



Escuela Técnica Superior de
Ingeniería Informática

TRABAJO FIN DE GRADO
Criptografía y computación cuántica

Realizado por
Irene Fernández Romero

Para la obtención del título de
Grado en Ingeniería Informática – Tecnología Informática

Dirigido por
Jose Andrés Armario
En el departamento de
Matemática Aplicada I

Convocatoria de julio, curso 2023/24

*A mis padres,
por apoyarme, cuidarme, enseñarme y quererme incondicionalmente.*

Agradecimientos

En primer lugar, quiero agradecer a mi tutor, José Andrés, por su orientación, paciencia y dedicación a lo largo de todo el proceso de la elaboración de este Trabajo de Fin de Grado.

Quiero agradecer a mi familia, mis padres, por haberme apoyado en todas mis decisiones, por haberme comprendido en mis momentos más difíciles y, sobre todo, por haber confiado en mí.

Quiero expresar mi gratitud a mis amigos, Ana y Toni, por estar siempre cuando más los necesito y por ayudarme con todo día sí y día también.

Por último, agradecer a todos mis amigos y compañeros de clase, que han hecho de estos últimos años una experiencia y momentos inolvidables.

Resumen

En este trabajo de fin de grado, hemos introducido y desarrollado las nociones básicas sobre la computación cuántica, un campo emergente que promete revolucionar la informática y la criptografía. Hemos trabajado con dos algoritmos fundamentales: el algoritmo de Deutsch-Jozsa y el algoritmo de Grover, además de experimentar con ellos en Qiskit y observar su consecuencia con la criptografía. Se ofrece también un pequeño manual de usuario sobre cómo trabajar en este entorno.

Palabras clave: Qubit, entrelazamiento, superposición, algoritmo de Deutsch-Jozsa, algoritmo de Grover.

Abstract

In this final degree project, we have introduced and developed the basic notions of quantum computing, an emerging field that promises to revolutionize computing and cryptography. We have worked with two fundamental algorithms: the Deutsch-Jozsa algorithm and Grover's algorithm. Additionally, we have experimented with these algorithms in Qiskit and observed their implications for cryptography. A small user manual on how to work in this environment is also provided.

Keywords: qubit, entanglement, superposition, Deutsch-Jozsa algorithm, Grover's algorithm.

Índice general

Resumen.....	2
Abstract	3
Introducción	10
1. Principios de la computación cuántica	11
1.1. Qubits.....	11
1.2. Puertas cuánticas de un qubit	14
1.1.1. Puertas de Pauli.....	15
1.1.2. Puerta de Hadamard.....	16
1.3. Puertas cuánticas de múltiples qubits	18
1.4. Medición de qubits.....	23
1.5. Circuitos cuánticos.....	26
2. Algoritmo de Deutsch-Jozsa	28
2.1. Funciones booleanas	28
2.2. Transformada de Walsh.....	29
2.3. Implementación de las funciones booleanas	29
2.4. Descripción del algoritmo de Deutsch.....	32
2.5. Descripción del algoritmo de Deutsch-Jozsa	33
2.6. Descripción circuito del algoritmo con Qiskit	35
3. Algoritmo de Grover	37
3.1. Descripción del algoritmo.....	37
3.2. Ejemplo para $n = 3$	41

3.3. Desarrollo del algoritmo en Qiskit	43
4. Manual de usuario de Qiskit.....	46
4.1. Instalación de Qiskit	46
4.2. Estructura básica de un programa en Qiskit	50
4.2.1. Clases	50
4.2.2. Creación de circuitos cuánticos.....	50
4.2.3. Añadir puertas cuánticas.....	50
4.2.4. Medición de qubits.....	51
4.3. Ejecución de un circuito	51
5. Consecuencias del algoritmo de Grover.....	52
6. Posibles ampliaciones del trabajo.....	56
6.1. IBM Quantum Composer.....	56
6.2. Algoritmo de Shor.....	65
7. Conclusiones	67
Referencias	68

Índice de figuras

Figura 1 Inicialización de un qubit.....	13
Figura 2 Esfera de Bloch.....	13
Figura 3 Puertas cuánticas para un qubit	18
Figura 4 Puerta cuántica CNOT	21
Figura 5 Puerta cuántica SWAP	22
Figura 6 Medición de un qubit.....	24
Figura 7 Medición de qubit aplicando Hadamard.....	25
Figura 8 Circuito cuántico	27
Figura 9 Circuito cuántico de la función AND	30
Figura 10 Oráculo	31
Figura 11 Circuito cuántico del algoritmo de Deutsch.....	32
Figura 12 Circuito cuántico del algoritmo de Deutsch-Jozsa.....	34
Figura 13 Algoritmo de Deutsch-Jozsa.....	36
Figura 14 Circuito algoritmo de Grover.....	39
Figura 15 Salida algoritmo de Grover.....	45
Figura 16 Circuito del algoritmo de Grover	45
Figura 17 Agregar extensión a Visual Studio Code.....	47
Figura 18 Terminal	47
Figura 19 comando conda install pip.....	47
Figura 20 Crear carpeta.....	48
Figura 21 Crear archivo nuevo.....	48
Figura 22 Jupyter Notebook	48
Figura 23 Seleccionar Kernel.....	48
Figura 24 Elegir entorno	49
Figura 25 Entorno recomendado.....	49
Figura 26 Importación de qiskit.....	49
Figura 27 Versión de qiskit	49
Figura 28 Creación de circuitos cuánticos.....	50
Figura 29 Añadir puertas cuánticas.....	50
Figura 30 Medición de qubits.	51
Figura 31 Ejecución de código.....	51
Figura 32 Una iteración para el algoritmo de Grover.	54
Figura 33 Dos iteraciones para el algoritmo de Grover.	54
Figura 34 Tres iteraciones para el algoritmo de Grover.....	54

Figura 35 Cuatro iteraciones para el algoritmo de Grover.....	54
Figura 36 Cinco iteraciones para el algoritmo de Grover.	54
Figura 37 IBM Quantum Composer	56
Figura 38 IBM Composer: representación puerta X	57
Figura 39 IBM Composer: implementación puerta X sobre primer qubit	57
Figura 40 IBM Composer: implementación puerta X sobre segundo qubit	58
Figura 41 IBM Composer: implementación puerta X en ambos qubits.....	58
Figura 42 IBM Composer: representación puerta Y	58
Figura 43 IBM Composer: implementación puerta Y sobre ambos qubits	59
Figura 44 IBM Composer: representación puerta Z.....	59
Figura 45 IBM Composer: implementación puerta Z sobre el primer qubit.....	60
Figura 46 IBM Composer: implementación puerta Z sobre segundo qubit	60
Figura 47 IBM Composer: representación puerta H.....	60
Figura 48 IBM Composer: implementación puerta H sobre qubit estado 0.....	61
Figura 49 IBM Composer: implementación puerta H sobre qubit estado 1.....	61
Figura 50 IBM Composer: representación puerta CNOT.....	61
Figura 51 IBM Composer: implementación puerta CNOT sobre qubit estado 0.....	62
Figura 52 IBM Composer: implementación puerta CNOT sobre qubit 1	62
Figura 53 IBM Composer: representación puerta SWAP	62
Figura 54 IBM Composer: implementación puerta SWAP sobre estado 10.....	63
Figura 55 IBM Composer: implementación puerta SWAP sobre estado 01.....	63
Figura 56 IBM Composer: Algoritmo de Deutsch-Jozsa.....	64
Figura 57 IBM Composer: Algoritmo de Deutsch-Jozsa para una función constante.....	64
Figura 58 IBM Composer: Algoritmo de Deutsch-Jozsa para una función balanceada.....	65
Figura 59 IBM Composer: Algoritmo de Grover	65

Índice de tablas

Tabla 1 Puertas cuánticas de un qubit.....	17
Tabla 2 Puertas cuánticas para más de un qubit.....	23

Índice de extractos de código

Código 1 Inicialización de un qubit	12
Código 2 Establecer un estado cuántico	13
Código 3 Implementación de puertas cuánticas para un qubit	17
Código 4 Implementación de la puerta cuántica CNOT	21
Código 5 Implementación de la puerta cuántica SWAP	22
Código 6 Implementación de la medición de un qubit	24
Código 7 Implementación medición de qubit aplicando Hadamard	25
Código 8 Ejemplo de un pequeño circuito cuántico	27
Código 9 Implementación de un oráculo	31
Código 10 Implementación del algoritmo de Deutsch-Jozsa	36
Código 11 Implementación del algoritmo de Grover	44

Introducción

La computación cuántica ha emergido como un campo revolucionario en el mundo de la informática, resolviendo algunos problemas complejos de una forma más eficiente y eficaz que las computadoras clásicas. En este trabajo, exploramos diversos aspectos de la computación cuántica, desde los fundamentos teóricos hasta la implementación práctica de algoritmos cuánticos.

En el capítulo uno introduciremos conceptos básicos de la computación cuántica, como viene siendo los qubits y las puertas cuánticas más importantes y utilizadas para la manipulación de estos qubits para realizar cálculos complejos. Abordaremos además conceptos cruciales como son las propiedades de superposición y entrelazamiento.

En el capítulo dos, hablaremos sobre uno de los algoritmos más importantes de la computación cuántica, el algoritmo de Deutsch-Jozsa. Gracias a este algoritmo, podemos determinar si una función es constante o balanceada con una única consulta. Realizaremos en Qiskit algunas pruebas sobre cómo funciona el algoritmo.

En el capítulo tres, comentaremos acerca de otro de los algoritmos más importantes, el algoritmo de Grover, que proporciona una búsqueda cuántica más eficiente en base de datos no estructuradas. Exploraremos cómo este algoritmo puede encontrar la entrada deseada en una lista no ordenada en un tiempo cuadrático con respecto al algoritmo clásico.

Para facilitar la experimentación y el uso de Qiskit, el capítulo cuatro consiste en un pequeño manual de usuario donde se detallan instrucciones paso a paso sobre cómo configurar el entorno detallado, ejecuciones de programas cuánticos y análisis de resultados.

En el capítulo cinco hablaremos sobre las consecuencias que tiene el algoritmo de Grover en la criptografía, además de experimentar qué ocurre si realizamos demasiadas iteraciones del algoritmo.

Finalmente, acabaremos el trabajo con un sexto capítulo en el que trataremos sobre algunas posibles ampliaciones del trabajo.

1. Principios de la computación cuántica

Las primeras aportaciones sobre la computación cuántica se atribuyen a los físicos Paul Benioff y Richard Feynman, a principios de la década de 1980. Fueron los pioneros en plantear la idea de aplicar algunos de los principios de la mecánica cuántica a la computación. Aunque habría que esperar 30 años hasta la creación del primer ordenador cuántico, Benioff diseñó en 1981 la primera computadora cuántica, basándose en la máquina de Turing.

La computación cuántica es una forma de procesamiento de información que se basa en los principios de la mecánica cuántica. Une disciplinas como ciencias de la computación, física y matemáticas, y aprovecha aspectos de la mecánica cuántica para resolver algunos problemas complejos que ordenadores clásicos no pueden. Dentro de las principales funciones que se pueden hacer con la computación cuántica está la resolución de ciertos problemas de optimización, para encontrar la mejor solución entre múltiples posibilidades.

La computación cuántica se caracteriza por el uso de qubits, en lugar de bits clásicos como ocurre en la computación clásica. Éstos qubits utilizan propiedades de superposición y entrelazamiento para realizar cálculos en paralelo, tener una mayor potencia de cálculo, aumentar su capacidad de memoria y menor consumo de energía. Los ordenadores cuánticos aprovechan estas propiedades para procesar información de manera más eficiente que las computadoras tradicionales.

Varias empresas que están desarrollando proyectos innovadores en este campo, como IBM, que creó IBM Quantum, una plataforma que proporciona acceso a ordenadores cuánticos a través de la nube. IBM lanzó a finales de 2022 un procesador de 433 qubits, lo que supuso un salto importante en la capacidad de procesamiento, mientras que para 2023 lanzaron un procesador con más de 1.000 qubits, lo que supuso una enorme mejora en la capacidad de computadoras cuánticas. Otra empresa global que se ha unido ha sido Amazon, con Amazon Web Services, que facilita la experimentación con algoritmos cuánticos mediante la integración de hardware cuántico de diferentes proveedores. Estos proyectos están impulsando importantes avances en la computación cuántica, acercándonos cada vez más a una nueva era en el procesamiento de la información.

1.1. Qubits

Un qubit, o bit cuántico, es la unidad básica de información en la computación cuántica [1](pg.1-3), [2], [3](pg.3,4). Un qubit tiene dos estados ortogonales, denotados como $|0\rangle$ y $|1\rangle$, según la Notación de Dirac, o también conocida como notación bra-ket. Éstos se leen como “ket cero” y “ket 1”, respectivamente.

Cualquier qubit se puede escribir como una combinación lineal de los estados $|0\rangle$ y $|1\rangle$. El estado de un qubit se puede representar como un vector de dimensión 2. Los estados $|0\rangle$ y $|1\rangle$ representan los vectores:

$$|0\rangle \equiv \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad |1\rangle \equiv \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

La base formada por estos vectores se conoce como base computacional. Los qubits, a diferencia de los bits clásicos, pueden encontrarse en una superposición de los estados anteriores. La superposición [4] es una propiedad fundamental que permite a los qubits existir en múltiples estados diferentes al mismo tiempo. Suponiendo una situación en la que necesitemos encontrar una solución óptima de un problema con múltiples soluciones, en una computadora clásica, tendríamos que probar cada posible solución secuencialmente, lo que nos puede llevar mucho tiempo, sobre todo si hay que considerar una gran cantidad de soluciones. Esto es porque los bits clásicos solo pueden estar en un estado definido.

Sin embargo, gracias a la superposición, podemos explorar múltiples soluciones simultáneamente y por ello, podemos realizar cálculos mucho más complejos y a una velocidad exponencialmente mayor, debido a que, a diferencia de que un algoritmo clásico que trabaja sobre n bits, el aumento de un qubit supone un aumento exponencial de 2^{n+1} .

De tal forma, un estado de qubit se representa como

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

donde α y $\beta \in \mathbb{C}$ se denominan amplitudes de $|0\rangle$ y $|1\rangle$ respectivamente, que representan la probabilidad de encontrar el qubit en el estado $|0\rangle$ o $|1\rangle$, y \mathbb{C} es el conjunto de todos los números complejos. Debemos tener en cuenta que la suma de los cuadrados de los módulos de α y β debe ser igual a 1, es decir, $|\alpha|^2 + |\beta|^2 = 1$.

Con esta representación, podemos ver que un estado arbitrario se puede representar como

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \equiv \alpha \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \beta \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

En Qiskit [1](pg.3), un qubit se inicializa de la siguiente manera

```
1 #Importing the required classes and modules
2 from qiskit import QuantumRegister
3 qr = QuantumRegister(1, "q")
```

Código 1 Inicialización de un qubit.

El código que proporcionamos anteriormente inicializará un único qubit a $|0\rangle$ con el nombre de q . Si desglosamos este código, podemos ver que en la línea 2 importamos la clase *QuantumRegister*, en la línea 3 creamos la variable qr a la que le proporcionamos un objeto de la clase *QuantumRegister* cuyos parámetros son el número de qubits y el nombre que se le asocia.

El estado del qubit qr también puede establecerse en cualquier estado arbitrario válido, aparte de $|0\rangle$. Esto se hace ya sea evolucionando el qubit qr usando puertas cuánticas, que las veremos en las siguientes secciones, o utilizando el método *initialize()*. Por ejemplo, el qubit qr se puede establecer en el estado cuántico $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$ utilizando *initialize()*. El código se muestra a continuación [1](pg.3)

```

1 # Importar las clases y módulos necesarios
2 from qiskit import QuantumRegister, QuantumCircuit
3 from math import sqrt
4 # Definir el estado deseado
5 desired_state = [1/sqrt(2), 1/sqrt(2)]
6 # Crear un registro cuántico con un solo qubit llamado "q"
7 qr = QuantumRegister(1, "q")
8 # Crear un circuito cuántico
9 qc = QuantumCircuit(qr)
10 # Inicializar el qubit al estado deseado
11 qc.initialize(desired_state, qr[0])
12 # Dibujar el circuito cuántico
13 qc.draw(output='mpl')

```

Código 2 Establecer un estado cuántico

Si analizamos cuidadosamente el código de arriba, en la línea 2 importamos las clases *QuantumRegister* y *QuantumCircuit*, que serán necesarias para trabajar con circuitos cuánticos. En la siguiente línea, importamos la función matemática *sqrt*, que se utilizará para poder calcular la raíz cuadrada de un número. En la línea 5 asignamos a *desired_state* el vector $[\alpha, \beta]$, donde $\alpha|0\rangle + \beta|1\rangle$ es el estado del qubit que deseamos crear. Seguido, creamos un objeto *QuantumRegister* llamado *qr*. La línea 9 del código lo que hace es crear un objeto *QuantumCircuit* denominado *qc* con *qr* como el qubit del circuito. Finalmente, utilizamos el método *initialize()* para inicializar el qubit *qr* al estado especificado por *desired_state*. Finalmente, para dibujar el circuito emplearemos la función *qc.draw(output = 'mpl')* donde *output = 'mpl'* sirve para indicar que la visualización del circuito debe generarse utilizando matplotlib y poder así generar un gráfico. La salida de este código por tanto es la siguiente

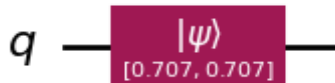


Figura 1 Inicialización de un qubit

El estado de un qubit se puede representar geoméricamente en la esfera de Bloch como se muestra en la figura 2.

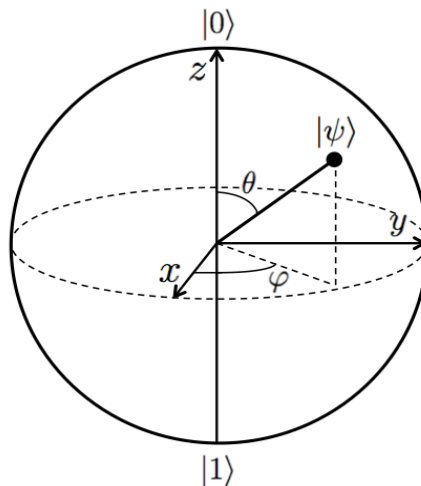


Figura 2 Esfera de Bloch

Los módulos son los que realmente caracterizan la distribución de la probabilidad que realmente importa, no las amplitudes α y β . Por lo tanto, dos qubits que presentan la misma distribución son indistinguibles computacionalmente. Por lo tanto, podemos elegir un representante para todos esos qubits.

Podemos forzar a α a ser un número real y, como dado un qubit $|\psi\rangle$ solo hay un qubit

$$|\psi_0\rangle = \cos\frac{\theta}{2} + e^{i\varphi} \sin\frac{\theta}{2}$$

con $\theta \in [0, \pi]$ y $\varphi \in [0, 2\pi)$. Escogiendo tal representación, tenemos que un qubit genérico se puede representar de manera única como un punto (θ, φ) .

1.2. Puertas cuánticas de un qubit

Las puertas cuánticas tienen como función cambiar el estado de uno o más qubits. Cada operación debe transformar un estado válido de un qubit en otro estado válido del mismo.

Una característica importante de una puerta cuántica es que todas las operaciones son reversibles [1](pg.4). Básicamente, si una puerta cuántica actúa sobre un estado de entrada $|\psi\rangle$ para dar algún estado de salida $|\varphi\rangle$, entonces, dado el estado $|\varphi\rangle$, deberíamos poder obtener el estado $|\psi\rangle$ a través de alguna operación de puerta cuántica válida. Estas operaciones no son más que transformaciones de un vector complejo normalizando a otro, haciendo uso de transformaciones unitarias. Dado que cualquier transformación unitaria se puede representar por una matriz unitaria, para cualquier puerta cuántica, existe una matriz unitaria correspondiente [5].

Recordemos que una matriz unitaria es una matriz cuadrada U cuya inversa es igual a su conjugado transpuesto. Una matriz U se considera unitaria si cumple la siguiente condición:

$$U^*U = UU^* = I$$

siendo U^* el conjugado transpuesto de U e I la matriz identidad. Por ejemplo, una matriz unitaria sería la siguiente

$$U = \begin{pmatrix} i & 0 \\ 0 & -i \end{pmatrix}$$

ya que la conjugada transpuesta de U sería

$$U^* = \begin{pmatrix} -i & 0 \\ 0 & i \end{pmatrix}$$

y la inversa de U sería

$$U^{-1} = \begin{pmatrix} -i & 0 \\ 0 & i \end{pmatrix}$$

Como $U^* = U^{-1}$ y, además, realizando la operación U^*U

$$U^*U = \begin{pmatrix} -i & 0 \\ 0 & i \end{pmatrix} \begin{pmatrix} i & 0 \\ 0 & -i \end{pmatrix} = \begin{pmatrix} -i^2 & 0 \\ 0 & -i^2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

nos sale una matriz identidad, U es una matriz unitaria.

Comenzaremos hablando primero sobre de las puertas de un solo qubit. Este tipo de puertas lo que hacen es tomar un estado de un solo qubit como entrada y producir un estado de un solo qubit como salida. Existen tres puertas cuánticas de un solo qubit muy importantes. Estas son la puerta de Pauli-X, la puerta de Pauli-Y y la puerta de Pauli-Z, también conocidas como X, Y y Z.

1.1.1. Puertas de Pauli

Para un qubit definido como $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, las puertas de Pauli se expresan como se muestra a continuación [1](pg.4,5).

Para la puerta X tenemos la siguiente definición

$$X : \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \alpha \end{bmatrix}$$

Si nos fijamos bien, podemos ver que esta puerta actúa de forma similar a una puerta clásica NOT. Su función consiste en cambiar el estado $|0\rangle$ a $|1\rangle$ y viceversa.

Para la puerta Y tenemos la siguiente expresión

$$Y : \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} -i\beta \\ i\alpha \end{bmatrix}$$

donde i es la unidad imaginaria. Esta puerta lo que hace es realizar una rotación de 180 grados alrededor del eje, lo mismo que aplicar una fase de π .

Finalmente tenemos la puerta Z cuya representación es la siguiente

$$Z : \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \alpha \\ -\beta \end{bmatrix}$$

Esta puerta introduce una diferencia de fase relativa de π entre $|0\rangle$ y $|1\rangle$. Esto quiere decir que la puerta Z multiplica por -1 la amplitud de $|1\rangle$.

