Universidad Politécnica de Cartagena

Escuela Técnica Superior de Ingeniería de Telecomunicación

Algoritmo cuántico para solución eficiente de problemas de enrutamiento

Proyecto Fin de Carrera

Grado en Ingeniería Telemática

Autor:
Antonio Díaz Hernández

Director:
Javier Cerrillo Moreno

Codirector:
Antonio Pérez Garrido

Septiembre 2022

Agradecimientos

Me gustaría agradecer en primer lugar a mi familia, por el apoyo que me ha dado siempre en cada decisión tomada.

A Marina, por ser mi pilar fundamental en cada momento.

Y por último, a Antonio y, especialmente a Javier, por haber depositado su confianza en mí y haberme dado la oportunidad de acercarme al mundo de la computación cuántica, ya que sin su ayuda esto no habría sido posible.

Resumen

La finalidad de este trabajo es adentrarnos primeramente en el mundo de la física cuántica, concretamente en la parte de computación cuántica. Para ello, una vez hecha una introducción teórica, se explicarán los conceptos más básicos y necesarios como son las formas de representación de un estado cuántico, las puertas cuánticas, así como los algoritmos cuánticos más comunes. Después, se resolverá el Problema del Vendedor Ambulante asimétrico para una distribución arbitraria de cinco ciudades utilizando un algoritmo cuántico heurístico basado en el conocido Algoritmo de Grover.

Para ello, se ha desarrollado una nueva codificación diferente a la propuesta en el algoritmo con la que se reduce sustancialmente el número de qubits necesarios para la resolución del problema. También ha sido necesaria la implementación de un oráculo capaz de asignar fases a estados cuánticos de manera correcta y acorde a unas condiciones. Gracias a ello, ha sido posible conseguir unos buenos resultados, que han sido analizados con gráficas y diversas herramientas matemáticas.

Abstract

The purpose of this work is to first enter the world of quantum physics, specifically in the quantum computing part. To do this, once a theoretical introduction has been made, the most basic and necessary concepts will be explained, such as the forms of representation of a quantum state, quantum gates, as well as the most common quantum algorithms. Finally, the asymmetric Travelling Salesman Problem will be solved for an arbitrary distribution of five cities using a heuristic quantum algorithm based on the well-known Grover Algorithm.

To do this, a new coding different from the one proposed in the algorithm has been developed, with which the number of qubits necessary to solve the problem is substantially reduced. It has also been necessary to implement an oracle capable of assigning phases to quantum states correctly and according to certain conditions. Thanks to this, it has been possible to achieve good results, which have been analyzed with graphs and various mathematical tools.

Índice

Índice de figuras

Índice de tablas

1. Introducción y objetivos

1.1. Introducción

En los últimos años está creciendo notablemente el interés por el mundo de la computación cuántica. Este paradigma comenzó a principio de los años 80, cuando Paul Benioff propuso un modelo cuántico de la máquina de Turin [7]. Más tarde, Richard Feymann y Yuri Manin sugirieron que un ordenador cuántico tenía la potencia suficiente para simular cosas que un ordenador clásico no era capaz [8]. Posteriormente, Peter Shor desarrolló el primer y más importante algoritmo de la computación cuántica, el algoritmo de Shor, capaz de factorizar números con una potencia exponencialmente superior a un ordenador clásico [9].

1.2. ¿Por qué un ordenador cuántico?

Un ordenador cuántico es una máquina que funciona gracias a las leyes de la física cuántica, por lo que aprovechando ciertas propiedades como el entrelazamiento cuántico o la superposición, se podrían llegar a ejecutar operaciones en un tiempo altamente menor al que tardaría un ordenador clásico al que todos estamos acostumbrados.

La clave de esta mejora exponencial en el tiempo de cómputo viene dada por el paralelismo en la ejecución de las tareas. Mientras que los ordenadores de hoy en día realizan las operaciones de forma secuencial, un ordenador cuántico podría realizar múltiples operaciones a la vez, gracias a la propiedad de la superposición cuántica.

