El Algoritmo de Grover

Autor:
David Castaño Bandín
SCBI (Universidad de Málaga)

Revisores:
Javier Mas Solé
Universidad de Santiago de Compostela
Andrés Gómez Tato
CESGA

27/03/2023

Quantum SPAIN
España | digital 2026
GOBIERNO DE ESPAÑA · VICEPRESIDENCIA PRIMERA DEL GOBIERNO · MINISTERIO DE ASUNTOS ECONÓMICOS Y TRANSFORMACIÓN DIGITAL · SECRETARÍA DE ESTADO DE DIGITALIZACIÓN E INTELIGENCIA ARTIFICIAL
Plan de Recuperación, Transformación y Resiliencia
Financiado por la Unión Europea NextGenerationEU
RES RED ESPAÑOLA DE SUPERCOMPUTACIÓN

Índice

Diccionario de notaciones

$(O)(z)$	Esta notación quiere decir "del orden de z".
N	Número de elemento del dataset en el que buscamos
n	Número de qubits que nos hacen falta
M	Número de soluciones que buscamos en el dataset de N elementos
t	Número de iteraciones del algoritmo de Grover
T	Número óptimo de iteraciones del algoritmo de Grover
$\lfloor z \rfloor$	Redondear z a un entero por truncamiento
$\lceil z \rceil$	Redondear z al siguiente entero.

1. Introducción

La computación cuántica es un campo relativamente joven y en el cual hay depositadas muchas esperanzas, pues se cree que tiene el potencial de poder resolver algunos problemas con los que la computación clásica no puede lidiar. Estamos hablando de los famosos problemas de la clase de complejidad NP-hard. Sin embargo, pese a esta creencia, pocas demostraciones teóricas hay sobre el tema. La demostración más famosa es la del **Algoritmo de Shor**, que es capaz de abordar el problema de la factorización de un número en sus factores primos. Esto es porque el algoritmo es capaz de calcular los factores primos en un tiempo polinónico, frente al tiempo exponencial que requieren los mejores algoritmos clásicos. El algoritmo sobre el que tratan estas notas, el también archiconocido **Algoritmo de Grover**, no ofrece una ventaja exponencial respecto al mejor algoritmo clásico, pero si ofrece una mejora sustancial (cuadrática).

Resumiendo mucho, el algoritmo de Grover es un algoritmo de búsqueda no estructurada. El objetivo del algoritmo es encontrar un valor (o varios) en una secuencia no ordenada de N componentes. Supongamos que tenemos una lista L con N elementos donde denominamos L_i, con $i = 0, 1, \ldots, N-1$, al i-ésido elemento de la lista. Supongamos que queremos encontrar un elemento q en la lista, es decir, lo que queremos encontrar es el índice ω tal que $L_\omega = q$. Si asumimos que la lista está desordenada, ningún algoritmo clásico conocido puede darnos un tiempo de búsqueda mejor que $\mathcal{O}(N)$. Es decir, el tiempo promedio de búsqueda crece linealmente con N, con el número de elementos entre los que buscamos. Esto es fácil de ver, pues para encontrar nuestro elemento tendremos que ir probando uno a uno los elementos de la lista. De esta forma en promedio tendremos que hacer $N/2$ pruebas para encontrar el resultado deseado.

Este tipo de problemas se denominan problemas de **búsqueda no estructurada** o problemas de búsqueda en **conjuntos de datos (datasets) no estructurados**. Como acabamos de comentar, estos problemas requieren un tiempo polinómico (en concreto, lineal) para ser resueltos. Se engloban dentro de esta categoría también problemas que presenten un dataset con alguna clase de estructura siempre que esta no se pueda aprovechar para acelerar la búsqueda.

La ventaja que aporta el algoritmo de Grover es **cuadrática**, ya que pasamos de necesitar un tiempo $\mathcal{O}(N)$ a un tiempo $\mathcal{O}(\sqrt{N})$. Esto puede parecer decepcionante si lo comparamos con la mejora exponencial que promete el algoritmo de Shor, pero está lejos de serlo. Aunque a priori no parezca una mejora sustancial, cuando tratamos con valores suficientemente grandes de N (datasets inmenso), la mejora en tiempo respecto a su contrapartida clásica puede ser de ordenes de magnitud, es decir, para nada despreciable.

El algoritmo de Grover no hace uso de la estructura interna del dataset en el que realiza la búsqueda, lo cual lo hace genérico y aplicable a una gran variedad de casos. Los problemas de búsqueda aparecen por doquier en las ciencias computacionales, con lo que una mejora en la eficiencia de estos es de gran interés. Además, este algoritmo no solo nos sirve para búsquedas, sino que nos sirve como subrutina para conseguir un aumento de velocidad cuadrático en otros algoritmo. A esto último se lo denomina el truco de la **amplificación de la amplitud**.

En estas notas vamos a ver todo lo necesario para entender el algoritmo de Grover, hablando tanto de matemática como de implementaciones. La estructura de las notas es la siguiente. En la sección 2 veremos una explicación geométrica del algoritmo, donde se usará como ejemplo el caso más simple (una solución, $N = 2^n$ y distribución uniforme) y se verán las diferentes partes del algoritmo. En las siguientes secciones iremos generalizando el algoritmo a medida que relajamos las condiciones del ejemplo anterior. En la sección 3 veremos que pasa cuando tenemos un número M de soluciones (conocido M). En la sección 4 relajaremos la condición de conocer el número M de soluciones y veremos que el algoritmo sigue siendo eficiente. Como continuación lógica de la sección anterior, en la sección 5 veremos un algoritmo inspirado en el algoritmo de Shor que usa el operador de Grover

para contar el número de soluciones (obtener M). En la sección 6 hablaremos sobre la implementación del difusor y como la formulación más habitual del mismo es la responsable de imponer la condición $N = 2^n$. Veremos en la subseción 6.1.2 como podemos eliminar también esta condición. Por último, en la sección 7 veremos que también es posible relajar la condición de partir de una distribución de probabilidad uniforme.

2. Explicación geométrica del algoritmo

Para facilitar la explicación y sin perdida de generalidad, pongámonos en el caso más simple de todos: queremos encontrar una única solución $|\omega_0\rangle$ y tenemos $N = 2^n$. Supongamos también que podemos partir del caso más simple, de una superposión uniforme de todos los estados. Más adelante iremos viendo generalizaciones del algoritmo para casos más complicados, como el caso de varias soluciones o el caso en el que partimos de una distribución de probabilidad aleatoria.

2.1. Estado inicial: superposición

Partimos de una superposición uniforme de N estados

$$|\Psi_0\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N} |i\rangle. \tag{2.1}$$

Esto no es más que decir que al inicio de la búsqueda, todas las opciones son igualmente probables (cualquier suposición de la ubicación de la solución es tan buena como cualquier otra).

Nota

Recordemos que si $N = 2^n$, donde n es el número de qubits, este estado es fácil de construir, pues solo tenemos que aplicar puertas de Hadamard en todos los Qubits (partiendo estos del estado $|0\rangle$):

$$|\Psi_0\rangle = H^{\otimes n}|0\rangle^n = \frac{1}{\sqrt{N}} \sum_{i=0}^{N=2^n} |i\rangle. \tag{2.2}$$

El objetivo del algoritmo de Grover es aumentar el valor del coeficiente que acompaña al estado $|\omega_0\rangle$ que queremos encontrar, es decir, aumentar la probabilidad de que al medir, obtengamos este estado. Como la suma de todas las probabilidades tiene que ser 1, este aumento en la probabilidad del estado $|\omega_0\rangle$ se produce a expensas de reducir la probabilidad del resto.

Nota

Recordemos que en mecánica cuántica podemos tener un estado que sea la **superposición** de varios estado (como es el caso anterior), pero al medir solo obtenemos **uno de estos estado**. La probabilidad de medir cada uno de estos estados que forman la superposición es igual al módulo cuadrado del coeficiente que lo acompaña en el vector de esto $|\Psi\rangle$ (el módulo cuadrado de la amplitud). Este caso, vemos que todos los estados tiene probabilidad $1/N$.

2.2. Amplificación de amplitud mediante iteraciones del algoritmo

Como vamos a ver a continuación, el algoritmo de Grover tiene una interpretación geométrica muy simple: dos reflexiones que rotan el vector de estado en un plano bidimensional.

El algoritmo de Grover consta de dos partes: un **oráculo** y un **difusor**. Para que el algoritmo maximice la probabilidad de la solución deseada tenemos realizar un número concreto de **iteraciones**. En cada iteración del algoritmo, primero se aplica el oráculo y después el difusor. Con cada iteración la probabilidad de medir la solución deseada va aumentando. Sin embargo, es muy importante aplicar el número correcto de iteraciones que nos maximiza la probabilidad, pues tanto si nos quedamos cortos como si nos pasamos, la probabilidad de medir $|\omega_0\rangle$ disminuye. Esto se va a entender muy bien a continuación, cuando veamos la explicación geométrica.

Pongamos el estado inicial de la Ec. (2.1) de una forma más adecuada para nuestro propósito

$$\boxed{|\Psi_0\rangle = |\Psi(k(0), l(0))\rangle = k(0)|\omega_0\rangle + \sum_{i\neq\omega_0} l(0)|i\rangle}, \quad \text{donde} \quad \boxed{k(0) = l(0) = \frac{1}{\sqrt{N}}}. \tag{2.3}$$

Denominaremos $k(t)$ y $l(t)$ a los coeficientes que tendremos en la t-esima iteración del algoritmo, es decir

$$\boxed{|\Psi(t)\rangle = |\Psi(k(t), l(t))\rangle = k(t)|\omega_0\rangle + \sum_{i\neq\omega_0}^{N} l(t)|i\rangle}. \tag{2.4}$$

Véase que aquí ya estamos adelantando que los coeficientes de todos los estados que no son el deseado son iguales en cada iteración. Nuestro vector de estado $|\Psi(t)\rangle$ vive en un espacio de Hilbert de N dimensiones y los estados $|i\rangle$, con $i = 0, 1, \ldots, N-1$, forman una base ortogonal del espacio de Hilbert (véase que aquí se incluye ω_0, pues no es nada más que un valor de i concreto). Para entender bien esto, podemos fijamos en las Ecs. (2.1) y (2.3) y ver que estas no son más que la expresión de un vector como la multiplicación de unos coeficientes por los elementos de la base[1]. Como no podemos dibujar un vectores de más de 3 dimensiones, para explicar geométricamente el algoritmo vamos a recurrir a un truco: vamos a descomponer nuestro vector de estado en la suma de dos vectores $|\omega_0\rangle$ y $|\omega^\perp\rangle$, donde este último es la suma de todos los demás elementos de la base, es decir:

$$\boxed{|\Psi(t)\rangle = |\Psi(k(t), l(t))\rangle = k(t)|\omega_0\rangle + l(t)\sqrt{N-1}|\omega^\perp\rangle}, \quad \text{donde} \quad \boxed{|\omega^\perp\rangle = \frac{1}{\sqrt{N-1}}\sum_{i\neq\omega_0}^{N}|i\rangle}. \tag{2.5}$$

Nota

Es importante darse cuenta de donde sale el factor $\sqrt{N-1}$ en la definición de $|\omega^\perp\rangle$. Este es debido a que estamos definiendo $|\omega^\perp\rangle$ como un vector unitario, es decir

$$\langle\omega^\perp|\omega^\perp\rangle = 1. \tag{2.6}$$

Por otro lado, $|\Psi_0\rangle$ también es unitario

$$\| \, |\Psi(t)\rangle \, \|^2 = \langle\Psi(t)|\Psi(t)\rangle = k(t)^2 + \sum_{i\neq\omega_0}^{N} l(t)^2 = k(t)^2 + l(t)^2(N-1) = 1. \tag{2.7}$$

Con lo cual, ese factor es necesario.

Teniendo en cuenta que $|\omega_0\rangle$ es un elemento de la base, eso quiere decir que es ortogonal al resto de elementos de la base. Concluimos entonces que $|\omega_0\rangle$ y $|\omega^\perp\rangle$ son ortogonales. Podemos pues dibujar nuestro vector de estado en un plano cuyos ejes son $|\omega_0\rangle$ y $|\omega^\perp\rangle$. La imagen de la izquierda de la Fig. 1 podemos ver el estado inicial $|\Psi_0\rangle$ dibujado en este plano.

[1] Esto es en esencia lo mismo que con vectores en tres dimensiones: $\vec{r} = a\hat{x} + b\hat{y} + c\hat{z}$

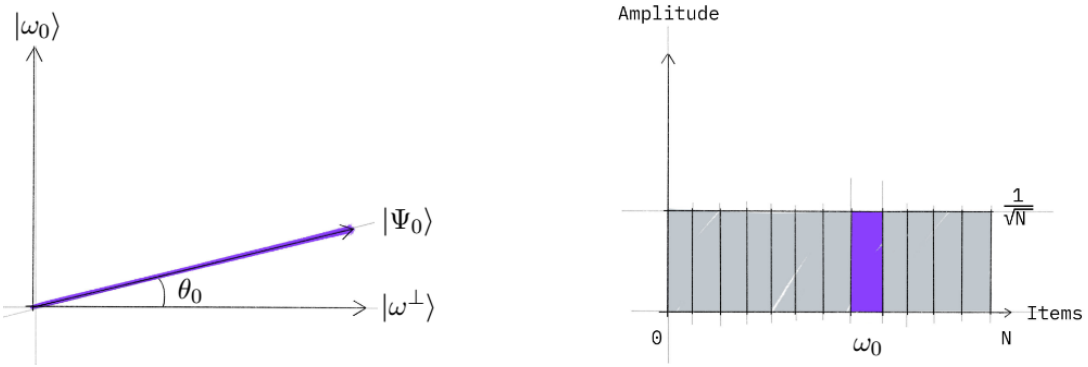


Figura 1: Esto inicial del algoritmo de Grover: superposición uniforme de los N estados. En la figura de la izquierda, $|\Psi_0\rangle$ representa el estado inicial, el eje $|\omega_0\rangle$ representa la solución y el eje $|\omega^\perp\rangle$ el resto de estados de espacio de Hilbert. En la figura de la derecha vemos la **amplitud** de cada estado. Recordemos que la probabilidad de cada estado es el cuadrado de la amplitud. Figura tomada de [1].