Usando la notación bra-ket, estas acciones se pueden describir de la siguiente manera:

$$X|\psi\rangle = X(\alpha|0\rangle + \beta|1\rangle) = \alpha X|0\rangle + \beta X|1\rangle = \alpha|1\rangle + \beta|0\rangle$$

$$Y|\psi\rangle = Y(\alpha|0\rangle + \beta|1\rangle) = \alpha Y|0\rangle + \beta Y|1\rangle = i\alpha|1\rangle - i\beta|0\rangle$$

$$Z|\psi\rangle = Z(\alpha|0\rangle + \beta|1\rangle) = \alpha Z|0\rangle + \beta Z|1\rangle = \alpha|0\rangle - \beta|1\rangle$$

1.1.2. Puerta de Hadamard

Otra puerta muy importante de un solo qubit que utilizaremos es la puerta de Hadamard [1](pg.5-7), [2]. Esta puerta genera superposición equiprobable, donde no se favorece ninguno de los estados básicos. La puerta de Hadamard se representa de la siguiente manera

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

Si la puerta H se aplica sobre el estado $|0\rangle$, obtendremos lo siguiente

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

cuya representación vectorial del estado es $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$.

Si la puerta H se aplica esta vez sobre el estado $|1\rangle$, obtendremos

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

que corresponde al estado $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$.

Si denotamos $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ y $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ como $|+\rangle$ y $|-\rangle$ respectivamente, podemos definir lo siguiente:

$$\begin{aligned} |+\rangle &\equiv H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \\ |-\rangle &\equiv H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \end{aligned}$$

Podemos pensar en el estado $|+\rangle$ como la superposición de los estados $|0\rangle$ y $|1\rangle$. Es decir, cuando se aplica una unidad sobre $|+\rangle$, básicamente se está aplicando tanto en $|0\rangle$ como en $|1\rangle$. Esta propiedad de la computación cuántica, donde una sola operación actúa sobre todos los estados simultáneamente en paralelo, se llama paralelismo cuántico. Gracias a ella, permite acelerar exponencialmente los cálculos respecto a la computación clásica.

Si observamos bien, vemos que los estados $\{|+\rangle, |-\rangle\}$ son ortogonales entre sí, y al igual que en la base computacional, estos estados forman otra base. Dos estados se consideran ortogonales entre sí, si el producto punto de sus vectores correspondientes es 0. Por lo tanto, el conjunto $\{|+\rangle, |-\rangle\}$ también forma una base para el espacio vectorial de los estados de un qubit, y a esto se le conoce como la base de Hadamard.

Para que cualquier qubit en el estado $|+\rangle$ o $|-\rangle$, la probabilidad de observar $|0\rangle$ en una medición es $1/2$ y la de $|1\rangle$ también es $1/2$.

A continuación, se muestra un resumen de las puertas cuánticas sobre un qubit.





PUERTA CUÁNTICA	SÍMBOLO	MATRIZ
X		$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Y		$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
Z		$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Hadamard		$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$

Tabla 1 Puertas cuánticas de un qubit

A continuación, se muestra el código en Qiskit sobre cómo se implementarían estas puertas [1](pg.6).

```

1 #Importar clases y módulos requeridos
2 from qiskit import QuantumRegister , QuantumCircuit
3 # Crear un registro cuántico con 4 qubits
4 qr = QuantumRegister(4)
5 qc = QuantumCircuit(qr)
6 # Aplicar las puertas a los qubits específicos
7 qc.h(qr[0]) # Aplicar puerta H al qubit 0
8 qc.z(qr[1]) # Aplicar puerta Z al qubit 1
9 qc.x(qr[2]) # Aplicar puerta X al qubit 2
10 qc.y(qr[3]) # Aplicar puerta Y al qubit 3
11 # Dibujar el circuito
12 qc.draw(output='mpl')
```

Código 3 Implementación de puertas cuánticas para un qubit

Si explicamos un poco el código, vemos que en la línea 2 importamos las clases *QuantumRegister* y *QuantumCircuit* que serán necesarias para trabajar con circuitos y registros cuánticos. Después, en las líneas 4 y 5, creamos 4 qubits que lo almacenamos en *qr* y después creamos un circuito *qc* con esos qubits, donde a cada qubit le corresponde una puerta cuántica. Primero asignamos la puerta de Hadamard al qubit 0, después la puerta Z al qubit 1, después la puerta X al qubit 2 y por último, la puerta Y al qubit 3.

La visualización del circuito sería el siguiente.

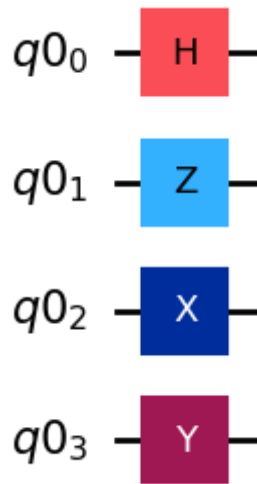


Figura 3 Puertas cuánticas para un qubit

1.3. Puertas cuánticas de múltiples qubits

Los sistemas de múltiples qubits pueden verse como una extensión de los sistemas de un solo qubit. Estos los utilizaremos para resolver problemas de escala considerable que no se pueden realizar con qubits individuales. Antes de comenzar a explicar acerca de múltiples qubits, vamos a hablar sobre otra propiedad muy importante que tienen los qubits que los diferencia de los bits clásicos.

Esta propiedad es el entrelazamiento [4], que ocurre cuando dos sistemas están tan conectados entre sí de tal manera que el conocimiento sobre uno se lo concede sobre el otro sin importar a qué distancia estén entre sí. Es decir, los procesadores pueden sacar conclusiones sobre una partícula midiendo otra. Pueden por ejemplo determinar que, si un bit gira hacia arriba, el otro siempre girará hacia abajo y viceversa. Esto permite que los ordenadores cuánticos resuelvan problemas complejos más rápido.

Para poder definir un estado cuántico de n qubits [1](pg.13,16), es necesario poder expresarlo como el producto tensorial de los estados de cada uno de los n qubits que lo componen. Si disponemos de dos bits clásicos, todos los posibles estados del sistema que tenemos de dos bits son 00, 01, 10 y 11, conocidos como el 0, 1, 2 y 3 en número binario. Sin embargo, en un entorno cuántico, un sistema de dos qubits puede estar en una superposición de los cuatro estados posibles.

El estado $|\psi\rangle_n$ de un sistema genérico de n qubits es una superposición de los 2^n estados de la base computacional $|0\rangle_n, |1\rangle_n, \dots, |2^n - 1\rangle_n$ con módulo 1 [3](pg..

$$|\psi\rangle_n = \sum_{i=0}^{2^n-1} \alpha_j |j\rangle_n$$

tal que

$$\sum_{i=0}^{2^n-1} |\alpha_j|^2 = 1$$

donde $|\alpha_j| \in \mathbb{C}$ son las amplitudes.

Tomando el ejemplo anterior, el sistema de dos qubits se puede escribir como

$$|\psi\rangle_2 = \alpha_0|00\rangle + \alpha_1|01\rangle + \alpha_2|10\rangle + \alpha_3|11\rangle$$

Aquí [1](pg.14), $|ab\rangle$ es una notación abreviada del producto tensorial $|a\rangle \otimes |b\rangle$, es decir, $|00\rangle$ significa $|0\rangle \otimes |0\rangle$, $|01\rangle$ implica $|0\rangle \otimes |1\rangle$ y así sucesivamente. A continuación, vamos a explicar qué son los productos tensoriales. Un producto tensorial es una asignación definida a partir de un par de espacios vectoriales a un espacio vectorial.

Es decir, vectorialmente, para dos vectores $|a\rangle = \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix}$ y $|b\rangle = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}$, el producto tensorial $|a\rangle \otimes |b\rangle$ se define como el vector

$$|a\rangle \otimes |b\rangle = \begin{pmatrix} a_1 \cdot b_1 \\ \vdots \\ a_1 \cdot b_m \\ \vdots \\ a_n \cdot b_1 \\ \vdots \\ a_n \cdot b_m \end{pmatrix}$$

Un dato importante a tener en cuenta es que el producto tensorial no es conmutativo, es decir, $|a\rangle \otimes |b\rangle$ no necesariamente es igual a $|b\rangle \otimes |a\rangle$. En cuanto a las dimensiones, si nos fijamos en $|a\rangle$, ésta tiene dimensión $n \times 1$, mientras que $|b\rangle$ tiene dimensión $m \times 1$. Por lo tanto, las dimensiones del producto tensorial $|a\rangle \otimes |b\rangle$ son $(n \cdot m) \times 1$.

Veamos un ejemplo. Si tenemos los estados $|0\rangle$ y $|1\rangle$, su producto tensorial viene definido de la siguiente manera:

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

obteniendo como resultado una matriz de dimensión 4×1 .

En una computadora clásica, representamos un entero $a \in \mathbb{Z}_{\geq 0}$ tal que $a < 2^n$ con el sistema numérico binario [2], [3](pg.6,7),:

$$a = \sum_{l=0}^{n-1} a_l \cdot 2^l$$

donde $a_l \in \{0,1\}$ son los dígitos binarios de a . En cambio, en una computadora cuántica,

también podemos representar un entero $a < 2^n$ con n qubits de la siguiente manera:

$$|a\rangle_n = |a_{n-1} \dots a_1 a_0\rangle = \bigotimes_{l=0}^{n-1} |a_l\rangle$$

Siguiendo esto, por ejemplo, el número 11 se puede representar con 4 qubits, ya que $11 < 2^4$, tal que:

$$|11\rangle_4 = |1011\rangle = |1\rangle \otimes |0\rangle \otimes |1\rangle \otimes |1\rangle$$

De esta forma, los enteros siempre están siempre representados por elementos de la base que se obtiene a partir de productos tensoriales de las bases computacionales de un solo qubit.

Los estados base de un sistema de dos qubits son los productos tensoriales de los estados base de un sistema de un solo qubit.

$$|0\rangle_2 = |00\rangle = |0\rangle \otimes |0\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$|1\rangle_2 = |01\rangle = |0\rangle \otimes |1\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

$$|2\rangle_2 = |10\rangle = |1\rangle \otimes |0\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

$$|3\rangle_2 = |11\rangle = |1\rangle \otimes |1\rangle = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

1.3.1. Puerta CNOT

Hablemos ahora de la puerta CNOT [1](pg.17,18), [6]. La puerta CNOT o Control-NOT afecta a dos qubits, un qubit control y un qubit objetivo. El qubit de control determina si se aplica una operación de negación al qubit objetivo. Si el qubit de control está en el estado $|0\rangle$, el estado del qubit objetivo permanece sin cambios, sin embargo, si el qubit de control está en el estado $|1\rangle$, el estado del qubit objetivo se invierte, es decir, se aplica una operación de negación (NOT) al qubit objetivo. La representación matricial de la puerta CNOT es la siguiente:

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

La acción de esta puerta en un estado de dos qubits de la forma $|\psi\rangle = \alpha_0|00\rangle + \alpha_1|01\rangle + \alpha_2|10\rangle + \alpha_3|11\rangle$ se puede expresar de la siguiente manera

$$CNOT|\psi\rangle = \alpha_0|00\rangle + \alpha_1|01\rangle + \alpha_2|11\rangle + \alpha_3|10\rangle$$

Observamos que, mientras las amplitudes de los estados $|00\rangle$ y $|01\rangle$ permanecen iguales, las amplitudes de $|10\rangle$ y $|11\rangle$ han sido intercambiados. Una manera de representar la acción de la compuerta CNOT en la notación de ket es especificar su acción en los estados de la base computacional de los qubits individuales.

$$CNOT|a\rangle|b\rangle \rightarrow |a\rangle|a \oplus b\rangle$$

La sintaxis para aplicar una puerta CNOT en un circuito cuántico *qc* en Qiskit se muestra a continuación.

```
1 # Importar las clases y módulos necesarios
2 from qiskit import QuantumRegister, QuantumCircuit
3 qr = QuantumRegister(2)
4 qc = QuantumCircuit(qr)
5 # Aplicar la puerta CNOT
6 qc.cx(qr[0], qr[1])
7 qc.draw(output='mpl')
```

Código 4 Implementación de la puerta cuántica CNOT

Como vemos en el código de arriba, al inicio implementamos las clases *QuantumRegister* y *QuantumCircuit*. En las líneas 3 y 4, esta vez, como nos encontramos con una puerta que actúa con dos qubits, vamos a colocar en el registro *qr* 2 qubits, que serán el *q0* y el *q1* que añadiremos al circuito *qc*. Finalmente, aplicamos la puerta CX en la que el *qr[0]* representa el qubit de control y *qr[1]* representa el qubit objetivo.

A continuación, se visualiza cómo quedaría este código.

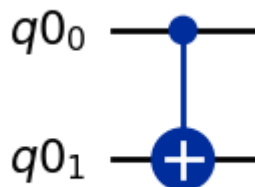


Figura 4 Puerta cuántica CNOT

1.3.2. Puerta SWAP

La siguiente puerta de la que hablaremos es la puerta SWAP [2]. Esta toma dos qubits como entrada y los intercambia. Su representación matricial es la siguiente:

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

En cuanto a su implementación en Qiskit, sería la siguiente

```
1 # Importar las clases y módulos necesarios
2 from qiskit import QuantumRegister, QuantumCircuit
3 qr = QuantumRegister(2)
4 qc = QuantumCircuit(qr)
5 # Aplicar la puerta SWAP
6 qc.swap(qr[0], qr[1])
7 qc.draw ()
```

Código 5 Implementación de la puerta cuántica SWAP

Es exactamente igual que para la puerta CNOT solo que la puerta SWAP se implementa con el comando `qc.swap` donde se realiza un intercambio entre el primer y segundo qubit.

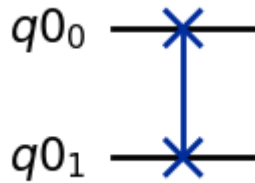


Figura 5 Puerta cuántica SWAP

A continuación, se muestra en una tabla un resumen de las puertas cuánticas sobre múltiples qubits

PUERTA CUÁNTICA	SÍMBOLO	MATRIZ
CNOT		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$

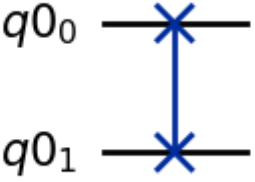
SWAP		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$
------	---	--

Tabla 2 Puertas cuánticas para más de un qubit

1.4. Medición de qubits

La medición [1](pg.9) es el proceso de observar el estado de un qubit y, a diferencia de las puertas cuánticas, la medición no es una operación unitaria y no es reversible. Al medir un qubit, obtenemos información. La peculiaridad de la medición es que al observar un qubit, este “colapsa”, es decir, cambia el estado del qubit a uno de los estados base.

Por ejemplo, si medimos un qubit de la forma $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, lo que vamos a obtener es el estado $|0\rangle$ con una probabilidad de $\frac{1}{2}$, y el estado $|1\rangle$ con una probabilidad también de $\frac{1}{2}$. Esta probabilidad se obtiene al elevar al cuadrado el valor absoluto de la amplitud de los estados. Supongamos ahora que tenemos acceso a n qubits del estado $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. Si queremos medir cada qubit en la base computacional, en cada instancia obtendremos $|0\rangle$ o $|1\rangle$. Nunca obtendremos $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ ya que éste no es uno de los estados de la base computacional. El número de veces que observamos $|0\rangle$ será casi igual al número de veces que observa $|1\rangle$, es decir, de las n mediciones, observaríamos $|0\rangle$ aproximadamente $\frac{n}{2}$ veces y observaríamos $|1\rangle$ aproximadamente $\frac{n}{2}$ veces.

Por tanto, se puede decir que, en la medición, el estado del qubit cambia al estado $|0\rangle$ o al estado $|1\rangle$.

Vamos a ver un ejemplo. Para ello, vamos a aplicar la puerta X a un qubit y a medirlo en Qiskit [1](pg.9,10). El código se muestra a continuación

```

1 #Importamos las clases y módulos necesarios
2 from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister,
  transpile
3 from qiskit_aer import Aer
4 #Creamos los registros cuánticos, registros clásicos y el circuito cuántico
5 qr = QuantumRegister(1)
6 cr = ClassicalRegister(1)
7 qc = QuantumCircuit(qr, cr)
8 #Invertir el qubit qr y medirlo
9 qc.x(qr)
10 qc.measure(qr, cr)
11 #Ejecutar el circuito en el simulador cuántico
12 backend = Aer.get_backend('qasm_simulator')
13 qc_transpiled = transpile(qc, backend)

```

```

14 # Ejecutar el circuito en el simulador
15 qjob = backend.run(qc_transpiled, shots=100)
16 measurement = qjob.result().get_counts()
17 print(measurement)
18 qc.draw(output='mpl')

```

Código 6 Implementación de la medición de un qubit

Vamos a explicar un poco el código que acabamos de implementar. Además de las importaciones de las clases habituales, hemos importado una nueva clase, la clase *ClassicalRegister*. En la línea 6, creamos una instancia llamada *cr* de la clase nueva creada para almacenar la salida de una medición de un qubit. La función *qc.measure(qr, cr)* es la que se encargará de medir el qubit *qr* y almacenar la salida en *cr*. Hemos proporcionado un nuevo parámetro llamado *shots*, que se utiliza para especificar el número de veces que deseamos que se repita el circuito. En este ejemplo, lo hemos establecido a 100, es decir, que el circuito se ejecutará un total de 100 veces.

Los resultados de la medición se obtienen del objeto de resultado utilizando *result().get_counts()*.

A la hora de ejecutar este código, nos muestra lo siguiente

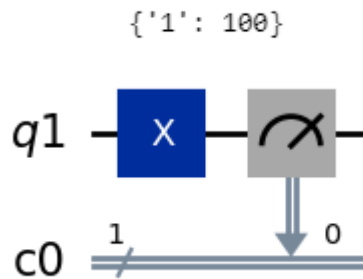


Figura 6 Medición de un qubit

A modo de resumen [1](pg.10,11), en nuestro circuito la puerta X se está aplicando en el qubit *qr*. Como el estado predeterminado de *qr* era $|0\rangle$, el estado después de que se aplica la puerta X pasará a ser $|1\rangle$. Por lo tanto, al medir el qubit *qr*, esperamos obtener $|1\rangle$ con una probabilidad de 1. Como hemos establecido el número de *shots* en 100, se espera que la salida sea el resultado de $|1\rangle$ 100 veces.

Sin embargo, si lo que queremos es saber si la salida es $|+\rangle$ o $|-\rangle$, tendríamos que medir el estado en la base de Hadamard, es decir, en la base $\{|+\rangle, |-\rangle\}$. Para ello, suponemos que tenemos el estado

$$|\phi\rangle = \frac{(\alpha + \beta)}{\sqrt{2}}|+\rangle + \frac{(\alpha - \beta)}{\sqrt{2}}|-\rangle$$

Si aplicamos H^t en $|\phi\rangle$ donde H^t es el conjugado transpuesto de H, obtenemos lo siguiente

$$H^t|\phi\rangle = \frac{(\alpha + \beta)}{\sqrt{2}}|0\rangle + \frac{(\alpha - \beta)}{\sqrt{2}}|1\rangle$$

Observamos que H^t mapea $|+\rangle$ a $|0\rangle$ y $|-\rangle$ a $|1\rangle$ pero manteniendo las amplitudes. Si ahora medimos $H^t|\phi\rangle$ en la base computacional, es equivalente a medir el estado $|\phi\rangle$ en la base de Hadamard. El siguiente código aplica la puerta X a un qubit y lo mide en la base de Hadamard [1](pg.11,12).

```
1 #Importamos las clases y módulos necesarios
2 from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister,
  transpile
3 from qiskit_aer import Aer
4 #Creamos los registros cuánticos, registros clásicos y el circuito cuántico
5 qr = QuantumRegister(1)
6 cr = ClassicalRegister(1)
7 qc = QuantumCircuit(qr,cr)
8 qc.x(qr)
9 # Mapear la base de Hadamard a la base computacional aplicando  $H^\dagger = H$ 
  en qr
10 qc.h(qr)
11 qc.measure(qr,cr)
12 #Ejecutar el circuito en el simulador cuántico
13 backend = Aer.get_backend('qasm_simulator')
14 qc_transpiled = transpile(qc, backend)
15 # Executing the circuit in the simulator
16 qjob = backend.run(qc_transpiled, shots=100)
17 counts = qjob.result().get_counts()
18 print(counts)
19 qc.draw(output='mpl')
```

Código 7 Implementación medición de qubit aplicando Hadamard

A la hora de ejecutar este código, nos muestra lo siguiente

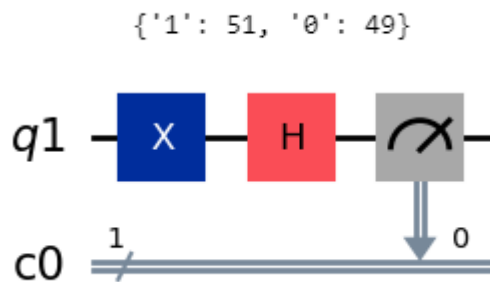


Figura 7 Medición de qubit aplicando Hadamard

Para poder convertir la base de medición de la base computacional a la base de Hadamard, aplicaremos la puerta H en qr , como se ve en la línea 10 del código.

A partir de la salida, encontramos que $|0\rangle$ se obtiene como resultado de la medición en 51 veces, mientras que $|1\rangle$ se obtiene en 49 veces de un total de 100 casos. Por lo tanto,

la probabilidad de obtener $|0\rangle$ es de aproximadamente el 51% y la probabilidad de obtener $|1\rangle$ es de aproximadamente el 49%. Teóricamente, las probabilidades deberían ser iguales, sin embargo, debido a la aleatoriedad en la medición, observamos la desviación de $\frac{1}{2}$. Recordemos que aquí el resultado $|0\rangle$ corresponde al estado $|+\rangle$ y el resultado $|1\rangle$ corresponde al estado $|-\rangle$.