Esto supondría un cambio radical en prácticamente todos los ámbitos de la ciencia. Algunas aplicaciones serían la resolución de problemas que actualmente no pueden ser resueltos en un tiempo razonable, una mejora en la seguridad informática que nos protege, o incluso descubrir nuevas medicinas y materiales para su uso clínico.

1.3. Objetivos de la computación cuántica

La computación cuántica tiene como objetivo principal ayudarnos a resolver cualquier tipo de problema que no pueda ser resuelto mediante computación clásica en un tiempo razonable. Sin embargo, para que un algoritmo cuántico sea realmente útil, este debe ser mucho mejor que cualquier algoritmo clásico existente [3].

2. Conceptos de computación cuántica

2.1. Definición de Qubit

El concepto más esencial de la computación cuántica es el qubit, el cual es el equivalente cuántico del bit. Un ordenador clásico trabaja con bits, los cuales pueden tomar los valores 0 o 1. Sin embargo, un ordenador cuántico trabaja con el qubit como unidad de información más pequeña, el cual puede encontrarse en el estado $|0\rangle$, en el estado $|1\rangle$, o en cualquier combinación lineal de ambos [3]. El estado de un qubit viene representado por

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \tag{1}$$

donde α y β son números complejos y deben cumplir la condición de normalización, la cual viene dada por

$$|\alpha|^2 + |\beta|^2 = 1. \tag{2}$$

Esto nos permite reescribir (1) de tal manera que

$$|\psi\rangle = cos\frac{\theta}{2}|0\rangle + e^{i\phi}sen\frac{\theta}{2}|1\rangle. \tag{3}$$

Así, se puede representar cualquier estado de un qubit a través de una esfera de radio unidad, llamada la esfera de Bloch [3].

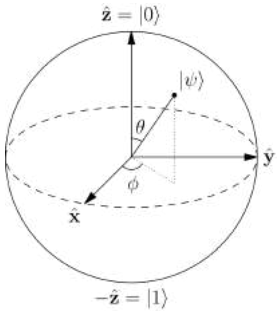

Figura 1: Esfera de Bloch. La Esfera de Bloch es una esfera de radio unidad en la que podemos representar cualquier estado $|\psi\rangle$ a través de las fases θ y ϕ. Fuente: Wikipedia.

De la misma forma, también podemos hablar del qudit, el cual es un objeto cuántico en el que el número de estados posibles no se limita a $|0\rangle$, $|1\rangle$, o a una combinación de ambos, si no que este se puede aumentar a dos o más. Este concepto es útil a la hora de la representación en ciertos problemas,

ya que un qubit solo permitiría asociar dos elementos a estados cuánticos, uno al $|0\rangle$ y otro al $|1\rangle$. Sin embargo, utilizando un qudit de cuatro niveles, equivalente a dos qubits de dos niveles, podríamos asociar hasta cuatro elementos, teniendo como posibles estados $|00\rangle$, $|01\rangle$, $|10\rangle$ y $|11\rangle$.

2.2. Puertas cuánticas

En computación clásica, las operaciones entre bits se realizan a través de puertas lógicas, mientras que en computación cuántica esto se lleva a cabo mediante puertas cuánticas. Estas puertas cuánticas deben ser reversibles, es decir, cualquier operación que se haga sobre uno o varios qubits tiene que poder deshacerse, por lo que no nos valen operaciones como duplicar un bit o borrarlo. Las puertas cuánticas se representan mediante matrices unitarias, por lo que una puerta que opera sobre n qubits, quedaría representada en una matriz $2^n \times 2^n$.

2.2.1. Puertas cuánticas de un qubit

Las puertas cuánticas de un solo qubit se corresponden con la puerta de Hadamard y aquellas que nos permiten jugar con la fase del mismo. A través de la puerta de Hadamard, podemos poner un qubit en un estado de superposición, lo cual es algo esencial para el desarrollo de cualquier algoritmo. Con las puertas de fase, o también llamadas las matrices de Pauli, podemos representar cualquier rotación del vector de Bloch como una combinación lineal de estas matrices. Además, también existe una puerta llamada puerta FASE, la cual asigna una fase global al estado en cuestión. La puerta Z en realidad es una puerta de fase global, pero con fase igual a π. Esta última será una puerta esencial en el desarrollo del algoritmo.