Nota

Veamos un ejemplo sencillo para ver a que nos referimos con esta descomposición. Pongamos que tenemos un vector tridimensional unitario de la forma

$$\vec{r} = a\vec{x} + b\vec{y} + c\vec{z}, \qquad \text{donde } a^2 + b^2 + c^2 = 1.$$

y donde $\vec{x}$, $\vec{y}$ y $\vec{z}$ son los vectores unitarios en las direcciones de los ejes X, Y y Z. Estos vectores unitarios son los **elementos de la base** que comentamos anteriormente (los equivalentes a los estados $|i\rangle$ en $\mathbb{R}^3$). Supongamos ahora que nuestra solución es $\vec{z}$. Lo que podemos hacer es juntar $a\vec{x} + b\vec{y}$ en un único vector unitario $\vec{v}$ que represente al plano formado por $\vec{x}$ y $\vec{y}$

$$\vec{v} = \frac{1}{\sqrt{a^2 + b^2}} \left(a\vec{x} + b\vec{y}\right).$$

Con lo que

$$\vec{r} = c\vec{z} + \sqrt{a^2 + b^2}\,\vec{v}.$$

De esta forma, al aumentar c el vector r se va poniendo cada vez más paralelo al eje Z, mientras que si aumentamos a o b el vector se acerca a $\vec{v}$, que representa el plano XY.

Sabemos también que todo vector en un plano podemos definirlo mediante su módulo y el ángulo respecto a uno de los ejes. Como el vector de estado tiene módulo uno (la suma de las probabilidades es uno), podemos escribirlo solo en función de un ángulo. Si llamamos θ_0 al ángulo que forman el vector de estado con el eje $|\omega^\perp\rangle$, podemos escribir

$$|\Psi_0\rangle = \sin\theta_0\,|\omega_0\rangle + \cos\theta_0\,|\omega^\perp\rangle, \qquad \text{donde} \qquad \boxed{\sin\theta_0 = \frac{1}{\sqrt{N}}}. \tag{2.8}$$

En el gráfico de barras de la derecha en la Fig. 1 podemos ver las amplitudes de los estados. Recordemos que la probabilidad de un estado es el cuadrado de la amplitud.

2.2.1. Primera parte de las iteraciones: El oráculo.

El algoritmo de Grover usa un **oráculo** cuya función es aplicar una fase negativa al estado que buscamos, es decir

$$\boxed{U_{\omega_0}|i\rangle = \begin{cases} |i\rangle & \text{si } i \neq \omega_0 \\ -|i\rangle & \text{si } i = \omega_0. \end{cases}} \tag{2.9}$$

Este oráculo no es más que una matriz diagonal con todo 1 en la diagonal menos en el elemento correspondiente al estado $|\omega_0\rangle$, donde tenemos un -1. Veremos más adelante ejemplos de como construir este tipo de oráculos. Podemos adelantar que, aunque no es trivial construirlos pues a priori parece que tenemos que conocer la solución de antemano, tampoco es (en muchos caso) excesivamente complicado.

Una de las características importantes de este algoritmo es lo fácil que resulta convertir un problema a un oráculo de esta forma. Hay muchos problemas computacionales en los que es difícil encontrar una solución, pero relativamente fácil verificarla (problemas NP). Por ejemplo, podemos verificar fácilmente la solución de un sudoku comprobando que se cumplen todas las reglas. Para estos problemas, podemos crear una función f que tome una propuesta de solución i y nos devuelva $f(i) = 0$ si i no es solución $(i \neq \omega_0)$ y $f(i) = 1$ si i es solución $(i \neq \omega_0)$. Podemos entonces definir el oráculo de la forma

$$\boxed{U_{\omega_0}|i\rangle = (-1)^{f(i)}|i\rangle}. \tag{2.10}$$

De esta forma, a matriz es ahora una matriz diagonal con

$$Diag(U_{\omega_0}) = \left[(-1)^{f(0)}, (-1)^{f(1)}, \ldots, (-1)^{f(2^n-1)}\right]. \tag{2.11}$$

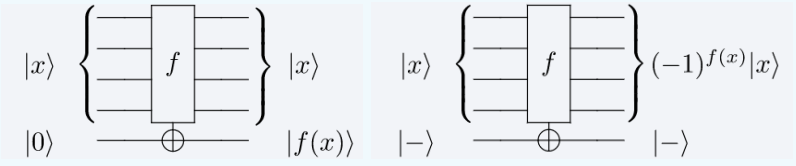
Nota

Podemos usar el retorno de fase (*phase kickback*) para construir este tipo de oráculos. Si tenemos nuestra función clásica $f(x)$, podemos convertirla en un circuito reversible de la forma (ver Fig. 2(a))

$$|x\rangle|0\rangle \to |x\rangle|0 \oplus f(x)\rangle = |x\rangle|f(x)\rangle.$$

Si inicializamos la ancilla en el estado $|-\rangle$, tenemos (ver Fig. 2(b))

$$|x\rangle|-\rangle = \frac{1}{\sqrt{2}}|x\rangle\left(|0\rangle - |1\rangle\right) \to |x\rangle\left(|0 \oplus f(x)\rangle - |1 \oplus f(x)\rangle\right) = (-1)^{f(x))}|x\rangle|-\rangle.$$



(a) (b)

Figura 2: Retorno de fase (*phase kickback*) para construir un oráculo de la forma de la Ec. (2.10).

En la Fig. 3 podemos ver el efecto del oráculo U_{ω_0} sobre el vector y las amplitudes. El cambio de signo en la amplitud del estado $|\omega_0\rangle$ se traduce en un cambio de signo en la proyección del vector de estado sobre este eje. A efectos prácticos, esto no es más que una reflexión del vector de estado respecto al eje $|\omega^{\perp}\rangle$. Como las probabilidades son el cuadrado de las amplitudes, este cambio de signo no se traduce en un cambio en la probabilidades. A efectos de las medidas, nada ha cambiado. En esta

figura también se representa la media de las amplitudes (linea punteada). Vemos que ha disminuido la media al cambiar el signo de la amplitud del estado $|\omega_0\rangle$.

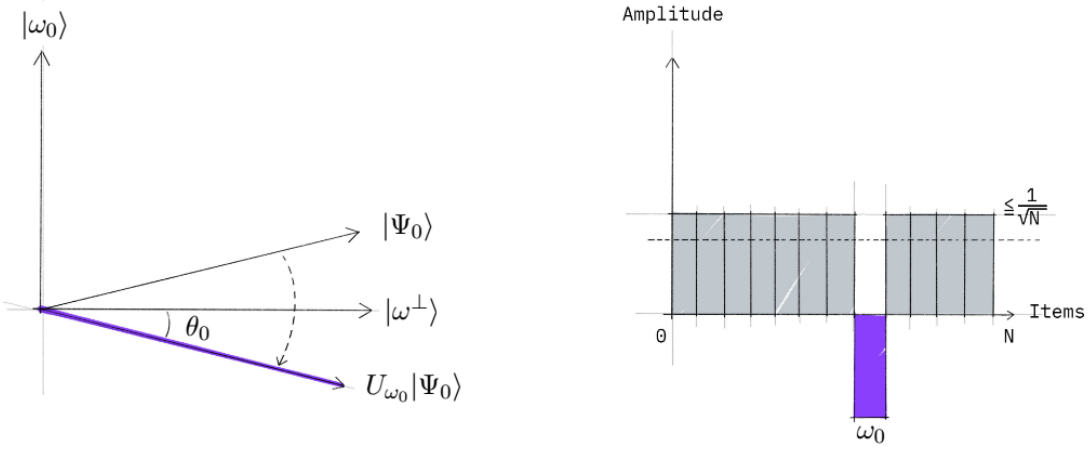


Figura 3: Primer paso del algoritmo de Grover: aplicación del **oráculo** U_{ω_0} para cambiar el signo de la amplitud del estado deseado. En la figura de la izquierda, $|\Psi_0\rangle$ representa el estado inicial, el eje $|\omega_0\rangle$ representa la solución y el eje $|\omega^\perp\rangle$ el resto de estados de espacio de Hilbert. En la figura de la derecha vemos la **amplitud** de cada estado, donde la línea punteada representa la media. Figura tomada de [1].

2.2.2. Segunda parte de las iteraciones: El difusor.

El **difusor** consiste en aplicar el operador

$$\boxed{U_{\Psi_0} = 2|\Psi_0\rangle\langle\Psi_0| - I}. \tag{2.12}$$

Este operador no es más que una reflexión respecto el esto inicial $|\Psi_0\rangle$.

En la Fig. 4 podemos ver el efecto del difusor. En el diagrama de barras de la amplitud, podemos entender esta transformación como una reflexión respecto a la media de las amplitudes (la media queda igual). Como habíamos disminuido la media al aplicar el oráculo, lo que tenemos ahora es una amplificación de la amplitud del estado deseado. Esto se ve también en el plano de la izquierda, pues las amplitudes no son más que las proyecciones del vector sobre los ejes.

Lo que estamos haciendo mediante la aplicación del oráculo y el difusor no es mas que **rotar el vector de estado un ángulo $2\theta_0$** (donde θ_0 está definido en la ec. (2.8)) hacia el eje que representa nuestra solución, aumentando así su proyección, es decir, su amplitud, y con ello la probabilidad de medirlo.

Después de t iteraciones hemos aumentado el ángulo en $2t\theta_0$, con lo que tenemos estado:

$$\boxed{|\Psi(t)\rangle = (U_{\Psi_0}U_{\omega_0})^t\,|\Psi_0\rangle} = \sin\left((2t+1)\theta_0\right)\,|\omega_0\rangle + \cos\left((2t+1)\theta_0\right)\,|\omega^\perp\rangle. \tag{2.13}$$

Podemos ahora definir el **operador de Grover**

$$\boxed{G = U_{\Psi_0}U_{\omega_0}} \quad \Rightarrow \quad |\Psi(t)\rangle = G^t|\Psi_0\rangle. \tag{2.14}$$

Ahora es fácil entender porqué tenemos que aplicar un número concreto de iteraciones y porqué si nos pasamos, la probabilidad disminuye. Como acabamos de ver, el resultado después de cada iteración es que rotamos el vector de estado $2\theta_0$ en el sentido contrario de las agujas del reloj. Lo que queremos

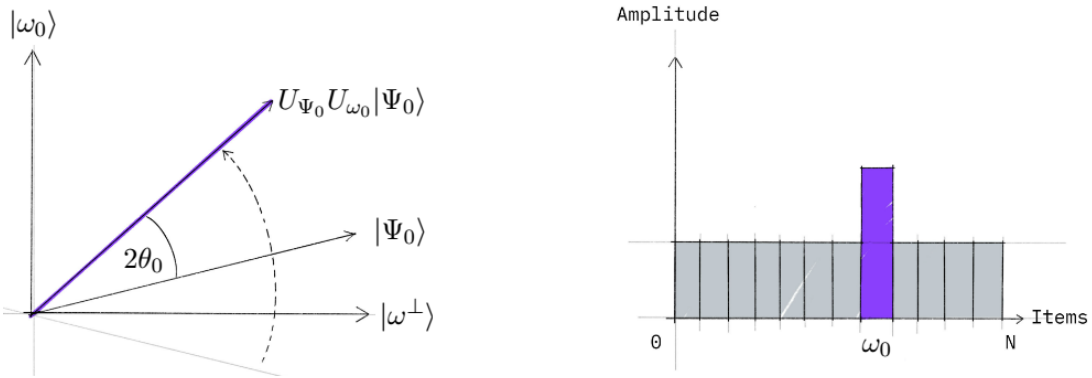


Figura 4: Segundo paso del algoritmo de Grover: aplicar el operador de **difusión** (o reflexión) $U_{\Psi_0} = 2|\Psi_0\rangle\langle\Psi_0| - I$. En la figura de la izquierda, $|\Psi_0\rangle$ representa el estado inicial, el eje $|\omega_0\rangle$ representa la solución y el eje $|\omega^\perp\rangle$ el resto de estados de espacio de Hilbert. En la figura de la derecha vemos la **amplitud** de cada estado. Recordemos que la probabilidad de cada estado es el cuadrado de la amplitud. Figura tomada de [1].

es que el vector quede lo más vertical posible, es decir, que quede lo más cerca posible del eje $|\omega\rangle$. Si hacemos demasiadas iteraciones, lo que vamos a conseguir es "pasarnos de largo" del eje. Discutiremos el número exacto de iteraciones en la sección 3, pero ya comentamos que es del orden de $\sqrt{N}$.

Si comparamos las ecuaciones (2.5) y (2.13) vemos que

$$\boxed{k(t) = \sin\Big[(2t+1)\theta_0\Big]} \qquad \boxed{l(t) = \frac{1}{\sqrt{N-1}}\cos\Big[(2t+1)\theta_0\Big]}. \tag{2.15}$$

3. Número conocido de soluciones.

Vamos a empezar el análisis formal tratando el caso en el que **conocemos el número M de soluciones** que hay en nuestro dataset. Es decir, tenemos M valores diferentes i que cumplen $L_i = x$. Denominemos ω al conjunto de los M valores i que son solución, y denominemos $\omega^\perp$ al conjunto de los $N - M$ valores i que no son solución

$$\boxed{\omega = \{i | L_i = x\}} \qquad \boxed{\omega^\perp = \{i | L_i \neq x\}}. \tag{3.1}$$

Supondremos también que **estamos en el caso en el que** $N = 2^n$ y que además partimos del estado de la Ec. (2.1), es decir, de una **superposición uniforme**.

Nota Importante!!

Uno de los pasos del algoritmo de Grover es una reflexión de las amplitudes respecto a la media. Esto implica que podemos tener los siguientes casos:

- $M < N/2$: El algoritmo funciona normal.
- $M = N/2$: El algoritmo no funciona
- $M > N/2$: El algoritmo amplifica las soluciones incorrectas.

