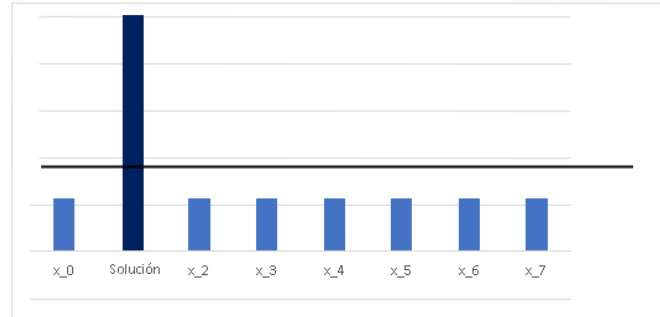


Figura 2.3: Inversión sobre la media

2.3. Interpretación geométrica

En esta sección vamos a demostrar que el número de iteraciones necesarias para conseguir una solución es exactamente del orden de $O(\sqrt{N/M})$, dando una interpretación geométrica de lo que realiza el operador de Grover $G = (2|\psi\rangle\langle\psi| - I)O$. Veremos que este operador puede ser interpretado como una rotación en el espacio 2-dimensional generado por el vector inicial $|\psi\rangle$ y el estado consistente en una superposición de los estados solución del problema de búsqueda. Para comenzar la explicación consideremos $\sum_{i=0}^J x_i$ la suma de estados que son solución del problema y $\sum_{i=J+1}^{N-1} x_i$ la suma del resto de estados que no son solución. Se tiene

por tanto que $\sum_{i=0}^{N-1} |x_i\rangle = \sum_{i=0}^J x_i + \sum_{i=J+1}^{N-1} x_i$. Tenemos por tanto los estados en superposición siguientes:

$$|\alpha\rangle = \frac{1}{\sqrt{N-M}} \sum_{i=0}^J x_i$$

$$|\beta\rangle = \frac{1}{\sqrt{M}} \sum_{i=J+1}^{N-1} x_i$$

Por tanto el estado inicial $|\psi\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |x_i\rangle$ se puede expresar como:

$$|\psi\rangle = \sqrt{\frac{N-M}{N}} |\alpha\rangle + \sqrt{\frac{M}{N}} |\beta\rangle \quad (2.1)$$

De esta manera el estado inicial se encuentra en el espacio vectorial generado por $|\alpha\rangle$ y $|\beta\rangle$.

Primero tenemos que observar que el oráculo realiza una simetría (reflejo) sobre el vector $|\alpha\rangle$ en el plano definido por $|\alpha\rangle$ y $|\beta\rangle$. Como ya explicamos, de forma similar $2|\psi\rangle\langle\psi| - I$ realiza una simetría (reflejo) sobre el vector $|\psi\rangle$ en el mismo plano y como es bien conocido, la composición de dos simetrías nos da una rotación. Por tanto, $G^k|\psi\rangle$, el resultado de aplicar el operador de Grover k veces, se mantiene en el plano formado por $|\alpha\rangle$ y $|\beta\rangle$. Con esta representación también obtenemos el ángulo de rotación. Sea $\cos(\theta/2) = \sqrt{(N-M)/N}$, de aquí deducimos:

$$\sin \frac{\theta}{2} = \sqrt{1 - \cos^2 \frac{\theta}{2}} = \sqrt{1 - \frac{N-M}{N}} = \sqrt{\frac{M}{N}}$$

De donde:

$$|\psi\rangle = \cos(\theta/2) |\alpha\rangle + \sin(\theta/2) |\beta\rangle \quad (2.2)$$

Si observamos la Figura 2.4, vemos que en primer lugar tenemos el vector $|\psi\rangle$ formando un ángulo de $\frac{\theta}{2}$ con $|\alpha\rangle$. Tras aplicar el oráculo O observamos la simetría de $|\psi\rangle$ respecto de $|\alpha\rangle$, obteniendo $O|\psi\rangle$, que forma un ángulo θ con $|\psi\rangle$. Después se realiza la inversión sobre la media, es decir, la simetría respecto de $|\psi\rangle$, llegando a obtener $G|\psi\rangle$, que no es más que una rotación de θ radianes de $|\psi\rangle$. Luego finalmente G manda $|\psi\rangle$ a:

$$G|\psi\rangle = \cos \frac{3\theta}{2} |\alpha\rangle + \sin \frac{3\theta}{2} |\beta\rangle$$

Un número k de aplicaciones del operador de Grover llevan el estado del sistema a:

$$G^k|\psi\rangle = \cos\left(\frac{2k+1}{2}\theta\right) |\alpha\rangle + \sin\left(\frac{2k+1}{2}\theta\right) |\beta\rangle$$

Entonces podemos afirmar que el operador de Grover G realiza una rotación sobre el plano formado por $|\alpha\rangle$ y $|\beta\rangle$ rotando el estado un ángulo θ en cada aplicación de G . Si

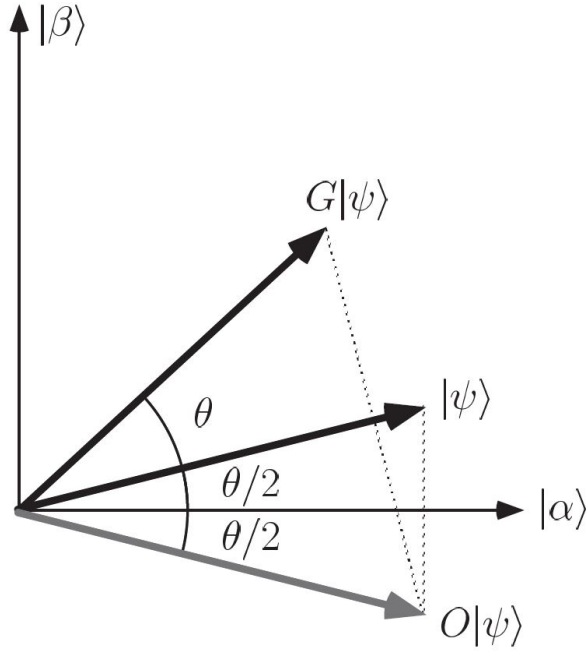


Figura 2.4: Interpretación geométrica de la operación de Grover

aplicamos G sucesivas veces rotaremos el vector del estado del sistema alejándonos de $|\alpha\rangle$ en cada iteración hasta acercarnos o llegar a $|\beta\rangle$ como se muestra en la figura 2.4. Aquí, hay que notar que ya que en cada iteración se hace una rotación de θ grados, el algoritmo de Grover no es convergente, si no que es un algoritmo cíclico, lo cual hace que calcular el número de iteraciones sea de vital importancia (en un algoritmo convergente podemos iterar cuantas veces como queramos y nos acercaremos cada vez más al objetivo; por el contrario, en un algoritmo cíclico, cuando pasemos del número correcto de iteraciones, nos alejaremos de la solución, aunque más adelante después de un número igual de iteraciones volvamos a acercarnos). Entonces cuando midamos el sistema, habrá una alta probabilidad de que colapse a un estado solución. A continuación demostraremos que realmente el número de iteraciones en el algoritmo de Grover para conseguir una solución es una $O(\sqrt{N/M})$.

2.3.1. Número de iteraciones

El estado inicial del sistema al aplicar el operador de Grover es

$$|\psi\rangle = \sqrt{(N-M)/N}|\alpha\rangle + \sqrt{M/N}|\beta\rangle$$

luego para llegar a la solución debemos rotar el sistema un ángulo $\alpha = \pi/2 - \theta/2$ radianes para llegar a $|\beta\rangle$, luego, ya que

$$\arccos(x) + \arcsin(x) = \pi/2, \arcsin(\sqrt{M/N}) = \theta/2$$

se tiene que

$$\alpha = \pi/2 - \theta/2 = \pi/2 - \arcsen(\sqrt{M/N}) = \arccos(\sqrt{M/N})$$

Teniendo en cuenta que en cada iteración el ángulo de rotación es θ , que $M \leq N/2$ y considerando la función redondeo tal que $RD(RD(5.4) = 5, RD(5.5) = 5, RD(5.51) = 6)$, repitiendo la iteración de Grover un número de veces igual a

$$R = \left(\frac{\arccos \sqrt{M/N}}{\theta} \right) \approx RD \left(\frac{\arccos \sqrt{M/N}}{\theta} \right) \quad (2.3)$$

rotaremos $|\psi\rangle$ hasta $|\beta\rangle$ con un error angular menor que $\theta/2 \leq \pi/4$. Esto es debido a que al utilizar la función redondeo en el cálculo de iteraciones, como mucho el error en este número será de media iteración. Si en cada iteración el ángulo de rotación es θ , en media iteración el ángulo será $\theta/2$. Este ángulo será menor que $\pi/4$ ya que el número de soluciones es menor que la mitad del número de resultados posibles. Por tanto si observamos el sistema tras esa rotación obtendremos la solución esperada con una probabilidad de al menos $1/2$.