Nombre	Símbolo	Estado inicial	Estado final	Matriz
Hadamard	$H\|\psi\rangle$	$\|0\rangle$,$\|1\rangle$	$\frac{1}{\sqrt{2}}(\|0\rangle+\|1\rangle), \frac{1}{\sqrt{2}}(\|0\rangle-\|1\rangle)$	$H=\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}$
Puerta X	$X\|\psi\rangle$	$\|0\rangle$,$\|1\rangle$	$\|1\rangle$,$\|0\rangle$	$X=\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}$
Puerta Y	$Y\|\psi\rangle$	$\|0\rangle$,$\|1\rangle$	$i\|1\rangle, -i\|0\rangle$	$Y=\begin{pmatrix}0 & -i\\ i & 0\end{pmatrix}$
Puerta Z	$Z\|\psi\rangle$	$\|0\rangle$,$\|1\rangle$	$\|0\rangle$, $-\|1\rangle$	$Z=\begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}$

Tabla 1: Puertas cuánticas de un qubit. Se representa el símbolo, transformación y la matriz asociada de cada puerta.

2.2.2. Puertas cuánticas de dos o más qubits

Al igual que en computación clásica, en computación cuántica existen puertas de varios qubits. Estas puertas se componen a partir de las puertas de un qubit, pero añadiéndoles un qubit de control, es decir, la salida de la puerta vendrá condicionada por el valor de dicho qubit. A la hora de diseñar

un circuito, podemos elegir tantos qubits de control y destino como queramos, es decir, una misma puerta podrá afectar a varios qubits a la vez, y esta puede estar controlada también por varios qubits simultáneamente.

Una de las más utilizada es la puerta CNOT, la cual consiste en una puerta X, controlada por un qubit adicional. Esta solamente actuará como puerta X en el qubit de destino siempre y cuando el qubit de control esté en estado $|1\rangle$.

Además, también existe la puerta de Toffoli, equivalente cuántica de la puerta NAND clásica. Esta puerta tiene la misma funcionalidad que la puerta CNOT, pero con dos qubits de control, por lo tanto, actúa como una puerta X en el qubit destino siempre y cuando los dos qubits de control estén en el estado $|1\rangle$, de ahí su equivalencia con la puerta NAND.

2.3. Algoritmos y protocolos cuánticos más relevantes

Al igual que en la computación clásica, en cuántica también existen ciertos algoritmos y protocolos importantes, los cuales son la base a utilizar para la resolución de cualquier problema.

Los más conocidos y, a su vez, más importantes son los siguientes.

- Algoritmo de Grover
- Algoritmo de Shor
- Protocolo BB84
- Protocolo de teletransporte cuántico

El algoritmo de Grover es uno de los algoritmos más importantes en computación cuántica. A través de él, un problema de búsqueda no estructurado puede ser resuelto en un tiempo cuadráticamente menor al tiempo utilizado por un algoritmo clásico [5]. Como veremos más adelante, este algoritmo es útil en problemas tales como problemas de enrutamiento o búsquedas en bases de datos no indexadas.

Por otra parte, el algoritmo de Shor es conocido por factorizar números enteros en un tiempo polinómico [4]. Este algoritmo es muy interesante, ya que consigue una mejora exponencial con respecto a los algoritmos de computación clásica. A su vez, este algoritmo es algo peligroso, ya que, muchas criptografías de clave pública, como el algoritmo RSA, basan su seguridad en la factorización de números enteros, por lo que un mensaje cifrado con dicho algoritmo podría ser descifrado utilizando el algoritmo de Shor en un tiempo polinómico.