3.1. Generalización de las expresiones de la sección 2 para M soluciones.

Vemos a reescribir las ecuaciones enmarcadas de la sección 2 para el caso de M soluciones. Empecemos reescribiendo las expresiones del estado inicial de la Ec. (2.3) y del estado $|\Psi_j\rangle$ de la Ec. (2.13)

$$\boxed{|\Psi_0\rangle = |\Psi(k(0), l(0))\rangle = \sum_{i\in\omega} k(0)|i\rangle + \sum_{i\in\omega^\perp} l(0)|i\rangle}, \quad \text{donde} \quad \boxed{k(0) = l(0) = \frac{1}{\sqrt{N}}}. \tag{3.2}$$

$$\boxed{|\Psi(t)\rangle = |\Psi(k(t), l(t))\rangle = \sum_{i\in\omega} k(t)|i\rangle + \sum_{i\in\omega^\perp} l(t)|i\rangle}. \tag{3.3}$$

Vemos que ahora el primer sumatorio tiene M elementos, mientras que el segundo tiene $N - M$. Por supuesto, se cumple que $|\Psi(t)\rangle$ tiene módulo 1, es decir

$$\langle\Psi(t)|\Psi(t)\rangle = \boxed{Mk(t)^2 + (N - M)l(t)^2 = 1 \ \forall t}. \tag{3.4}$$

Podemos ahora también redefinir el estado $|\omega^\perp\rangle$ de la Ec. (2.5) y definir $|\omega\rangle$

$$\boxed{|\omega^\perp\rangle = \frac{1}{\sqrt{N - M}} \sum_{i\in\omega^\perp} |i\rangle} \qquad \boxed{|\omega\rangle = \frac{1}{\sqrt{M}} \sum_{i\in\omega} |i\rangle} \tag{3.5}$$

de forma que

$$\boxed{|\Psi(t)\rangle = k(t)\sqrt{M}|\omega\rangle + l(t)\sqrt{N - M}|\omega^\perp\rangle}. \tag{3.6}$$

Nota

Nuevamente, los factores $\sqrt{N - M}$ y $\sqrt{M}$ en las definiciones de $|\omega^\perp\rangle$ y $|\omega\rangle$ son para hacerlos vectores unitarios

$$\langle\omega^\perp|\omega^\perp\rangle = 1 \qquad \langle\omega|\omega\rangle = 1. \tag{3.7}$$

Veamos ahora la expresión del esto inicial en función del ángulo θ de la Ec. (2.8)

$$|\Psi_0\rangle = \sin\theta\,|\omega\rangle + \cos\theta\,|\omega^{\perp}\rangle, \qquad \text{donde} \qquad \boxed{\sin\theta = \sqrt{\frac{M}{N}}}. \tag{3.8}$$

Nuevamente, comparando las Ecs. (3.8) y (3.6) obtenemos las expresiones de $k(t)$ y $l(t)$ en función del ángulo θ

$$\boxed{k(t) = \frac{1}{\sqrt{M}}\sin\Big[(2t+1)\theta\Big]} \qquad \boxed{l(t) = \frac{1}{\sqrt{N-M}}\cos\Big[(2t+1)\theta\Big]}. \tag{3.9}$$

Vemos que estas expresiones cumplen la Ec. (3.4) Finalmente, por ser rigurosos (y por recopilar todas las ecuaciones juntas), tenemos que el oráculo (2.9), el difusor (2.12) nos quedan:

$$\boxed{U_\omega|i\rangle = \begin{cases} |i\rangle & \text{si } i \notin \omega \\ -|i\rangle & \text{si } i \in \omega \end{cases}} \qquad \boxed{U_{\Psi_0} = 2|\Psi_0\rangle\langle\Psi_0| - I}. \tag{3.10}$$

donde recordemos que denominamos operador de Grover a

$$\boxed{G = U_{\Psi_0}U_\omega} \quad \Rightarrow \quad |\Psi(t)\rangle = G^t|\Psi_0\rangle. \tag{3.11}$$

Véase que el caso particular de una solución es, efectivamente, aquel con $M = 1$.

3.2. Número de iteraciones.

Denominemos $\boldsymbol{T}$ al **número de iteraciones para el cual la probabilidad de medir la solución correcta se maximiza**. Es decir, $l(T) \approx 0$ y $k(T) \approx 1$. Sabemos que $l(\tilde{T}) = 0$ cuando el coseno es igual cero, es decir:

$$l(\tilde{T}) = 0 \quad \Rightarrow \quad (2\tilde{T}+1)\theta = \frac{\pi}{2} \quad \Rightarrow \quad \tilde{T} = \frac{\pi}{4\theta} - \frac{1}{2}. \tag{3.12}$$

Por regla general, este valor $\tilde{T}$ no será un entero. Tomemos

$$T = \lfloor \pi/4\theta \rfloor \tag{3.13}$$

donde la notación $\lfloor\alpha\rfloor$ quiere decir que **redondeamos a un entero por truncamiento**. Véase que $|m - \tilde{m}| \leq 1/2$. Se sigue que $|(2T+1)\theta - (2\tilde{T}+1)\theta| \leq \theta$. Pero $(2\tilde{T}+1)\theta = \pi/2$ por definición de $\tilde{T}$. Con lo cual $\Big|\cos\Big((2T+1)\theta\Big)\Big| \leq |\sin\theta|$. Concluimos entonces que la probabilidad de fallo después de $T = \lfloor\pi/4\theta\rfloor$ es

$$(N-M)l^2(T) = \cos^2\Big((2T+1)\theta\Big) \leq \sin^2\theta = \frac{M}{N} \tag{3.14}$$

que es despreciable si $M \ll N$. Véase que el $(N-M)$ de la expresión anterior es porque hay $(N-M)$ estados incorrectos que podemos medir, cada uno de ellos con probabilidad $l^2(T)$.

Véase que el algoritmo corre en un tiempo del orden de $\mathcal{O}(\sqrt{N/M})$ ya que

$$T \leq \frac{\pi}{4\theta} \approx \frac{\pi}{4\sin\theta} \approx \frac{\pi}{4}\sqrt{\frac{N}{M}} \quad \Rightarrow \quad \boxed{T = \left\lfloor \frac{\pi}{4}\sqrt{\frac{N}{M}} \right\rfloor}. \tag{3.15}$$

3.3. Extra: Formulación recursiva de $k(t)$ y $l(t)$.

Ya hemos comentado en cada iteración se aplican lo dos operadores de la Ec. (3.10), es decir

$$|\Psi(t+1)\rangle = U_{\Psi_0} U_\omega |\Psi(t)\rangle. \tag{3.16}$$

Desarrollemos esta expresión para ver la relación de recursivida de los coeficientes, es decir, para ver la expresión de $k(t+1)$ y $l(t+1)$ en función de $k(t)$ y $l(t)$. Primero vamos el término $U_\omega|\Psi(t)\rangle$ usando la expresión de U_ω de la Ec. (3.10):

$$U_\omega|\Psi(t)\rangle = -k(t)\sqrt{M}|\omega\rangle + l(t)\sqrt{N-M}|\omega^\perp\rangle. \tag{3.17}$$

Desarrollemos un poco la Ec. (3.16) usando la expresión de U_{Ψ_0} de la Ec. (3.10)

$$\begin{aligned} |\Psi(t+1)\rangle &= U_{\Psi_0} U_\omega |\Psi(t)\rangle \\ &= \Big(2|\Psi_0\rangle\langle\Psi_0| - I\Big) U_\omega |\Psi(t)\rangle \\ &= 2|\Psi_0\rangle\langle\Psi_0|U_\omega|\Psi(t)\rangle - U_\omega|\Psi(t)\rangle. \end{aligned} \tag{3.18}$$

Usando la Ec. (3.17) el término $\langle\Psi_0|U_\omega|\Psi(t)\rangle$ nos queda

$$\langle\Psi_0|U_\omega|\Psi(t)\rangle = \frac{1}{\sqrt{N}}\Big(-k(t)M + (N-M)l(t)\Big). \tag{3.19}$$

Calculo de $\langle\Psi_0|U_\omega|\Psi(t)\rangle$

Hacemos ahora el producto $\langle\Psi_0|U_\omega|\Psi(t)\rangle = \langle\Psi_0|\Big(U_\omega|\Psi(t)\rangle\Big)$ usando la Ec. (3.17)

$$\begin{aligned} \langle\Psi_0|U_\omega|\Psi(t)\rangle &= \langle\Psi_0|\Big(-k(t)\sqrt{M}|\omega\rangle + l(t)\sqrt{N-M}|\omega^\perp\rangle\Big) \\ &= \Big(k(0)\sqrt{M}\langle\omega| + l(0)\sqrt{N-M}\langle\omega^\perp|\Big)\Big(-k(t)\sqrt{M}|\omega\rangle + l(t)\sqrt{N-M}|\omega^\perp\rangle\Big). \end{aligned}$$

Debido a la ortogonalidad de los estado (esto es, $\langle i|j\rangle = 0$ si $j \neq i$), y teniendo en cuanta que $k(0) = l(0) = 1/\sqrt{N}$ tenemos

$$\langle\Psi_0|U_\omega|\Psi(t)\rangle = -k(0)k(t)M + (N-M)l(0)l(t) = \frac{1}{\sqrt{N}}\Big(-k(t)M + (N-M)l(t)\Big).$$

Sustituyendo (3.17) y (3.19) en (3.18) tenemos:

$$|\Psi(t+1)\rangle = \left(\frac{N-2M}{N}k(t) + \frac{2(N-M)}{N}l(t)\right)\sum_{i\in\omega}|i\rangle + \left(\frac{N-2M}{N}l(t) - \frac{2M}{N}k(t)\right)\sum_{i\in\omega^\perp}|i\rangle. \tag{3.20}$$

Calculo de la Ec. (3.20)

Sustituyendo (3.17) y (3.19) en (3.18) tenemos:

$$\begin{aligned}
|\Psi(t+1)\rangle &= 2\frac{1}{\sqrt{N}}\Big(-k(t)M + (N-M)l(t)\Big)|\Psi_0\rangle - k(t)\sqrt{M}|\omega\rangle + l(t)\sqrt{N-M}|\omega^{\perp}\rangle = \\
&= \frac{2}{\sqrt{N}}\Big(-k(t)M + (N-M)l(t)\Big)\frac{1}{\sqrt{N}}\Big(\sqrt{M}|\omega\rangle + \sqrt{N-M}|\omega^{\perp}\rangle\Big) \\
&\quad + k(t)\sqrt{M}|\omega\rangle + l(t)\sqrt{N-M}|\omega^{\perp}\rangle = \\
&= \left[\frac{2}{N}\Big(-k(t)M + (N-M)l(t)\Big) + k(t)\right]\sqrt{M}|\omega\rangle \\
&\quad + \left[\frac{2}{N}\Big(-k(t)M + (N-M)l(t)\Big) - l(t)\right]\sqrt{N-M}|\omega^{\perp}\rangle.
\end{aligned}$$

Simplificando y teniendo en cuenta que las expresión de $|\omega\rangle$ y $|\omega^{\perp}\rangle$ en la Ec. (3.5), llegamos a la Ec. (3.20).

Finalmente llegamos a las relaciones:

$$\boxed{k(t+1) = \frac{N-2M}{N}k(t) + \frac{2(N-M)}{N}l(t)}, \qquad \boxed{l(t+1) = \frac{N-2M}{N}l(t) - \frac{2M}{N}k(t)}. \tag{3.21}$$

4. Número desconocido de soluciones

Como ya hemos comentado, para poder aplicar el algoritmo de Grover tal y como lo hemos visto hasta ahora, nos hace falta conocer el número de soluciones. Esto es debido a que debemos aplicar un número concreto de iteraciones para maximizar la probabilidad de las soluciones correctas. Si no aplicamos el número correcto de iteraciones, la probabilidad puede ser incluso nula. Como vemos en la Ec. (3.15), para saber el número de iteraciones tenemos que conocer el número de soluciones.

En esta sección vamos a ver un algoritmo para poder abordar el caso en el que no conocemos el número de soluciones y por ende, no sabemos cuantas iteraciones debemos aplicar. Este algoritmo no es más que una forma inteligente de aplicar el algoritmo de Grover. Lo bueno de este algoritmo es que nuevamente podemos encontrar una solución con un número de iteraciones $\mathcal{O}(\sqrt{N/M})$, aunque a priori no sabemos cuantas iteraciones son. Vamos a ver primero dos Lemmas que nos servirán para entender el algoritmo que se plantea en el Teorema 1.

Nota Importante!!

Uno de los pasos del algoritmo de Grover es una reflexión de las amplitudes respecto a la media. Esto implica que podemos tener los siguientes casos:

- $M < N/2$: El algoritmo funciona normal.
- $M = N/2$: El algoritmo no funciona
- $M > N/2$: El algoritmo amplifica las soluciones incorrectas.

4.1. Conocimientos previos.

Lemma 1 *Para cualquier par de números α y β, y cualquier posible entero r tenemos*

$$\sum_{j=0}^{r-1} \cos\left(\alpha + 2\beta j\right) = \frac{\sin\left(r\beta\right)\cos(\alpha + (r-1)\beta)}{\sin\beta}. \tag{4.1}$$

En particular, si $\alpha = \beta$,

$$\sum_{j=0}^{r-1} \cos\left((2j+1)\,\alpha\right) = \frac{\sin\left(2r\alpha\right)}{2\sin\alpha}. \tag{4.2}$$

Lemma 2 *Sea M el número (desconocido) de soluciones de un problema tipo Grover con N elementos y sea θ tal que $\sin^2\theta = M/N$. Sea r un entero positivo aleatorio. Sea t un entero aleatorio (con distribución uniforme) entre 0 y $r-1$. Si observamos el registro después de t iteraciones del algoritmo de Grover empezando desde el estado $|\Psi_0\rangle = \sum_i \frac{1}{\sqrt{N}}|i\rangle$, la probabilidad de obtener la solución correcta es exactamente*

$$P_r = \frac{1}{2} - \frac{\sin(4r\theta)}{4r\sin(2\theta)}. \tag{4.3}$$

En particular, tenemos $P_r \geq 1/4$ si $r \geq 1/\sin(2\theta)$.

Demostración: Como ya sabemos, si tenemos M soluciones la probabilidad de que tras t iteraciones se mida el resultado correcto es M veces el valor del coeficiente $k^2(t)$ (la amplitud al cuadrado del estado $i \in \omega$ en la Ec. (3.16)). Teniendo en cuenta la Ec. (3.21) sabemos que la probabilidad es

$$P(t) = Mk^2(t) = \sin^2((2t+1)\theta).$$

De esto se sigue que la probabilidad promedio si tomamos un número de iteraciones t tal que $0 \leq t < r$ es

$$P_r = \frac{1}{r}\sum_{t=0}^{r-1} P(t) = \frac{1}{r}\sum_{t=0}^{r-1} \sin^2((2t+1)\theta) = \frac{1}{2r}\sum_{t=0}^{r-1}\Big(1 - \cos((2t+1)\theta)\Big) = \frac{1}{2} - \frac{\sin(4r\theta)}{4r\,\sin(2\theta)},$$

donde en la última igualdad se ha usado el Lemma 1. Si estamos en el caso de $r \geq 1/\sin(2\theta)$ tenemos

$$\frac{\sin(4r\theta)}{4r\,\sin(2\theta)} \leq \frac{1}{4r\,\sin(2\theta)} \leq \frac{1}{4} \quad \Rightarrow \quad P_r \geq \frac{1}{2} - \frac{1}{4},$$

donde la primera desigualdad se cumple siempre (independientemente del valor de r) pues viene de que la función seno está acotada entre -1 y 1. ■

Nota

A veces al ver las formulaciones tan formales que se dan en los teoremas o lemmas, perdemos un poco el norte sobre lo que nos quieren transmitir. Veámoslos en palabras más llanas:

- El Lemma 1 simplemente es una relación trigonométrica un poco exótica y nos sirve para demostrar el Lemma 2
- El Lemma 2 nos da la expresión de la probabilidad promedio que tenemos de medir el resultado correcto usando el algoritmo de Grover en un caso particular. En este caso lo que hacemos es, dado un número r, ejecutar t iteraciones de Grover donde t es un número aleatorio entre 0 y $r-1$. Probando muchas veces con muchos número aleatorio t, en promedio la probabilidad P_r de obtener el resultado correcto está dada por (4.3). Véase que, para un problema de Grover concreto (θ fijo), esta probabilidad depende solo de r.