Ahora bien, si nos encontramos en un sistema de n qubits donde el estado base es $|0\rangle$, a la hora de medir tendríamos lo siguiente [3](pg.11)

$$H^{\otimes n}|\psi 0\rangle_n = H^{\otimes n}|0\rangle_n = \frac{1}{\sqrt{2^n}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}^{\otimes n} \begin{pmatrix} 1 \\ 0 \end{pmatrix}^{\otimes n} = \frac{1}{\sqrt{2^n}} \begin{pmatrix} 1 \\ 1 \end{pmatrix}^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} |j\rangle_n$$

Lo que obtenemos aquí es una superposición de todos los estados de base del sistema con una probabilidad idéntica. En otras palabras, si medimos nuestro registro de n qubits, obtendremos un cierto entero j en el conjunto $\{0, \dots, 2^n - 1\}$ con una probabilidad de $\frac{1}{2^n}$.

1.5. Circuitos cuánticos

Los circuitos cuánticos son un modelo utilizado en el ámbito de la computación cuántica para diseñar y describir algoritmos cuánticos. Funcionan de manera similar a los circuitos lógicos de las computadoras clásicas, pero con algunas diferencias debido a la naturaleza de la mecánica cuántica. Estos circuitos están compuestos por una serie de compuertas cuánticas aplicadas a qubits de entrada para realizar operaciones específicas, y, gracias a los principios cuánticos como la superposición y la interferencia cuántica, hacen que sea posible realizar ciertos problemas de manera más eficiente.

Los circuitos cuánticos se representan mediante diagramas de circuitos. Estos constan de líneas horizontales, que representan los qubits, y bloques que representan las puertas cuánticas, que realizan operaciones a los qubits aplicados.

Veamos un ejemplo.

```
1 # Importar las clases y módulos necesarios
2 from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister,
3 transpile
4 from qiskit_aer import Aer
5 # Crear un circuito cuántico con 4 qubits y 4 bits clásicos
6 qr = QuantumRegister(4)
7 cr = ClassicalRegister(4)
8 qc = QuantumCircuit(qr, cr)
9 # Aplicar una puerta Hadamard al primer y tercer qubit
10 qc.h(0)
11 qc.h(2)
12 # Aplicar una puerta X (Pauli-X) al segundo y cuarto qubit
13 qc.x(1)
14 qc.x(3)
15 # Aplicar una puerta CNOT entre el primer qubit (control) y el segundo qubit
16 (objetivo)
```

```

17 qc.cx(0, 1)
18 # Aplicar una puerta CNOT entre el tercer qubit (control) y el cuarto qubit
19 (objetivo)
20 qc.cx(2, 3)
21 # Medir todos los qubits y almacenar el resultado en los bits clásicos
22 correspondientes
23 for i in range(4):
24     qc.measure(qr[i], cr[3 - i])
25 # Transpila el circuito para el backend deseado
26 backend = Aer.get_backend('qasm_simulator')
27 qc_transpiled = transpile(qc, backend)
28 # Ejecuta el circuito en el simulador
29 result = backend.run(qc_transpiled).result()
30 # Obtiene los resultados
31 counts = result.get_counts()
32 # Imprime los resultados
33 print(counts)
34 # Dibujar el circuito
35 qc.draw(output='mpl')

```

Código 8 Ejemplo de un pequeño circuito cuántico

Lo que hacemos en este circuito de 4 qubits y 4 bits consiste en aplicar unas puertas Hadamard a los qubits 0 y 2, mientras que a los qubits 1 y 3 les vamos a aplicar unas puertas X. A continuación, se les va a añadir unas puertas CNOT a los qubits a los que les aplicamos la puerta de Hadamard, que cambia del estado del bit objetivo de $|0\rangle$ a $|1\rangle$ si el bit de control está a $|1\rangle$, pero se mantiene a $|0\rangle$ si el bit de control es $|0\rangle$. En la siguiente figura podemos ver las probabilidades de obtener los estados $|1001\rangle$, $|1010\rangle$, $|0110\rangle$ y $|0101\rangle$ en un total de 1024 ejecuciones.

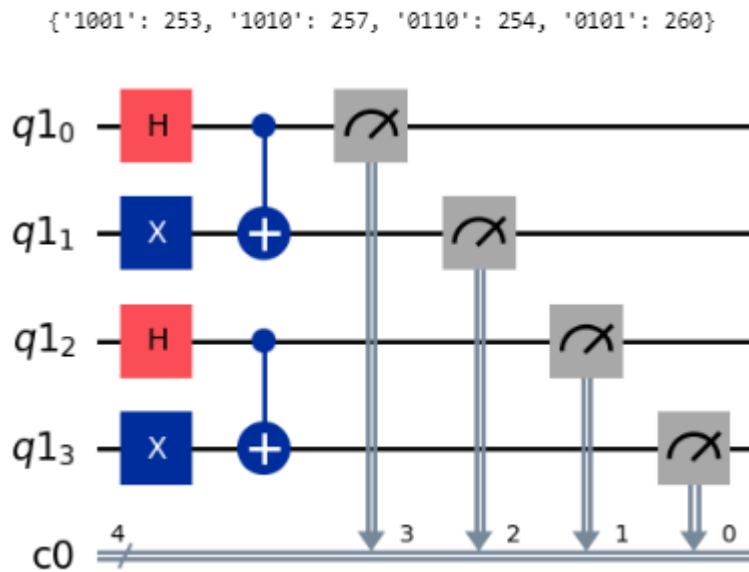


Figura 8 Circuito cuántico

2. Algoritmo de Deutsch-Jozsa

En este capítulo estudiaremos uno de los algoritmos más importantes de la computación cuántica, el algoritmo de Deutsch-Jozsa, propuesto por David Deutsch y Richard Jozsa en 1992, y lo relacionaremos con el espectro de Walsh de una función booleana. Este espectro no es más que una transformación matemática que descompone una función booleana en una suma de funciones más simples, facilitando así su implementación y comprensión. Este algoritmo es conocido por ser uno de los primeros en mostrar una ventaja cuántica significativa sobre los algoritmos clásicos, además de resolver el problema de si una función booleana es constante o balanceada. Antes de comenzar con el desarrollo del algoritmo, introduciremos los conceptos básicos de las funciones booleanas.

2.1. Funciones booleanas

Las funciones booleanas [1] (pg.39-40) desempeñan un papel muy importante en muchos campos como pueden ser la criptografía simétrica, la teoría de códigos y en el diseño y análisis de algoritmos criptográficos.

Vamos a definir una función booleana de entrada de n bits y salida de m bits la siguiente asignación $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$.

Sabemos que una función booleana puede representarse en diversas formas. Una forma es mediante la tabla de verdad, donde ésta contiene todas las combinaciones posibles de los valores de entrada y sus salidas correspondientes. Otra forma es mediante la Forma Normal Algebraica. Cualquier función booleana de n variables puede considerarse un polinomio de n variables sobre el campo finito. Este polinomio puede escribirse de la siguiente manera:

$$a_0 \oplus a_i x_i \oplus a_{ij} x_i x_j \oplus \dots \oplus a_{12\dots n} x_1 x_2 \dots x_n$$

donde $a_i, a_{ij}, \dots, a_{12\dots n}$ son los coeficientes cuyo valor son 0 o 1. El símbolo \oplus significa la adición módulo 2, también conocida como suma binaria o suma XOR, que devuelve como resultado un 0 si ambos bits son iguales o 1 si son diferentes. El número de variables presentes en el término de producto de mayor orden con coeficientes no nulos se le denomina grado algebraico de la función booleana.

Vamos a definir primero unos conceptos base que necesitaremos saber para entender este algoritmo. Estos son una función balanceada y una función constante [1] (pg.42). Una función booleana $f(x)$ de n variables se dice que está balanceada si f contiene un número igual de ceros y unos. De manera similar, una función booleana $f(x)$ se dice que es constante si la salida de la función es independiente de los bits de entrada y siempre devuelve el mismo valor. En otras palabras, una función booleana constante es una función tal que $f(x) = 1$ para todos los x en $\{0, 1\}^n$ o $f(x) = 0$ para todos los x en $\{0, 1\}^n$.

2.2. Transformada de Walsh

La transformada de Walsh [1] (pg.43) es una de las herramientas más importantes para el análisis de las funciones booleanas. Lo que hace es convertir una función booleana en una suma de funciones más simples.

Para una función booleana $f(x)$ de n bits, la Transformada de Walsh de $f(x)$ en cualquier punto $a \in \{0,1\}^n$ se define como:

$$W_f(a) = \sum_{x \in \{0,1\}^n} (-1)^{f(x) \oplus a \cdot x}$$

donde $a \cdot x$ denota el producto $a_1x_1 \oplus a_2x_2 \oplus \dots \oplus a_nx_n$.

El valor de la Transformada de Walsh en un punto a se llama el coeficiente de Walsh de $f(x)$ en el punto a . El espectro de Walsh de la función booleana $f(x)$ es la lista de todos los 2^n coeficientes de Walsh dados por:

$$W_f = [W_f(0), W_f(1), \dots, W_f(2^n - 1)]$$

donde $W_f(i)$ es el coeficiente de Walsh en un punto cuya representación decimal es dada por i .

Si observamos cuidadosamente, el coeficiente de Walsh en el punto $|0^{\otimes n}\rangle$ se puede simplificar como

$$W_f(0^{\otimes n}) = \sum_{x \in \{0,1\}^n} (-1)^{f(x)}$$

Esta expresión calcula la diferencia entre el número de valores de entrada para los cuales $f(x) = 0$ y aquellos para los cuales $f(x) = 1$. Es decir,

$$W_f(0^{\otimes n}) = |\{x : f(x) = 0\}| - |\{x : f(x) = 1\}| = 2^n - 2wt(f)$$

donde f es la representación de la tabla de verdad de $f(x)$. Entonces, para una función balanceada, tenemos $W_f(0^{\otimes n}) = 0$.

2.3. Implementación de las funciones booleanas

Debemos tener en cuenta un dato importante; las funciones booleanas generalmente no son biyectivas. Sin embargo, dado que cualquier compuerta cuántica debe ser reversible, debemos tener algún mecanismo para implementar una función booleana en el dominio cuántico. Ese mecanismo es la función oráculo, que lo vamos a explicar a continuación [1](pg.45).

Sea $F : \{0,1\}^n \rightarrow \{0,1\}^m$ una función booleana de entrada de n bits y salida de m bits. Necesitaremos implementar F en un circuito cuántico, pero F no es reversible en

general. Si vemos un ejemplo sencillo, por ejemplo, la función AND $f(x_1x_2) = x_1 \cdot x_2$ no es reversible ya que no podemos obtener la entrada de la función si solo se nos da su valor de salida correspondiente. Como todas las compuertas en un circuito cuántico deben ser reversibles, para solucionar este problema de las funciones booleanas, vamos a ver la función oráculo U_F .

Dada una función $F : \{0,1\}^n \rightarrow \{0,1\}^m$, U_F no es más que una función booleana de $(n+m)$ entradas y $(n+m)$ salidas que se define de la siguiente manera

$$U_F(x_1, x_2, \dots, x_n, b) = (x_1, x_2, \dots, x_n, b \oplus F(x))$$

donde $b \in \{0,1\}^m$ y $x = (x_1, x_2, \dots, x_n)$.

Para cualquier entrada $x \in \{0,1\}^n$, obtener el valor de $F(x)$ utilizando el oráculo U_F es bastante sencillo. Vamos a verlo en un ejemplo, y para ello, vamos a suponer $b = 0$. El último bit de $U_F(x_1, \dots, x_n, 0) = (x_1, \dots, x_n, F(x))$ es el valor requerido.

Dada una salida y de U_F , siempre es posible obtener la entrada de U_F ya que $U_F(U_F(x_1, \dots, x_n, b)) = (x_1, \dots, x_n, b)$; es decir, lo que hace el oráculo U_F consiste en retener los bits de entrada (x_1, \dots, x_n) en sus bits de salida, para luego así poder recuperar la entrada a partir de la salida de $U_F(x_1, \dots, x_n, b)$. A diferencia de F , la función U_F es siempre reversible; además, cualquier función reversible puede implementarse en un circuito cuántico utilizando el conjunto de puertas universales.

Veamos un ejemplo. Sea la función F la función AND que hemos propuesto anteriormente, $f(x_1x_2) = x_1 \cdot x_2$. El circuito cuántico correspondiente de U_F se muestra en la figura 9.

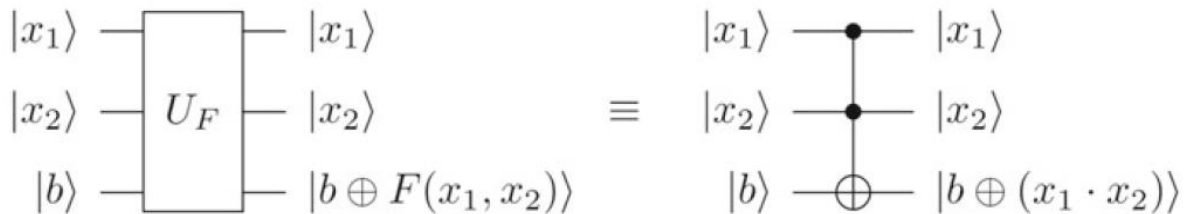


Figura 9 Circuito cuántico de la función AND

Podemos codificar el circuito oráculo en Qiskit de la siguiente manera [1](pg.46,47):

```
1 # Importar las clases y módulos necesarios
2 from qiskit import *
3 # Definir la función que devuelve el circuito cuántico del oráculo U_F para la
  función AND
4 def oracle():
5     circ = QuantumCircuit(3)
6     circ.ccx(0,1,2)
7     return circ
8 # Definir el circuito oráculo
9 circuito_oraculo = oracle()
11 # Dibujar el circuito
```



```
11 circuito_oraculo.draw(output='mpl')
```

Código 9 Implementación de un oráculo

La visualización del circuito quedaría de la siguiente manera

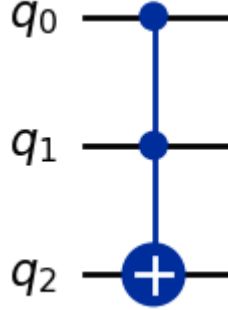


Figura 10 Oráculo

A modo de repaso, antes de ver los algoritmos de Deutsch y Deutsch-Jozsa, vamos a ver cómo se comportan la puerta de Hadamard y el oráculo U_F sobre los qubits [3](pg.12-16).

Recordemos que la puerta de Hadamard, lo que hace es poner en superposición los qubits, de tal forma que los estados pasan de $|0\rangle$ y $|1\rangle$ a los estados $|+\rangle$ y $|-\rangle$ respectivamente, como se muestra a continuación

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \equiv |+\rangle$$

$$H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \equiv |-\rangle$$

Para el algoritmo de Deutsch-Jozsa, a la hora de aplicar las puertas de Hadamard para un estado base de n qubits donde j es un número entre $\{0, \dots, 2^n - 1\}$, utilizaremos la siguiente expresión

$$H^{\otimes n}|j\rangle_n = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} (-1)^{j \cdot k} |k\rangle_n$$

En cuanto al oráculo, tenemos que tener en cuenta varias expresiones. La más genérica de cómo funciona el oráculo es la siguiente

$$U_F: |j\rangle_n \otimes |k\rangle_m \rightarrow |j\rangle_n \otimes |k \oplus f(j)\rangle_m$$

donde tenemos dos qubits de entrada, de los cuales al segundo se le aplica el operador \oplus que consiste en realizar la operación XOR.

En la siguiente expresión, lo que vamos a ver es cómo se aplicaría el oráculo teniendo en cuenta el primer bit de entrada.

$$U_F(|j\rangle_n \otimes |- \rangle) = (-1)^{f(j)} |j\rangle_n \otimes |- \rangle$$

De esta manera, si tenemos que aplicar el oráculo después de aplicar la puerta de Hadamard, tenemos lo que sigue

$$U_F(|+\rangle \otimes |- \rangle) = \frac{(-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle}{\sqrt{2}} \otimes |- \rangle$$

2.4. Descripción del algoritmo de Deutsch

El algoritmo de Deutsch [1](pg. 47,48), [7], es uno de los algoritmos cuánticos más simples y uno de los primeros. Supongamos que se nos proporciona una función booleana de 1 bit $f(x)$ como una caja negra. Una caja negra, (también llamada oráculo) es un modelo donde no se sabe el funcionamiento interno, solo que proporcionándole una entrada x , ésta produce la salida $f(x)$. Nuestro objetivo, como explicamos al principio, consiste en determinar si la función que se nos propone es constante o balanceada. A este problema se denomina el problema de Deutsch.

Para resolver el problema, necesitaríamos realizar dos llamadas a la caja negra de $f(x)$ para obtener tanto $f(0)$ como $f(1)$ antes de saber si la función es constante o balanceada. Sin embargo, al encontrarnos en un entorno cuántico, el mismo problema se puede resolver utilizando únicamente una sola consulta al oráculo de $f(x)$ mediante el uso del algoritmo de Deutsch. El oráculo de $f(x)$ es la función $U_F(x)$ que se comentó en el apartado anterior.

Este algoritmo se implementa como en el circuito de la figura 11.

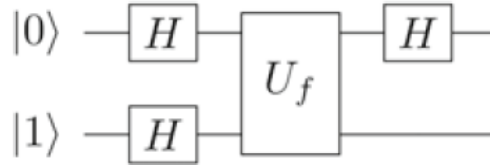


Figura 11 Circuito cuántico del algoritmo de Deutsch

Vamos a calcular paso a paso cómo va evolucionando el estado del sistema. Para ello, aplicaremos las puertas de Hadamard para los estados iniciales $|0\rangle$ y $|1\rangle$, seguido aplicaremos el oráculo U_f , y finalmente, la puerta de Hadamard para el estado $|0\rangle$.

$$|0\rangle \otimes |1\rangle \xrightarrow{H \otimes H} \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle) \otimes |- \rangle$$

$$\xrightarrow{U_f} \frac{1}{\sqrt{2}} ((-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle) \otimes |- \rangle$$

$$\xrightarrow{H \otimes I} \frac{1}{2} [((-1)^{f(0)} + (-1)^{f(1)})|0\rangle + ((-1)^{f(0)} - (-1)^{f(1)})|1\rangle] \otimes |- \rangle$$

Si queremos medir el estado final del sistema, vemos que la probabilidad de obtener el estado $|0\rangle$ es

$$\Pr(|0\rangle) = \frac{1}{4} [(-1)^{f(0)} + (-1)^{f(1)}]^2$$

y que de obtener $|1\rangle$ es

$$\Pr(|1\rangle) = \frac{1}{4} [(-1)^{f(0)} - (-1)^{f(1)}]^2 = 1 - \Pr(|0\rangle)$$

Teniendo esto en cuenta, sacamos las siguientes conclusiones:

Si la función $f(x)$ es balanceada, entonces la probabilidad de obtener $|0\rangle$ se desvanece a 0, y por tanto, obtenemos solo $|1\rangle$ al realizar la medición.

Si la función $f(x)$ es constante, entonces la probabilidad de obtener $|0\rangle$ es igual a 1 y así obtenemos solo $|0\rangle$ al realizar la medición.

Podemos observar que con una sola llamada al oráculo U_F , la función es constante si el resultado de la medición es $|0\rangle$ o está balanceada si el resultado de la medición es $|1\rangle$.

Aunque el algoritmo de Deutsch tiene un uso práctico limitado, sirve para ilustrar el paralelismo cuántico y la interferencia cuántica. Este algoritmo, por tanto, aprovecha esta condición para obtener la solución utilizando solo una consulta a la función.

2.5. Descripción del algoritmo de Deutsch-Jozsa

El algoritmo de Deutsch-Jozsa [1](pg.48-50), propuesto por David Deutsch y Richard Jozsa en 1992, es uno de los primeros algoritmos cuánticos que demostró una clara ventaja cuántica sobre sus contrapartes clásicas, ya que aprovecha el paralelismo de los estados de superposición. El paralelismo cuántico es una característica clave de la computación cuántica que permite acelerar exponencialmente ciertos tipos de cálculos.

El algoritmo de Deutsch-Jozsa es una extensión del algoritmo de Deutsch para resolver funciones booleanas de n bits. El objetivo de este algoritmo consiste en resolver el siguiente problema: dada una función booleana de n bits $f(x)$ como un oráculo, encontrar si la función $f(x)$ está balanceada o es constante, dado el compromiso de que $f(x)$ es constante o balanceada.

Una función booleana de n bits $f(x)$ tiene 2^n valores de salida, es decir, uno para cada $x \in \{0,1\}^n$. Sabemos desde el principio que $f(x)$ es constante o balanceada. Una función balanceada tiene una tabla de verdad con exactamente $\frac{2^n}{2}$ ceros y $\frac{2^n}{2}$ unos. Esto significa que necesitaremos $\frac{2^n}{2} + 1$ consultas a la función $f(x)$ para determinar si todos los $\frac{2^n}{2} + 1$ valores de $f(x)$ son iguales. Si esto es así, diremos entonces que la función es constante, y en caso contrario, diremos que la función está balanceada.

A continuación, en la figura 12, se muestra el circuito cuántico del algoritmo de Deutsch-Jozsa.

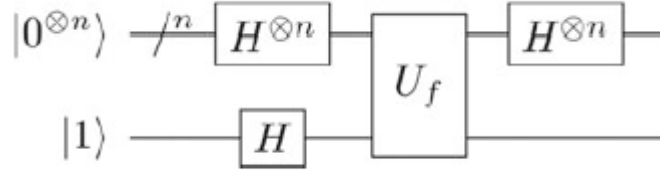


Figura 12 Circuito cuántico del algoritmo de Deutsch-Jozsa

Vamos a observar la evolución del estado del sistema, que es exactamente igual que el algoritmo de Deutsch, solo que esta vez tenemos n qubits

$$|0^{\otimes n}\rangle \otimes |1\rangle \xrightarrow{H^{\otimes n} \otimes H} \frac{1}{\sqrt{2^n}} \sum_x |x\rangle \otimes |-\rangle$$

$$\xrightarrow{U_f} \frac{1}{\sqrt{2^n}} \sum_x (-1)^{f(x)} |x\rangle \otimes |-\rangle$$

$$\xrightarrow{H^{\otimes n} \otimes I} \frac{1}{\sqrt{2^n}} \sum_y \left[\sum_x (-1)^{f(x) \oplus x \cdot y} \right] |y\rangle \otimes |-\rangle$$

Donde $x \cdot y$ denota el producto bit a bit $x_1 y_1 + x_2 y_2 + \dots + x_n y_n$.