Para ciertos valores de M y de N es posible conseguir una probabilidad de éxito mucho mayor. Por ejemplo, en caso de que $M \ll N$ (M mucho menor que N) tendríamos que $\sin(\theta/2) = \sqrt{M/N} \rightarrow 0$, luego $\sin(\theta/2) \approx \theta/2$, por lo que $\theta/2 \approx \sqrt{M/N} \ll \pi/4$.

Como hemos dicho, el algoritmo de Grover es cíclico. Por ello, de igual manera que hemos calculado el número de iteraciones necesarias para llegar a una solución, podemos también calcular cuantas iteraciones son necesarias para volver a obtener una solución una vez ya hemos iterado las primeras R veces, es decir, podemos calcular cuál es el período. Volver a obtener una solución significa rotar el sistema 2π radianes, por tanto el cálculo es idéntico que para el número de iteraciones, solo que esta vez $\alpha = 2\pi$. Por tanto el periodo T es igual a:

$$T = \left(\frac{2\pi}{\theta} \right) \approx RD \left(\frac{2\pi}{\theta} \right) \quad (2.4)$$

Hemos de fijarnos en que el número de iteraciones R depende del número de soluciones M pero no de lo que tengan que cumplir las mismas, luego simplemente conociendo M podemos aplicar el algoritmo como se ha descrito.

La ecuación 2.3 nos da una expresión exacta del número de llamadas R que hacer al operador de Grover a la hora de efectuar el algoritmo, pero vamos a buscar una expresión más sencilla que represente realmente el significado de R . Debemos notar que $R \leq \lceil \pi/2\theta \rceil$ (siendo $\lceil \cdot \rceil$ la función parte entera *techo*) ya que $RD(x) \leq \lceil x \rceil$ y $\arccos \sqrt{M/N} \leq \pi/2$, por lo que dando una cota inferior de θ obtendremos una cota superior para R . Asumiendo $M \leq N/2$ tenemos:

$$\frac{\theta}{2} \geq \sin \frac{\theta}{2} = \sqrt{\frac{M}{N}}$$

manipulamos la expresión y obtenemos

$$\sqrt{\frac{N}{M}} \geq \frac{2}{\theta}$$

luego

$$\frac{1}{2}\sqrt{\frac{N}{M}} \geq \frac{1}{\theta}$$

de donde obtenemos la cota superior que buscábamos desde un principio para el número de iteraciones a realizar:

$$R \leq \left\lceil \frac{\pi}{2\theta} \right\rceil \leq \left\lceil \frac{\pi}{4} \sqrt{\frac{N}{M}} \right\rceil \quad (2.5)$$

Por lo que se tiene que $R = O(\sqrt{N/M})$ son las iteraciones que se deben realizar para obtener una solución correcta con elevada probabilidad. Podemos así ver la mejora de complejidad cuadrática sobre la complejidad lineal del algoritmo clásico.

A continuación vamos a ver un caso que es importante tener en cuenta del problema de búsqueda. ¿Qué pasa si más de la mitad de los elementos de la lista son solución, es decir, $M \geq N/2$?

Sabemos que de las ecuaciones 2.1 y 2.2 se tiene

$$\begin{aligned} \cos \frac{\theta}{2} &= \sqrt{\frac{N-M}{N}} \\ \sin \frac{\theta}{2} &= \sqrt{\frac{M}{N}} \end{aligned}$$

De donde aplicando la fórmula del seno del ángulo doble, obtenemos

$$\theta = \arcsin \left(\frac{2\sqrt{M(N-M)}}{N} \right) \quad (2.6)$$

De la expresión anterior podemos observar que conforme M aumenta de $N/2$ a N , el ángulo θ disminuye. Como consecuencia el número de iteraciones necesarias se incrementa conforme aumenta M , para $M \geq N/2$. Esto parece irónico, ¿tenemos una lista en la que los elementos que buscamos son más de la mitad de la lista, y nos va a costar aún más trabajo que solo buscando uno de ellos? Esto tiene fácil solución, veamos dos maneras diferentes de abordar el problema.

La primera consiste en, si sabemos de antemano que el número de soluciones M es mayor que $N/2$ entonces podemos elegir un elemento aleatoriamente y ver si es una solución utilizando el oráculo. La probabilidad de acertar es de al menos $1/2$ y solo requerimos de una llamada. La desventaja es que puede que no sepamos de antemano cual es el número de soluciones M .

Otra solución muy interesante y útil para cuando no conozcamos si $M \geq N/2$, es la siguiente. La idea es duplicar el número de elementos del listado añadiendo N elementos extra, ninguno de ellos solución. Esto se traduce en añadir un nuevo qubit $|q\rangle$, duplicando el número de estados básicos posibles a $2N = 2^{n+1}$. Construimos un nuevo oráculo que realice la misma función pero que se aplique a este qubit extra también. Nuestro problema de búsqueda ahora tiene solamente M soluciones de entre $2N$ elementos, luego aplicando el algoritmo, necesitaremos realizar como mucho un número de iteraciones $R = \pi/4\sqrt{2N/M}$, de donde sigue que volvemos a tener $R \approx O(\sqrt{N/M})$. [5]

Capítulo 3

Implementación del algoritmo

Tras exponer la base matemática existente detrás de la computación cuántica y explicar el funcionamiento del algoritmo de Grover, llega el momento de dar la implementación del algoritmo. Para ello daremos la implementación del mismo con 2 y 3 qubits y nos apoyaremos en una herramienta de desarrollo utilizada para trabajar con ordenadores cuánticos de IBM: Qiskit[6].

3.1. Implementación con 2 qubits

Para el caso en que el número de elementos sea $N = 4$ y la solución sea única $M = 1$ necesitaremos $n = 2$ ($N = 2^2 = 4$). Según la ecuación 2.5, necesitaríamos dos iteraciones como máximo para obtener el resultado esperado.

$$R \leq \left\lceil \frac{\pi}{4} \sqrt{\frac{N}{M}} \right\rceil = \left\lceil \frac{\pi}{4} \sqrt{\frac{4}{1}} \right\rceil = 2$$

Realmente en este caso particular solo una iteración es necesaria para rotar el estado inicial $|\psi\rangle$ hasta el estado buscado $|\beta\rangle$. Esto se debe a que por ser $N = 4$, tenemos:

$$\frac{\theta}{2} = \arcsen \sqrt{\frac{M}{N}} = \arcsen \sqrt{\frac{1}{4}} = \arcsen \frac{1}{2} = \frac{\pi}{6}$$

o sea que $|\psi\rangle$ está a 30 grados con respecto a $|\alpha\rangle$. Por lo que en cada iteración rotaremos $2\frac{\pi}{6} = \frac{\pi}{3} = 60$ grados, que es justo lo que necesitamos rotar $|\psi\rangle$ para llegar a $|\beta\rangle$. De hecho, si aplicamos la ecuación 2.3 que nos da el número de iteraciones exacto sin necesidad de redondear:

$$R = \frac{\arccos \sqrt{M/N}}{\theta} = \frac{\arccos \sqrt{2/8}}{\theta} = \frac{\arccos \sqrt{1/4}}{\pi/3} = \frac{\pi/3}{\pi/3} = 1$$

Calculemos también el periodo. Para ello utilizamos la fórmula 2.4:

$$T = \frac{2\pi}{\theta} = \frac{2\pi}{\pi/3} = 6$$

Por tanto serán necesarias 6 iteraciones para volver a obtener una solución una vez la hallamos encontrado.