En la parte de protocolos, hablamos de uno que está muy relacionado con el algoritmo de Shor, ya que si bien hemos dicho que este sería capaz de romper todo el cifrado que produce RSA, el protocolo BB84 es un protocolo de criptografía cuántica, en el cual la transmisión está basada en la polarización de fotones, los cuales se envían mediante un canal cuántico. A su vez, con el uso de este algoritmo, podríamos saber si alguien está interviniendo en el envío de la información, ya que el hecho de medir alteraría el estado cuántico de los fotones, cosa que podría detectar el receptor [6].

Por último, tenemos el protocolo de teletransporte cuántico. Este protocolo hace posible la transmisión de información cuántica de una posición a otra suficientemente alejada, incluso en ausencia de un canal de comunicaciones cuánticas que vincule al emisor de dicha información con el receptor [3]. Este proceso se realiza mediante un canal clásico, por lo que la máxima velocidad a la que se puede realizar es la velocidad de la luz. Un matiz muy importante en este protocolo es la utilización de los estados de la base de Bell, la cual es un conjunto de estados específico de dos qubits que representa el ejemplo más simple del entrelazamiento cuántico.

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

Figura 2: Estados de la Base de Bell.

3. Problema del vendedor ambulante

3.1. Descripción del problema

El problema del vendedor ambulante o también llamado problema del agente viajero consiste en, dada una distribución de N ciudades, encontrar el camino más corto con el cual se visitan todas las ciudades, con las restricciones de solo poder pasar una vez por cada ciudad y de empezar y acabar el tour en la misma ciudad. Podemos considerar dos tipos de problemas, el simétrico y el asimétrico. La diferencia entre ambos reside en que en la variante asimétrica, las distancias de ida no coinciden con las de vuelta entre cada par de ciudades, por lo que se deben contemplar el doble de soluciones posibles.

Este ejemplo pertenece al tipo de problemas NP-hard, por lo que actualmente no es posible encontrar la solución exacta en un tiempo polinómico. A raíz de esto, ha sido necesario recurrir a la utilización de algoritmos heurísticos, con los cuales es posible obtener buenas soluciones en poco tiempo, que pueden ser óptimas o no.

3.2. ¿Por qué es importante?

El problema del vendedor ambulante es un problema que abarca un gran conjunto de aplicaciones, tales como problemas de encaminamiento de vehículos, problemas de logística y planificación, y, a su vez, se puede aplicar en la fabricación de circuitos electrónicos. También aparece como subproblema en muchos campos, como puede ser la secuenciación de ADN [10].

En el ámbito de las telecomunicaciones, este problema se puede aplicar a diversos problemas de enrutamiento de redes, en los cuales las ciudades se pueden sustituir por nodos de una red, y las distancias pueden hacer referencia a la velocidad o ancho de banda de cada enlace. Para este tipo de casos, estaríamos hablando de la variante asimétrica del problema del vendedor ambulante, ya que las velocidades en un sentido y en otro no tienen por qué ser iguales.

4. Algoritmo de Grover

4.1. Descripción del algoritmo

El algoritmo de Grover es uno de los algoritmos cuánticos más importantes, y se corresponde con un algoritmo de búsqueda en una secuencia de datos no ordenada con N elementos y en un tiempo $O(\sqrt{N})$ [1]. Supongamos que tenemos una lista de elementos, por ejemplo una agenda de números de teléfono, y queremos encontrar uno de ellos. Clásicamente, sería necesario realizar en media un total de $N/2$ consultas a la agenda telefónica. Gracias a la computación cuántica y en concreto, al algoritmo de Grover, podríamos reducir ese número de consultas a un total de $\sqrt{N}$.

Para entender este algoritmo de una forma más matemática, podríamos verlo como la inversión de una función, es decir, si tenemos una función $y = f(x)$, el algoritmo de Grover nos permitiría encontrar de forma más rápida el valor de x cuando tenemos como entrada el valor de y. Este algoritmo resulta muy útil en problemas en los cuales es sencillo comprobar si una solución es válida o no, como por ejemplo un sudoku, ya que cuanto más sencillo sea comprobarlo, menos trabajo costará implementar el oráculo, un componente muy importante en este algoritmo.