Vamos a calcular cuál es la probabilidad de obtener el estado $|0^{\otimes n}\rangle$.

$$\Pr(|0^{\otimes n}\rangle) = \frac{1}{2^{2n}} \left[\sum_{x \in \{0,1\}^n} (-1)^{f(x)} \right]^2$$

Como se nos indica si la función $f(x)$ es constante o balanceada, vamos a asumir primero que la función es constante, es decir, vamos a suponer que $f(x) = 1$ para todos los x . Por lo tanto, tenemos:

$$\Pr(|0^{\otimes n}\rangle) = \frac{1}{2^{2n}} \left[\sum_{x \in \{0,1\}^n} (-1)^{f(x)} \right]^2 = \frac{1}{2^{2n}} \left[\sum_{x \in \{0,1\}^n} -1 \right]^2 = \frac{1}{2^{2n}} \cdot 2^{2n} = 1$$

Así, una medición siempre daría el estado $|0^{\otimes n}\rangle$ si la función $f(x)$ es constante.

Vamos a asumir ahora que la función $f(x)$ es balanceada. Entonces, tenemos exactamente $\frac{2^n}{2}$ puntos $x \in \{0,1\}^n$ tales que $f(x) = 0$ y exactamente el mismo número de puntos tales que $f(x) = 1$. Por lo tanto, la probabilidad de obtener el estado $|0^{\otimes n}\rangle$ se vuelve 0. Esto es porque habría exactamente $\frac{2^n}{2}$ números de 1 y $\frac{2^n}{2}$ números de -1 en la suma de la ecuación. Por lo tanto, si la función $f(x)$ es balanceada, es imposible obtener el estado $|0^{\otimes n}\rangle$ como resultado de la medición.

Concluimos, por tanto, que $|0^n\rangle$ se observa siempre si $f(x)$ es constante y nunca se observa si $f(x)$ es balanceada.

Si observamos bien, la función dada $f(x)$ puede caracterizarse como balanceada o como constante con solo una consulta al oráculo U_f . Con esto, obtenemos un algoritmo cuántico que es exponencialmente más rápido que el mejor algoritmo clásico conocido para distinguir entre una función booleana balanceada y una constante.

Un dato importante a observar sobre la salida del algoritmo es que la probabilidad de obtener un estado $|a\rangle$ es

$$\Pr(|a\rangle) = \frac{1}{2^{2n}} \left[\sum_x (-1)^{f(x) \oplus x \cdot a} \right]^2 = \left(\frac{W_f(a)}{2^n} \right)^2$$

que es el cuadrado del coeficiente de Walsh normalizado de la función $f(x)$ en el punto a .

2.6. Descripción circuito del algoritmo con Qiskit

A continuación, vamos a implementar el algoritmo de Deutsch-Jozsa en Qiskit [1](pg.50). Sea la función dada $f(x) = x_2 \oplus x_3 \oplus x_4$:

```

1 # Importamos los módulos necesarios
2 from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister,
  transpile
3 from qiskit_aer import Aer
4 # Construimos un oráculo para la función f(x) = x2 xor x3 xor x4
5 orcl = QuantumCircuit(5)
6 orcl.cx(1, 4)
7 orcl.cx(2, 4)
8 orcl.cx(3, 4)
9 oracle = orcl.to_instruction()
10 # Creamos los registros cuánticos y clásicos
11 q = QuantumRegister(5)
12 c = ClassicalRegister(4)
13 qc = QuantumCircuit(q, c)
14 # Implementamos el algoritmo de Deutsch-Jozsa
15 qc.x(4)
16 qc.h(range(5))
17 qc.append(oracle, q)
18 qc.h(range(5))
19 for i in range(4):
20     qc.measure(q[i], c[3 - i])
21 # Ejecutamos el circuito para el backend deseado
22 backend = Aer.get_backend('qasm_simulator')
23 qc_transpiled = transpile(qc, backend)
24 # Ejecutamos el circuito en el simulador
25 result = backend.run(qc_transpiled).result()
26 # Obtenemos los resultados

```

```

27 counts = result.get_counts()
28 # Imprime los resultados
29 print(counts)
30 # Dibujamos el circuito
31 qc.draw(output='mpl')

```

Código 10 Implementación del algoritmo de Deutsch-Jozsa

Vamos a analizar el código que implementamos anteriormente. En las líneas 2 y 3, importamos las clases *QuantumCircuit*, *ClassicalRegister*, *QuantumRegister*, *transpile* y *aer*, que sirven para construir circuitos cuánticos, definir los registros cuánticos y clásicos, ejecutar circuitos para un backend específico y simularlos, respectivamente. De las líneas 5 a 9 definimos un circuito cuántico denominado *orcl* con 5 qubits. Seguido, agregamos las compuertas CNOT al circuito, haciendo que el qubit objetivo sea el qubit 4 y se controlan por los qubits 1, 2 y 3 respectivamente. Con la función *orcl.to_instruction()* convertimos el circuito en una instrucción, lo que permite agregarlo como un bloque en otros circuitos. Seguido, definimos un registro cuántico llamado *q* con 5 qubits, y un registro clásico llamado *c* con 4 bits, para luego crear un nuevo circuito cuántico llamado *qc* que utiliza tanto el registro cuántico *q* y el registro clásico *c*. De las líneas 15 a 20 lo que hacemos es aplicar una compuerta X al qubit 4 del circuito *qc*, para establecer el estado inicial de ese qubit en $|1\rangle$. Aplicamos compuertas Hadamard a todos los qubits en el registro cuántico *q*. Esto coloca los qubits en una superposición uniforme de todos los posibles estados. Al usar *qc.append(oracle, q)*, agregamos el circuito del oráculo al registro de qubits *q* al circuito *qc*, es decir, implementamos la función $f(x) = x_2 \oplus x_3 \oplus x_4$. Aplicamos las compuertas Hadamard a todos los qubits nuevamente para deshacer la superposición antes de la medición. Aplicamos un bucle for para la medición a los primeros 4 qubits del registro cuántico *q* y los asigna a los bits clásicos correspondientes en el registro clásico *c*. Finalmente, ejecutamos y, en la salida, podemos ver que el resultado de la medición es $|0111\rangle$, que no es el estado $|0000\rangle$ por lo que la función $f(x) = x_2 \oplus x_3 \oplus x_4$ es balanceada.

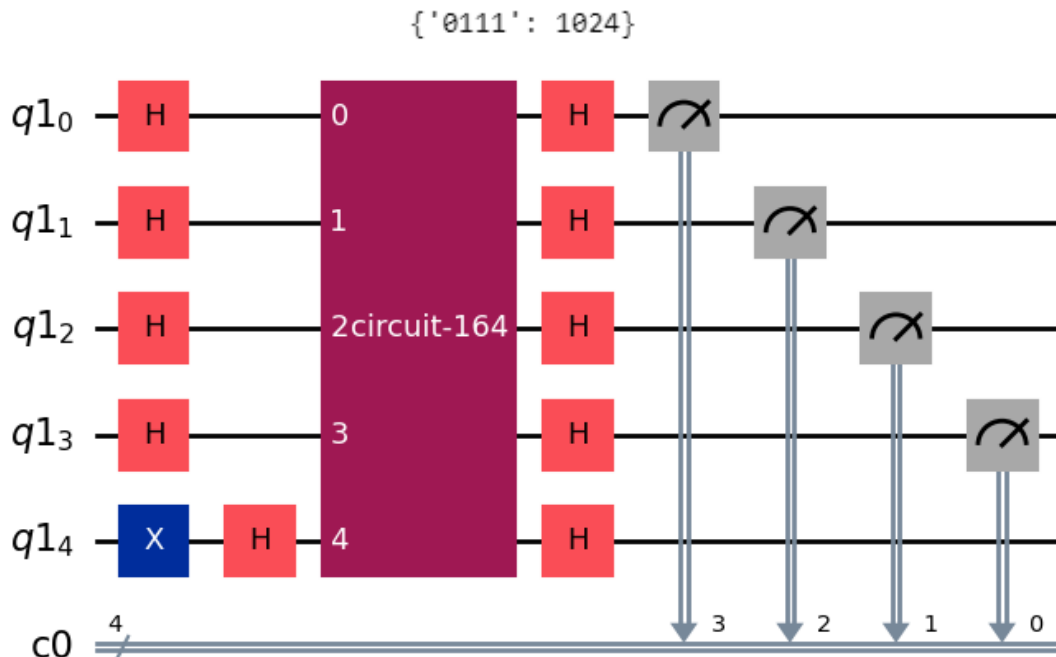


Figura 13 Algoritmo de Deutsch-Jozsa

3. Algoritmo de Grover

El algoritmo de Grover [3](pg.29-31) es uno de los algoritmos cuánticos más populares que existen. Fue propuesto por Lov Grover en 1996 y representa un hito significativo en la computación cuántica al ofrecer una solución eficiente que se utiliza para la búsqueda no estructurada en bases de datos no ordenadas.

Con esto, lo que queremos buscar es un elemento entre N elementos. Cuando los elementos no están ordenados y no hay una estructura de datos adicional, como puede ser el caso de un árbol de búsqueda binario, el mejor algoritmo clásico conocido debe ser el que consulte todos los N elementos en el peor de los casos. Sorprendentemente, el algoritmo de Grover es capaz de encontrar el elemento en solo $O(\sqrt{N})$ consultas.

3.1. Descripción del algoritmo

Para este algoritmo, necesitamos un ordenador cuántico con $n + 1$ qubits, donde los primeros n qubits se inicializarán a $|0\rangle$ y el último a $|1\rangle$, de tal forma que tenemos la siguiente expresión

$$|\psi_0\rangle_{n,1} = |0\rangle_n \otimes |1\rangle$$

En este algoritmo, el primer paso consiste en aplicar la puerta de Hadamard a todos los qubits de nuestro sistema.

$$|\psi_1\rangle_{n,1} = (H^{\otimes n} |0\rangle_n) \otimes (H|1\rangle) = (H|0\rangle)^{\otimes n} \otimes (H|1\rangle) = \left(\frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} |j\rangle_n \right) \otimes |-\rangle$$

El resultado es una combinación de todos los estados base de nuestros primeros n qubits, y el estado $|-\rangle$ en el qubit restante. A partir de ahora, a modo de simplicidad, se va a introducir un estado cuántico que nos será útil y lo utilizaremos repetidas veces a lo largo del algoritmo.

$$|\gamma\rangle_n = \frac{1}{\sqrt{2^n}} \sum_{j=0}^{2^n-1} |j\rangle_n$$

Esta definición nos ayuda a expresar nuestro estado cuántico como $|\psi_1\rangle_{n,1} = |\gamma\rangle_n \otimes |-\rangle$

El siguiente paso del algoritmo consiste en hacer uso de un oráculo U_f , que vimos en el capítulo anterior. Este oráculo hace uso de una función capaz de reconocer el elemento de la base de datos que estamos buscando. Esta función f se puede definir de la siguiente manera

$$f(j) = \begin{cases} 1 & \text{if } j = j_0 \\ 0 & \text{en otro caso} \end{cases}$$

Como nos encontramos en un sistema de $n+1$ qubits, el oráculo tiene el siguiente efecto

$$U_f : |j\rangle_n \otimes |k\rangle \mapsto |j\rangle_n \otimes |k \oplus f(j)\rangle$$

Vamos a introducir otro estado para simplificar la comprensión del algoritmo. Éste nos ayudará a notar la separación del elemento buscado j_0 del resto de los estados base cuánticos.

$$|\rho\rangle_n = \frac{1}{\sqrt{2^n - 1}} \sum_{\substack{j=0 \\ j \neq j_0}}^{2^n - 1} |j\rangle_n$$

Este estado tiene relación con el estado que creamos anteriormente, con $|\gamma\rangle_n$.

$$|\gamma\rangle_n = \frac{\sqrt{2^n - 1}}{\sqrt{2^n}} |\rho\rangle_n + \frac{1}{\sqrt{2^n}} |j_0\rangle_n$$

Hay que tener en cuenta para el resto del algoritmo, que $|\rho\rangle$ depende del valor de j_0 .

Después de haber aplicado nuestro oráculo U_f a nuestro sistema, ser verá de la siguiente manera

$$\begin{aligned} |\psi_2\rangle_{n,1} &= U_f (|\gamma\rangle_n \otimes |-\rangle) = U_f \left(\left(\frac{\sqrt{2^n - 1}}{\sqrt{2^n}} |\rho\rangle_n + \frac{1}{\sqrt{2^n}} |j_0\rangle_n \right) \otimes |-\rangle \right) \\ &= \left(\frac{\sqrt{2^n - 1}}{\sqrt{2^n}} |\rho\rangle_n - \frac{1}{\sqrt{2^n}} |j_0\rangle_n \right) \otimes |-\rangle = \left(|\gamma\rangle_n - \frac{2}{\sqrt{2^n}} |j_0\rangle_n \right) \otimes |-\rangle \end{aligned}$$

Recordemos que el oráculo lo que hace es invertir el signo de la amplitud que corresponde al estado base que codifica el elemento buscado en los primeros n qubits mientras que el resto de las amplitudes las mantiene intactas.

Para el último paso, vamos a ver la iteración de Grover [1](pg.60-, que implementa el siguiente operador

$$G = H^{\otimes n} U_{\bar{0}} H^{\otimes n} U_f$$

donde $H^{\otimes n}$ es el operador Hadamard en n qubits, U_f es el oráculo que marca el elemento de interés y $U_{\bar{0}}$ es el operador que invierte la fase de todos los estados base excepto el estado $|0\rangle^{\otimes n}$. Por tanto, la acción de $U_{\bar{0}}$ en los estados base se puede expresar como:

$$U_{\bar{0}}|x\rangle = \begin{cases} -|x\rangle & \text{if } x \neq 0^n \\ |x\rangle & \text{if } x = 0^n \end{cases} = 2|0^{\otimes n}\rangle\langle 0^{\otimes n}| - I$$

Podemos expandir $U_{\bar{0}}$ como un producto de dos operadores. En esta nueva expresión para G , usaremos $|S\rangle$ para denotar la superposición igual de todos los N estados correspondientes a los N elementos.

$$G = H^{\otimes n}(2|0^n\rangle\langle 0^n| - I)H^{\otimes n}U_f = (2|S\rangle\langle S| - I)U_f$$

El funcionamiento de U_f consiste en marcar cuál es el estado bueno, es decir, actúa como:

$$U_f(\alpha|bad\rangle + \beta|good\rangle) \rightarrow \alpha|bad\rangle - \beta|good\rangle$$

Esta es una reflexión sobre el estado *bad*. A continuación, consideremos el operador $2|S\rangle\langle S| - I$. Para entender su comportamiento, denotaremos por $|S^\perp\rangle$ un estado ortogonal a $|S\rangle$ y aplicaremos $2|S\rangle\langle S| - I$ sobre cualquier estado que sea una superposición de los dos estados ortogonales $|S\rangle$ y $|S^\perp\rangle$. Vamos a ver a continuación que el operador realiza una reflexión de este estado sobre el estado $|S\rangle$.

$$\begin{aligned} & (2|S\rangle\langle S| - I)(\alpha|S\rangle + \beta|S^\perp\rangle) \\ &= \alpha(2|S\rangle\langle S|S\rangle - |S\rangle) + \beta(2|S\rangle\langle S|S^\perp\rangle - |S^\perp\rangle) \\ &= \alpha(2|S\rangle - |S\rangle) + \beta(0 - |S^\perp\rangle) \\ &= \alpha|S\rangle - \beta|S^\perp\rangle \end{aligned}$$

A modo de resumen, el algoritmo de Grover comienza preparando una superposición igual de N estados correspondientes a todos los N elementos de la lista, luego, la iteración de Grover se aplica en esta superposición \sqrt{N} veces, y finalmente, se mide el estado resultante, sabiendo que el estado observado representa el elemento de interés con alta probabilidad.

En la siguiente figura, se muestra un circuito para el algoritmo de Grover para que se entienda mejor su funcionamiento.

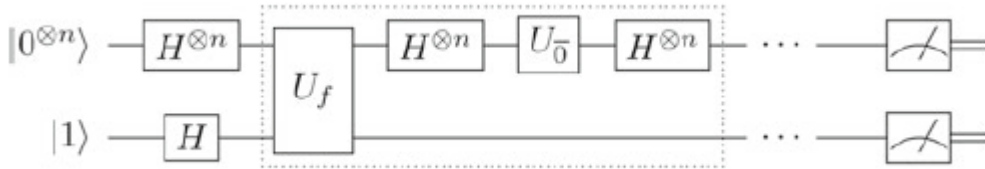


Figura 14 Circuito algoritmo de Grover

Vamos a realizar ahora un ejemplo genérico. Supongamos que tenemos una lista de $N = 2^n$ elementos de los cuales estamos interesados en encontrar un elemento particular. Para ello, vamos a llamar al elemento que nos interesa *good* y *bad* al resto de elementos. Entonces, un estado que contiene la superposición igual que todos los N elementos tiene la forma:

$$\psi = \frac{1}{\sqrt{N}}|good\rangle + \sqrt{\frac{N-1}{N}}|bad\rangle$$

Podemos ver que la probabilidad de obtener el estado *good* al medir $|\psi\rangle$ es $\frac{1}{N}$. Vamos a escoger θ en el intervalo $[0, \frac{\pi}{2}]$ tal que $\sin \theta = \frac{1}{\sqrt{N}}$. Entonces tendremos lo siguiente:

$$|\psi\rangle = \sin \theta |good\rangle + \cos \theta |bad\rangle$$

El ángulo θ es el ángulo formado por el estado $|\psi\rangle$ con el estado $|bad\rangle$, por lo que si aplicamos el oráculo U_f , obtenemos el estado:

$$|\psi'\rangle = -\sin \theta |good\rangle + \cos \theta |bad\rangle$$

Ahora, sabemos que

$$|S\rangle = \sum_{x \in \{0,1\}^n} |x\rangle = \frac{1}{\sqrt{N}} |good\rangle + \sqrt{\frac{N-1}{N}} |bad\rangle = \sin \theta |good\rangle + \cos \theta |bad\rangle$$

Como $|S\rangle$ y $|\psi\rangle$ representan el mismo estado, podemos expresar el estado $|S^\perp\rangle$ como

$$|S^\perp\rangle = \cos \theta |good\rangle - \sin \theta |bad\rangle.$$

De tal forma

$$\begin{aligned} \langle S|S^\perp\rangle &= (\sin \theta \langle good| + \cos \theta \langle bad|)(\cos \theta |good\rangle - \sin \theta |bad\rangle) \\ &= \cos \theta \sin \theta - \cos \theta \sin \theta = 0 \end{aligned}$$

y

$$|S\rangle\langle S| + |S^\perp\rangle\langle S^\perp| = |good\rangle\langle good| + |bad\rangle\langle bad| = I$$

Por lo tanto, $|S\rangle$ y $|S^\perp\rangle$ forman una base. Esto nos va a permitir escribir $|\psi'\rangle$ como una superposición de $|S\rangle$ y $|S^\perp\rangle$ de la siguiente forma:

$$\begin{aligned} |\psi'\rangle &= -\sin \theta |good\rangle + \cos \theta |bad\rangle \\ &= \sin(\theta - 2\theta) |good\rangle + \cos(\theta - 2\theta) |bad\rangle \\ &= (\cos 2\theta \sin \theta - \sin 2\theta \cos \theta) |good\rangle + (\cos 2\theta \cos \theta + \sin 2\theta \sin \theta) |bad\rangle \\ &= \cos 2\theta (\sin \theta |good\rangle + \cos \theta |bad\rangle) - \sin 2\theta (\cos \theta |good\rangle - \sin \theta |bad\rangle) \\ &= \cos 2\theta |S\rangle - \sin 2\theta |S^\perp\rangle \end{aligned}$$

Si ahora aplicamos el operador $2|S\rangle\langle S| - I$ en $|\psi'\rangle$, tenemos

$$\begin{aligned} |\psi''\rangle &= (2|S\rangle\langle S| - I)|\psi'\rangle \\ &= (2|S\rangle\langle S| - I)(\cos 2\theta |S\rangle - \sin 2\theta |S^\perp\rangle) \\ &= 2\cos 2\theta |S\rangle\langle S|S\rangle - \cos 2\theta |S\rangle - 2\sin 2\theta |S\rangle\langle S|S^\perp\rangle + \sin 2\theta |S^\perp\rangle \end{aligned}$$

Por lo tanto, $|\psi''\rangle$ en términos de estados $|good\rangle$ y $|bad\rangle$ se puede expresar como:

$$\begin{aligned} |\psi''\rangle &= \cos 2\theta |S\rangle + \sin 2\theta |S^\perp\rangle \\ &= \cos 2\theta (\sin \theta |good\rangle + \cos \theta |bad\rangle) + \sin 2\theta (\cos \theta |good\rangle - \sin \theta |bad\rangle) \\ &= \sin 3\theta |good\rangle + \cos 3\theta |bad\rangle \end{aligned}$$

Si el estado $|\psi''\rangle$ se mide, observaríamos el estado $|good\rangle$ con una probabilidad de $\sin^2 3\theta$. Por lo tanto, una sola aplicación de G puede aumentar la probabilidad de observar

un estado bueno de $\sin^2\theta$ a $\sin^23\theta$. El cálculo anterior se puede extender para mostrar que k aplicaciones de G pueden aumentar la probabilidad de observar un estado bueno a $\sin^2(2k+1)\theta$.

Volviendo a donde lo dejamos [3](pg.31), veamos qué pasa cuando aplicamos $U_{\bar{0}}$ a los primeros n qubits de nuestro estado cuántico $|\psi_2\rangle_{n,1}$.

$$\begin{aligned} U_{\bar{0}}(|\psi_2\rangle_n) &= (2|\gamma\rangle_n\langle\gamma|_n - I) \left(|\gamma\rangle_n - \frac{2}{\sqrt{2^n}} |j_0\rangle_n \right) \\ &= 2|\gamma\rangle_n\langle\gamma|_n - \frac{4}{\sqrt{2^n}} |\gamma\rangle_n\langle\gamma|_n |j_0\rangle_n - |\gamma\rangle_n + \frac{2}{\sqrt{2^n}} |j_0\rangle_n \\ &= 2|\gamma\rangle_n - \frac{4}{2^n} |\gamma\rangle_n - |\gamma\rangle_n + \frac{2}{\sqrt{2^n}} |j_0\rangle_n = \frac{2^{n-2}-1}{2^{n-2}} |\gamma\rangle_n + \frac{2}{\sqrt{2^n}} |j_0\rangle_n \end{aligned}$$

Es importante tener en cuenta las propiedades $\langle\gamma|\gamma\rangle_n = 1$ y que $\langle\gamma|i_0\rangle_n = \frac{1}{\sqrt{2^n}}$.