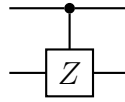
Veamos un ejemplo concreto en el que el estado buscado es $|\beta\rangle = |11\rangle$. En primer lugar tenemos que construir el oráculo O , el cual invertirá la amplitud del estado $|11\rangle$:

$$O|\psi\rangle = O\frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle) = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle - |11\rangle)$$

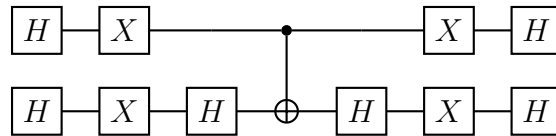
Construyamos este operador según lo que vimos en la sección 1.2.8. Se tiene que f es una función booleana tal que $f(x) = 1$ si $x = 3$ y $f(x) = 0$ si $x \neq 3$ (notar que 3 es 11 en binario). Por tanto el operador de Grover será la matriz diagonal $U_f = O$ tal que $O[x, x] = -1$ si $x = 3$ y $O[x, x] = 1$ si $x \neq 3$, de forma que:

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

lo que se corresponde con la puerta **Controlled-Z** (CZ). El oráculo en forma de circuito tendría la siguiente forma:



Ahora, para la transformación de difusión, utilizaremos la implementación que se da en el libro ‘Quantum Computation and Quantum Information’ [5] de Isaac L. Chuang y Michael A. Nielsen., la cual está basada en la utilización de puertas X, Hadamard y CNOT. La implementación es la siguiente:



Llegados a este punto podemos comparar matricialmente esta transformación de difusión con la que dimos teóricamente en el capítulo anterior. Según lo explicado en el capítulo 2, la transformación de difusión que denotamos por $D = HRH = 2P - I$, tal que

$$D_{ij} = \frac{2}{N} \text{ para } i \neq j$$

$$D_{ii} = -1 + \frac{2}{N}$$

para 2 qubits tiene la forma:

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

Ahora, si calculamos la matriz correspondiente al circuito anterior como explicamos en el capítulo 1, mediante el producto matricial y el producto tensorial, obtenemos:

$$D' = (H \otimes H) * (X \otimes X) * (I \otimes H) * \text{CNOT} * (I \otimes H) * (X \otimes X) * (H \otimes H)$$

Donde

$$H \otimes H = \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}, X \otimes X = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

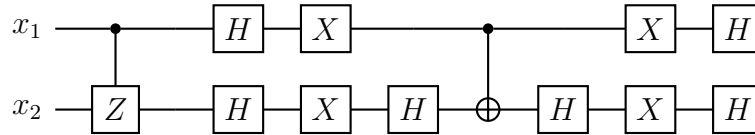
$$I \otimes H = \frac{1}{2} \begin{pmatrix} \sqrt{2} & \sqrt{2} & 0 & 0 \\ \sqrt{2} & -\sqrt{2} & 0 & 0 \\ 0 & 0 & \sqrt{2} & \sqrt{2} \\ 0 & 0 & \sqrt{2} & -\sqrt{2} \end{pmatrix}, \text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Operamos y obtenemos la siguiente matriz:

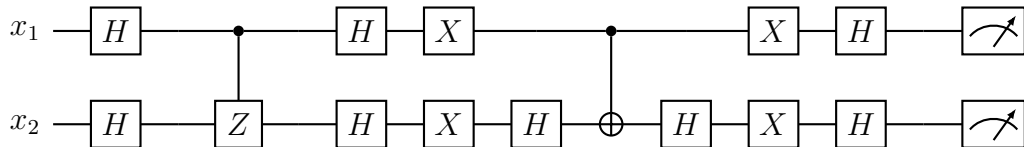
$$D' = \frac{1}{2} \begin{pmatrix} 1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 \\ -1 & -1 & 1 & -1 \\ -1 & -1 & -1 & 1 \end{pmatrix}$$

Como podemos observar, $D' = -D$. Esto no influye en la obtención del resultado final, pues como sabemos la probabilidad de que un qubit colapse a un estado concreto depende del cuadrado del valor absoluto de sus amplitudes, luego al final la probabilidad de colapsar a cierto estado será independiente del signo de la matriz de difusión.

Una vez vistos la transformación de difusión y el oráculo, el circuito asociado al operador de Grover correspondiente sería el siguiente:



Finalmente, como solo es necesaria una iteración, para completar el circuito solo queda añadir puertas Hadamard al principio para la inicialización de estados y medidores al final:



3.1.1. Implementación del algoritmo en Qiskit

Qiskit es un kit de desarrollo de software de código abierto creado por **IBM**, hecho para trabajar con sus ordenadores cuánticos y simuladores de los mismos y así poder experimentar creando circuitos, aplicaciones, etc.

Qiskit está basado en el lenguaje de programación **Python** y podemos trabajar tanto en local con la interfaz gráfica **Anaconda**, como en la nube en las plataformas **IBM Quantum Lab** y **Strangeworks**. En este caso la plataforma escogida ha sido IBM Quantum Lab, que permite crear circuitos cuánticos tanto gráficamente como mediante código y así poder experimentar con nuestros propios programas de una manera bastante cómoda.[6]

Utilizaremos la componente *Aer* de Qiskit, la cual proporciona simuladores de ordenadores cuánticos.

Mostraremos paso a paso como los pasos teóricos que hemos explicado se cumplen en el simulador del ordenador cuántico.

1. **Inicialización de estados:** En la figura A.1 podemos ver como tras aplicar puertas Hadamard sobre cada uno de los qubits, la amplitud de cada uno de los estados es la misma: $\frac{1}{\sqrt{N}} = \frac{1}{\sqrt{4}} = \frac{1}{2}$.
2. **Aplicación del oráculo:** En la figura A.2 podemos ver la construcción del oráculo y en la figura A.3 podemos ver como tras aplicarlo la amplitud del estado $|11\rangle$ se invierte.
3. **Transformación de difusión:** En la figura A.4 podemos ver cómo se construye la transformación de difusión, en la figura A.5 el circuito final y en la figura A.6 podemos ver cómo tras aplicar la transformación, la amplitud del estado $|11\rangle$ pasa a ser -1 y la de los otros estados 0, por tanto la probabilidad de que el sistema colapse al estado $|11\rangle$ es 1.
4. **Resultado final:** En la figura A.7 podemos ver como el resultado final de la ejecución del algoritmo 1000 veces nos da como resultado el estado buscado $|11\rangle$.

Ahora, si seguimos iterando hasta $R + T = 1 + 6 = 7$ iteraciones, volveremos a llegar a la solución. Como el periodo es exacto, llegaremos al mismo sistema y obtendremos el estado buscado en todas las ejecuciones del algoritmo que hagamos. En la figura A.8 vemos que tras 1000 ejecuciones el vector de estados que obtenemos es idéntico al que obteníamos con 1 iteración. En la misma figura vemos cómo obtenemos el estado deseado el 100 % de las veces.

3.2. Implementación con 3 qubits

En este ejemplo consideraremos $N = 8$ elementos y $M = 2$ soluciones. Según la ecuación 2.5, necesitaremos como mucho dos iteraciones:

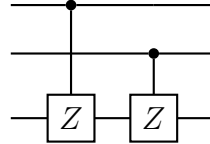
$$R \leq \left\lceil \frac{\pi}{4} \sqrt{\frac{N}{M}} \right\rceil = \left\lceil \frac{\pi}{4} \sqrt{\frac{8}{2}} \right\rceil = 2$$

Realmente en este caso ocurre lo mismo que en el ejemplo anterior para dos qubits, solo una iteración es necesaria para rotar el estado inicial $|\psi\rangle$ hasta el estado buscado $|\beta\rangle$. Procediendo de igual manera, calculemos el ángulo inicial $\theta/2$:

$$\frac{\theta}{2} = \arcsen \sqrt{\frac{M}{N}} = \arcsen \sqrt{\frac{2}{8}} = \arcsen \sqrt{\frac{1}{4}} = \arcsen \frac{1}{2} = \frac{\pi}{6}$$

o sea que $|\psi\rangle$ vuelve a estar a 30 grados con respecto a $|\alpha\rangle$. Por lo que en cada iteración rotaremos $2\frac{\pi}{6} = \frac{\pi}{3} = 60$ grados, que es justo lo que necesitamos rotar $|\psi\rangle$ para llegar a $|\beta\rangle$.

Construiremos nuestro operador para que detecte los estados $|101\rangle$ y $|011\rangle$. Para que nuestro oráculo detecte esos dos estados, una opción es utilizar dos puertas Controlled-Z. Sabemos que la puerta Z niega el estado $|1\rangle$ y deja el $|0\rangle$ invariante. Por tanto configuraremos estas puertas CZ la primera con control en el primer qubit y objetivo en el tercero y la segunda con control en el segundo y objetivo en el tercero. El oráculo tendrá la siguiente forma:



Esto podemos verlo matricialmente. Si procedemos calculando la matriz como hicimos con dos qubits, al querer detectar los estados $|101\rangle$ y $|011\rangle$ (5 y 3) llegamos a que la matriz O es tal que:

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

Veamos que si construimos la matriz a partir de las puertas CZ obtenemos lo mismo: CZ con control en el primer qubit y objetivo en el tercero niega los estados $|101\rangle, |111\rangle$ (5 y 7),

luego tiene como matriz:

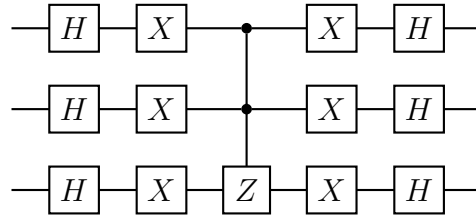
$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix}$$

Por otro lado, CZ con control en el segundo qubit y objetivo en el tercero niega los estados $|011\rangle, |111\rangle$ (3 y 7), luego tiene como matriz:

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

Podemos ver fácilmente que su producto produce la matriz O .