Como podemos ver en la gráfica de la figura 3, para un número pequeño de elementos, se necesitaría un mayor número de consultas utilizando el algoritmo de Grover, pero una vez la cantidad de elementos crece, la mejora es bastante notable.

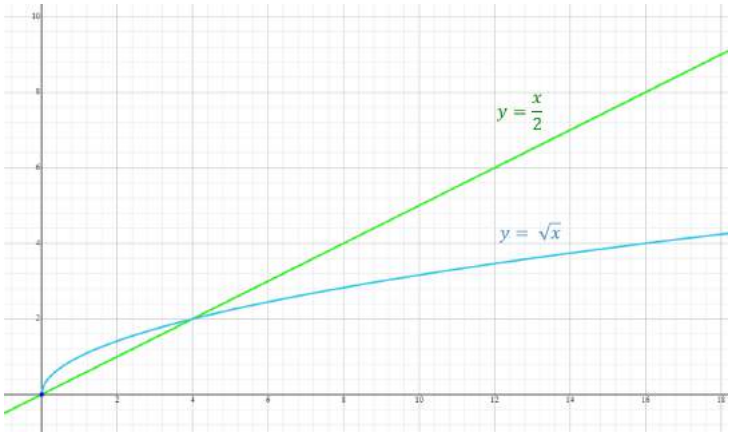

Figura 3: Comparación de funciones $y = \sqrt{x}$ e $y = \frac{x}{2}$ para observar la mejora de una con respecto a la otra. Fuente: Symbolab.

El algoritmo consiste básicamente en la aplicación de forma repetida de lo que se llama el operador Grover, que viene dado por la expresión

$$G = H^{\otimes n}(2\left|0\right\rangle\left\langle 0\right| - I)H^{\otimes n}O, \tag{4}$$

siendo $H^{\otimes n}$ el equivalente a la puerta de Hadamard de dimensión $n \times n$ y O el oráculo. A partir de esta expresión, podemos dividir el algoritmo en cuatro pasos, los cuales son

1. Aplicación del oráculo.
2. Aplicación de las puertas de Hadamard a los n primeros qubits.
3. Aplicación de la transformación unitaria a los n primeros qubits.
4. Aplicación de las puertas de Hadamard también a los n primeros qubits.

En la figura 4, podemos ver la estructura de forma gráfica, es decir, cómo quedaría en un circuito cuántico.

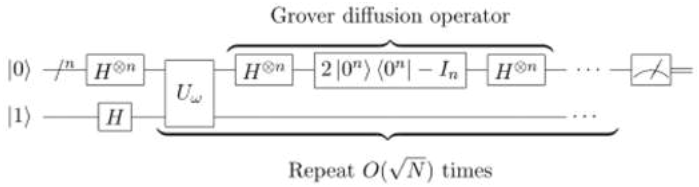

Figura 4: Estructura general del algoritmo de Grover. Fuente: https://commons.wikimedia.org.

Donde U_ω es el bloque correspondiente al oráculo y se explica con detenimiento en el siguiente apartado.

4.2. Oráculo y amplificador

Lo que hace el algoritmo de Grover tan útil es lo fácil que resulta convertir un problema de búsqueda local a un oráculo [5]. El oráculo que resuelve el algoritmo se corresponde con un bloque que asigna una fase negativa al estado solución, es decir, dejando los demás estados tal y como estaban. Para cualquier estado $|x\rangle$ de la base computacional, tomando como solución un estado arbitrario $|\omega\rangle$, el oráculo vendría dado por

$$f(x) = \begin{cases} |x\rangle & si \quad x \neq \omega \\ -|x\rangle & si \quad x = \omega \end{cases} . \tag{5}$$

También se podría expresar de la siguiente manera, siendo $f(x)$ una función que únicamente vale 1 cuando x es igual al estado solución ω y 0 en los demás casos.

$$U_\omega(|x\rangle) = (-1)^{f(x)}\,|x\rangle \tag{6}$$

En resumen, el oráculo se corresponde con una matriz diagonal en la cual la entrada correspondiente al estado ω tendrá una fase negativa [5]. Dependiendo del tipo de problema, la implementación de este bloque puede tener una mayor complejidad, llegando a necesitar qubits extra como podemos ver en la figura 4, o también llamados, qubits ancilla.