3.2. Ejemplo para $n = 3$

Para comprender mejor este algoritmo, vamos a proponer un ejemplo. Para ello, nos hemos basado en el ejemplo que aparece en la referencia [1] (pg. 32,33), solo que en vez de para $n=4$ qubits, lo vamos a realizar para $n=3$ qubits.

Supongamos que disponemos de una base de datos no estructura con los elementos enumerados del 0 al 7, indexados mediante 3 qubits. Queremos encontrar un elemento que está indexado con el número 5, aunque esto no se sabe, pero se supone para explicar mejor el algoritmo.

Necesitaremos para ello un ordenador cuántico con $n+1$ qubits, siendo $n=3$ el número de bits necesarios para codificar los índices de los elementos de la base de datos. Para configurar nuestro sistema cuántico, vamos a establecer los primeros 3 qubits en el estado $|0\rangle$, y el qubit restante en el estado $|1\rangle$, para que nuestro sistema cuántico empiece de la siguiente forma

$$|\psi_0\rangle_{3,1} = |0\rangle_3 \otimes |1\rangle$$

El primer paso consiste en aplicar a este sistema la puerta de Hadamard a todos los qubits, convirtiendo lo anterior en el siguiente estado

$$|\psi_1\rangle_{3,1} = (H^{\otimes 3}|0\rangle_3) \otimes (H|1\rangle) = \left(\frac{1}{\sqrt{8}} \sum_{j=0}^7 |j\rangle_3 \right) \otimes \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) = \left(\frac{1}{\sqrt{8}} \sum_{j=0}^7 |j\rangle_3 \right) \otimes |-\rangle$$

Usaremos los estados $|\gamma\rangle_3$ y $|\rho\rangle_3$, estados que definimos en el apartado anterior, para simplificar el proceso.

$$|\gamma\rangle_3 = \frac{1}{\sqrt{8}} \sum_{j=0}^7 |j\rangle_3$$

$$|\rho\rangle_3 = \frac{1}{\sqrt{7}} \sum_{\substack{j=0 \\ j \neq 5}}^7 |j\rangle_3$$

$$|\gamma\rangle_3 = \sqrt{\frac{7}{8}} |\rho\rangle_3 + \frac{1}{\sqrt{8}} |5\rangle_3$$

Con estas expresiones, procedemos a escribir el estado $|\psi_1\rangle_{3,1}$ como

$$|\psi_1\rangle_{3,1} = \left(\sqrt{\frac{7}{8}} |\rho\rangle_3 + \frac{1}{\sqrt{8}} |5\rangle_3 \right) \otimes |-\rangle$$

Después de aplicar la puerta de Hadamard, aplicábamos U_f , que lo que hacía era aplicar a todos los posibles estados dentro de los primeros 3 qubits un oráculo U_f que identificase al 5 como índice correcto para el elemento que estamos buscando.

$$|\psi_2\rangle_{3,1} = U_f \left[\left(\sqrt{\frac{7}{8}} |\rho\rangle_3 + \frac{1}{\sqrt{8}} |5\rangle_3 \right) \otimes |-\rangle \right] = \left(\sqrt{\frac{7}{8}} |\rho\rangle_3 - \frac{1}{\sqrt{8}} |5\rangle_3 \right) \otimes |-\rangle = \left(|\gamma\rangle_3 - \frac{2}{\sqrt{8}} |5\rangle_3 \right) \otimes |-\rangle$$

Por último, aplicábamos $U_{\bar{0}}$ a los primeros 3 qubits. Vamos a usar $|\rho\rangle_3$ para ver más claro qué parte del estado contiene el índice 5 y cuál no lo contiene.

$$\begin{aligned} U_{\bar{0}} \left(|\gamma\rangle_3 - \frac{2}{\sqrt{8}} |5\rangle_3 \right) &= (2|\gamma\rangle_3 \langle \gamma|_3 - I) \left(|\gamma\rangle_3 - \frac{2}{\sqrt{8}} |5\rangle_3 \right) \\ &= 2|\gamma\rangle_3 \langle \gamma|_3 - \frac{4}{\sqrt{8}} |\gamma\rangle_3 \langle \gamma|_5 - |\gamma\rangle_3 + \frac{2}{\sqrt{8}} |5\rangle_3 \\ &= 2|\gamma\rangle_3 - \frac{4}{\sqrt{8}} \frac{1}{\sqrt{8}} |\gamma\rangle_3 - |\gamma\rangle_3 + \frac{2}{\sqrt{8}} |5\rangle_3 = 2|\gamma\rangle_3 - \frac{1}{2} |\gamma\rangle_3 - |\gamma\rangle_3 + \frac{2}{\sqrt{8}} |5\rangle_3 \\ &= \frac{1}{2} |\gamma\rangle_3 + \frac{2}{\sqrt{8}} |5\rangle_3 = \frac{1}{2} \left(\frac{\sqrt{7}}{\sqrt{8}} |\rho\rangle_3 + \frac{1}{\sqrt{8}} |5\rangle_3 \right) + \frac{2}{\sqrt{8}} |5\rangle_3 \\ &= \frac{\sqrt{7}}{2\sqrt{8}} |\rho\rangle_3 + \frac{5\sqrt{2}}{8} |5\rangle_3 \end{aligned}$$

Finalmente, tenemos que el estado $|\psi_3\rangle_{3,1}$ es

$$|\psi_3\rangle_{3,1} = \left(\frac{\sqrt{7}}{2\sqrt{8}} |\rho\rangle_3 + \frac{5\sqrt{2}}{8} |5\rangle_3 \right) \otimes |-\rangle$$

Si medimos nuestro estado cuántico, obtenemos una probabilidad $p = \left(\frac{5\sqrt{2}}{8} \right)^2 = \frac{50}{64} \approx 0,78125$ de obtener un índice 5, y una probabilidad $\bar{p} = \left(\frac{\sqrt{7}}{2\sqrt{8}} \right)^2 = \frac{7}{32} \approx 0,21875$ de obtener cualquier otro índice.

Vamos a realizar otra iteración del algoritmo ya que la probabilidad de obtener el índice buscado no es lo suficientemente grande. Vamos a volver a aplicar U_f .

$$|\psi_4\rangle_{3,1} = U_{f,3}(|\psi_3\rangle_{3,1}) = \left(\frac{\sqrt{7}}{2\sqrt{8}} |\rho\rangle_3 - \frac{5\sqrt{2}}{8} |5\rangle_3 \right) \otimes |-\rangle = \left(\frac{1}{2} |\gamma\rangle_3 - \frac{3}{\sqrt{8}} |5\rangle_3 \right) \otimes |-\rangle$$

Aplicando una vez más $U_{\bar{0}}$

$$\begin{aligned} |\psi_5\rangle_3 &= (2|\gamma\rangle_3\langle\gamma|_3 - I) \left(\frac{1}{2} |\gamma\rangle_3 - \frac{3}{\sqrt{8}} |5\rangle_3 \right) \\ &= \frac{2}{2} |\gamma\rangle_3\langle\gamma|_3 - \frac{6}{\sqrt{8}\sqrt{8}} |\gamma\rangle_3 - \frac{1}{2} |\gamma\rangle_3 + \frac{3}{\sqrt{8}} |5\rangle_3 \\ &= |\gamma\rangle_3 - \frac{3}{4} |\gamma\rangle_3 - \frac{1}{2} |\gamma\rangle_3 + \frac{3}{\sqrt{8}} |5\rangle_3 = -\frac{1}{4} |\gamma\rangle_3 + \frac{3}{\sqrt{8}} |5\rangle_3 \\ &= -\frac{1}{4} \left(\frac{\sqrt{7}}{\sqrt{8}} |\rho\rangle_3 + \frac{1}{\sqrt{8}} |5\rangle_3 \right) + \frac{3}{\sqrt{8}} |5\rangle_3 = -\frac{\sqrt{14}}{16} |\gamma\rangle_3 + \frac{11\sqrt{2}}{16} |5\rangle_3 \end{aligned}$$

Después de estas iteraciones, obtenemos la siguiente probabilidad $p = \left(\frac{11\sqrt{2}}{16} \right)^2 = \frac{242}{256} \approx 0,9453125$.

Si nos fijamos, con respecto a la iteración anterior, ha aumentado la probabilidad de encontrar el índice que queremos. Sin embargo, si seguimos iterando, esto no nos garantiza un mayor aumento, y esto es debido a que el algoritmo no siempre mejora a medida que vamos aumentando iteraciones. Esto se verá con más detalle en el capítulo 4 de este trabajo.

3.3. Desarrollo del algoritmo en Qiskit

A continuación, demostraremos la implementación del algoritmo de Grover en Qiskit [1](pg.64-66). Nuestro objetivo es encontrar el punto de entrada x para el cual la función booleana de 4 bits $f(x) = x_2x_3 \oplus x_1x_2x_3 \oplus x_2x_3x_4 \oplus x_1x_2x_3x_4$ produce 1. Para ello, realizamos el siguiente código.

```
1 #Importamos los módulos necesarios
2 from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister,
  transpile
3 from qiskit_aer import Aer
4 from qiskit.circuit.library import *
5 #Building an oracle for the function
6 #f(x)=x2x3+x1x2x3+x2x3x4+x1x2x3x4
7 #This is the U_f.
8 def oracle():
9     orcl = QuantumCircuit(4)
10    orcl.x(3)
11    orcl.append(ZGate().control(3, ctrl_state='110'), range(4))
12    orcl.x(3)
13    return orcl
```

```

15 #Número de iteraciones implementadas
16 num_iter = 3
17 #Creación del registro cuántico, registro clásico
18 #y circuito cuántico
19 q = QuantumRegister (4)
20 c = ClassicalRegister (4)
21 qc = QuantumCircuit(q,c)
22 #Iniciación del estado inicial a superposición
23 qc.h(range (4))
24 #Implementación de la iteración de Grover
25 #por num_iter
26 for k in range(num_iter):
27     qc.append(oracle (), range (4))
28     qc.h(range (4))
29     qc.x(3)
30     qc.append(ZGate ().control(3, ctrl_state='000'),range (4))
31     qc.x(3)
32     qc.h(range (4))
33 #Medición de qubits
34 for i in range (4):
35     qc.measure(i,3-i)
36 # Transpilar el circuito
37 backend = Aer.get_backend('qasm_simulator')
38 qc_transpiled = transpile(qc, backend)
39 # Ejecutar el circuito en el simulador
40 qjob = backend.run(qc_transpiled, shots=100)
41 counts = qjob.result().get_counts()
42 print(counts)
43 qc.draw(output='mpl')

```

Código 11 Implementación del algoritmo de Grover

Vamos a explicar el código que proporcionamos antes. Como siempre, importamos todas las clases que nos van a hacer falta para poder crear y ejecutar circuitos. En la línea 8 vamos a definir la función oráculo *oracle()* que nos va a devolver el oráculo para la función $f(x)$. Dado que $f(x)$ es una función booleana de 4 bits, el tamaño de su tabla de verdad es $2^4 = 16$. Por lo tanto, se espera que la iteración de Grover se aplique en un total de aproximadamente $\sqrt{16} = 4$ veces. Por ello, creamos el oráculo con 4 qubits llamado *orcl*. Aplicamos después una puerta X al qubit en la posición 3, para así cambiar el estado del qubit de $|0\rangle$ a $|1\rangle$. En la línea 11, lo que hacemos es aplicar la puerta Z para aplicar una fase de -1 a $|1\rangle$. Esta puerta se controla con 3 qubits y se aplica al cuarto qubit. Especificamos la cadena 110 para indicar que la puerta se aplica cuando los tres primeros qubits están en el estado $|1\rangle$ y el cuarto qubit está en el estado $|0\rangle$. Volvemos a aplicar la puerta X al qubit de la posición 3 para reestablecer el estado del qubit a $|0\rangle$, y devolvemos el resultado de la función *oracle*. En la línea 15 establecemos el número de iteraciones de Grover que se van a implementar, que en este caso es 3. Seguido, creamos un registro cuántico *q* de 4 qubits y un registro clásico *c* de 4 bits, y luego creamos el circuito cuántico *qc*. Después, en la línea 23 se le aplica una puerta Hadamard a todos los qubits del registro cuántico *q*, lo que crea superposición igual de todos los posibles estados. De las líneas 26 a 32, vamos a implementar un bucle for donde se van a implementar las iteraciones de

Grover, y para cada iteración, se va a aplicar el oráculo, seguido de la puerta de Hadamard, para aplicar superposición, la puerta X, para cambiar el qubit de la tercera posición del estado $|0\rangle$ al $|1\rangle$, y la puerta Z, donde esta vez la cadena de control que especificamos es 000, es decir, que la puerta se aplica cuando los tres primeros qubits están en el estado $|0\rangle$ y el cuarto qubit está en el estado $|1\rangle$. Volvemos a aplicar tanto la puerta X al qubit de la tercera posición para restablecer el estado del qubit a $|0\rangle$ como la puerta Hadamard a todos los qubits del circuito. En la línea 34 creamos un bucle for que itera sobre los índices de los qubits y los almacena. Para terminar, se obtienen los resultados finales y se muestra el circuito.

Tenemos que tener en cuenta que la amplitud en el estado $|0110\rangle$ alcanza el máximo cuando el número de iteraciones es 3. En la salida podemos ver que el estado que se observa con una alta probabilidad es $|0110\rangle$, que corresponde al número 6 en binario.

```
{'0110': 94, '0011': 2, '1100': 1, '0001': 1, '1001': 1, '1011': 1}
```

Figura 15 Salida algoritmo de Grover

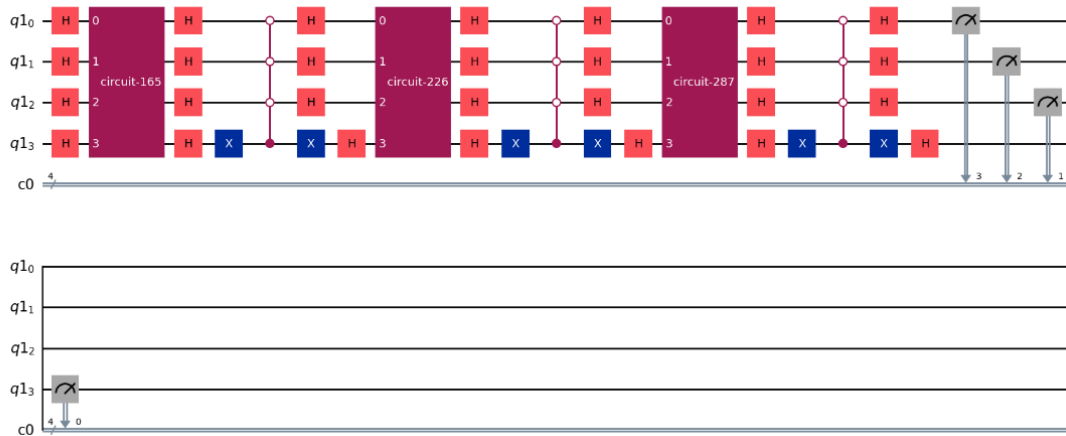


Figura 16 Circuito del algoritmo de Grover

4. Manual de usuario de Qiskit

Para la implementación de los algoritmos, vamos a utilizar Qiskit [8], un kit de desarrollo software creado por IBM para trabajar con computadoras cuánticas a nivel de circuitos, pulsos y algoritmos.

Qiskit se basa en Python, uno de los lenguajes de programación más populares y accesibles del mundo. Esto hace que el proceso de desarrollo de algoritmos cuánticos sea más intuitivo y familiar para una amplia gama de usuarios. Desde la creación de circuitos cuánticos hasta la simulación y ejecución en hardware real, Qiskit proporciona un conjunto de herramientas completo y flexible que permite a los usuarios aprovechar al máximo el potencial de la computación cuántica.

Qiskit permite trabajar en diferentes niveles, cada uno con una función específica.

Qiskit Terra, que es el núcleo de Qiskit, y se encarga de la creación y manipulación de circuitos cuánticos y estructuras de datos básica.

Qiskit Aer proporciona simuladores de alto rendimiento para realizar experimentos virtuales y validar algoritmos antes de ejecutarlos en hardware real.

Qiskit Aqua está diseñada especialmente para funcionar en ordenadores cuánticos para resolver problemas de optimización, química y aprendizaje automático.

Qiskit Ignis, que se centra en la caracterización y corrección de errores, además de la mitigación del ruido para mejorar la precisión de programas cuánticos.

Los circuitos cuánticos que se implementan en Qiskit pueden ser ejecutados en nuestro propio ordenador utilizando Aer de Qiskit, o bien en un computador cuántico. Gracias a IBM, que nos proporciona una plataforma en la nube llamada IBM Quantum Experience, tenemos acceso gratuito a sistemas cuánticos reales y simuladores cuánticos.

4.1. Instalación de Qiskit

Vamos a explicar paso a paso cómo instalar y hacer uso de Qiskit. En este trabajo, hemos utilizado el entorno de Visual Studio Code. Se recomienda usar una versión de Python 3.6 o superior. En nuestro caso, utilizaremos la versión 3.12.2.

Una vez iniciado el entorno de Visual Studio Code, lo primero que haremos será instalarnos una extensión de Jupyter Notebook. Para ello, nos dirigiremos al menú de la izquierda, y clicaremos en el quinto submenú, que es el apartado de las extensiones. Una vez ahí, buscaremos la extensión de Jupyter y la instalamos. Esto nos servirá para poder aprovechar las características de Jupyter Notebooks dentro de Visual Studio Code.

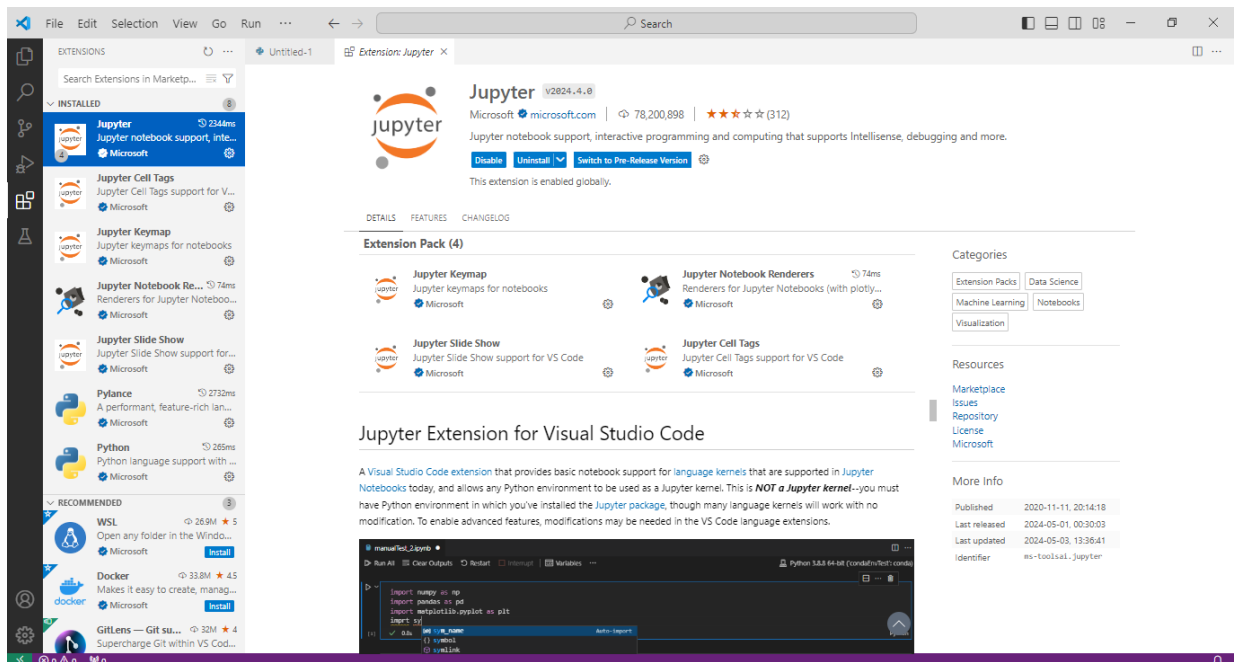


Figura 17 Agregando extensión a Visual Studio Code

Vamos a continuación, a instalar pip. Para ello, vamos a abrir un terminal, que se encuentra en el menú de **VIEW>>TERMINAL**.

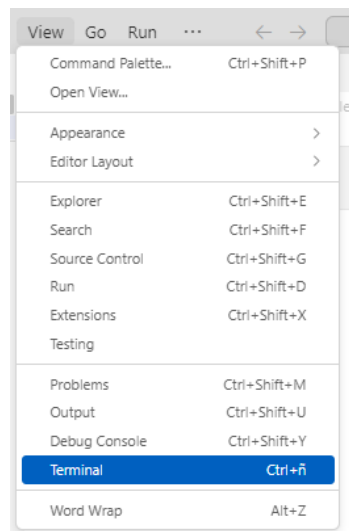


Figura 18 Terminal

Ejecutaremos la siguiente línea de comando.

```
conda install pip
```

Figura 19 comando conda install pip

Vamos a crear ahora una carpeta para guardar nuestros archivos. Para ello, nos iremos al menú de la izquierda, y seleccionaremos el primer icono que veamos, para que se nos abra una ventana al lado, y cliquemos en “Open Folder”.

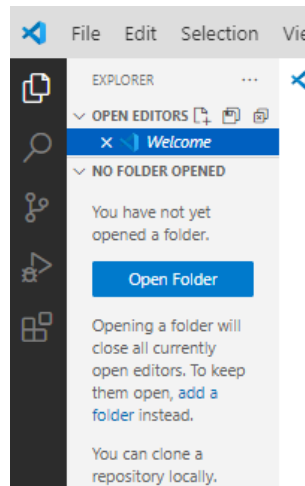


Figura 20 Crear carpeta

Una vez creada la carpeta, procedemos a crear un nuevo archivo de Python. Para ello, nos iremos al menú y seleccionamos en FILE > NEW FILE y nos aparecerá un submenú donde se nos pregunta si queremos un archivo de tipo texto, un archivo de tipo Python o un archivo de Jupyter Notebook, y escogeremos este último.