4.2. Algoritmo para el caso de M desconocido.

Teorema 1 *El siguiente algoritmo encuentra una solución en un tiempo esperado $\mathcal{O}(\sqrt{N/M})$ (si tomamos $1 \leq M \leq 3N/4$)*

1. *Inicializamos $r = 1$ y $\lambda = 6/5$. (En realidad, cualquier valor $1 \leq \lambda \leq 4/3$ sirve).*
2. *Elegimos un valor aleatorio t con distribución uniforme tal que $0 \leq t < r$.*
3. *Aplicamos t iteraciones del algoritmo de Grover empezando desde el estado inicial $|\Psi_0\rangle = \sum_i \frac{1}{\sqrt{N}}|i\rangle$.*
4. *Medimos, de forma que obtenemos uno de los estados $|i\rangle$ del paso anterior.*
5. *Si $L_i = x$, hemos encontrado una solución: **Exit**.*
6. *En caso contrario, tomemos $r = min(\lambda r, \sqrt{N})$ y volvemos al paso 2.*

4.3. Demostración del Teorema 1.

Demostración: Sea θ un ángulo tal que

$$\sin\theta = \sqrt{N/M}, \tag{4.4}$$

(como en la Ec. (3.8)). Denominemos r_s al valor de r en la s-esima iteración del bucle principal, es decir

$$r_s = \lambda^{s-1}, \qquad \text{con } s = 1, 2, \ldots \qquad \text{y } \lambda = \frac{4}{3}. \tag{4.5}$$

(No confundir las **iteraciones del bucle principal** con las **iteraciones de Grover**: en cada iteración s del bucle principal hacemos t iteraciones de Grover.) Sea r_0 tal que

$$r_0 = \frac{1}{\sin(2\theta)}. \tag{4.6}$$

(Véase que r_0 no es valor inicial de r_s). Aplicando un poco de trigonometria

$$\sin^2\theta = \frac{1-\cos(2\theta)}{2} \quad \Rightarrow \quad 1-2\sin^2\theta = \cos(2\theta) = \sqrt{1-\sin^2(2\theta)} \quad \Rightarrow \quad \sin^2(2\theta) = 1-\Big(1-2\sin^2\theta\Big)^2.$$

Desarrollando el cuadrado y teniendo en cuenta la Ec. (4.4) llegamos a

$$r_0 = \frac{1}{\sin(2\theta)} = \frac{N}{2\sqrt{(N-M)M}} < \sqrt{\frac{N}{M}}. \tag{4.7}$$

donde en la desigualdad se ha tenido en cuenta que $M \leq 3N/4$. Veámoslo

$$\frac{N}{2\sqrt{(N-M)M}} < \sqrt{\frac{N}{M}} \quad \Rightarrow \quad \frac{\sqrt{N}}{2\sqrt{(N-M)}} < 1 \quad \Rightarrow \quad \frac{N}{(N-M)} < 4 \quad \Rightarrow \quad 1-\frac{1}{4} > \frac{M}{N}.$$

Debemos estimar el número esperado total de iteraciones de Grover que tenemos que hacer (sumando todas las interaciones de Grover en todas las iteraciones del bucle principal). El tiempo total será del orden de este número. Sabemos que **el número promedio de iteraciones de Grover que se realizan en la iteración s-esima del bucle principal es menor que $r_s/2$**. Escribiéndolo de forma bonita:

$$\bar{t}_s < \frac{r_s}{2} = \frac{\lambda^{s-1}}{2} \qquad \text{donde la barra significa } \textit{promedio}. \tag{4.8}$$

Esto es debido a que en cada iteración del bucle principal hacemos un número aleatorio de iteraciones t_s de Grover con $0 \leq t_s < r_s$. Denominamos **fase crítica** al momento en el que hemos realizado suficientes iteraciones del bucle principal como para tener $r_s \geq r_0$, es decir

$$r_s > r_0 \quad \Rightarrow \quad \lambda^{s-1} > r_0 \quad \Rightarrow \quad s > 1+\log_\lambda r_0 \quad \Rightarrow \quad s > \lceil \log_\lambda r_0 \rceil. \tag{4.9}$$

con lo cual, **estamos en la fase crítica cuando el número de iteraciones del bucle principal es mayor que $\lceil \log_{\boldsymbol{\lambda}} \boldsymbol{r_0} \rceil$** (donde esta notación significa *redondear hacia el siguiente entero*). Si nos fijamos en el Lemma 2, esta fase corresponde a aquella donde $P_r > 1/4$.

Teniendo en cuenta la Ec. (4.8), vemos que el número promedio total de iteraciones de Grover que se realizan antes de llegar a la fase crítica es

$$\begin{aligned} \sum_{s=1}^{\lceil \log_\lambda r_0 \rceil} \bar{t}_s = \frac{1}{2} \sum_{s=1}^{\lceil \log_\lambda r_0 \rceil} \lambda^{s-1} = \frac{1}{2} \sum_{\tilde{s}=0}^{\lceil \log_\lambda r_0 \rceil - 1} \lambda^{\tilde{s}} = \frac{1}{2}\frac{\lambda^{\lceil \log_\lambda r_0 \rceil}-1}{\lambda-1} < \\ < \frac{1}{2}\frac{\lambda^{1+\log_\lambda r_0}-1}{\lambda-1} = \frac{1}{2}\frac{r_0\lambda-1}{\lambda-1} < \frac{1}{2}\frac{\lambda}{\lambda-1} r_0 = \boxed{3r_0}. \end{aligned} \tag{4.10}$$

donde se ha aplicado la formula e la suma de la serie geométrica[2]. Vemos pues que el algoritmo llega a la fase crítica en un tiempo $\mathcal{O}(r_0) < \mathcal{O}(\sqrt{N/M})$.

Como ya comentamos, una vez alcanzado el estado crítico, por el Lemma 2, sabemos que en cada nueva iteración del bucle principal la **probabilidad de éxito $\boldsymbol{P_r \geq 1/4}$** ya que $r_s > 1/\sin(2\theta)$. De ello se deduce que el número esperado de iteraciones de Grover necesarias para tener éxito una vez alcanzada la fase crítica está acotado superiormente por

$$\sum_{\tilde{u}=1}^{\infty} (1-P_r)^{\tilde{u}-1} P_r\, \bar{t}_{\tilde{u}+1+\lceil \log_\lambda r_0 \rceil} < \frac{1}{2}\sum_{\tilde{u}=1}^{\infty} \frac{3^{\tilde{u}-1}}{4^{\tilde{u}-1}} \frac{1}{4} \lambda^{\tilde{u}+1+\lceil \log_\lambda r_0 \rceil}, \tag{4.12}$$

[2]Suma de la serie geométrica:

$$\sum_{k=0}^{n} r^k = \frac{1-r^{n+1}}{1-r} = \frac{r^{n+1}-1}{r-1}. \tag{4.11}$$

donde $(1-P_r)^{\tilde{u}-1}$ es la probabilidad de fallar $\tilde{u}-1$, con lo que $(1-P_r)^{\tilde{u}-1}P_r$ es la probabilidad de haber fallado $\tilde{u}-1$ veces y acertar en la $\tilde{u}$-esima. $\bar{t}_s$ viene dada por (4.8). Véase que para acotar las iteraciones nos hemos puesto en el peor caso, en el que la probabilidad de exito es la mínima y no aumenta ($P_r = 1/4$). Podemos hacer al cambio de variable $u = \tilde{u} - 1$ para poner la expresión anterior más acorde para realizar la suma de la serie geométrica (ver Ec. (4.11))

$$\frac{1}{2}\sum_{u=0}^{\infty}\frac{3^u}{4^u}\frac{1}{4}\lambda^{u+\lceil\log_\lambda r_0\rceil} < \frac{1}{2}\frac{1}{4}\lambda^{1+\log_\lambda r_0}\sum_{u=0}^{\infty}\frac{3^u}{4^u}\lambda^u = \frac{r_0\lambda}{8}\frac{1}{1-3\lambda/4} = \frac{\lambda}{8-6\lambda}r_0 = \boxed{\frac{3}{2}r_0}. \tag{4.13}$$

El número total esperado de iteraciones de Grover, en caso de que se alcance la fase crítica, está, por tanto, limitado por la suma de (4.10) más (4.13), es decir,

$$T = r_0 + \frac{3}{2}r_0 = \frac{9}{2}r_0 < \frac{9}{2}\sqrt{\frac{M}{N}}, \tag{4.14}$$

con lo que el tiempo esperado total es $\mathcal{O}(\sqrt{N/M})$ siempre que $0 < M \leq 3N/4$. Véase que $\frac{9}{2}r_0 \approx \frac{9}{2}\sqrt{N/M}$ cuando $M \ll N$, que es menos de 4 veces el tiempo esperado en el caso de conocer M de antemano (ver Ec. (3.15)). El caso $M > 3N/4$ puede resolverse en tiempo esperado constante mediante muestreo clásico. El caso $M = 0$ se maneja mediante un tiempo de espera apropiado en el algoritmo anterior, que permite afirmar en un tiempo en $\mathcal{O}(\sqrt{N})$ que no hay soluciones cuando este es el caso, con una probabilidad de fallo arbitrariamente pequeña cuando de hecho hay una solución. ■

5. Conteo de soluciones (Quantum counting)

En esta sección vamos a ver un algoritmo para obtener el número M de soluciones. Este algoritmo puede decirse que en cierta medida está inspirado en el algoritmo de Shor, pues lo que vamos a hacer es usar el **algoritmo de estimación de fase cuántico (QPE)** para obtener el ángulo θ a partir del operador de Grover G. Después, podemos usar la Ec. (3.8) para obtener M partir de θ.

Vamos a seguir en el caso en el que $N = 2^n$ y la distribución de probabilidad inicial es uniforme.

5.1. Breve resumen de la estimación de fase cuántica (QFE).

Dado un operador unitario U y un autovector $|\psi\rangle$ del mismo, tenemos:

$$U\,|\psi\rangle = e^{2\pi i\alpha}\,|\psi\rangle$$

El **algoritmo de estimación de fase cuántica** lo que hace es calcular un valor aproximado del ángulo α. En la Fig. 5 podemos ver el circuito que implementa este algoritmo. No vamos a entrar a hablar en detalle del mismo (puede verse una explicación más detallada en las notas del algoritmo de Shor). Simplemente comentar dos cosas:

- Por el registro de qubits de abajo en la Fig. 5 debe de entrar el autoestado de U del cual queremos medir la fase.
- Si por el registro de conteo entran p qubits (el de arriba en la Fig. 5), en la salida vamos a medir el estado $|2^p\alpha\rangle$.

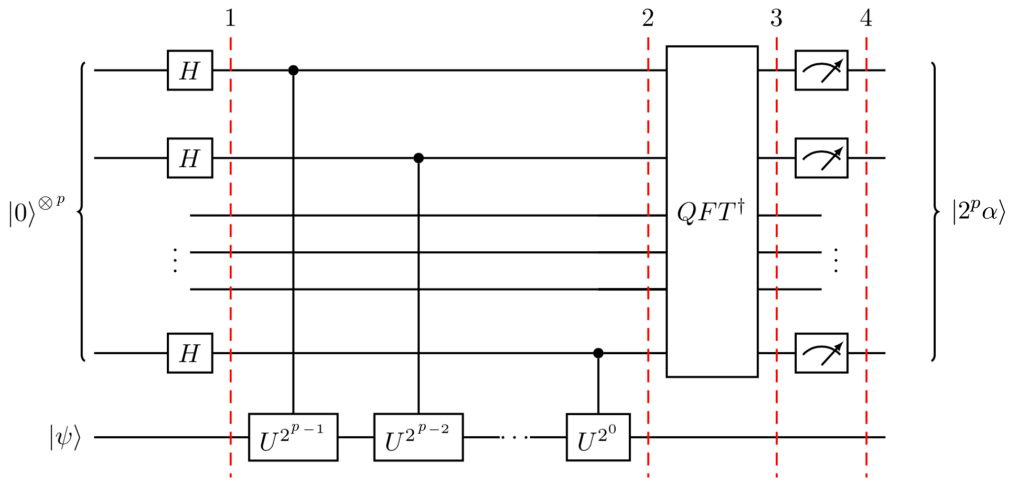


Figura 5: Implementación del algoritmo de estimación de fase cuántica (en el convenio estándar, siendo el bit más significativo el de arriba).

5.2. Estimación de fase con el operador de Grover.

Como ya comentamos anteriormente, el operador de Grover G (ver Ec. (3.11)) rota el vector de estado en un ángulo 2θ, donde θ está dado por (3.8), es decir

$$|\Psi(t+1)\rangle = G|\Psi(t)\rangle = e^{i2\theta}|\Psi(t)\rangle. \tag{5.1}$$

En concreto, se puede aplicar sobre el estado inicial

$$\boxed{G|\Psi_0\rangle = e^{i2\theta}|\Psi_0\rangle}, \qquad \text{donde} \qquad |\Psi_0\rangle = H^{\otimes n}|0\rangle = \frac{1}{2^n}\sum_{i=0}^{2^n}|i\rangle \tag{5.2}$$

Podemos pues usar el algoritmo de QFE poniendo el registro de abajo en el estado $|\Psi_0\rangle$ y mediremos a la salida el estado $|2^p 2\theta/2\pi\rangle = |2^p\theta/\pi\rangle$. Podemos ver esto en la Fig. 6.