Para la implementación de la transformación de difusión seguiremos la dada en la documentación de Qiskit[7], que realmente es una extensión de la implementación para 2 qubits dada por Nielseng y Chuang[5]. Veámosla:



Ahora hagamos el mismo análisis que en el caso de 2 qubits. En primer lugar, para 3 qubits, la matriz D según Grover tomaría la forma:

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

Ahora, la forma matricial del circuito anterior es la siguiente:

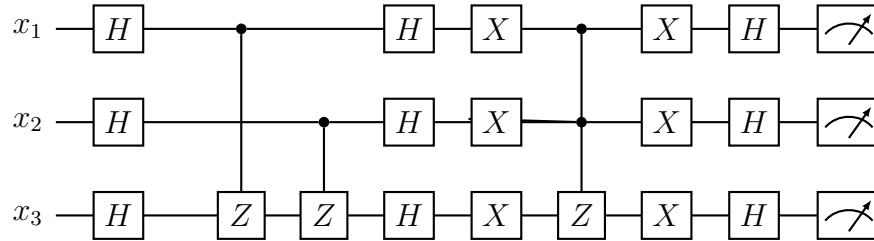
$$D' = (H \otimes H \otimes H) * (X \otimes X \otimes X) * CCZ * (X \otimes X \otimes X) * (H \otimes H \otimes H)$$

Operando exactamente igual que antes, la matriz que obtenemos es:

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

Que es nuevamente la dada en el artículo original de Grover[4] multiplicada por -1, es decir $D' = -D$. Esto, como ya hemos explicado, no supone un problema para llegar al resultado final.

Una vez vistos la transformación de difusión y el oráculo, el circuito asociado a este problema, teniendo en cuenta que solo una iteración es necesaria sería el siguiente:



3.2.1. Implementación del algoritmo en Qiskit

De igual manera a como hemos hecho en la sección anterior, mostraremos paso a paso como los pasos teóricos que hemos explicado se cumplen en el simulador del ordenador cuántico.

1. **Inicialización de estados:** En la figura B.1 podemos ver como tras aplicar puertas hadamard sobre cada uno de los qubits, la amplitud de cada uno de los estados es la misma: $\frac{1}{\sqrt{N}} = \frac{1}{\sqrt{8}} = \frac{1}{2}\sqrt{2}$.
2. **Aplicación del oráculo:** En la figura B.2 podemos ver la construcción del oráculo y en la figura B.3 podemos ver como tras aplicarlo, la amplitud de los estados $|011\rangle$ y $|101\rangle$ se invierten.
3. **Transformación de difusión:** En la figura B.4 podemos ver como se construye la transformación de difusión, en la figura B.5 el circuito final. En las figuras B.6 y B.7

podemos ver cómo tras aplicar la transformación, el resultado varía entre la amplitud del estado $|011\rangle$ a -1 y la de los otros estados 0, o bien la amplitud del estado $|101\rangle$ a -1 y la de los otros estados 0, que son justo los estados que buscábamos detectar.

4. **Resultado final:** En la figura B.8 podemos ver como el resultado final de la ejecución del algoritmo 1000 veces nos da como resultado los estados buscados.

3.2.2. Otro ejemplo con más iteraciones

Hemos visto dos implementaciones del algoritmo con 2 y 3 qubits en las que solo era necesaria una iteración. A continuación veremos un caso en el que para 3 qubits son necesarias dos iteraciones y veremos lo que ocurre si iteramos más veces.

Sea $n = 3$ entonces $N = 2^3 = 8$ y $M = 1$. Calculemos el número de iteraciones:

Según la ecuación 2.5, necesitaremos como mucho tres iteraciones:

$$R \leq \left\lceil \frac{\pi}{4} \sqrt{\frac{N}{M}} \right\rceil = \left\lceil \frac{\pi}{4} \sqrt{\frac{8}{1}} \right\rceil = 3$$

Veamos que realmente necesitamos solo dos iteraciones calculando θ . Se tiene que:

$$\frac{\theta}{2} = \arcsen \sqrt{\frac{M}{N}} = \arcsen \sqrt{\frac{1}{8}} = \arcsen \frac{1}{2\sqrt{2}} \approx 0.361367 \text{ rad} \approx 20.7048^\circ$$

o sea que $|\psi\rangle$ está a 20.7048 grados aproximadamente de $|\alpha\rangle$. Por lo que en cada iteración rotaremos $2 * 20.7048 = 41.4096$ grados. Por tanto haciendo dos iteraciones es cuando más cerca de $\pi/2$ estaremos. De hecho, si aplicamos la fórmula 2.3, obtenemos:

$$R = \frac{\arccos \sqrt{M/N}}{\theta} = \frac{\arccos \sqrt{1/8}}{\theta} = 1.67341 \approx RD(1.67341) = 2$$

En este caso también vamos a calcular el periodo. Luego utilizando la fórmula 2.4:

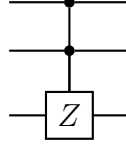
$$T = 2\pi/\theta \approx RD(\pi/\theta) = RD(\pi/0.361367) = RD(8.69363) = 9$$

Por tanto serán necesarias 9 iteraciones para volver a obtener una solución una vez la hallamos encontrado.

Supongamos que queremos detectar el estado $|\beta\rangle = |111\rangle$. Para ello construimos el oráculo de la forma habitual. Como 111 es 7 en decimal nuestro oráculo será de la forma:

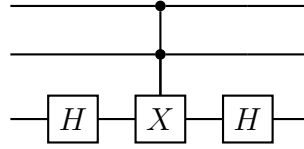
$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix}$$

Lo cual coincide con una puerta Controlled-Controlled-Z (CCZ):

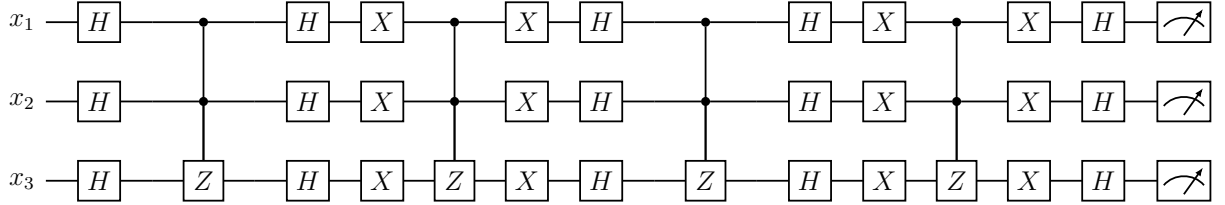


Es fácil ver que esta puerta solo negará el estado $|111\rangle$ ya que la puerta Z solo se activará si el primer y segundo qubit forman el estado $|11\rangle$ y esta puerta niega el estado $|1\rangle$ dejando el resto iguales.

En Qiskit no podemos implementar esta puerta, pero ya vimos en la implementación con dos qubits que una puerta Z se puede obtener a partir de puertas Hadamard y puertas X por lo que nuestro oráculo compatible con Qiskit quedaría tal que:



Tomando la misma transformación de difusión del ejemplo anterior, el circuito para este problema quedaría tal que:



Podemos ver en la figura C.1 como construimos el oráculo en Qiskit. En la figura C.2 vemos como queda el circuito para dos iteraciones, donde en un bucle añadimos dos veces el operador de Grover. Finalmente en la figura C.3 podemos ver como el resultado es el esperado el 95,2 % de las veces (952 de 1000 veces).

Por otro lado, en la figura C.4 vemos como queda el circuito para tres iteraciones, donde en un bucle añadimos tres veces el operador de Grover. En la figura C.5 vemos como ya no obtenemos el estado $|111\rangle$ solo un 31.8 % de las veces, habiendo mucha más variación en los resultados. Por tanto así nos damos cuenta de que haciendo más iteraciones de las debidas nos alejamos de la solución.