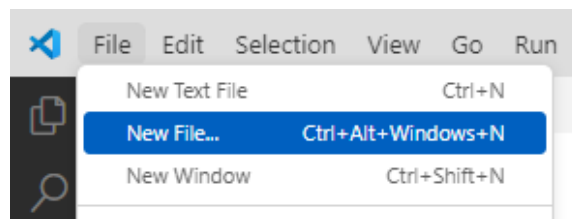


Figura 21 Crear archivo nuevo

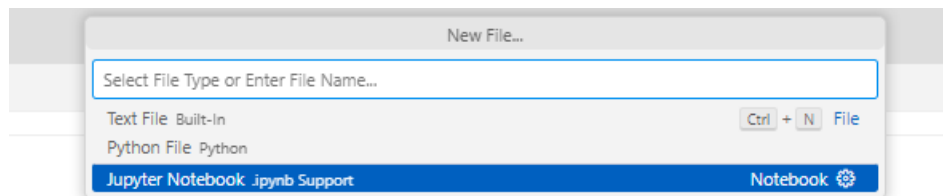


Figura 22 Jupyter Notebook

Después de haber creado el archivo, tenemos que seleccionar un Kernel, que es lo que se encarga de interpretar y ejecutar el código escrito en las celdas. Para ello, seleccionamos en la parte de la derecha donde pone Select Kernel.

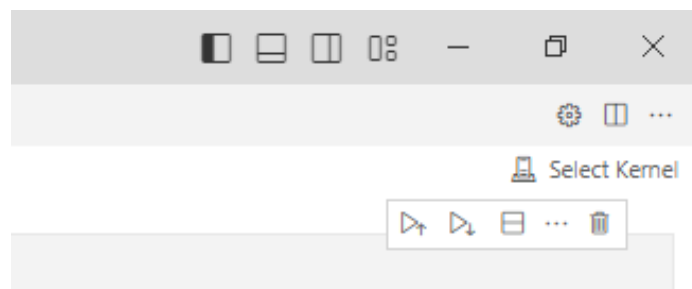


Figura 23 Seleccionar Kernel

Nos preguntará a continuación qué Kernel queremos. En nuestro caso, escogeremos la opción de Python Enviroments.

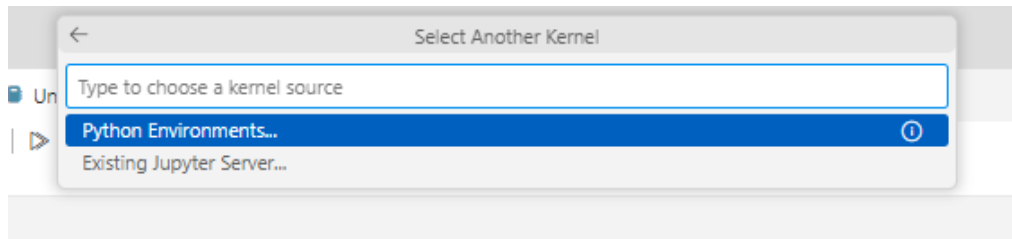


Figura 24 Elegir entorno

Elegiremos el entorno que nos recomienda, el “base (Python 3.12.2)”. Una vez seleccionado, al intentar escribir en la línea de comando, debería aparecer que se necesita instalarse el “ipykernel”, a lo que diremos que sí.

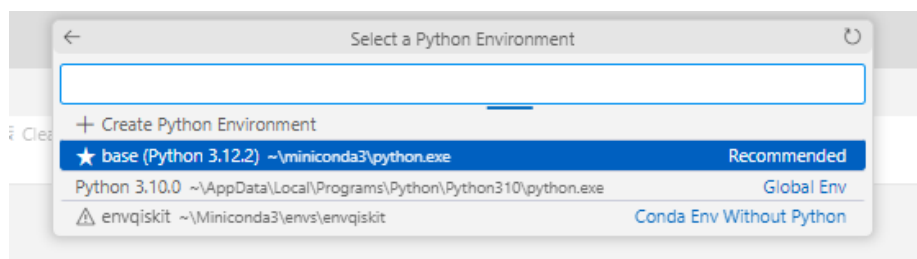


Figura 25 Entorno recomendado

Una vez tengamos el Kernel seleccionado, empezamos con la instalación de Qiskit. Para ello, ejecutaremos la siguiente línea de comando.

```
import qiskit
```

Figura 26 Importación de qiskit.

Una vez importado, veremos en qué versión nos encontramos. Para ello, ejecutaremos el siguiente comando.

```
qiskit.__version__
```

✓ 0.0s

'1.0.2'

Figura 27 Versión de qiskit.

Como vemos, utilizaremos la versión 1.0.2.

Si no nos aparecieron errores, entonces significa que todo ha ido correctamente y que la instalación fue exitosa.

4.2. Estructura básica de un programa en Qiskit

4.2.1. Clases

A continuación, se detallan las clases principales que se utilizan en Qiskit y sus respectivas funciones. De qiskit disponemos de las siguientes clases

QuantumCircuit: Clase principal para crear y manipular circuitos cuánticos. Permite definir una secuencia de operaciones cuánticas que quieres realizar.

Classical Register: Define registros clásicos que contienen bits clásicos.

QuantumRegister: Define registros cuánticos, que contienen qubits.

Transpile: Esta función sirve para optimizar y convertir un circuito cuántico a una forma que pueda ser ejecutada en un backend específico.

De qiskit_aer disponemos de la siguiente clase

Aer: Es el módulo de simulación de Qiskit que permite la simulación de circuitos cuánticos.

4.2.2. Creación de circuitos cuánticos

Para crear un circuito cuántico, primero se definen los registros cuánticos y clásicos, una vez importadas las clases necesarias.

```
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister

# Crear un registro cuántico con 2 qubits y un registro clásico con 2 bits
qr = QuantumRegister(2)
cr = ClassicalRegister(2)

# Crear un circuito cuántico con los registros creados
qc = QuantumCircuit(qr, cr)
```

Figura 28 Creación de circuitos cuánticos.

4.2.3. Añadir puertas cuánticas

Para añadir puertas cuánticas a un circuito, usaremos lo siguiente.

```
# Añadir una puerta Hadamard en el primer qubit
qc.h(qr[0])

# Añadir una puerta CNOT con control en el primer qubit y objetivo en el segundo qubit
qc.cx(qr[0], qr[1])
```

Figura 29 Añadir puertas cuánticas.

4.2.4. Medición de qubits

Para medir los qubits y almacenar el resultado en los bits clásicos, usaremos el método `measure`.

```
qc.measure(qr, cr)
```

Figura 30 Medición de qubits.

4.3. Ejecución de un circuito

Para ejecutar el circuito, utilizaremos las siguientes líneas de código

```
# Ejecutar el circuito en el simulador
backend = Aer.get_backend('qasm_simulator')
qc_transpiled = transpile(qc, backend)

# Ejecutar el circuito en el simulador
qjob = backend.run(qc_transpiled, shots=100)
counts = qjob.result().get_counts()
print(counts)
qc.draw(output='mpl')
```

Figura 31 Ejecución de código.

5. Consecuencias del algoritmo de Grover

El algoritmo de Grover es uno de los algoritmos cuánticos más destacados, conocido por su capacidad de buscar en una base de datos no estructurada en un tiempo significativamente más corto que cualquier algoritmo clásico. Mientras que éstos requieren de $O(N)$ consultas para encontrar un elemento en una base de datos de N elementos, el algoritmo de Grover reduce este número a $O(\sqrt{N})$ consultas, aprovechando la amplificación de amplitud cuántica.

Sin embargo, un aspecto crucial es que este algoritmo depende del número correcto de iteraciones [3](pg.34,35). La amplificación de amplitud funciona incrementando la probabilidad de encontrar el estado deseado, pero este incremento sigue un patrón oscilatorio, y, después de alcanzar un pico de probabilidad máxima, continuar iterando puede comenzar a disminuir la probabilidad de éxito en lugar de aumentarla.

El algoritmo de búsqueda de Grover, para una solución única, necesita $m \sim O(2^n)$ iteraciones de la puerta G para maximizar la probabilidad de obtener el elemento deseado con índice desconocido j_0 .

Primero, vamos a redefinir los estados posibles que ocurren durante la ejecución del algoritmo como una función de las distintas amplitudes que se involucran. Como se mostró anteriormente, la única amplitud que diferirá del resto después de cada iteración de Grover es la asociada con el estado base que identifica el elemento buscado. Vamos a definir un estado genérico de la siguiente manera

$$|\psi(\alpha, \beta)\rangle_n = \alpha|j_0\rangle_n + \beta \sum_{\substack{j=0 \\ j \neq j_0}}^{2^n-1} |j\rangle_n$$

con α y β restringidos por $\alpha^2 + (2^n - 1)\beta^2 = 1$

Solo estamos teniendo en cuenta el primer registro, el que tiene n qubits, ya que el qubit restante comenzará en $|1\rangle$ y permanecerá como $|-\rangle$ durante el resto del algoritmo, pero lo omitiremos para que se pueda comprender mejor.

Vamos a observar qué sucede cuando aplicamos todas las puertas cuánticas que componen la transformación de Grover G a un estado genérico $|\psi(\alpha, \beta)\rangle_n$. Empezaremos utilizando U_f para que reconozca el elemento buscado y voltee su amplitud.

$$|\psi(\alpha, \beta)\rangle_n = -\alpha|j_0\rangle_n + \beta \sum_{\substack{j=0 \\ j \neq j_0}}^{2^n-1} |j\rangle_n$$

Antes de seguir avanzando, vamos a ver nuestro estado actual como una función de $|\gamma\rangle_n$, como hicimos cuando explicamos el algoritmo de Grover en la sección 3.

$$|\psi'(\alpha, \beta)\rangle_n = -(\alpha + \beta)|j_0\rangle_n + \sqrt{2^n}\beta|\gamma\rangle_n$$

Ahora podemos aplicar $U_{\bar{0}}$. Recordemos que

$$U_{\bar{0}} = (2|\gamma\rangle_n \langle \gamma|_n - I)$$

Por consiguiente

$$\begin{aligned} |\psi''(\alpha, \beta)\rangle_n &= U_{\bar{0}}(|\psi'(\alpha, \beta)\rangle_n) = U_{\bar{0}}(-(\alpha + \beta)|j_0\rangle_n + \sqrt{2^n}\beta|\gamma\rangle_n) \\ &= (2|\gamma\rangle_n \langle \gamma|_n - I)(-(\alpha + \beta)|j_0\rangle_n + \sqrt{2^n}\beta|\gamma\rangle_n) \\ &= (\alpha + \beta)|j_0\rangle_n + \left(2\sqrt{2^n}\beta - \frac{2}{\sqrt{2^n}}(\alpha + \beta) - \sqrt{2^n}\beta\right)|\gamma\rangle_n \\ &= \left(\frac{2^{n-1} - 1}{2^{n-1}}\alpha + \frac{2^n - 1}{2^{n-1}}\beta\right)|j_0\rangle_n + \left(-\frac{1}{2^{n-1}}\alpha + \frac{2^n - 1}{2^{n-1}}\beta\right)|\gamma\rangle_n \end{aligned}$$

Vamos a predecir a continuación en cuál iteración del algoritmo tendremos más posibilidades de obtener el elemento deseado i_0 . Para ello, vamos a definir un par de expresiones que nos ayudarán a encontrar una fórmula más manejable para la amplitud de i_0 .

$$|\psi_{k+1}(\alpha_{k+1}, \beta_{k+1})\rangle_n = G(|\psi_k(\alpha_k, \beta_k)\rangle_n)$$

donde

$$\begin{aligned} \alpha_1 &= \beta_1 = \frac{1}{\sqrt{2^n}} \\ \alpha_{k+1} &= \frac{2^{n-1} - 1}{2^{n-1}}\alpha_k + \frac{2^n - 1}{2^{n-1}}\beta_k \\ \beta_{k+1} &= \frac{1}{2^{n-1}}\alpha_k + \frac{2^n - 1}{2^{n-1}}\beta_k \end{aligned}$$

para $j \geq 1$. Note que para $j=0$ no es posible definir $|\psi_0\rangle$ como una función de α_0 y β_0 .

Si designamos θ tal que $\sin^2 \theta = \frac{1}{2^n}$, podemos probar por inducción matemática que $\alpha_j = \sin((2j - 1)\theta)$ y, por lo tanto

$$\beta_j = \frac{1}{\sqrt{2^n - 1}} \cos((2j - 1)\theta)$$

Si suponemos que, para un paso desconocido $j = m + 1$, siendo m el número de veces que hemos aplicado G , queremos asegurar que $\alpha_m = 1$. Esto ocurrirá cuando $(2m + 1)\theta = \frac{\pi}{2}$ y cuando $m = \frac{\pi - 2\theta}{4\theta}$.

Como es obvio, no podemos realizar un número no entero de iteraciones de G . Si tomamos por ejemplo $m = \left\lfloor \frac{\pi}{4\theta} \right\rfloor$, podemos concluir que el número de iteraciones de G necesarias para alcanzar la máxima probabilidad de éxito es cercano a $\left(\frac{\pi}{4}\right)\sqrt{2^n}$. Nótese

que $\theta \approx \sin \theta = \frac{1}{\sqrt{2^n}}$ cuando 2^n es suficientemente grande.

Con todo esto, podemos concluir que el número m de iteraciones de G tiene un orden de $m \sim O(\sqrt{2^n})$.

Veámoslo con el código que implementamos en la sección 3. Si lo ejecutamos con una iteración, los resultados que se nos muestra son los siguientes.

```
{'1011': 4, '0110': 55, '1110': 5, '1001': 1, '0001': 3, '1010': 3, '0100': 4,
'1100': 9, '0101': 2, '1101': 1, '0011': 4, '0111': 2, '1000': 3, '1111': 3, '0000': 1}
```

Figura 32 Una iteración para el algoritmo de Grover.

Si nos fijamos, podemos ver cómo el estado $|0110\rangle$ se observa 55 de 100 veces. Para dos iteraciones del algoritmo obtendríamos lo siguiente.

```
{'0110': 92, '0010': 1, '1011': 1, '1101': 2, '1000': 1, '0111': 2, '1110': 1}
```

Figura 33 Dos iteraciones para el algoritmo de Grover.

Podemos ver ahora cómo el estado $|0110\rangle$ pasa de observarse de 55 a 92 veces. Podemos decir que ha aumentado la probabilidad de encontrar ese estado. Vamos a ver los resultados para una tercera iteración.

```
{'0110': 98, '0111': 1, '1000': 1}
```

Figura 34 Tres iteraciones para el algoritmo de Grover.

Se ha vuelto a incrementar las veces de que salga el estado $|0110\rangle$ que buscamos. Para una cuarta iteración ocurre lo siguiente.

```
{'0110': 59, '1100': 4, '0101': 3, '0001': 2, '1010': 1, '0000': 2, '1000': 3, '1011': 8,
'1111': 2, '1101': 1, '0010': 2, '0011': 2, '0111': 5, '1110': 5, '1001': 1}
```

Figura 35 Cuatro iteraciones para el algoritmo de Grover.

Vemos que ha descendido y pasamos de 98 veces a 59. Si realizamos una quinta iteración, ocurriría lo siguiente.

```
{'0001': 4, '1010': 9, '1001': 3, '1000': 7, '0100': 5, '0010': 5, '0110': 17, '0101': 4,
'1110': 6, '0011': 12, '1011': 8, '0111': 6, '1111': 5, '0000': 3, '1101': 4, '1100': 2}
```

Figura 36 Cinco iteraciones para el algoritmo de Grover.

Con esto concluimos que, a medida que vamos iterando en el algoritmo de Grover, éste no garantiza un aumento continuo.

El algoritmo de Grover es el método de ataque cuántico más representativo que amenaza la seguridad de la criptografía de clave simétrica [9]. Al realizarse un ataque de fuerza bruta clásico contra un sistema simétrico, un atacante prueba con todas las posibles claves hasta encontrar la correcta. Para una clave de n bits, existen 2^n posibles claves. Esto quiere decir que el tiempo promedio para encontrar la clave correcta es 2^{n-1} pruebas.

Sin embargo, el algoritmo de Grover mejora significativamente la eficiencia de los ataques de fuerza bruta, ya que utiliza un procedimiento cuántico que reduce el número de pruebas necesarias de 2^n a aproximadamente $2^{n/2}$ pruebas. Por ejemplo, una clave de 128 bits, que se considera segura contra ataques clásicos, ofrece una seguridad efectiva de solo 64 bits contra un ataque cuántico con Grover. Esto significa que, para mantener el mismo nivel de seguridad contra ataques de fuerza bruta cuánticos, las claves deben ser el doble de largas.

6. Posibles ampliaciones del trabajo

A continuación, se van a tratar un par de aspectos que pueden resultar interesantes y que pueden ayudar a comprender mejor este proyecto.

6.1. IBM Quantum Composer

Vamos a hacer uso de IBM Quantum Composer, una plataforma para ver algunos ejemplos de cómo se emplean algunas puertas cuánticas y algoritmos, para observar cómo cambian las probabilidades y gráficamente cómo se ven en la esfera de Bloch.

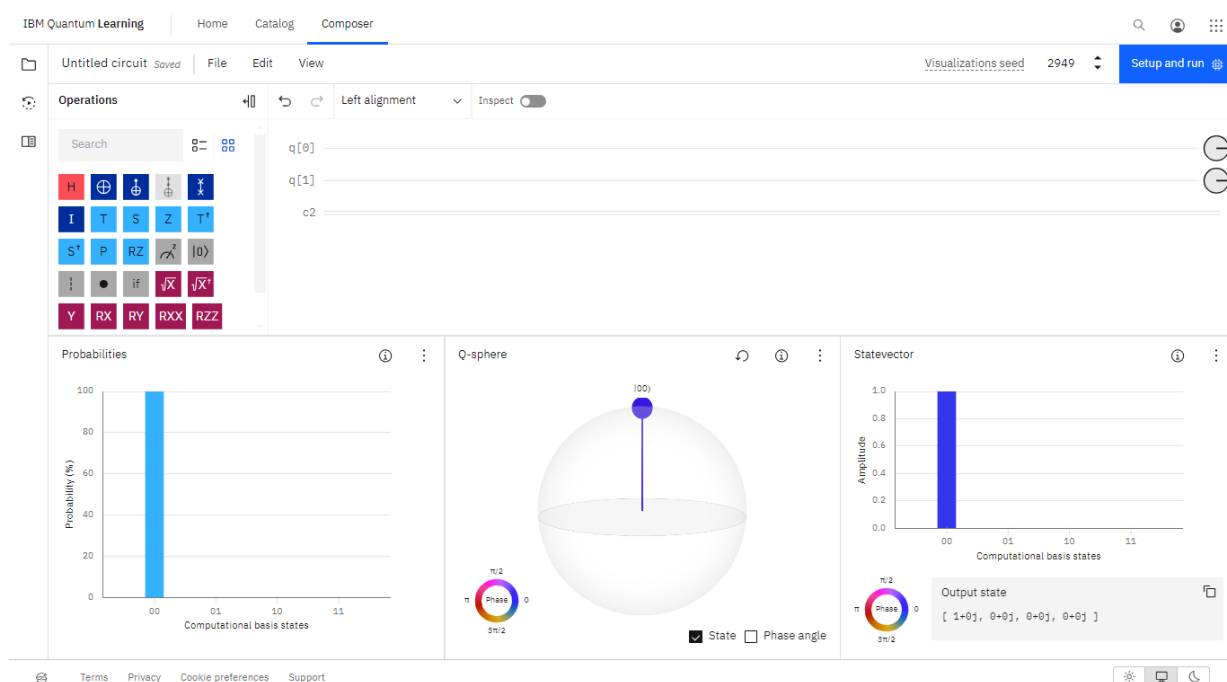


Figura 37 IBM Quantum Composer

A continuación, explicaremos un poco qué nos encontramos en el apartado de Composer de IBM. En la parte de la izquierda, nos encontramos con un menú donde aparecen varias puertas cuánticas. Podemos arrastrarlas y soltarlas en las líneas que aparecen a la derecha, que es donde se crean los circuitos. De momento solo tenemos q[0], y q[1], que son los que van a representar los qubits, aunque se pueden añadir más qubits. Debajo disponemos de c[1], que representa el bit clásico.

Tenemos que tener mucho cuidado a la hora de dónde colocar las puertas, ya que, en esta plataforma, se sitúan los bits más significativos en la parte de abajo y los menos significativos en la parte de arriba, por lo que podemos hacer dos cosas; o bien construimos el circuito al revés, el cual nos resultaría un poco confuso, o bien, a la hora de medir, en vez de medir incrementando, lo hacemos decrementando. Lo que ocurre de esta última manera, es que a la hora de verse en la esfera, no nos va a salir bien. Para este trabajo, vamos a utilizar la primera opción, pero simplemente para que se vean las gráficas claras.

En la parte inferior, de izquierda a derecha nos encontramos con una gráfica donde se nos muestra las probabilidades de los estados del circuito. Inicialmente, se está midiendo

el estado $|00\rangle$, con una probabilidad de un 100% en ese estado. Después tenemos la representación visual del estado mediante la esfera de Bloch, y por último, disponemos de una gráfica donde se nos muestra las amplitudes de los diferentes estados. Existe otra vista, que no está habilitada, en la que te muestra el código, tanto en qiskit como en OpenQASM 2.0, mientras vas creando el circuito. Para habilitarlo, le damos en el menú de View>>Panels>>CodeEditor. Para este caso, lo vamos a deshabilitar, ya que queremos centrarnos en los circuitos y probabilidades.