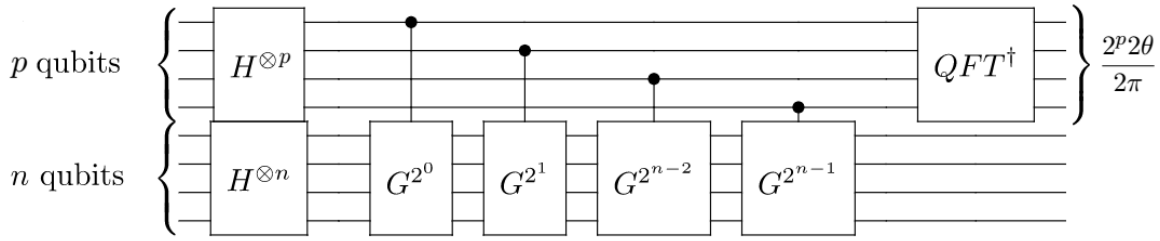


Figura 6: Circuito para la QFE en el caso de Grover.

Como acabamos de comentar, al aplicar QFE mediremos un valor $\tilde{f} = 2^p\tilde{\theta}/\pi$. Podemos despejar $\tilde{\theta}$ y usar (3.8) para calcular $\tilde{M}$

$$\tilde{\theta} = \frac{\tilde{f}\pi}{2^p} \quad \Rightarrow \quad \tilde{M} = 2^n \sin^2\tilde{\theta} = 2^n \sin^2\frac{\tilde{f}\pi}{2^p}, \tag{5.3}$$

donde las $\tilde{\theta}$, $\tilde{f}$ y $\tilde{M}$ llevan una tilde para diferenciarlos de los valores reales. Esto es, porque dependiendo del como de grande sea p (cuantos qubits tengamos en el registro de conteo) nos acercaremos más a medir el valor real del ángulo θ.

Nota

Como veremos en la sección 6.1, muchas veces en vez de implementar el operador de difusión U_{Ψ_0} (ver Ec. (3.10)) en realidad se implementa $-U_{\Psi_0}$. En una búsqueda de Grover normal, esta fase es global y no afecta al resultado, pero ahora estamos aplicando versiones controladas del operador de Grover G, con lo que esta fase afecta. En esencia, el único cambio es que en realidad estamos contando aquellos estados que *no son solución*. Lo único que tenemos que hacer para obtener el número de soluciones es restar al total de estados, N, el valor que obtenemos de aplicar QFE.

5.2.1. Precisión de $\tilde{M}$ respecto a M.

Podemos evaluar la precisión del valor de $\tilde{M}$ respecto al valor real M, acotando la desviación entre los mismos a medida que variamos p. Para ello, vamos a empezar asumiendo que la diferencia entre el valor real y el medido es menor que uno, es decir, $|f - \tilde{f}| < 1$. Esto sucede con una probabilidad razonable si f es suficientemente grande (si p es suficientemente grande). Teniendo en cuenta la Ec. (5.3), vemos que

$$|f - \tilde{f}| < 1 \quad \Rightarrow \quad |\theta - \tilde{\theta}| < \frac{\pi}{2^p} \quad \Rightarrow \quad |\sin\theta - \sin\tilde{\theta}| < \frac{\pi}{2^p},$$

donde en la última expresión se ha tenido en cuenta que $\sin\theta \approx \theta$ si θ es pequeño, lo cual es lógico si consideramos $M \ll N$. Jugando con las desigualdades puede verse que

$$\boxed{|M - \tilde{M}| < \frac{2\pi}{2^p}\sqrt{MN} + \frac{\pi^2}{(2^p)^2}N}. \tag{5.4}$$

Derivación de la Ec. (5.4)

Teniendo en cuenta las desigualdades anteriores, podemos derivar la expresión deseada

$$|M - \tilde{M}| < N \left| sin^2\theta - sin^2\tilde{\theta} \right| = N \left| \sin\theta - \sin\tilde{\theta} \right| \left| \sin\theta + \sin\tilde{\theta} \right| < N \frac{\pi}{2^p} \left(\sin\theta + \sin\tilde{\theta} \right).$$

En la última igualdad hemos quitado el valor absoluto porque es una suma de dos términos positivos ($0 < \theta < \pi/2$). Precisamente, como estos dos senos son positivos podemos escribir

$$\left| \sin\theta - \sin\tilde{\theta} \right| < \frac{\pi}{2^p} \quad \Rightarrow \quad \begin{cases} \sin\tilde{\theta} < \sin\theta + \frac{\pi}{2^p} \\ \sin\theta < \sin\tilde{\theta} + \frac{\pi}{2^p}. \end{cases}$$

Esto es porque tenemos dos números positivos que se diferencian en menos de una cierta cantidad α, así que es siempre cierto que la suma de uno de los números más α es mayor que el otro. Podemos usar la primera de estas desigualdades para seguir con el cálculo

$$|M - \tilde{M}| < N \frac{\pi}{2^p} \left(\sin\theta + \sin\tilde{\theta} \right) < N \frac{\pi}{2^p} \left(2\sin\theta + \frac{\pi}{2^p} \right) \tag{5.5}$$

Finalmente, llegamos a la Ec. (5.4)

Como podemos ver, la precisión depende de p. Además, el tiempo de ejecución depende de p, con lo que lo ideal es elegir un valor de p suficientemente grande como para tener una buena precisión, pero que este valor de p no sea demasiado grande y el algoritmo no tarde demasiado. Tomemos c como un parámetro y veamos diferentes casos:

- Si tomamos $2^p = c\sqrt{N}$, el error de nuestra estimación de M esta acotado por $\frac{2\pi}{c}\sqrt{M} + \frac{\pi^2}{c^2}$ siempre que $|f - \tilde{f}| < 1$. Esto recuerda a encontrar la respuesta hasta unas pocas desviaciones estándar.
- Si nos conformamos con tener un error *relativo* pequeño, podemos correr el algoritmo para sucesivos valores de p hasta que $\tilde{f}$ sea razonablemente grande. Esto sucederá cuando $2^p = c\sqrt{N/M}$. Después de un tiempo proporcional a $\sqrt{N/M}$, esto nos dará una estimación para M que probablemente estén dentro de un factor $(1 + \pi/c)^2$ de la respuesta correcta.
- Si queremos que el error *absoluto* esté probablemente limitado por una constante, aplicamos el algoritmo una vez para $2^p = c\sqrt{N}$ con el objetivo de estimar M. Entonces, ejecutamos otra vez, pero esta vez con $2^p = c\sqrt{\tilde{M}N}$. De acuerdo con la Ec. (5.4), pero suponiendo $2^p = c\sqrt{MN}$ por simplicidad, el error resultante en nuestra segunda estimación de M es probable que esté acotado por $\frac{2\pi}{c} + \frac{\pi^2}{c^2 M}$. En particular, obtenemos una solución exacta, siempre que $|f - \tilde{f}| < 1$, si tomamos $c \geq 14$ ya que $\frac{2\pi}{c} + \frac{\pi^2}{c^2 M} < \frac{1}{2}$ en ese caso. (Obsérvese que si aplicaciones sucesivamente el algoritmo de Grover y vamos tachamos las soluciones a medida que se encuentran también nos proporcionará un recuento exacto con alta probabilidad en un tiempo en $\mathcal{O}(\sqrt{MN})$, pero con un consumo enorme de memoria. Ver [2].)
- Finalmente, comentar que si estamos en el caso en el que el número de soluciones es un cuadrado perfecto pequeño, podemos encontrar el valor exacto en un tiempo $\mathcal{O}(\sqrt{N})$ con una probabilidad de error muy pequeña.

Para más detalles sobre el tema, puede verse [2].

6. Consideraciones sobre la implementacion

6.1. Creación de un difusor U_{Ψ_0}.

Vamos a ver como podemos hacer para crear de forma genérica un difusor de la forma de (3.10). Refrescando un poco la memoria, el difusor es un operador realiza una reflexión respecto al estado inicial $|\Psi_0\rangle$, es decir, **le cambia el signo a las componentes perpendiculares a $|\Psi_0\rangle$**. Lo que vamos a hacer para construir el difusor es, en realidad, construir un operador que **le cambia el signo a las componentes paralelas a $|\Psi_0\rangle$**, es decir, vamos a implementar $-U_{\Psi_0}$.

Empecemos definiendo la familia de **operadores de reflexión** S_A

$$\boxed{S_A|i\rangle = \begin{cases} |i\rangle & \text{si } i \notin A \\ -|i\rangle & \text{si } i \in A \end{cases}} \tag{6.1}$$

Es fácil ver que podemos escribir tanto el oráculo como el difusor en función de los operadores S_A

$$U_\omega = S_\omega \qquad\qquad \boxed{U_{\Psi_0} = -S_{\Psi_0}} \tag{6.2}$$

Nosotros lo que vamos a construir y implementar es S_{Ψ_0}, no U_{Ψ_0}

6.1.1. Caso con $N = 2^n$.

En el caso en el que el estado inicial es una superposición uniforme de la forma (2.2) podemos construir el difusor teniendo en cuenta que

$$|\Psi_0\rangle = H^{\otimes n}|0\rangle \quad \Rightarrow \quad H^{\otimes n}|\Psi_0\rangle = |0\rangle. \tag{6.3}$$

Viendo esta propiedad, podemos darnos cuenta de que si aplicamos el operador de Walsh-Hadamard $H^{\otimes n}$ a la salida del oráculo lo que obtenemos es el estado $|0\rangle$ más una serie de estado que corresponderán a los cambios respecto al estado inicial que ha realizado el oráculo. Lo que tenemos que hacer para aplicar el difusor es cambiarle el signo al estado $|0\rangle$ (aplicar S_0) y volver a aplicar $H^{\otimes n}$ para deshacer los cambios introducidos por la última aplicación el mismo. Es decir, el difusor será de la forma

$$U_{\Psi_0} = -S_{\Psi_0} = -H^{\otimes n} S_0 H^{\otimes n} \tag{6.4}$$

Podemos construir S_0 a partir de la **puerta multicontrolada Z** (MCZ)

$$MCZ = \begin{pmatrix} 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & \dots & 0 & -1 \end{pmatrix} \tag{6.5}$$

que lo que hace es cambiarle el signo al estado $|2^n - 1\rangle = |11\dots1\rangle$. Tenemos pues

$$S_0 = X^{\otimes n}(MCZ)X^{\otimes n} \tag{6.6}$$

La puerta $X^{\otimes n}$ (que consiste en aplicar puertas X a todos los qubits) lo que hace es

$$|00\dots0\rangle \to |11\dots1\rangle \ \Big[|0\rangle \to |2^n - 1\rangle\Big]$$
$$|11\dots1\rangle \to |00\dots0\rangle \ \Big[|2^n - 1\rangle \to |0\rangle\Big]$$

De esta forma, lo que hace S_0 es:

1. Aplicar la puerta $X^{\otimes n}$ para cambiar el estado $|00\ldots0\rangle$ por el estado $|11\ldots1\rangle$. (Ver la Nota de abajo)

2. Aplicar la puerta MCZ con la que cambiamos el signo a $|11\ldots1\rangle$

3. Aplicar la puerta $X^{\otimes n}$ para deshacer los cambios del primer paso. De esta forma, el único cambio real es el del signo del estado $|0\rangle = |00\ldots0\rangle$.

Nota

En realidad la puerta $X^{\otimes n}$ afecta a todos los estados (no solo a $|00\ldots0\rangle$ y $|11\ldots1\rangle$). Sin embargo, como es su propia inversa ($X^{\otimes n}X^{\otimes n} = I$) y como entre la primera y la segunda aplicación de $X^{\otimes n}$ lo único que hacemos es cambiarle al signo a $|11\ldots1\rangle$, todos los cambios se deshacen menos este signo, que pasa a estar en el estado $|00\ldots0\rangle$.

Como ya comentamos, **el operador que se implementa es $-U_{\Psi_0}$**, es decir

$$\boxed{-U_{\Psi_0} = S_{\Psi_0} = H^{\otimes n}S_0H^{\otimes n} = H^{\otimes n}X^{\otimes n}(MCZ)X^{\otimes n}H^{\otimes n}} \Rightarrow \tag{6.7}$$

$$\boxed{G_{imple} = -U_{\Psi_0}U_\omega = -H^{\otimes n}S_0H^{\otimes n}U_\omega} \tag{6.8}$$

6.1.2. Caso con $N \neq 2^n$.

Las limitación que hemos impuesto hasta ahora diciendo que N debía ser una potencia de 2 viene de la transformación de Walsh-Hadammard

$$H^{\otimes n}|j\rangle = \frac{1}{\sqrt{N}}\sum_{i=0}^{N-1}(-1)^{i\cdot j}|i\rangle, \qquad \text{(donde } i\cdot j \text{ denota el producto escalar binario)} \tag{6.9}$$

Esta transformación, que se usa para generar el estado inicial y se en el difusor, no está bien definida si no se cumple que $N = 2^n$.

Esta condición puede ser relajada si **sustituimos la transformación de Walsh-Hadammard por cualquier otra transformación unitario F que cumpla**

$$\boxed{F|0\rangle = \frac{1}{\sqrt{N}}\sum_{i=0}^{N-1}|i\rangle} \tag{6.10}$$

y seguiremos teniendo interaciones de Grover válidas aplicando

$$FS_0F^{-1}U_\omega \tag{6.11}$$

De hecho, cuando estamos en el caso $N = 2^n$, la Walsh-Hadammard es simplemente la elección más sencilla de F. Para el caso en el que N no es una potencia de dos, podemos usar la transformación de Fourier de Kitaev [3]. Puede verse también [4]

7. Distribución de probababilidad inicial aleatoria

En esta sección vamos a ver una generalización del algoritmo de Grover para el caso en el que partimos de una distribución de probabilidad aleatoria (no uniforme). Veremos como este algoritmo generalizado sigue requiriendo un número de iteraciones $\mathcal{O}(\sqrt{N/M})$, aunque veremos también que hay ciertos casos particularmente desfavorables donde no podemos encontrar ninguna solución.

Este caso es de especial interés si tenemos en cuenta que muchas veces hay errores al aplicar las puertas. De esta forma, podemos tener errores en el paso de inicialización de la distribución uniforme y acabar con una distribución que se distancia un poco de esta. Como veremos a continuación, el algoritmo de Grover sigue funcionando en presencia de estos errores modestos.

En la siguientes subsecciones presentaremos el algoritmo y derivaremos las ecuaciones diferenciales que rigen la evolución de las amplitudes. Usaremos la solución (exacta) de las mismas para calcular la probabilidad de éxito y analizaremos la diferente casuística. Como el cálculo, aunque simple, puede hacerse pesado, se incluye al final un resumen de con las conclusiones importantes (sección 7.4).