La segunda parte del algoritmo, llamada amplificador, difusor, o inversión sobre la media, es algo más sencilla, ya que es general para todos los problemas que se resuelvan con este tipo de algoritmo. Vendría dado por la expresión

$$U_s = H^{\otimes n}(2\left|0\right\rangle\left\langle 0\right| - I)H^{\otimes n}, \tag{7}$$

la cual podemos simplificar y quedaría

$$U_s = (2\left|s\right\rangle\left\langle s\right| - I), \tag{8}$$

siendo $\left|s\right\rangle = H^{\otimes n}\left|0\right\rangle = \frac{1}{\sqrt{2^n}}\sum_{x=0}^{2^n-1}\left|x\right\rangle$. Por último, para poder construir nuestro circuito, es necesario saber previamente cuántas iteraciones vamos a tener que aplicar del operador Grover. Para ello, primero debemos calcular el ángulo de rotación θ el cual podemos verlo gráficamente en la figura 5. Este vendría dado por

$$\theta = arcsin\frac{1}{\sqrt{N}}, \tag{9}$$

siendo N el número total de elementos en la lista, después de t pasos, tendremos

$$(U_sU_w)^t\left|s\right\rangle = sin\theta_t\left|w\right\rangle + cos\theta_t\left|s'\right\rangle, \tag{10}$$

donde $\left|s\right\rangle$ se corresponde con el estado inicial, $\left|w\right\rangle$ el estado solución y $\left|s'\right\rangle$ el conjunto de estados no solución. A su vez, la fase vendría dada por

$$\theta_t = (2t+1)\theta. \tag{11}$$

Para obtener $\left|w\right\rangle$ necesitamos que $\theta_t = \frac{\pi}{2}$, por lo que, sustituyendo, obtendríamos el número t de iteraciones necesarias.

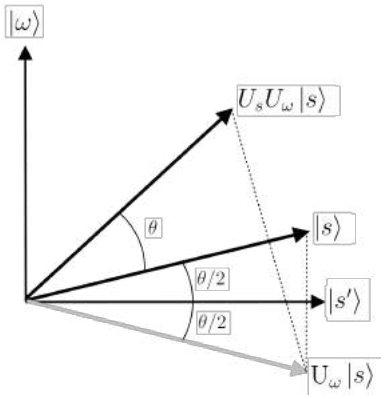

Figura 5: Rotaciones del estado inicial tras aplicar el oráculo y el amplificador.

4.3. Variación del algoritmo de Grover

Como hemos comentado en la introducción, en este trabajo no se ha utilizado directamente el algoritmo de Grover, sino una variación del mismo. Concretamente, se utilizará un algoritmo cuántico heurístico para el problema del vendedor ambulante, generalizando el algoritmo de Grover [2].

Al igual que en el algoritmo de Grover, tendremos un oráculo que nos asignará unas determinadas fases y un amplificador. Si los caminos van a ser representados de forma cuántica, necesitamos una base ortonormal de N estados [2], en la que cada estado se corresponderá con un camino diferente. Esto lo veremos detenidamente en el apartado número cinco de codificaciones, ya que según la codificación de estado a camino que elijamos, se pueden necesitar más o menos qubits.