Empezaremos con las puertas de Pauli. Recordemos que existían la puerta X, la puerta Y y la puerta Z. Vamos a empezar por la puerta X. Ésta tiene la siguiente representación



Figura 38 IBM Composer: representación puerta X

Si arrastramos tal puerta, obtenemos lo siguiente

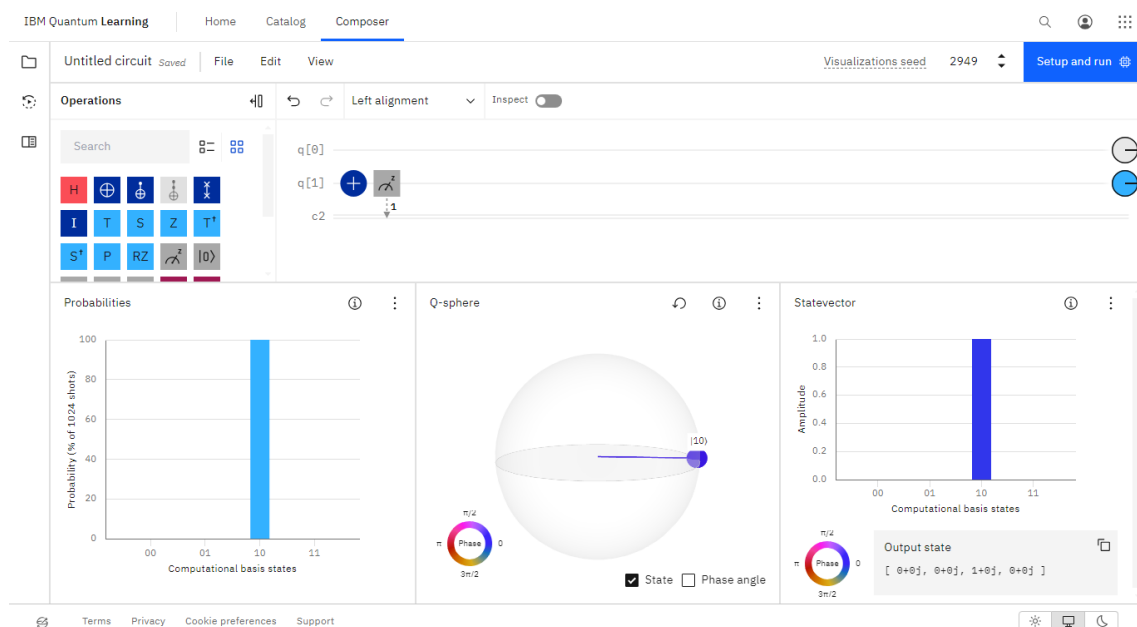


Figura 39 IBM Composer: implementación puerta X sobre primer qubit

Si repasamos lo que hace la puerta X, ésta cambia el estado de $|0\rangle$ a $|1\rangle$ y de $|1\rangle$ a $|0\rangle$. Si observamos la figura de arriba, en la gráfica de la izquierda se nos muestra que el circuito tiene una probabilidad del 100% en conseguir el estado $|10\rangle$, ya que hemos aplicado la puerta X, y hemos cambiado su estado del $|0\rangle$ al $|1\rangle$, como comentamos anteriormente.

Si lo que queremos es pasar al estado $|01\rangle$, lo que haremos será aplicarle la puerta X al qubit q[1], de tal forma que ahora obtendremos una probabilidad del 100% de conseguir ese estado. Lo podemos ver en la figura que se muestra a continuación.

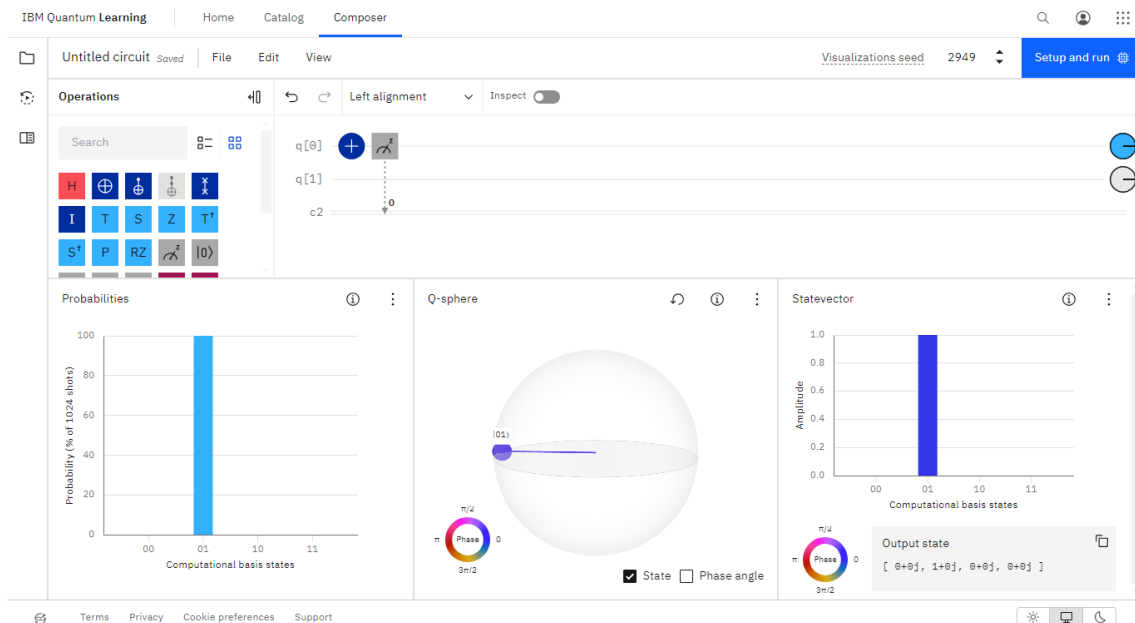


Figura 40 IBM Composer: implementación puerta X sobre segundo qubit

Finalmente, si queremos conseguir el estado $|11\rangle$, lo que tenemos que hacer es aplicar a cada qubit la puerta X.

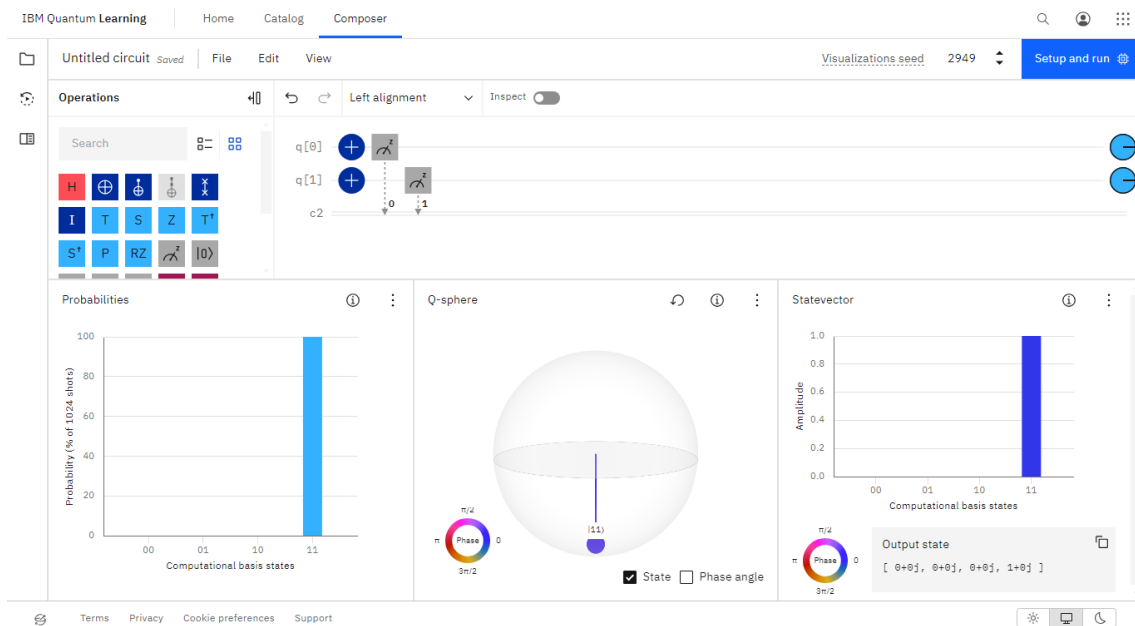


Figura 41 IBM Composer: implementación puerta X en ambos qubits

Para la puerta Y, cuya representación es la siguiente,



Figura 42 IBM Composer: representación puerta Y

si la aplicamos sobre los dos qubits, lo que vamos a obtener como resultado es una

probabilidad del 100% de obtener el estado $|11\rangle$, con una amplitud de 1 pero con una fase de π . Esto quiere decir, que la amplitud es de -1.

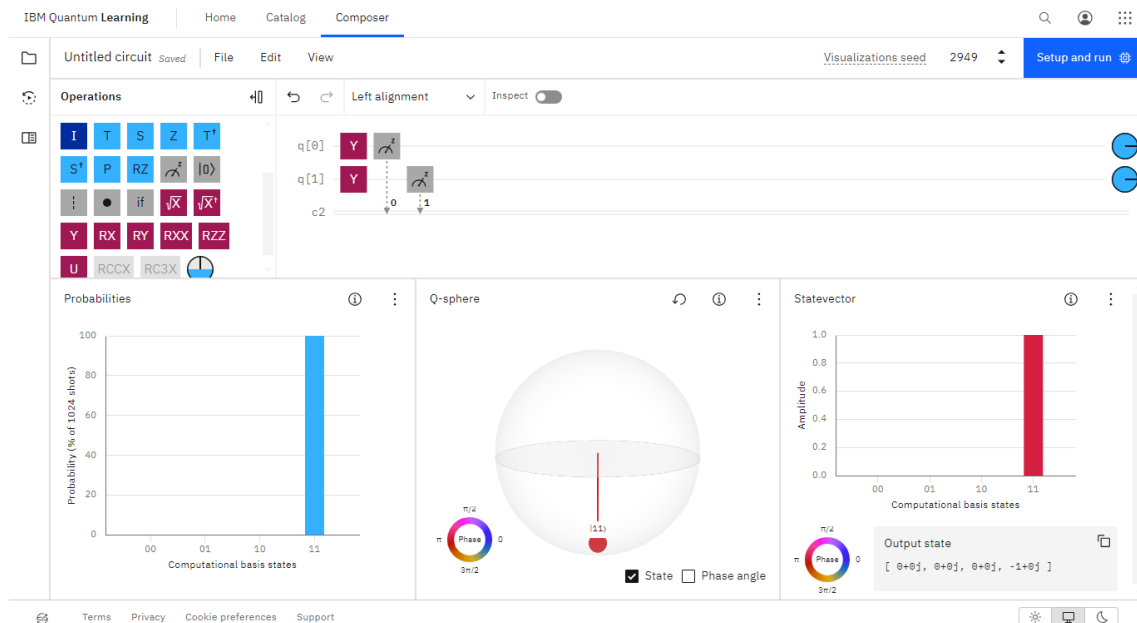


Figura 43 IBM Composer: implementación puerta Y sobre ambos qubits

Para la puerta Z, que su símbolo es el siguiente



Figura 44 IBM Composer: representación puerta Z

lo que nos interesa es aplicar la puerta a los estados $|10\rangle$ y para $|01\rangle$. Para ambos casos, lo que vamos a conseguir es obtener una probabilidad del 100% de conseguir el mismo estado respectivamente, pero con una amplitud de 1 con una fase de π , es decir, una amplitud de -1. Lo podemos ver a continuación.

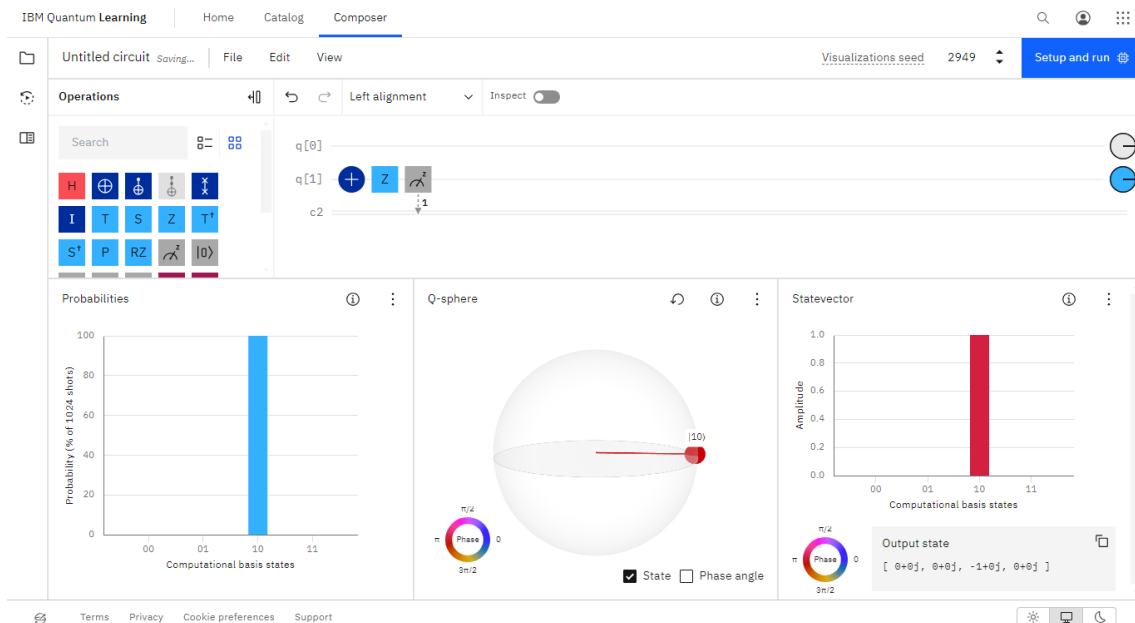


Figura 45 IBM Composer: implementación puerta Z sobre el primer qubit

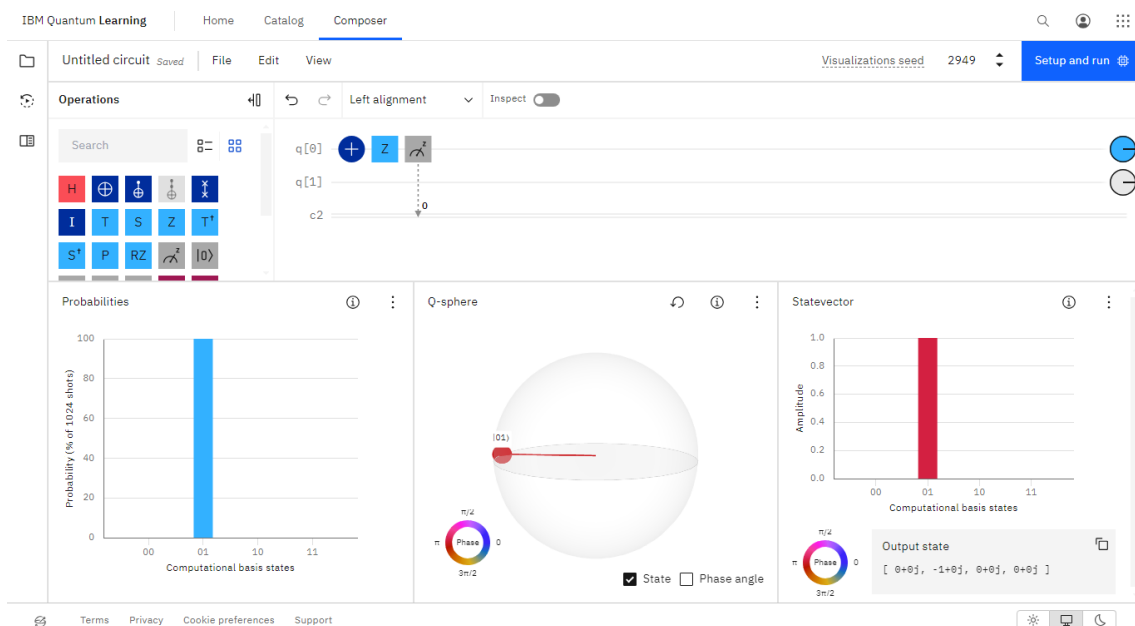


Figura 46 IBM Composer: implementación puerta Z sobre segundo qubit

Vamos ahora a ver los resultados de aplicar la puerta de Hadamard. La representación de la puerta H es la que muestra a continuación



Figura 47 IBM Composer: representación puerta H

Si aplicamos la puerta H al estado $|0\rangle$, obtendremos una probabilidad de $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$,

con una amplitud de 1, como podemos ver a continuación. Las probabilidades varían un poco debido a que existen algunos márgenes y errores.

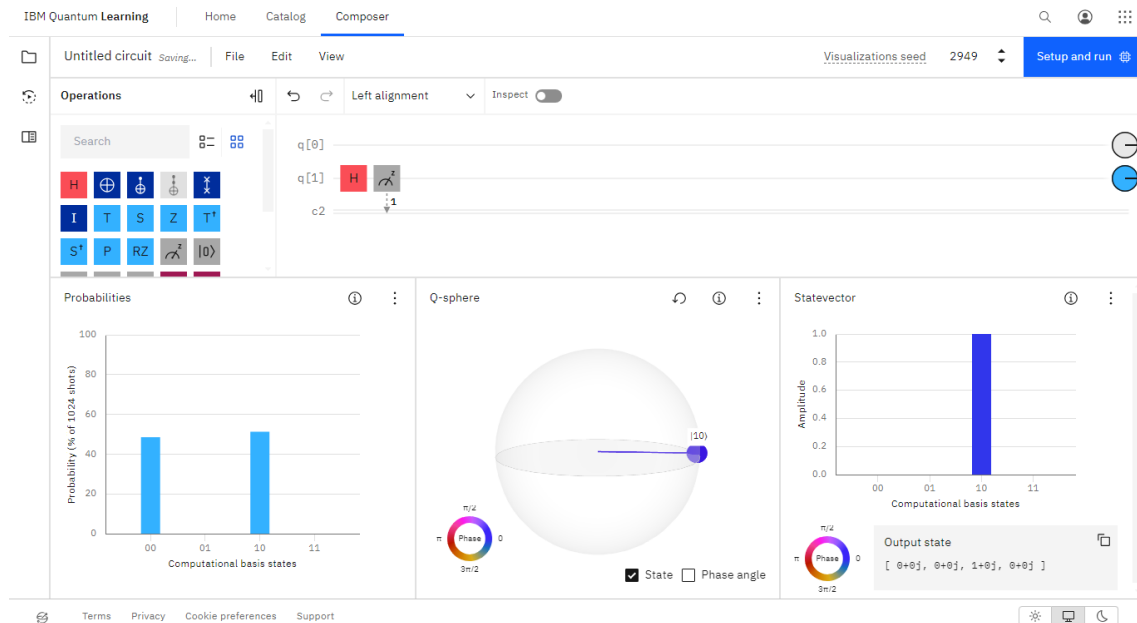


Figura 48 IBM Composer: implementación puerta H sobre qubit estado 0

Sin embargo, si aplicamos la puerta H al estado $|1\rangle$, lo que vamos a obtener es una probabilidad de $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, con una amplitud de -1.

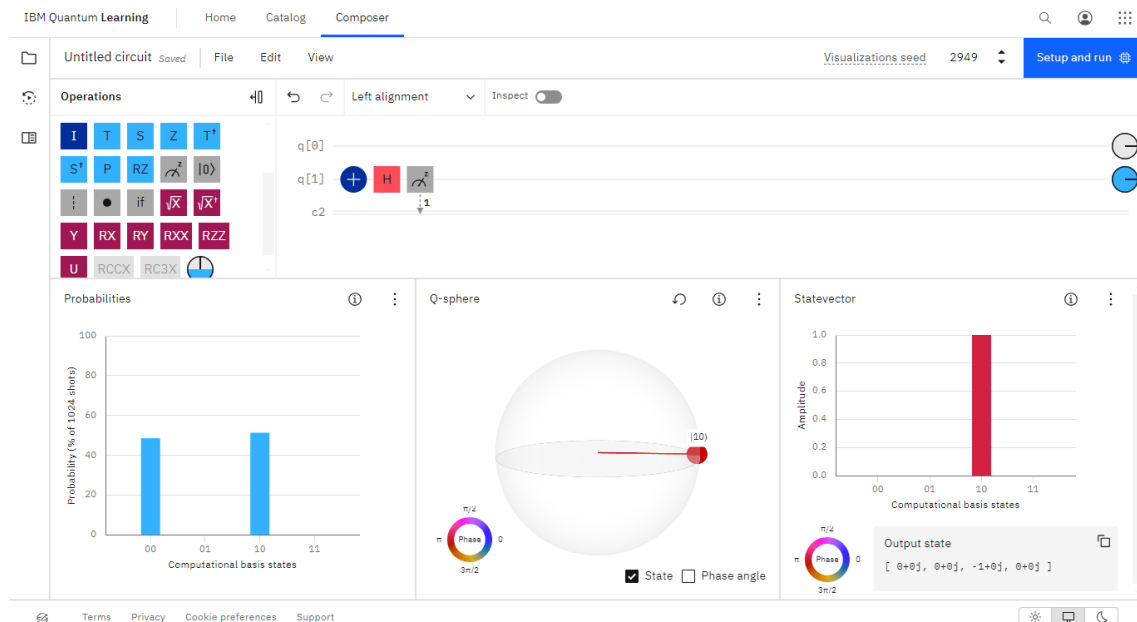


Figura 49 IBM Composer: implementación puerta H sobre qubit estado 1

Para la puerta CNOT, cuyo símbolo es el que sigue,



Figura 50 IBM Composer: representación puerta CNOT

lo que va a hacer esta puerta es, si el qubit de control se encuentra en el estado $|0\rangle$, dejar en el estado $|0\rangle$ el qubit objetivo, pero si el qubit de control se encuentra en el estado $|1\rangle$, va a invertir el estado del qubit objetivo. Se puede ver a continuación.