Ahora, si seguimos iterando hasta $R+T = 2+9 = 11$ iteraciones, nos volveremos a acercar a la solución. Como el período no es exactamente 9, el sistema no se acercará tanto como para dos iteraciones. Aún así, en la figura C.6 vemos que tras 1000 ejecuciones obtenemos el estado deseado el 80,7 % de las veces.

3.3. Conclusiones

Tras ver los ejemplos anteriores uno puede pensar que hemos construido algo muy tedioso para realizar una búsqueda en una lista de tan solo 8 elementos. Si bien este algoritmo teóricamente se puede implementar para un número indefinido de qubits (por tanto indefinidos elementos), si investigamos en la literatura y en la red no encontramos implementaciones que utilicen más de cuatro o cinco qubits. Hay que tener en cuenta que el número de puertas cuánticas a utilizar aumenta considerablemente cuando aumentamos el número de qubits y que ni los simuladores cuánticos disponibles para su libre uso ni los ordenadores cuánticos existentes son capaces ni de implementar todas las puertas cuánticas posibles ni de superar un cierto número de operaciones en cada ejecución.

También es importante notar que el algoritmo de Grover se centra no en la implementación del mismo, sino en la reducción cuadrática en el número de veces que tenemos que aplicar el oráculo respecto al algoritmo clásico, y así queda reflejado en el artículo original. De alguna forma se intenta ‘camuflar’ la complejidad de la implementación del propio algoritmo.

A parte de la limitación en cuanto a número y tipo de puertas, también hay que tener en cuenta que en la actualidad, aunque existan ordenadores de hasta 127 qubits, por la naturaleza física de los computadores existe un gran problema con el ruido y con los errores de precisión generados en procedimientos y algoritmos que utilizan muchos qubits. Por eso, aunque su implementación pueda ser posible, en el punto en que nos encontramos en la computación cuántica, puede que incluso no merezca la pena implementar este algoritmo o su implementación pues no sería satisfactoria.

Además, a pesar de que el algoritmo de Grover no proporciona una mejora tan significativa como la que consiguen otros algoritmos como el de Shor o el de Deutsch-Jozsa, debemos tener en cuenta que el uso de este algoritmo no queda aislado simplemente en problemas de búsquedas desordenadas. La búsqueda desordenada es un problema recurrente en muchos otros problemas y algoritmos. En concreto, el algoritmo de Grover se utiliza para acelerar algoritmos NP-completos, pues normalmente estos contienen búsquedas exhaustivas como subrutinas.

Por esto mismo debemos valorar el trabajo teórico que hay detrás del algoritmo y enfocarlo como una herramienta que será de gran utilidad en un futuro cercano (y que ya esta siendo de ayuda actualmente).

Capítulo 4

Retrospectiva y conclusiones finales

En este hemos abordado el estudio del algoritmo de Grover desde las matemáticas más básicas que hay detrás de la computación cuántica.

Cuando se habla de computación cuántica, la mayor parte de la gente piensa que es un tema un tanto oscuro, desconocido y complicado. Yo mismo estaba algo asustado al elegir el tema del proyecto y cuando empecé a trabajar en él por primera vez. Claro que había oído hablar de computación cuántica y había leído algún artículo, pero nunca había profundizado en el tema. El realizar el trabajo de fin de grado sobre un tema que ni siquiera se menciona en la carrera de matemáticas era realmente un reto que estaba dispuesto a superar.

Si bien la computación cuántica no es un tema sencillo, en este trabajo hemos visto como sin tener ninguna noción previa del tema, simplemente teniendo nociones de álgebra lineal y de trigonometría, podemos llegar a explicar el funcionamiento de un algoritmo cuántico tan potente como el algoritmo de Grover. Hemos ido introduciendo herramientas matemáticas poco a poco, construyendo un modelo cada vez más complejo que nos ha permitido entender y explicar uno de los mayores hallazgos del mundo de la computación cuántica.

En ocasiones nos abrumamos y no nos vemos capaces de abordar un problema por que aparentemente es complicado. La experiencia nos dice que basta con informarse correctamente en las fuentes adecuadas y dar el primer paso para perder el miedo y poco a poco ir adentrándonos en el problema.

Al igual que realizar la carrera de matemáticas, que tantas cosas me ha enseñado, la experiencia realizando este trabajo de fin de grado no ha sido menos. He pasado por momentos de todo tipo: me he atascado, me he perdido, me he desesperado... pero también me he vuelto a encontrar, he remontado y me he sentido muy agusto trabajando en el proyecto. He disfrutado el proceso y he quedado satisfecho con el resultado final, pero sobre todo he aprendido cosas que no solo aplicaré en mi vida profesional si no que también aplicaré en mi vida personal.

Por último quiero agradecer a mis tutores la propuesta de un tema tan interesante y su apoyo cuando no sabía por donde seguir o cómo resolver cierto tipo de problemas dándome

soluciones y buenos consejos. Este trabajo no hubiera sido posible sin su ayuda.

Anexos

Anexo A

Implementación en Qiskit con 2 qubits

A continuación se muestra el código utilizado para la implementación del algoritmo estructurado en diferentes figuras.

```
[3]: grover=QuantumCircuit(2,2) #Creamos un circuito de 2 qubits
grover.h([0,1]) #INICIALIZACIÓN DE ESTADOS: Puertas hadamard en los dos qubits

job = execute(grover,backend) #Ejecutamos el circuito y guardamos el resultado de la ejecución
result= job.result()
sv = result.get_statevector() #Obtenemos el vector de estados
sv

[3]: array([0.5+0.j, 0.5+0.j, 0.5+0.j, 0.5+0.j])
```

Figura A.1: Inicialización de estados

```
[4]: #Creamos el oráculo y lo añadimos a nuestro circuito
oracle = QuantumCircuit(2,name='oracle')
oracle.cz(0,1) #Puerta Controlled-Z con control en el primer qubit y objetivo en el segundo

grover.append(oracle,[0,1]) #Insertamos el oráculo
grover.draw()
```

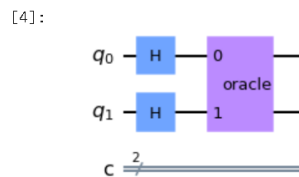


Figura A.2: Construcción del oráculo

```
[5]: job = execute(grover,backend) #Ejecutamos el circuito y guardamos el resultado de la ejecución
      result= job.result()
      sv = result.get_statevector() #Obtenemos el vector de estados
      sv

[5]: array([ 0.5+0.j,  0.5+0.j,  0.5+0.j, -0.5+0.j])
```

Figura A.3: Aplicación del oráculo

```
[6]: diffusion=QuantumCircuit(2,name='diffusion')
      diffusion.h([0,1]) #Puertas hadamard en los dos qubits (desde el qubit 0 hasta el 1)
      diffusion.x([0,1]) #Puertas X en los dos qubits (desde el qubit 0 hasta el 1)
      diffusion.h(1) #Puerta hadamard en el qubit 1
      diffusion.cnot(0,1) #Puerta CNOT con control en el qubit 0 y objetivo en el qubit 1
      diffusion.i(0) #Puerta identidad simplemente para mostrar las siguientes puertas
                      #X y hadamard a la misma altura
      diffusion.h(1)
      diffusion.x([0,1])
      diffusion.h([0,1])
      diffusion.draw() #Dibujamos el circuito
```

[6]:

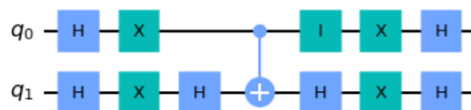


Figura A.4: Construcción de la transformación de difusión

```
[7]: grover.append(diffusion,[0,1])
      grover.measure([0,1],[0,1])
      grover.draw()
```

[7]:

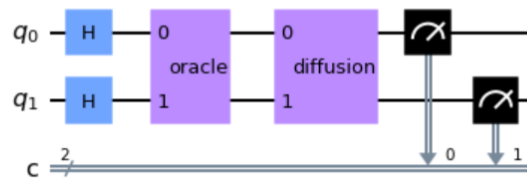


Figura A.5: Circuito completo

```
[8]: job = execute(grover,backend,shots=1000)#Ejecutamos el algoritmo 1000 veces
      result = job.result()
      sv = result.get_statevector() #Obtenemos el vector de estados
      np.around(sv,2)

[8]: array([[ 0.+0.j,  0.-0.j,  0.+0.j, -1.-0.j])
```