7.1. Algoritmo

En realidad, el algoritmo no tiene ningún misterio. Consiste simplemente en saltarse al paso de la inicialización con las puertas de Hadammard y partir del estado con una distribución aleatoria:

1. Utilizar cualquier distribución inicial, por ejemplo, el estado final de cualquier otro algoritmo cuántico (no inicializar el sistema con la distribución uniforme).
2. Aplicar el operador de Grover T veces (calcularemos T). El operador de Grover al que nos referimos aquí es el mismo de las anteriores secciones (con la Walsh-Hadammard o con una trasformación de la forma (6.10)). No cambia nada en ese sentido.
3. Medir el resultado

7.2. Evolución de las amplitudes (M soluciones)

Analicemos la evolución de las amplitudes en el algoritmo modificado. Supongamos (como siempre) que tenemos N estados y M soluciones. Asumamos, sin perdida de generalidad, que $0 \leq M \leq N/2$. El estado total ahora será de la forma

$$\boxed{|\Psi(t)\rangle = \sum_{i\in\omega} k_i(t)|i\rangle + \sum_{i\in\omega^\perp} l_i(t)|i\rangle}. \tag{7.1}$$

Nótese que ahora las amplitudes tenemos un índices i que corresponde al estado al que acompañan. Esto es fácil de entender: como la distribución de partida ahora no es uniforme, cada estado tendrá su amplitud. Las medias de las amplitudes de los estados solución ($i \in \omega$) y no solución ($i \in \omega^\perp$) son

$$\boxed{\bar{k}(t) = \frac{1}{M}\sum_{i\in\omega} k_i(t)} \qquad \boxed{\bar{l}(t) = \frac{1}{N-M}\sum_{i\in\omega^\perp} l_i(t)}. \tag{7.2}$$

Una de las conclusiones clave del desarrollo que vamos a ver a continuación es que *la dinámica dictada por el algoritmo de Grover se puede describir por completo basándose en la dependencia de las medias de las amplitudes con las iteraciones.*

7.2.1. Derivación de las ecuaciones diferenciales

Comencemos viendo la expresión de la media total de las amplitudes el estado $|\Psi(t)\rangle$

$$\bar{\psi}(t) = \frac{1}{N}\left[\bar{k}(t) + (N-M)\bar{l}(t)\right], \tag{7.3}$$

y definamos la cantidad $C(t)$

$$C(t) = \frac{2}{N}\left[(N-M)\bar{l}(t) - M\bar{k}(t)\right]. \tag{7.4}$$

Veamos como evolucionan las medias de las amplitudes con cada iteracións:

- Después a aplicar el oráculo sobre el esto $|\Psi(t)\rangle$ lo que sucede es el signo de la amplitud de los estados solución se invierte, con lo que

$$\begin{cases} \bar{k}(t) \;\to\; \bar{k}'(t) = -\bar{k}(t) \\ \bar{l}(t) \;\to\; \bar{l}'(t) = \bar{l}(t) \end{cases} \quad \Rightarrow \quad \overline{U_\omega\psi}(t) = \frac{1}{N}\left[-\bar{k}(t) + (N-M)\bar{l}(t)\right] = \frac{C(t)}{2}.$$

- Después a aplicar el difusor sobre el esto $|\Psi(t)\rangle$ lo que sucede es una reflexión respecto a la media $\overline{U_\omega\psi}(t)$. Esto se pude expresar de la siguiente manera

$$k_i'(t) \;\to\; k_i(t) = 2\overline{U_\omega\psi}(t) - k_i'(t) = C(t) - k_i'(t) = C(t) + k_i(t) \tag{7.5}$$

$$l_i'(t) \;\to\; l_i(t) = 2\overline{U_\omega\psi}(t) - l_i'(t) = C(t) - l_i'(t)C(t) - l_i(t). \tag{7.6}$$

  Promediando

$$\boxed{\bar{k}(t+1) = C_j + \bar{k}(t)} \tag{7.7}$$

$$\boxed{\bar{l}(t+1) = C_j - \bar{l}(t)}. \tag{7.8}$$

Las Ecs. (7.7) y (7.8) son las ecuaciones diferenciales que rigen la evolución de la media de las amplitudes.

7.2.2. Soluciones de las ecuaciones diferenciales

Vamos a resolver de forma analítica las Ecs. (7.7) y (7.8) para ver la evolución exacta de las medias de las amplitudes de los estados solución y no solución. Procedemos a resolver las fórmulas de recursión para condiciones iniciales complejas arbitrarias. Empecemos definiendo las funciones

$$\boxed{f_+(t) = \bar{l}(t) + i\sqrt{\frac{M}{N-M}}\bar{k}(t)} \tag{7.9}$$

$$\boxed{f_-(t) = \bar{l}(t) - i\sqrt{\frac{M}{N-M}}\bar{k}(t)}. \tag{7.10}$$

Vemos que no son más que un **cambio de variables** para facilitarnos la resolución de las ecuaciones. Empleando este cambio de variables podemos reescribir las Ecs. (7.7) y (7.8) de la siguiente forma

$$f_+(t+1) = e^{i\beta} f_+(t) \tag{7.11}$$

$$f_-(t+1) = e^{-i\beta} f_-(t), \tag{7.12}$$

donde β es **real** y cumple

$$\boxed{\cos\beta = 1 - 2\frac{M}{N}}. \tag{7.13}$$

Vemos que ahora la solución de las ecuaciones es trivial:

$$f_+(t) = e^{i\beta t} f_+(0) \tag{7.14}$$

$$f_-(t) = e^{-i\beta t} f_-(0). \tag{7.15}$$

Vemos claramente que $|f_+(t)|$ y $|f_-(t)|$ son independientes de la iteración t en la que estemos. Podemos ahora deshacer el cambio de variable

$$\boxed{\bar{k}(t) = -i\sqrt{\frac{N-M}{4M}}\left[e^{i\beta t}f_+(0) - e^{-i\beta t}f_-(0)\right]} \tag{7.16}$$

$$\boxed{\bar{l}(t) = \frac{1}{2}\left[e^{i\beta t}f_+(0) - e^{-i\beta t}f_-(0)\right]}. \tag{7.17}$$

7.3. Propiedades de las soluciones: Probabilidad de acierto.

7.3.1. Evolución de las amplitudes en función de la evolución de las medias.

Lo primero que vamos a hacer es ver que, efectivamente, solo con la evolución de las medias podemos describir la evolución. Para ello, restemos a la Ec. (7.5) la Ec. (7.7) y a la Ec. (7.5) restémosle la Ec. (7.7)

$$\begin{aligned} k_i(t+1) - \bar{k}(t+1) &= k_i(t) - \bar{k}(t) \\ l_i(t+1) - \bar{l}(t+1) &= -\left[l_i(t) - \bar{l}(t)\right]. \end{aligned}$$

Esto significa que las cantidades

$$\Delta k_i \equiv k_i(0) - \bar{k}(0) \tag{7.18}$$

$$\Delta l_i \equiv l_i(0) - \bar{l}(0) \tag{7.19}$$

son *constantes del movimiento*. Esto nos permite simplificar las expresiones para la dependencia temporal de las amplitudes:

$$\boxed{k_i(t) = \bar{k}(t) + \Delta k_i} \tag{7.20}$$

$$\boxed{l_i(t) = \bar{l}(t) + (-1)^t \Delta l_i}. \tag{7.21}$$

Vemos que la distribución de las amplitudes de los estados solución $k_i(t)$ respecto a su medias $\bar{k}(t)$ es constante. Es decir, las amplitudes varían al unísono entorno a la medias. De esta forma, con saber las distribución inicial respecto a las medía (Δk_i) podemos describir la evolución de las amplitudes solo conociendo la evolución de las medias. Lo mismo sucede para los estados no solución $l_i(t)$, salvo que estos se invierten entorno a su media en cada iteración.

De las Ecs. (7.20) y (7.21) se ve inmediatamente que las varianzas

$$\begin{aligned} \sigma_k^2(t) &= \frac{1}{M}\sum_{i\in\omega} |k_i(t) - \bar{k}(t)|^2 \\ \sigma_l^2(t) &= \frac{1}{N-M}\sum_{i\in\omega^\perp} |l_i(t) - \bar{l}(t)|^2 \end{aligned}$$

son independientes del tiempo [$\sigma_k^2(t) = \sigma_k^2(0)$ y $\sigma_l^2(t) = \sigma_l^2(0)$ $\forall t$].

7.3.2. Probabilidad de acierto.

Para facilitar el análisis de las soluciones, definamos las variables (complejas) α y ϕ de la forma

$$\alpha = \sqrt{f_+(0)f_-(0)}, \qquad\qquad e^{2i\phi} = f_+(0)/f_-(0). \tag{7.22}$$

Reescribamos las Ecs. (7.16) y (7.17) en función de estas variables

$$\boxed{\bar{k}(t) = \sqrt{\frac{N-M}{M}}\alpha \sin(\beta t + \phi)} \tag{7.23}$$

$$\boxed{\bar{l}(t) = \alpha \cos(\beta t + \phi)} \tag{7.24}$$

Esto nos permite ver que hay una diferencia de fase de $\pi/2$ entre las medias de las amplitudes de los estados solución y los no solución. Se ve fácilmente que cuando una es máxima, la otra es mínima.

La probabilidad de medir alguno de los estados solución será $P(t) = \sum_{i\in\omega} |k_i(t)|^2$. Como todos los operadores son unitarios, las amplitudes cumplen la condición de normalización:

$$\sum_{i\in\omega} |k_i(t)|^2 + \sum_{i\in\omega^\perp} |l_i(t)|^2 = 1 \quad \forall t \tag{7.25}$$

Usando la identidad

$$\overline{(y-\bar{y})^2} = \overline{y^2} - \bar{y}^2 \quad \Rightarrow \quad \sigma_y^2 = \overline{y^2} - \bar{y}^2 \quad \Rightarrow \quad \overline{y^2} = \sigma_y^2 + \bar{y}^2 \tag{7.26}$$

podemos escribir

$$\overline{|k_i(t)|^2} = \frac{1}{M}\sum_{i\in\omega} |k_i(t)|^2 = \sigma_k^2 + |\bar{k}(t)|^2$$

$$\overline{|l_i(t)|^2} = \frac{1}{N-M}\sum_{i\in\omega} |l_i(t)|^2 = \sigma_l^2 + |\bar{l}(t)|^2$$

Despejando $\sum_{i\in\omega} |k_i(t)|^2$ en (7.25), sustituyendo en $P(t)$, usando la expresión anterior de $\sum_{i\in\omega} |l_i(t)|^2$ y desarrollando con cuidado, puede verse que

$$\boxed{P(t) = P_{av} - \Delta P \cos 2\left[\beta t + \mathrm{Re}(\phi)\right],} \tag{7.27}$$

donde

$$P_{av} = 1 - (N-M)\sigma_l^2 - \frac{1}{2}\left[(N-r)|\bar{l}(0)|^2 + r|\bar{k}(0)|^2\right]$$

$$\Delta P = \frac{1}{2}\left|(N-M)\bar{l}(0)^2 + M\bar{k}(0)^2\right|$$

7.3.3. Número optimo, T, de iteraciones.

El valor máximo de la probabilidad que se puede obtener durante la evolución del algoritmo es

$$P_{max} = P_{av} + \Delta P \tag{7.28}$$

Dada una distribución arbitraria inicial de M estados solución y $N-M$ estados no solución, con medias $\bar{k}(0)$ y $\bar{l}(0)$ respectivamente, el valor máximo de la probabilidad P_{max} se alcanzará cuando realicemos T iteraciones tal que

$$\cos 2\left[\beta T + \mathrm{Re}(\phi)\right] = 1 \quad \Rightarrow \quad \boxed{T = \left[(u + 1/2)\pi - \mathrm{Re}(\phi)\right]/\beta} \tag{7.29}$$

con $u = 0, 1, 2, \ldots$. Una importante conclusión es que para determinar el valor óptimo de iteraciones, todo lo que necesitamos es conocer *las medias iniciales de las amplitudes y el número de estados marcados.*

Si M es pequeño tenemos que $1 - 2M/N \approx 1$, así que podemos aproximar β a primer orden en la Ec. (7.13)

$$\cos\beta \approx 1 - \frac{1}{2}\beta^2 = 1 - 2\frac{M}{N} \quad \Rightarrow \quad \beta = 2\sqrt{\frac{M}{N}} \tag{7.30}$$

Así que tenemos que T es del orden de $\mathcal{O}(\sqrt{N/M})$ (para $u = 0$).

7.3.4. Casos particulares.

El valor máximo de la probabilidad puede variar mucho dependiendo de las propiedades estáticas (medias y varianzas) de las distribución inicial de amplitudes. Como es lógico, cuanto mayor sea P_{max} menos repeticiones del algoritmo tendremos que hacer para encontrar una solución (donde cada repetición son m iteraciones). Es fácil darse cuenta de que el número esperado de repeticiones del algoritmo hasta encontrar un estado solución es $1/P_{max}$.

Nota

Porque *repeticiones del algoritmo*?. En el parágrafo anterior llamamos repeticiones del algoritmo a realizar varias veces las m iteraciones necesarias para maximizar la probabilidad.
Si la probabilidad máxima es menor de uno, tenemos entonces la posibilidad de tras realizar las m iteraciones no midamos un estado solución. Si esto sucede, como al medir se destruye el estado, lo que hay que hacer es volver a empezar: volver a partir de la distribución inicial y aplicar m iteraciones. En promedio, tendremos que repetir este proceso $1/P_{max}$ veces. Vemos que si la probabilidad máxima es, por ejemplo, $1/4$, tendremos que medir en promedio 4 veces para obtener un estado solución.

Veamos los diferentes casos:

- Cuando el ratio $\bar{l}(0)/\bar{k}(0)$ es real, puede verse fácilmente que $|f_+(0)| = f_-(0)|$. En este caso $P_{max} = 1 - (N-r)\sigma_l^2$. El mejor caso, aquel con $P_{max} = 1$, se obtiene cuando $\sigma_l = 0$, es decir, cuando la distribución inicial es uniforme.
- Cuando tenemos $f_+(0) = 0$ o $f_-(0) = 0$, el algoritmo es inútil, pues $P(t)$ es constante.
- El peor caso se da cuando $\sigma_k^2 = f_+(0) = f_-(0) = \bar{k}(0) = \bar{l}(0) = 0$ y $(N-r)\sigma_l^2 = 1$. En este caso tenemos $P_{max} = P(t) = 0\ \forall t$, con lo que nunca encontraremos una solución.