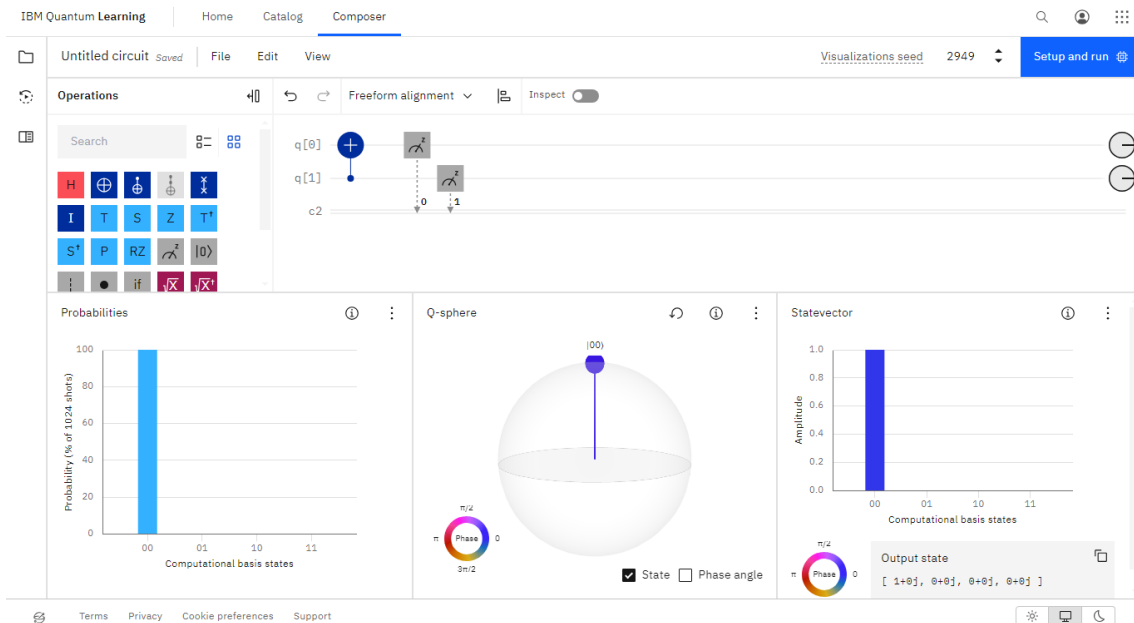


Figura 51 IBM Composer: implementación puerta CNOT sobre qubit estado 0

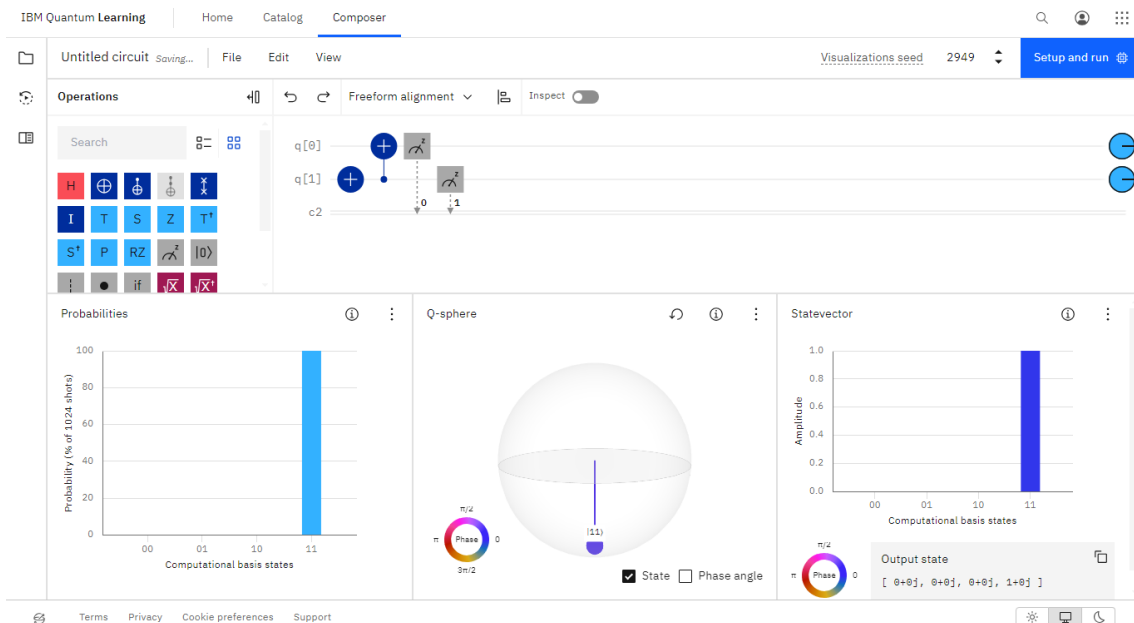


Figura 52 IBM Composer: implementación puerta CNOT sobre qubit 1

Finalizamos con la puerta SWAP, cuya representación se puede ver a continuación



Figura 53 IBM Composer: representación puerta SWAP

La función de esta puerta consiste en intercambiar los estados. Es decir, si nos

encontramos en el estado $|10\rangle$, lo que nos devolverá será una probabilidad de que salga el estado $|01\rangle$, y viceversa. Lo podemos ver en la figura que se muestra abajo.

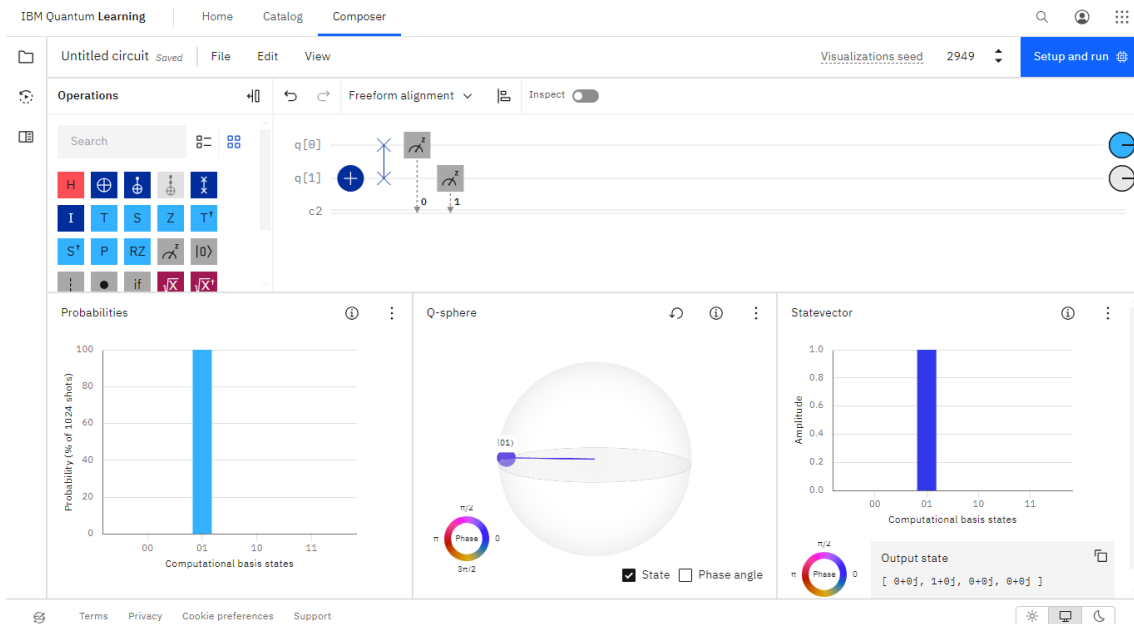


Figura 54 IBM Composer: implementación puerta SWAP sobre estado 10

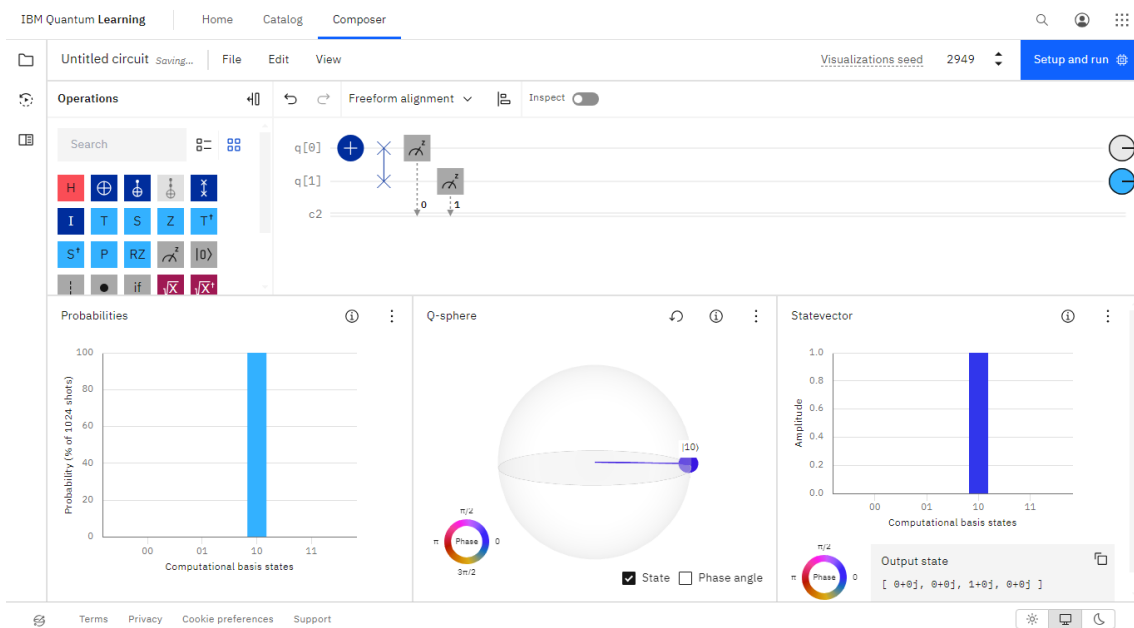


Figura 55 IBM Composer: implementación puerta SWAP sobre estado 01

Por último, vamos a ver cómo se realizarían los algoritmos de Deutsch-Jozsa y el algoritmo de Grover. Se recuerda que los circuitos deben de crearse de manera inversa ya que en IBM Composer los datos más significativos se encuentran abajo y los menos significativos se encuentran arriba.

Para el algoritmo de Deutsch, el circuito quedaría de la siguiente manera

Para el algoritmo de Deutsch-Jozsa, el circuito quedaría como se muestra en la siguiente figura



Figura 56 IBM Composer: Algoritmo de Deutsch-Jozsa

Recordemos que lo que hacíamos era poner los qubits de entrada en estado de superposición, creábamos un oráculo donde escribíamos una función, que es lo que se encuentra entre las dos barreras, el cual nos decía si una función era constante o balanceada. Hemos propuesto un circuito para cada caso. Para una función constante, en el oráculo se contraría una función Z, y el circuito quedaría tal que así

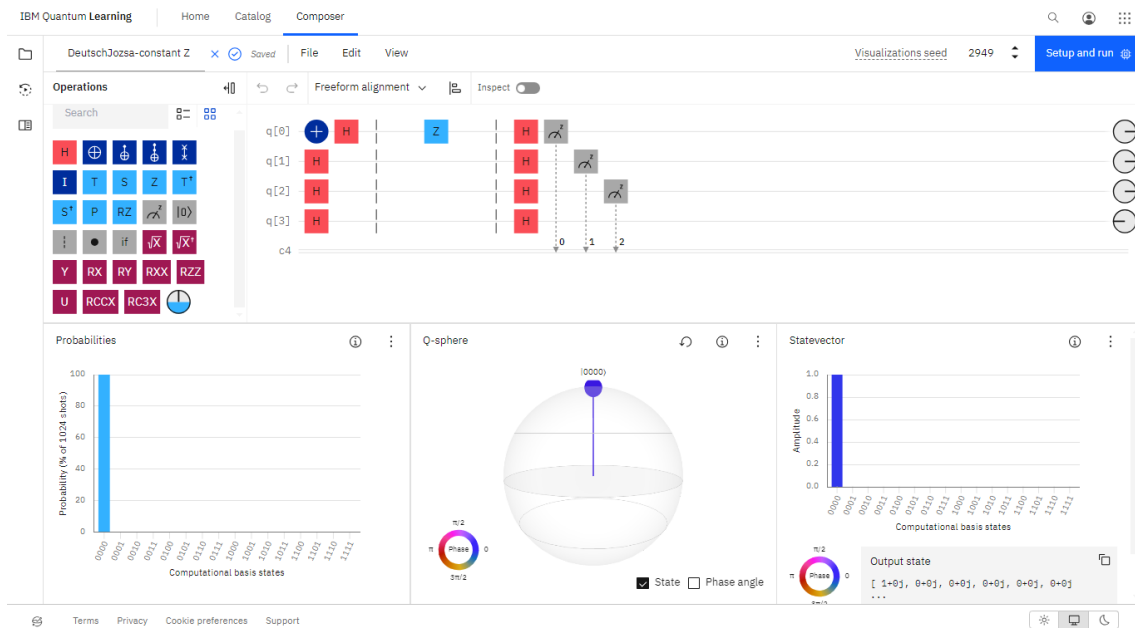


Figura 57 IBM Composer: Algoritmo de Deutsch-Jozsa para una función constante

y para una función balanceada, en el oráculo se encuentra la función $f(x) = x_2 \oplus x_3 \oplus x_4$, que ya la trabajamos anteriormente en el capítulo 2, y tendríamos un circuito tal que así

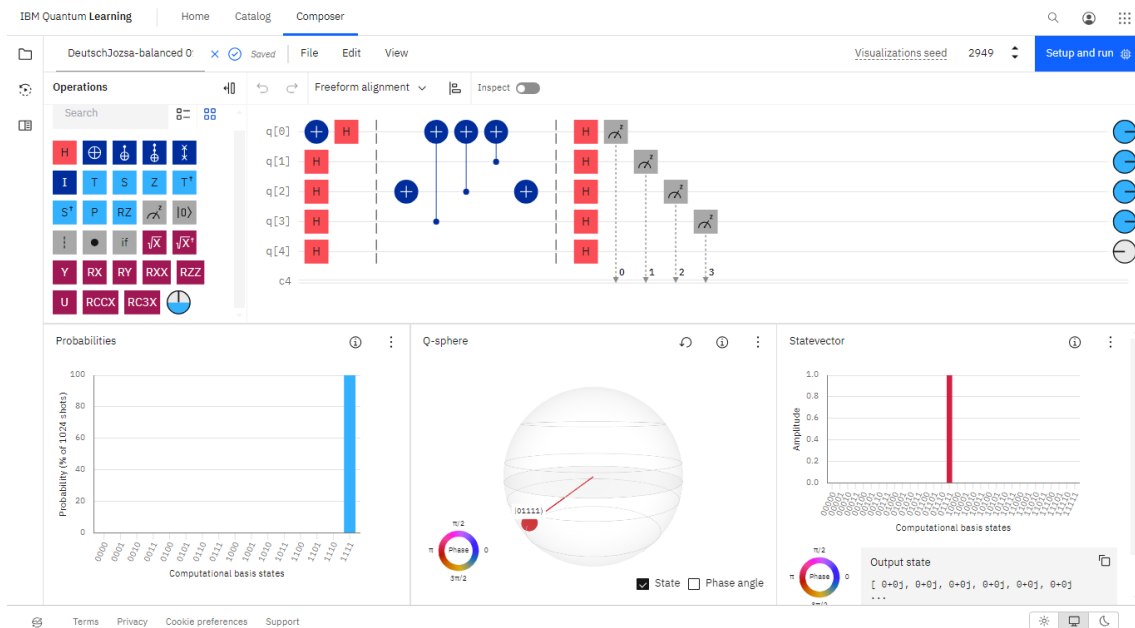


Figura 58 IBM Composer: Algoritmo de Deutsch-Jozsa para una función balanceada

Y para finalizar, un ejemplo de un circuito del algoritmo de Grover, para buscar el estado $|01\rangle$.

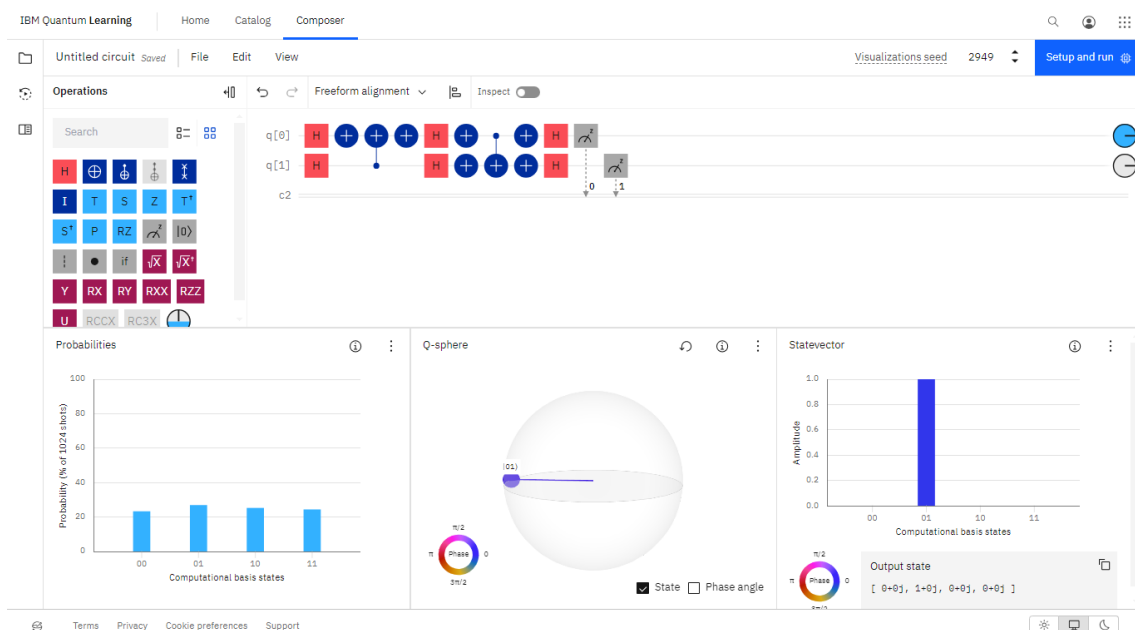


Figura 59 IBM Composer: Algoritmo de Grover

6.2. Algoritmo de Shor

Otro aspecto que podemos comentar en esta sección es mencionar otro algoritmo muy importante y conocido como es el algoritmo de Shor [10], [11]. En 1994, Peter Shor, un matemático del Instituto Tecnológico de Massachusetts, presentó un algoritmo cuántico revolucionario que tenía el potencial de resolver uno de los problemas más difíciles en la teoría de números: la factorización de enteros. Este problema implica descomponer un número entero compuesto en sus factores primos, es decir, encontrar los números primos

que multiplicados entre sí producen el número original. Aunque esta tarea parece sencilla para números pequeños, se vuelve exponencialmente más difícil a medida que los números crecen en tamaño.

Sin embargo, el algoritmo cuántico desarrollado por Shor cambió por completo esta perspectiva. Utilizando los principios de la computación cuántica, Shor demostró que era posible resolver el problema de la factorización de enteros en tiempo polinómico, lo que significa que el tiempo requerido para resolver el problema aumenta de manera mucho más controlada con el tamaño del número. Este avance nos ofrece una solución considerablemente más rápida y eficiente en comparación con los algoritmos clásicos conocidos hasta el momento.

La capacidad de factorizar grandes números en tiempo polinómico tiene implicaciones en el campo de la criptografía. Muchos sistemas de criptografía de clave pública, como por ejemplo RSA, dependen de la dificultad de factorizar números grandes para garantizar la seguridad de las comunicaciones y transacciones en línea. Sin embargo, el algoritmo de Shor plantea una amenaza potencial para estos sistemas, ya que podría desbloquear la capacidad de descomponer las claves criptográficas en un tiempo relativamente corto utilizando un ordenador cuántico lo suficientemente potente.

Para evitar eso, se está trabajando en varias soluciones para abordar esta amenaza, como la búsqueda de nuevos algoritmos criptográficos que sean seguros frente a los ordenadores cuánticos, o el desarrollo de ordenadores cuánticos más potentes y fiables.

7. Conclusiones

En este trabajo de Fin de Grado, se ha llevado a cabo un estudio exhaustivo de varios aspectos fundamentales de la computación cuántica, abarcando desde sus principios teóricos hasta la implementación práctica de algoritmos cuánticos.

Los temas que se han abordado han sido los siguientes:

Se han explicado los fundamentos de los qubits, así como las diferencias entre los bits clásicos.

Se han estudiado los principios cuánticos más importantes para el funcionamiento de los algoritmos cuánticos; la superposición y el entrelazamiento.

Se han definido las puertas cuánticas que más se utilizan para la realización de circuitos cuánticos.

Se ha profundizado en dos algoritmos cuánticos muy importantes; el algoritmo de Deutsch-Jozsa, donde se ha implementado y probado el algoritmo para funciones balanceadas y constantes, y el algoritmo de Grover, donde se ha analizado la capacidad para realizar búsquedas en base de datos no estructurada.

Se han implementado y realizado un manual de usuario para todo aquél que quiera comenzar a trabajar con Qiskit.

Este trabajo me ha permitido no solo adquirir conocimientos teóricos profundos sobre la computación cuántica, sino también a desarrollar habilidades prácticas en la implementación y experimentación con circuitos cuánticos.

Mencionar por último, que todo el código está disponible en un repositorio de gitub: <https://github.com/irenefernandezromero/TrabajoFinDeGrado>

Referencias

- [1] T. SAPV, D. Bera, A. Maitra y S. Maitra, Quantum Algorithms for Cryptographically Significant Boolean Functions, An IBMQ Experience, 2021.
- [2] C. R. Jiménez, «<http://www.fisicafundamental.net/misterios/computacion.html>,» [En línea].
- [3] J. Ossorio-Castillo y J. M. Tornero, Quantum computing from a mathematical perspective: a description of the quantum circuit model., 2018.
- [4] AWS, «¿Qué es la computación cuántica?,» [En línea]. Available: <https://aws.amazon.com/es/what-is/quantum-computing/#:~:text=principios%20a%20continuaci%C3%B3n,Superposici%C3%B3n,dos%20o%20m%C3%A1s%20estados%20distintos..>
- [5] J. E. G. Cornejo, «Conceptos Matemáticos Básicos de Computación Cuántica, Puertas Lógicas y Cuánticas, Circuitos cuánticos, Matriz Unitaria,» 2020. [En línea]. Available: https://docirs.cl/Puertas_Circuitos_Cuanticos.asp.
- [6] J. A. Maldonado, «Computación cuántica, compuertas o circuitos cuánticos,» 2017. [En línea]. Available: <https://josueacevedo.medium.com/computaci%C3%B3n-cu%C3%A1ntica-compuertas-o-circuitos-cu%C3%A1nticos-27910f5338c8>.
- [7] Z. Li, J. Dai, S. Pan, W. Zhang¹ y J. Hu, Synthesis of Deutsch-Jozsa Circuits and Verification, 2020.
- [8] «Qiskit,» [En línea]. Available: <https://docs.quantum.ibm.com/>.
- [9] K. Jang, G. Song, H. Kim y H. Kwon, Efficient Implementation of PRESENT and GIFT on Quantum Computers, 2021.
- [10] «<https://utimaco.com/es/servicio/base-de-conocimientos/criptografia-postcuantica/que-es-el-algoritmo-de-shor>,» [En línea].
- [11] «<https://www.tomorrow.bio/es/post/qu%C3%A9-es-el-algoritmo-de-shor-s-2023-06-4669709562-quantum>,» [En línea].