Figura A.6: Aplicación de la transformación de difusión

```
[9]: counts=result.get_counts() #Contamos los resultados obtenidos y los mostramos
      counts

[9]: {'11': 1000}
```

Figura A.7: Resultado de la ejecución del algoritmo

```
[15]: grover2=QuantumCircuit(2,2)
      grover2.h([0,1])
      for i in range(7):
          grover2.append(oracle,[0,1])
          grover2.append(diffusion2,[0,1])
      grover2.measure([0,1],[0,1])
      job = execute(grover2,backend,shots=1000)
      result = job.result()
      sv = result.get_statevector()
      np.around(sv,2)
```

```
[15]: array([0.+0.j, 0.+0.j, 0.+0.j, 1.+0.j])
```

```
[16]: counts=result.get_counts()
      counts
```

```
[16]: {'11': 1000}
```

Figura A.8: Resultado iterando siete veces

Anexo B

Implementación en Qiskit con 3 qubits

A continuación se muestra el código utilizado para la implementación del algoritmo estructurado en diferentes figuras.

```
[3]: grover=QuantumCircuit(3,3) #Creamos un circuito de 2 qubits
      grover.h(range(3)) #INICIALIZACIÓN DE ESTADOS: Puertas hadamard en los dos qubits

      job = execute(grover,backend) #Ejecutamos el circuito y guardamos el resultado de la ejecución
      result= job.result()
      sv = result.get_statevector() #Obtenemos el vector de estados
      np.around(sv,2) #Redondeamos

[3]: array([[0.35+0.j, 0.35+0.j, 0.35+0.j, 0.35+0.j, 0.35+0.j, 0.35+0.j,
            0.35+0.j, 0.35+0.j])
```

Figura B.1: Inicialización de estados

```
[4]: oracle = QuantumCircuit(3,name='oracle')

      oracle.cz(2,0) #Puerta Controlled-Z con control en el tercer qubit y objetivo en el primero
      oracle.cz(1,0) #Puerta Controlled-Z con control en el segundo qubit y objetivo en el primero
      grover.append(oracle,range(3))

      grover.draw()
```

[4]:

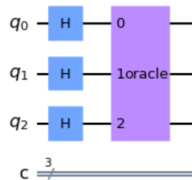


Figura B.2: Construcción del oráculo

```
[5]: job = execute(grover,backend)
result= job.result()
sv = result.get_statevector()
np.around(sv,2)

[5]: array([[ 0.35+0.j,  0.35+0.j,  0.35+0.j, -0.35+0.j,  0.35+0.j, -0.35+0.j,
            0.35+0.j,  0.35-0.j]])
```

Figura B.3: Aplicación del oráculo

```
[6]: diffusion=QuantumCircuit(3,name='diffusion')
diffusion.h(range(3)) #Puertas hadamard en los dos qubits (desde el qubit 0 hasta el 2)
diffusion.x(range(3)) #Puertas X en los dos qubits (desde el qubit 0 hasta el 2)
diffusion.h(2) #Puerta hadamard en el qubit 2
diffusion.mct([0,1],2,0) #Puerta CNOT con control en los qubits 0 y 1 y objetivo en el qubit 2
diffusion.i([0,1]) #Puerta identidad simplemente para mostrar las siguientes puertas
#X y hadamard a la misma altura

diffusion.h(2)
diffusion.x(range(3))
diffusion.h(range(3))
diffusion.draw() #Dibujamos el circuito
```

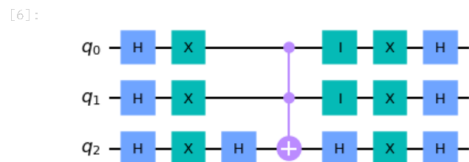


Figura B.4: Construcción de la transformación de difusión

```
[7]: grover.append(diffusion,range(3))
grover.measure(range(3),range(3))
grover.draw()
```

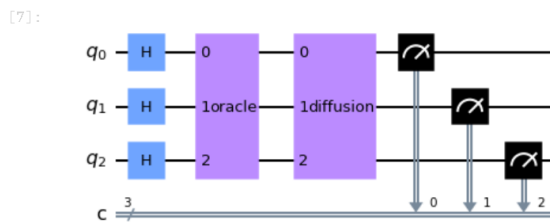


Figura B.5: Circuito completo

```
[8]: job = execute(grover,backend,shots=1000)#Ejecutamos el algoritmo 1000 veces
result = job.result()
sv = result.get_statevector() #Obtenemos el vector de estados
np.around(sv,2)

[8]: array([[ 0.+0.j,  0.+0.j,  0.+0.j,  0.+0.j,  0.+0.j, -1.-0.j, -0.+0.j,
            0.+0.j]])
```

Figura B.6: Aplicación de la transformación de difusión

```
[13]: job = execute(grover,backend,shots=1000)#Ejecutamos el algoritmo 1000 veces
      result = job.result()
      sv = result.get_statevector() #Obtenemos el vector de estados
      np.around(sv,2)

[13]: array([[ 0.+0.j,  0.+0.j,  0.+0.j, -1.-0.j,  0.+0.j,  0.+0.j, -0.+0.j,
              0.-0.j])
```

Figura B.7: Otra aplicación de la transformación de difusión

```
[14]: counts=result.get_counts()
      counts

[14]: {'101': 500, '011': 500}
```

Figura B.8: Resultado de la ejecución del algoritmo

```
[16]: counts=result.get_counts()
      counts

[16]: {'011': 485, '101': 515}
```

Figura B.9: Otro resultado de la ejecución del algoritmo

Anexo C

Implementación en Qiskit con 3 qubits y más iteraciones

A continuación se muestra el código utilizado para la implementación del algoritmo estructurado en diferentes figuras.

```
[12]: backend=Aer.get_backend('statevector_simulator')
      oracle = QuantumCircuit(3,name='oracle')
      oracle.h(2)
      oracle.ccx(0,1,2)
      oracle.h(2)
      oracle.draw()
```

[12]:

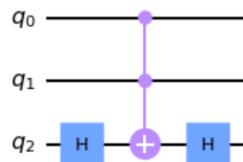


Figura C.1: Construcción del oráculo

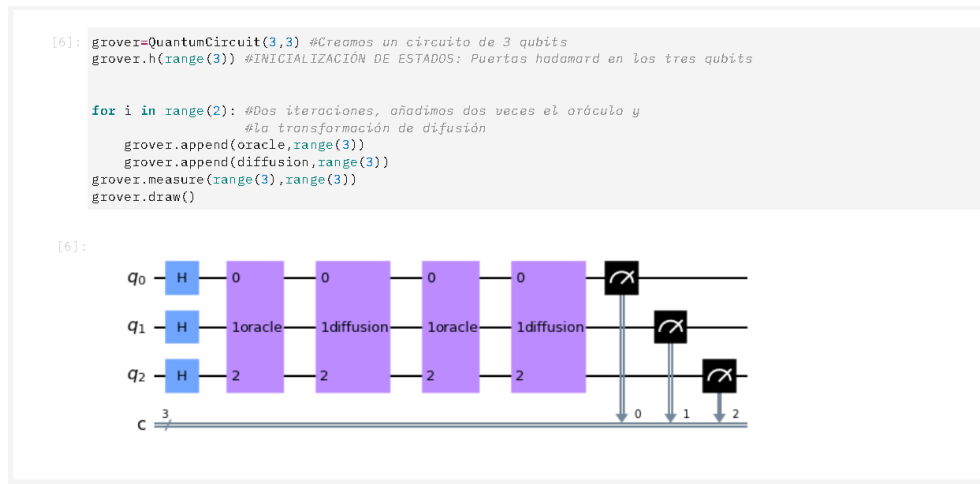


Figura C.2: Circuito completo con 2 iteraciones

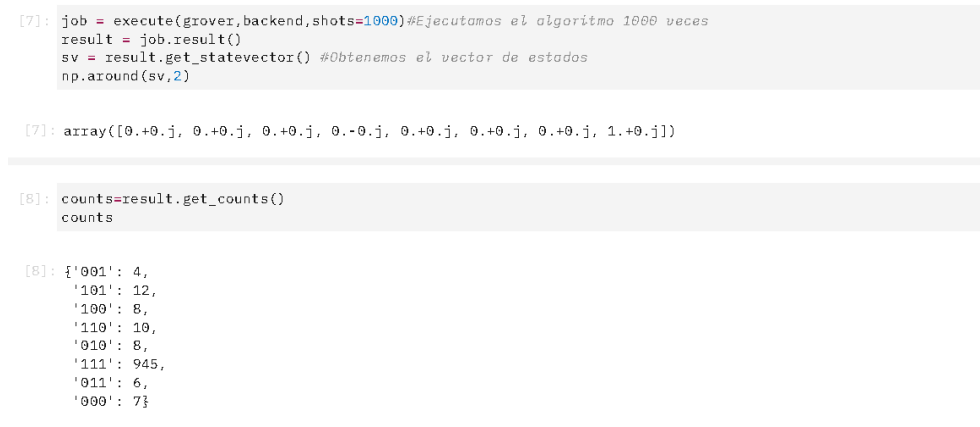


Figura C.3: Resultado iterando dos veces

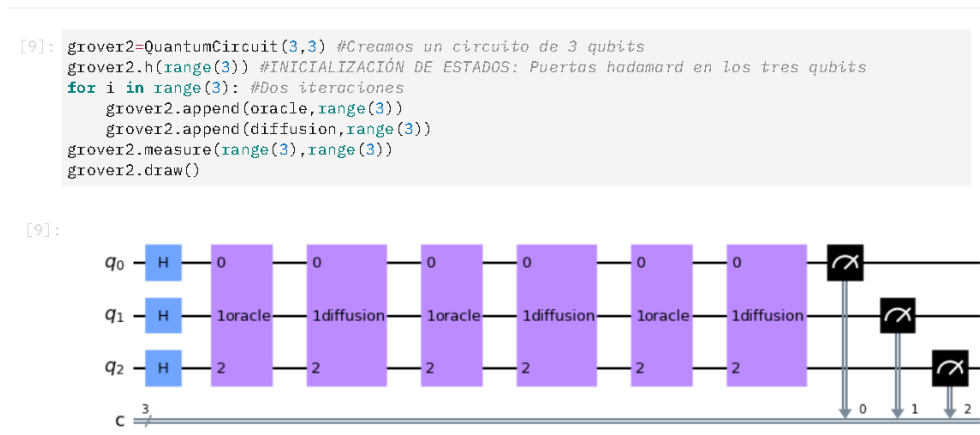


Figura C.4: Circuito completo con 3 iteraciones

```
[13]: job2 = execute(grover2, backend, shots=1000) #Ejecutamos el algoritmo 1000 veces
result2 = job2.result()
sv2 = result2.get_statevector() #Obtenemos el vector de estados
np.around(sv2, 2)
```

```
[13]: array([0.+0.j, 0.+0.j, 0.+0.j, 0.-0.j, 0.+0.j, 0.+0.j, 0.+0.j, 1.+0.j])
```

```
[14]: counts2=result2.get_counts()
counts2
```

```
[14]: {'010': 85,
      '011': 83,
      '000': 91,
      '111': 343,
      '110': 97,
      '100': 105,
      '101': 106,
      '001': 90}
```

Figura C.5: Resultado iterando tres veces

```
[12]: grover2=QuantumCircuit(3,3) #Creamos un circuito de 3 qubits
grover2.h(range(3)) #INICIALIZACIÓN DE ESTADOS: Puertas hadamard en los tres qubits
for i in range(11): #Dos iteraciones
    grover2.append(oracle, range(3))
    grover2.append(diffusion, range(3))
grover2.measure(range(3), range(3))
job2 = execute(grover2, backend, shots=1000) #Ejecutamos el algoritmo 1000 veces
result2 = job2.result()
sv2 = result2.get_statevector() #Obtenemos el vector de estados

counts2=result2.get_counts()
counts2
```

```
[12]: {'101': 21,
      '100': 30,
      '110': 29,
      '010': 38,
      '011': 25,
      '111': 807,
      '000': 28,
      '001': 22}
```

Figura C.6: Resultado iterando once veces

Bibliografía

- [1] Kenneth Regan Richard Linton. *Quantum algorithms via Linear Algebra: A Primer*. The MIT Press, 2014.
- [2] Varios autores. Quantum algorithm implementations for beginners.
- [3] E. Sáenz de Cabezón. Curso computación cuántica 1. <https://www.youtube.com/watch?v=KKwjeJzKew&t=3632s>, Abril 2019.
- [4] Lov K. Grover. A fast quantum mechanical algorithm for database search. page 8, 1996.
- [5] Isaac L. Chuang Michael A. Nielsen. *Quantum Computation and Quantum Information*. Cambridge University Pres, 2010.
- [6] MD SAJID ANIS, Abby-Mitchell, Héctor Abraham, AduOffei, Rochisha Agarwal, Gabriele Agliardi, Merav Aharoni, Vishnu Ajith, Ismail Yunus Akhalwaya, Gadi Aleksandrowicz, Thomas Alexander, Matthew Amy, Sashwat Anagolum, Anthony-Gandon, Eli Arbel, Abraham Asfaw, Anish Athalye, Artur Avkhadiev, Carlos Azaustre, PRATHA-MESH BHOLE, Abhik Banerjee, Santanu Banerjee, Will Bang, Aman Bansal, Panagiotis Barkoutsos, Ashish Barnawal, George Barron, George S. Barron, Luciano Bello, Yael Ben-Haim, M. Chandler Bennett, Daniel Bevenius, Dhruv Bhatnagar, Prakhar Bhatnagar, Arjun Bhobe, Paolo Bianchini, Lev S. Bishop, Carsten Blank, Sorin Bolos, Soham Bopardikar, Samuel Bosch, Sebastian Brandhofer, Brandon, Sergey Bravyi, Nick Bronn, Bryce-Fuller, David Bucher, Artemiy Burov, Fran Cabrera, Padraic Calpin, Lauren Capelluto, Jorge Carballo, Ginés Carrascal, Adam Carriker, Ivan Carvalho, Adrian Chen, Chun-Fu Chen, Edward Chen, Jielun (Chris) Chen, Richard Chen, Franck Chevallier, Karthik Chinda, Rathish Cholarajan, Jerry M. Chow, Spencer Churchill, CisterMoke, Christian Claus, Christian Clauss, Caleb Clothier, Romilly Cocking, Ryan Cocuzzo, Jordan Connor, Filipe Correa, Zachary Crockett, Abigail J. Cross, Andrew W. Cross, Simon Cross, Juan Cruz-Benito, Chris Culver, Antonio D. Córcoles-Gonzales, Navaneeth D, Sean Dague, Tareq El Dandachi, Animesh N Dangwal, Jonathan Daniel, Marcus Daniels, Matthieu Dartailh, Abdón Rodríguez Davila, Faisal Debouni, Anton Dekusar, Amol Deshmukh, Mohit Deshpande, Delton Ding, Jun Doi, Eli M. Dow, Patrick Downing, Eric Drechsler, Eugene