7.3.5. Distribución de probabilidad desconocida.

Veamos que sucede si no conocemos de antemano las medias y las varianzas de la distribución inicial de amplitudes. La solución es más sencilla de lo que podríamos pensar en un principio. Lo único que hay que hacer es ejecutar el algoritmo dos veces, ejecutando en un caso m_1 iteraciones y en el otro m_2 iteraciones, tal que $m_2 - m_1 = \pi/(2\beta)$. De la Ec. (7.27) está claro que al menos en uno de los dos caso vamos a tener $P(t) \geq P_{av} \geq P_{max}/2$. En este caso, necesitamos el doble de repeticiones para obtener al menos la mitad de probabilidades de éxito que cuando se conoce el tiempo de medición óptimo. Vemos además que lo único que necesitamos conocer en este caso es β, que según la Ec. (7.13) podemos calcularlo a partir del número de soluciones M.

7.4. Resumen de la sección.

Uno de los puntos importantes del análisis de esta sección es el hecho de demostrar que el algoritmo de Grover funciona incluso en el caso en el que tenemos pequeños errores a la hora de generar la distribución de probabilidad inicial. En realidad, en esta sección no se presenta ningún algoritmo nuevo. Simplemente se analiza que pasa si usamos el algoritmo de Grover donde en el primer paso sustituimos la inicialización de la distribución uniforme por una distribución no uniforme.

Otra de las conclusiones clave del desarrollo es que la dinámica dictada por el algoritmo de Grover se puede describir por completo basándose en la dependencia de las medias de las amplitudes $\bar{k}(t)$ y $\bar{l}(t)$ (definidas en la Ec. (7.2)).

Para estudiar la dependencia con las iteraciones de las medias $\bar{k}(t)$ y $\bar{l}(t)$, en la sección 7.2.1 deducimos las ecuaciones diferenciales que rigen su evolución (Ecs. (7.7) y (7.8)). En la sección 7.2.2 las resolvemos.

Para ello primero hacemos un cambio de variable de $\bar{k}(t)$ y $\bar{l}(t)$ a $f_+(t)$ y $f_-(t)$ (Ecs. (7.9) y (7.10)). Resolvemos usando estas variables y deshacemos el cambio. Nos quedan pues $\bar{k}(t)$ y $\bar{l}(t)$ en función de $f_+(0)$, $f_-(0)$ y β (definido β en la Ec. (7.13), que solo depende de N y M).

Una vez tenemos las soluciones, en las Ecs. (7.20) y (7.21) demostramos que las amplitudes mantienen su distribución respecto a las medias invariante (con una salvedad en $\bar{l}(t)$). De esta forma, confirmamos que solo tenemos que conocer la distribución inicial respecto a las medias para describir la evolución de las amplitudes usando solo la evolución de las medias.

En la sección 7.3.2 definimos dos parámetros más, α y ϕ (Ec. (7.22)), que depende de $f_+(0)$ y $f_-(0)$. Tras unas líneas de cálculo podemos llegar a la Ec. (7.27) que nos da la probabilidad $P(t)$ de medir un estado solución en función del número de iteraciones. Una vez tenemos esta expresión, en la sección 7.3.3 buscamos el número T de iteraciones que nos maximizan la probabilidad (Ec. (7.29)). Una importante conclusión es que para determinar el valor óptimo de iteraciones, todo lo que necesitamos es conocer las medias iniciales de las amplitudes y el número de estados marcados. Concluimos también que si M es pequeño, tenemos que T es del orden de $\mathcal{O}(\sqrt{N/M})$.

Finalmente, analizamos casos particulares de distribuciones iniciales favorables y desfavorables (sección 7.3.4) y vemos que pasa si no conocemos las distribución de probabilidad inicial (sección 7.3.5). Es este último caso, nuevamente podemos hallar una solución en tiempo $\mathcal{O}(N/M)$ conociendo solo M.

8. Implementaciones con qiskit.

8.0.1. Puerta multicontrolada Z (MCZ).

Vamos a empezar viendo la implementación de la puerta multicontrolada Z (MCZ) en qiskit, pues la usaremos bastante el las siguientes secciones. Esta puerta podemos construir a partir de la puerta multicontrolada Toffoli (MCT) de forma muy sencilla. Para ello, recordemos que la MCT no es más que una CNOT (es decir, una puerta X) con varios controles y recordemos también la propiedad

$$HXH = Z. \tag{8.1}$$

Podemos pues construir la MCZ aplicando puertas de Hadammard en el qubit objetivo de la MCT antes y después de la misma. En código de qiskit esto nos queda

```
def mcz(circuit, control_qubits, target_qubit):

   circuit.h(target_qubit)
   circuit.mct(control_qubits, target_qubit) # multi-controlled-toffoli
   circuit.h(target_qubit)
```

Nota

Esta función (con su documentación) puede encontrarse también en la carpeta *Code* en el script *Grover_Gates_and_functions.py*.

8.1. Difusor genérico.

Como ya comentamos en la sección 6.1, habitualmente en vez de implementar U_{Ψ_0} implementamos $-U_{\Psi_0}$. Vimos además que podemos hacer la implementación mediante transformadas de Walsh-Hadamard $H^{\otimes n}$, puertas $X^{\otimes n}$ y la puerta MCZ de la forma de la Ec. (6.7), es decir

$$\boxed{-U_{\Psi_0} = S_{\Psi_0} = H^{\otimes n} S_0 H^{\otimes n} = H^{\otimes n} X^{\otimes n} (MCZ) X^{\otimes n} H^{\otimes n}} \tag{8.2}$$

Veamos como traducir la Ec. (8.2) a código qiskit.

```
def Grover_Diffuser(circuit, target_reg):

   n = target_reg.size

   # Apply transformation |s> -> |00..0> (H-gates)
   for i in range(n):
      circuit.h(target_reg[i])

   # Apply transformation |00..0> -> |11..1> (X-gates)
   for i in range(n):
      circuit.x(target_reg[i])

   # Do multi-controlled-Z gate
   mcz(circuit, target_reg[list(range(n-1))], target_reg[n-1])

   # Apply transformation |11..1> -> |00..0>
   for i in range(n):
      circuit.x(target_reg[i])

   # Apply transformation |00..0> -> |s>
```

```
    for i in range(n):
        circuit.h(target_reg[i])
```

La función anterior es genérica: dado un circuito y un registro cuántico, esta función aplica un difusor de Grover en el registro. En la Fig. 7 el resultado de aplicar la función anterior sobre un circuito.

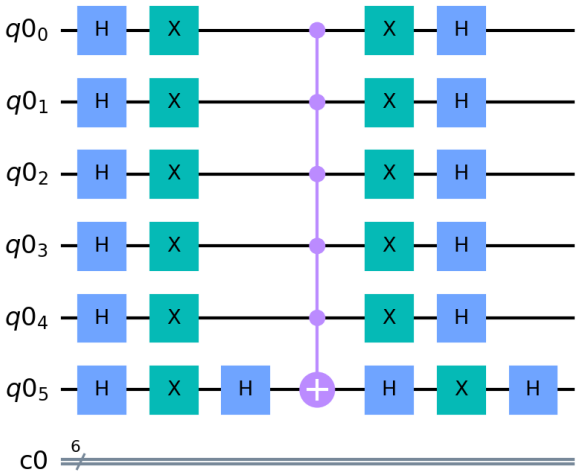


Figura 7: Difusor de Grover construido con Qiskit. Código en Grover's_algoritm\Code\Grover_Diffusers.py

Nota

Esta función (con su documentación) puede encontrarse también en la carpeta **Grover's_algoritm\Code.** (en mismo nivel que la carpeta Theory) en el script **Grover_Diffusers.py**. El script puede usarse como módulo para importar el difusor o simplemente ejecutarse para generar la gráfica.

8.2. Oráculo "trivial".

Nota Importante!!

Puede verse una implementación más general y avanzada del oráculo que se va a construir en esta sección en el script **1-Grover_trivial_oracle.py** situado en la carpeta **Grover's_algoritm\Code.** (en mismo nivel que la carpeta Theory).

Vamos a presentar en esta sección un código de qiskit para construir un oráculo que cambie el signo de los estados que nosotros le digamos. Este es uno de esos ejemplo típicos que se plantean cuando se habla de Grover, esos en los que sabemos con antelación los estados concretos que queremos buscar. De esta forma, el oráculo que construimos está echo "ad hoc" para marcar ciertos estado. Precisamente elegi llamarle a esta caso "trivial" porque no entraña ningún misterio, sino que como comento es un caso académico.

El código que se presenta a continuación es muy simple. Partimos de una lista de cadenas binarias de n bits que llamamos **M_list_bin_qiskit**. Por ejemplo, para $n = 5$

$$\text{M\_list\_bin\_qiskit} = [\text{'01101'}, \text{'11000'}, ...]$$

Nota

Recordemos que en qiskit el bit menos significativo es que va arriaba en el circuito, así que el "qiskit" en el nombre de la lista **M_list_bin_qiskit** hace referencia que estamos mandando las cadenas el revés. Es decir, para pasarlas a decimal las invertimos:

$$01101 \quad \rightarrow \quad 10110 = 22, \qquad 11000 \quad \rightarrow \quad 00011 = 3$$

Para cambiar el signo de un estado lo que tenemos que hacer es aplicar una MCZ. Como esta solo aplica sobre el estado $|11\ldots1\rangle$, lo que hay que hacer es aplicar puertas X antes y después del la MCZ en los qubit correspondientes a las posiciones donde tenemos un 0 en las cadenas bits.

```
def Grover_Oracle_trivial(circuit, target_reg, M_list_bin_qiskit):

   n = target_reg.size

   circuit.barrier()
   for m in M_list_bin_qiskit:
      for i in range(len(m)):
         if m[i] == '0':
            circuit.x(target_reg[i])

      mcz(circuit, target_reg[list(range(n-1))], target_reg[n-1])

      for i in range(len(m)):
         if m[i] == '0':
            circuit.x(target_reg[i])
```

En la Fig. 8 podemos ver el resultado de aplicar la función anterior sobre un circuito.

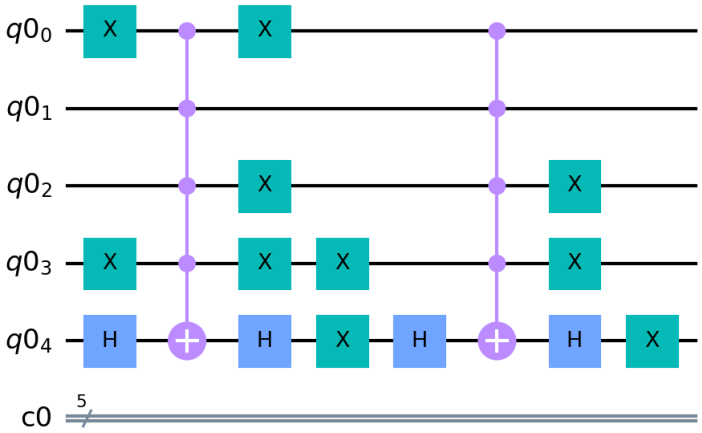


Figura 8: Oráculo trivial, es decir, para el caso en el que conocemos las soluciones de antemano. Código en Theory_Notes_Codes\Oracle_Trivial.py

Nota

Esta función (con su documentación) puede encontrarse también en la carpeta **Grover's_algoritm\Code.** en el script **Grover_Oracles.py**.
Además, el código que generó la Fig. 8 puede encontrarse en el script *Oracle_Trivial.py* en la carpeta *Theory_Notes_Codes*. En este script se un *import* del oráculo que está en el script anterior.

8.3. Oráculos que verifican condiciones.

En esta sección vamos a ver como se puede usar el algoritmo de Grover para buscar cadenas de bits que satisfacen unas ciertas condiciones. En concreto vamos a ver dos caso: como solucionar un **sudoku 2x2** y como calcular todas las posibles permutaciones de P números.

Nota Importante!!

Puede verse una implementación más general y avanzada del caso de las permutaciones situado en la carpeta **Grover's_algoritm\Code.** (en mismo nivel que la carpeta Theory). En esta implementación se usa la idea de buscar números diferentes para calcular las permutaciones de P números. Esta implementación es la "joya de la corona" de la explicación que se está dando sobre el algoritmo de Grover.

8.3.1. Sudoku 2x2.

8.3.1.1. El sudoku.

Vamos a ver en esta sección como construir un oráculo que resuelva un sudoku 2x2 de la forma:

$$\begin{bmatrix} V_0 & V_1 \\ V_2 & V_3 \end{bmatrix} \tag{8.3}$$

8.3.1.2. Condiciones y puerta XOR.

En un sudoku tiene que cumplirse que no se repitan números en las filas ni en las columnas, así que las condiciones que tienen que verificar las variables de sudoku 2x2 son:

$$V_0 \neq V_1, \qquad V_0 \neq V_2, \qquad V_2 \neq V_3, \qquad V_1 \neq V_3. \tag{8.4}$$

Por comodidad, podemos compilar este conjunto de comparaciones en una lista de cláusulas:

```
clause_list = [ [0,1], [0,2], [1,3], [2,3] ]
```

Para verificar estas condiciones podemos usar un qubit ancilla y usar la puerta XOR (se puede ver el código a continuación). Como el qubit ancilla parte del estado $|0\rangle$, esta puerta lo que hace ponerlo a en el estado $|1\rangle$ si la condición se verifica.

```
def XOR_2qubits(qc, a, b, output):
    qc.cx(a, output)
    qc.cx(b, output)
```

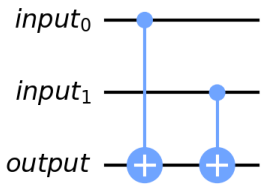


Figura 9: Puerta XOR. Código en Theory_Notes_Codes\XOR.py

Nota

Esta función (con su documentación) puede encontrarse también en la carpeta **Grover's_algoritm\Code.** en el script **Grover_Gates_and_functions.py**.
Además, el código que generó la Fig. 9 puede encontrarse en el script **XOR.py** en la carpeta **Theory_Notes_Codes**. En este script se un *import* de la puerta que está en el script anterior.