Dumitrescu, Karel Dumon, Ivan Duran, Kareem EL-Safty, Eric Eastman, Grant Eberle, Amir Ebrahimi, Pieter Eendebak, Daniel Egger, ElePT, Emilio, Alberto Espiricueta, Mark Everitt, Davide Facoetti, Farida, Paco Martín Fernández, Samuele Ferracin, Davide Ferrari, Axel Hernández Ferrera, Romain Fouilland, Albert Frisch, Andreas Fuhrer, Bryce Fuller, MELVIN GEORGE, Julien Gacon, Borja Godoy Gago, Claudio Gambella, Jay M. Gambetta, Adhisha Gammanpila, Luis Garcia, Tanya Garg, Shelly Garion, James R. Garrison, Jim Garrison, Tim Gates, Hristo Georgiev, Leron Gil, Austin Gilliam, Aditya Giridharan, Glen, Juan Gomez-Mosquera, Gonzalo, Salvador de la Puente González, Jesse Gorzinski, Ian Gould, Donny Greenberg, Dmitry Grinko, Wen Guan, Dani Guijo, John A. Gunnels, Harshit Gupta, Naman Gupta, Jakob M. Günther, Mikael Haglund, Isabel Haide, Ikko Hamamura, Omar Costa Hamido, Frank Harkins, Kevin Hartman, Areeq Hasan, Vojtech Havlicek, Joe Hellmers, Łukasz Herok, Stefan Hillmich, Hiroshi Horii, Connor Howington, Shaohan Hu, Wei Hu, Chih-Han Huang, Junye Huang, Rolf Huisman, Haruki Imai, Takashi Imamichi, Kazuaki Ishizaki, Ishwor, Raban Iten, Toshinari Itoko, Alexander Ivrii, Ali Javadi, Ali Javadi-Abhari, Wahaj Javed, Qian Jianhua, Madhav Jivrajani, Kiran Johns, Scott Johnstun, Jonathan-Shoemaker, JosDenmark, JoshDumo, John Judge, Tal Kachmann, Akshay Kale, Naoki Kanazawa, Jessica Kane, Kang-Bae, Annanay Kapila, Anton Karazeev, Paul Kassebaum, Tobias Kehrer, Josh Kelso, Scott Kelso, Hugo van Kemenade, Vismai Khanderao, Spencer King, Yuri Kobayashi, Kovi11Day, Arseny Kovyrshin, Rajiv Krishnakumar, Pradeep Krishnamurthy, Vivek Krishnan, Kevin Krsulich, Prasad Kumkar, Gawel Kus, Ryan LaRose, Enrique Lacal, Raphaël Lambert, Haggai Landa, John Lapeyre, Joe Latone, Scott Lawrence, Christina Lee, Gushu Li, Tan Jun Liang, Jake Lishman, Dennis Liu, Peng Liu, Lolcroc, Abhishek K M, Liam Madden, Yunho Maeng, Saurav Maheshkar, Kahan Majmudar, Aleksei Malyshev, Mohamed El Mandouh, Joshua Manela, Manjula, Jakub Marecek, Manoel Marques, Kunal Marwaha, Dmitri Maslov, Paweł Maszota, Dolph Mathews, Atsushi Matsuo, Farai Mazhandu, Doug McClure, Maureen McElaney, Cameron McGarry, David McKay, Dan McPherson, Srujan Meesala, Dekel Meirom, Corey Mendell, Thomas Metcalfe, Martin Mevissen, Andrew Meyer, Antonio Mezzacapo, Rohit Midha, Daniel Miller, Hannah Miller, Zlatko Minev, Abby Mitchell, Nikolaj Moll, Alejandro Montanez, Gabriel Monteiro, Michael Duane Mooring, Renier Morales, Niall Moran, David Morcuende, Seif Mostafa, Mario Motta, Romain Moyard, Prakash Murali, Daiki Murata, Jan Müggenburg, Tristan NEMOZ, David Nadlinger, Ken Nakanishi, Giacomo Nannicini, Paul Nation, Edwin Navarro, Yehuda Naveh, Scott Wyman Neagle, Patrick Neuweiler, Aziz Ngoueya, Thien Nguyen, Johan Nicander, Nick-Singstock, Pradeep Niroula, Hassi Norlen, NuoWenLei, Lee James O’Riordan, Oluwatobi Ogunbayo, Pauline Ollitrault, Tamiya Onodera, Raul Otaolea, Steven Oud, Dan Padilha, Hanhee Paik, Soham Pal, Yuchen Pang, Ashish Panigrahi, Vincent R. Pascuzzi, Simone Perriello, Eric Peterson, Anna Phan, Kuba Pilch, Francesco Piro, Marco Pistoia, Christophe Piveteau, Julia Plewa, Pierre Pocreau, Alejandro Pozas-Kerstjens, Rafał Pracht,

Milos Prokop, Viktor Prutyantov, Sumit Puri, Daniel Puzzuoli, Pythonix, Jesús Pérez, Quant02, Quintiii, Rafey Iqbal Rahman, Arun Raja, Roshan Rajeev, Isha Rajput, Nipun Ramagiri, Anirudh Rao, Rudy Raymond, Oliver Reardon-Smith, Rafael Martín-Cuevas Redondo, Max Reuter, Julia Rice, Matt Riedemann, Rietesh, Drew Risinger, Pedro Rivero, Marcello La Rocca, Diego M. Rodríguez, RohithKarur, Ben Rosand, Max Rossmann, Mingi Ryu, Tharmashastha SAPV, Nahum Rosa Cruz Sa, Arijit Saha, Abdullah Ash-Saki, Sankalp Sanand, Martin Sandberg, Hirmay Sandesara, Ritvik Sapra, Hayk Sargsyan, Aniruddha Sarkar, Ninad Sathaye, Niko Savola, Bruno Schmitt, Chris Schnabel, Zachary Schoenfeld, Travis L. Scholten, Eddie Schoute, Mark Schulterbrandt, Joachim Schwarm, James Seaward, Sergi, Ismael Faro Sertage, Kanav Setia, Freya Shah, Nathan Shammah, Will Shanks, Rohan Sharma, Yunong Shi, Jonathan Shoemaker, Adenilton Silva, Andrea Simonetto, Deeksha Singh, Divyanshu Singh, Parmeet Singh, Phattharaporn Singkanipa, Yukio Siraichi, Siri, Jesús Sistos, Iskandar Sitdikov, Seyon Sivarajah, Slavikmew, Magnus Berg Sletfjerding, John A. Smolin, Mathias Soeken, Igor Olegovich Sokolov, Igor Sokolov, Vicente P. Soloviev, SooluThomas, Starfish, Dominik Steenken, Matt Stypulkoski, Adrien Suau, Shaojun Sun, Kevin J. Sung, Makoto Suwama, Oskar Słowik, Hitomi Takahashi, Tanvesh Takawale, Ivano Tavernelli, Charles Taylor, Pete Taylor, Soolu Thomas, Kevin Tian, Mathieu Tillet, Maddy Tod, Miroslav Tomasik, Caroline Tornow, Enrique de la Torre, Juan Luis Sánchez Toural, Kenso Trabing, Matthew Treinish, Dimitar Trennev, TrishaPe, Felix Truger, Georgios Tsilimigkounakis, Davindra Tulsi, Doğukan Tuna, Wes Turner, Yotam Vaknin, Carmen Recio Valcarce, Francois Varchon, Adish Vartak, Almudena Carrera Vazquez, Prajjwal Vijaywargiya, Victor Villar, Bhargav Vishnu, Desiree Vogt-Lee, Christophe Vuillot, James Weaver, Johannes Weidenfeller, Rafal Wieczorek, Jonathan A. Wildstrom, Jessica Wilson, Erick Winston, WinterSoldier, Jack J. Woehr, Stefan Woerner, Ryan Woo, Christopher J. Wood, Ryan Wood, Steve Wood, James Wootton, Matt Wright, Lucy Xing, Jintao YU, Bo Yang, Unchun Yang, Jimmy Yao, Daniyar Yeralin, Ryota Yonekura, David Yonge-Mallo, Ryuhei Yoshida, Richard Young, Jessie Yu, Lebin Yu, Yuma-Nakamura, Christopher Zachow, Laura Zdanski, Helena Zhang, Iulia Zidaru, Bastian Zimmermann, Christa Zoufal, aeddins ibm, alexzhang13, b63, bartek bartlomiej, bcamorrison, brandhsn, chetmurthy, deeplokhande, dekel.meirom, dime10, dlasecki, ehchen, ewinston, fanizzamarco, fs1132429, gadial, galeinston, georgezhou20, georgios ts, gruu, hhorii, hhyap, hykavitha, itoko, jeppevin- kel, jessica angel7, jezerjojo14, jliu45, johannesgreiner, jscott2, klinvill, krutik2966, ma5x, michelle4654, msuwama, nico lgrs, nrhawkins, ntgiwsvp, ordmoj, sagar pahwa, pritamsinha2304, rithikaadiga, ryancocuzzo, saktar unr, saswati qiskit, septembr, sethmerkel, sg495, shaashwat, smturro2, sternparky, strickroman, tigerjack, tsura crisaldo, upsideon, vadebayo49, welien, willhbang, wmurphy collabstar, yang.luh, and Mantas Čepulkovskis. Qiskit: An open-source framework for quantum computing, 2021.

- [7] Amira Abbas, Stina Andersson, Abraham Asfaw, Antonio Corcoles, Luciano Bello, Yael Ben-Haim, Mehdi Bozzo-Rey, Sergey Bravyi, Nicholas Bronn, Lauren Capelluto, Almudena Carrera Vazquez, Jack Ceroni, Richard Chen, Albert Frisch, Jay Gambetta, Shelly Garion, Leron Gil, Salvador De La Puente Gonzalez, Francis Harkins, Takashi Imamiuchi, Pavan Jayasinha, Hwajung Kang, Amir h. Karamlou, Robert Lored, David McKay, Alberto Maldonado, Antonio Macaluso, Antonio Mezzacapo, Zlatko Minev, Ramis Movassagh, Giacomo Nannicini, Paul Nation, Anna Phan, Marco Pistoia, Arthur Rattew, Joachim Schaefer, Javad Shabani, John Smolin, John Stenger, Kristan Temme, Madeleine Tod, Ellinor Wanzambi, Stephen Wood, and James Wootton. Learn quantum computation using qiskit, 2020.