8.3.1.3. *Oráculo.*

Para generar el oráculo del sudoku podemos usar 4 qubits ancila (uno por condición a verificar) y usar la puerta XOR 4 veces. Pero con esto aún no tenemos el oráculo, pues nos falta verificar si todas las condiciones se cumplen y cambiar el signo del estado. Para ello, podemos usar una puerta MCT que tenga como controles los 4 cubits ancilla y que se aplique sobre un quinto qubit que esté en el estado $|-\rangle$. De esta forma, la puerta MCT solo se aplicará si se cumplen las 4 condiciones. Por último, solo nos faltaría volver a aplicar las puertas XOR para limpiar los qubits ancilla. El resultado final sería el siguiente:

```
def sudoku_oracle(circuit, clause_list, clause_qubits):
    # Compute clauses
    i = 0
    for clause in clause_list:
        XOR(circuit, clause[0], clause[1], clause_qubits[i])
        i += 1

    # Flip 'output' bit if all clauses are satisfied
    qc.mct(clause_qubits, output_qubit)

    # Uncompute clauses to reset clause-checking bits to 0
    i = 0
    for clause in clause_list:
        XOR(circuit, clause[0], clause[1], clause_qubits[i])
        i += 1
```

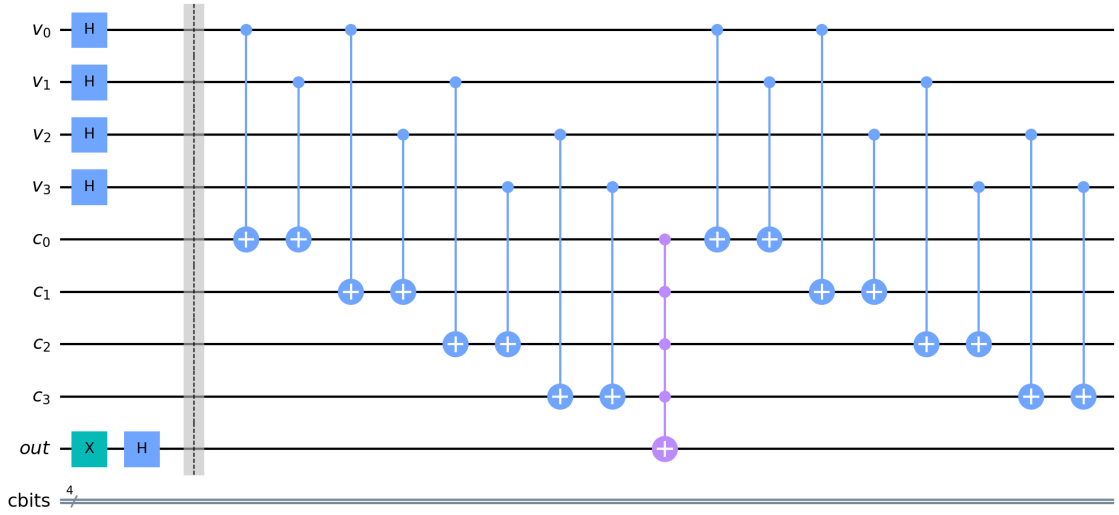


Figura 10: Oráculo para el sudoku 2x2. Código en Grover's_algoritm\Code\3-Sudoku_2x2.py

Nota

El código que generó la Fig. 10 puede encontrarse en el script **3-Sudoku_2x2.py** en la carpeta **Grover's_algoritm\Code**.

8.3.1.4. Simulación.

Podemos ahora construir el circuito completo y ejecutar la simulación.

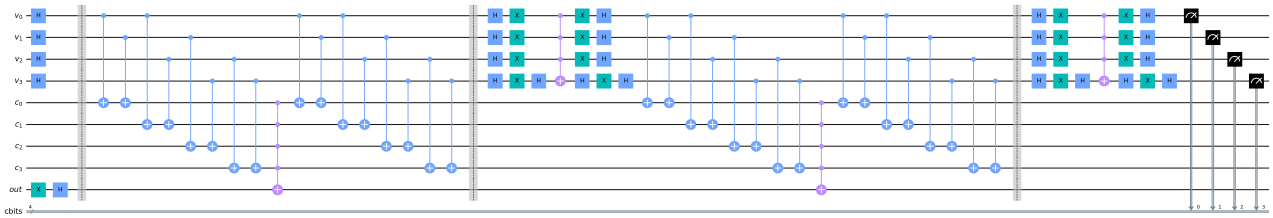


Figura 11: Circuito completo para la resolución con Grover del sudoku 2x2. Código en Grover's_algoritm\Code\sudoku_2x2.py

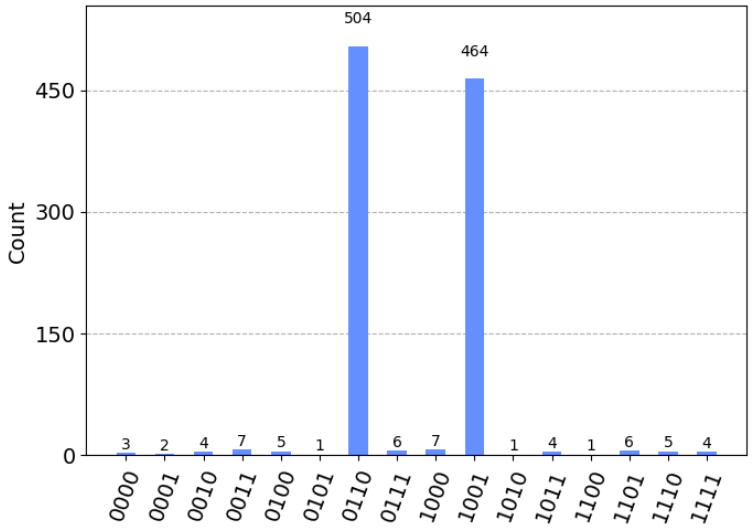


Figura 12: Resultado de la simulación del circuito de la Fig. 11.

Nota

El código que generó las simulación de las Figs. 11 y 12 puede encontrarse en el script **3-Sudoku_2x2.py** en la carpeta **Grover's_algoritm\Code**.

8.3.2. Permutaciones de P números.

En esta sección vamos a ver como construir un oráculo que nos dé todas las permutaciones posibles de P números $v_0 v_1 \dots v_{p-1}$ (con P siendo una potencia de 2). Para construir este oráculo necesitamos 4 registro de qubits:

- *var_reg*: Es el registro donde se almacenan los número que queremos que verifiquen las condiciones. Debe de tener $P * p$ qubits, donde p es el número de bis necesario para representar P en binario. Por ejemplo, para $P = 4$ necesitamos 8 qubits.
- *ancilla_reg*: Este registro lo usamos para compara qubit a qubit (bit a bit) los números codificados en el registro anterior. Debe de tener p qubits.
- *ancilla_reg_2*: Este registro tendrá tantos qubits como condiciones deben de verificarse. En cada qubit de este registro se almacenara el resultado de cada una de las condiciones (0 si no se cumple, 1 si se cumple). Para P número tenemos $(P-1)!$ condiciones. Por ejemplo, para $P = 4$ tenemos:
$$v_0 \neq v_1, \quad v_0 \neq v_2, \quad v_0 \neq v_3, \quad v_1 \neq v_2, \quad v_1 \neq v_3, \quad v_2 \neq v_3$$
- *ancilla_out_qubit*: Un registro con un solo qubit. Este qubit está en el estado $|-\rangle$ y sobre el se aplicará una puerta X si todos los qubits del registro *ancilla_reg_2* están en el estado $|1\rangle$ (es decir, si se cumplen todas las condiciones).

En las Figs. 13 y 14 podemos ver la primera parte del oráculo para el caso de $P = 4$. Podemos ver que está dividido en 7 bloques: un bloque por cada condición más un último bloque con un MCT que se aplica tomando como controles todos los qubit del registro ancilla_reg_2 y que tiene como objetivo el qubit *ancilla_out_qubit*.

Cada uno de los 6 bloques realiza una comparación bit a bit de dos números (compara el primer bit

con el primero, el segundo con el segundo,...). Cada comparación se realiza usando 2 CNOT. Una de ellas controlada por uno de los bits y la otra por el otro. Las CNOTs tiene como qubit objetivo un qubit del registro *ancilla_reg* $(c_0, c_1, \dots)$. De esta forma, si los bis son diferentes el estado del qubit del registro *ancilla_reg* pasa a ser $|1\rangle$. Para nuestro caso de $P = 4$, estas comparaciones las realizan las primeras 4 CNOTs de cada bloque.

Dos números son diferentes si al menos uno de sus bits es diferente, es decir, si tenemos al menos un $|1\rangle$ en el registro *ancilla_reg* después de las comparaciones de parágrafo anterior. El siguiente tramo de los bloques actúa sobre los qubits del registro *ancilla_reg* y un qubit del registro *ancilla_reg_2* $(c\_out_0, c\_out_1, \dots)$. El objetivo el mismo es poner el qubit del registro *ancilla_reg_2* en el estado $|1\rangle$ si tenemos al menos un 1 en el registro *ancilla_reg*. Esto lo hacemos en tres pastes:

1. Aplicamos sobre el qubit del registro *ancilla_reg_2* una CNOT por cada qubit del registro *ancilla_reg*. De esta forma, estamos cambiando el estado del qubit objetivo si tenemos un número impar de 1's en el registro *ancilla_reg*.

2. Aplicamos un puerta X sobre uno de los qubits de *ancilla_reg*. Volvemos a aplicar las CNOT y tras ellas volvemos a aplicar la puerta X. De esta forma, estamos cambiando el estado del qubit objetivo si tenemos un número par de 1's en el registro *ancilla_reg* o ningún 1 (si tenemos todo 0's)

3. Aplicamos puertas X sobre todos los qubis del registro *ancilla_reg*, aplicamos una MCT y aplicamos otra vez puertas X en todos los qubit de *ancilla_reg*. De esta forma, corregimos el hecho de haber cambiado el estado del qubit objetivo en el caso de haber tenido un estado con todo ceros en el registro *ancilla_reg*.

La última parte de los bloques es volver a aplicar las CNOT de comparación del circuito para limpiar los qubits del registro *ancilla_reg*.

La segunda parte del oráculo consistiría en repetir los 6 bloques (en realidad, los $(P-1)!$ bloques) de las condiciones para volver a poner a cero los qubits del registro ancilla_reg_2.

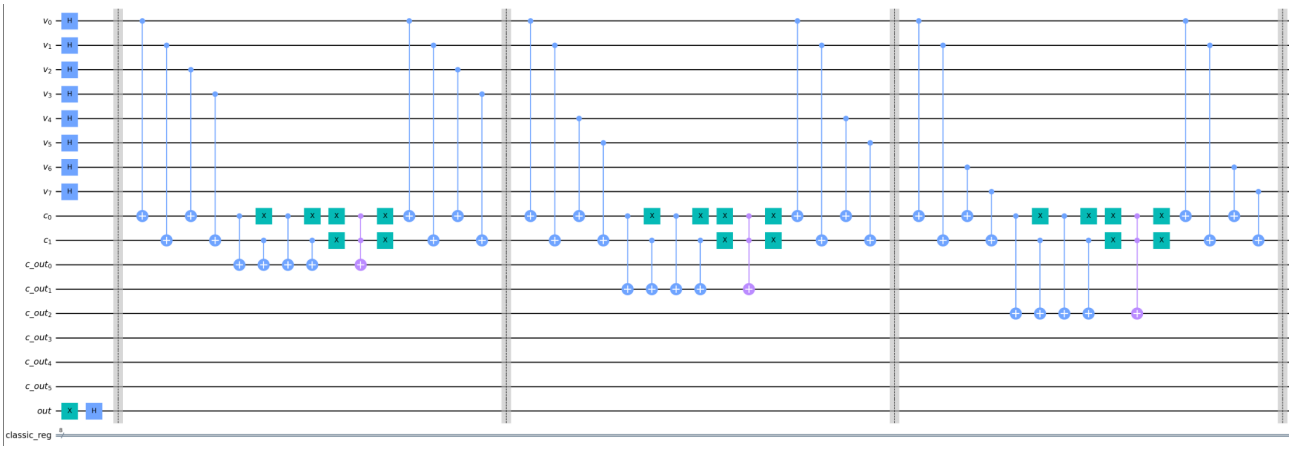


Figura 13: Oráculo para calcular las permutaciones, parte uno primer tramo.

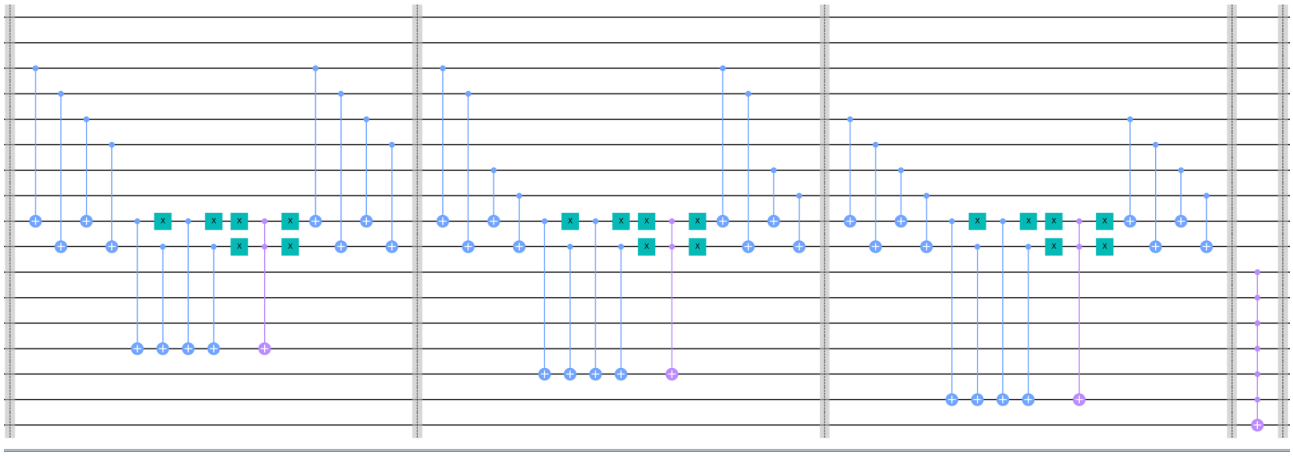


Figura 14: Oráculo para calcular las permutaciones, parte uno segundo tramo.

Nota Importante!!

Puede verse una implementación más general y avanzada del caso de las permutaciones situado en la carpeta **Grover's_algoritm\Code.** (en mismo nivel que la carpeta Theory). En esta implementación se usa la idea de buscar números diferentes para calcular las permutaciones de P números. Esta implementación es la "joya de la corona" de la explicación que se está dando sobre el algoritmo de Grover.

Referencias

[1] Textbook de Qiskit (IBM), "Grover's algorithm," 2021.

[2] G. Brassard, P. HØyer, and A. Tapp, "Quantum counting," in *Automata, Languages and Programming*, pp. 820–831, Springer Berlin Heidelberg, 1998.

[3] A. Y. Kitaev, "Quantum measurements and the abelian stabilizer problem," 1995.

[4] L. K. Grover, "Quantum computers can search rapidly by using almost any transformation," *Phys. Rev. Lett.*, vol. 80, pp. 4329–4332, May 1998.

[5] M. Boyer, G. Brassard, P. Høyer, and A. Tapp, "Tight bounds on quantum searching," *Fortschritte der Physik*, vol. 46, pp. 493–505, jun 1998.