La misión del oráculo ahora no va a ser poner una fase negativa a los estados solución, sino dar una fase escalada entre 0 y 2π, dependiendo de la distancia de cada camino. Es decir, si dividimos los caminos en cuatro fases entre 0 y 2π, los caminos más largos se quedarán con una fase de $3\pi/2$, y los más cortos se quedarán con fase 0, puesto que estos son los que dará el algoritmo como resultado. Los caminos con distancias intermedias quedarán repartidos entre las fases $\pi/2$ y π. A la hora de implementar este tipo de oráculo, es muy importante la utilización de las puertas de fase condicionadas, ya que serán las encargadas de asignar un tipo de fase u otro en cada caso. El oráculo $\hat{C}$ vendría dado por la expresión

$$\hat{C} |T\rangle = e^{i\phi(T)} |T\rangle , \tag{12}$$

siendo $|T\rangle$ el estado cuántico correspondiente a un camino. La segunda parte del algoritmo es equivalente al amplificador, o difusor, para el cual vamos a utilizar el mismo que se ha explicado en el apartado anterior, ya que este bloque es general para cualquier aplicación de algoritmo de Grover. En [2] se propone un operador Grover definido por

$$\hat{G} = -U_s \hat{C}, \tag{13}$$

donde $\hat{C}$ se corresponde con el oráculo de coste definido en la ecuación (12) y $U_s|s\rangle = -|s\rangle$, dejando los demás estados ortogonales invariantes, donde $|\psi_0\rangle$ viene dado como antes por

$$|s\rangle = \frac{1}{\sqrt{N}} \sum_T |T\rangle, \tag{14}$$

donde el sumatorio recorre todos los N caminos. En cuanto al número de iteraciones, este ya no se obtiene de la misma forma que en el algoritmo de Grover, si no que ahora depende de la distribución de fases. Vendría dado por la expresión

$$R = \frac{\frac{\pi}{2} - \sqrt{f_0}}{2\sqrt{f_0}}, \tag{15}$$

donde $f_0 = N_0/N$ siendo N_0 el número de caminos con fase 0.

Para que el algoritmo funcione de forma correcta, la distribución de las distancias de los caminos debe seguir una distribución específica, la cual se asemeje a una distribución normal. Es decir, si $f_j = N_j/N$, siendo N_j el número de caminos pertenecientes al grupo de fase $j * \pi/2$ para $j = 0, 1, 2, 3$, para un correcto funcionamiento se debe cumplir $f_0 + f_2 \simeq 1 \gg f_{1,3}$, con $f_2 \gg f_0, f_1, f_3$. En pocas palabras, es deseable que la mayoría de caminos sean de una longitud intermedia (fase π) y como consecuencia, que el número de caminos en los demás grupos de fase (fases 0, $\frac{\pi}{2}$ y $\frac{3\pi}{2}$) sea mínimo.

5. Codificaciones para la resolución del problema

5.1. Tipos de codificación

Para resolver este problema, es necesario tener una buena codificación de caminos a partir de estados cuánticos, para así optimizar de la mejor forma el algoritmo. En este trabajo se han tenido en cuenta dos tipos de codificación.

La primera de ellas, es la codificación propuesta en [2], y consiste en codificar cada ciudad con un estado diferente de la base utilizada. Es decir, teniendo N ciudades, necesitaríamos $n = \lceil \log_2 N \rceil$ qubits para codificarlas todas, o bien un qudit de dimensión N. Por lo que un camino vendría dado por el producto tensorial de las ciudades que lo forman, siendo cada una de ellas un estado cuántico diferente. Equivaldría a la expresión

$$|T_1\rangle = |C_0\rangle \otimes |C_1\rangle \otimes ... \otimes |C_{N-1}\rangle, \tag{16}$$

donde $|C_k\rangle$ se refiere a la k-ésima ciudad visitada, y $|C_0\rangle$ se corresponde tanto con la primera ciudad como con la última, ya que es un ciclo Hamiltoniano. Podemos ver que el espacio de Hilbert que abarca esta codificación es como mínimo de dimensión N^N, sin embargo, el total de caminos posibles para una distribución de N ciudades es de $(N-1)!$, por lo que vemos que habría bastantes estados que no equivaldrían a un camino válido.

A partir de esta, se ha desarrollado una nueva codificación más optimizada, la cual se ajusta más al número de caminos posibles para una distribución dada. En esta nueva codificación, no se codifican directamente las ciudades que forman un camino, sino que se codifica el número de saltos de una ciudad a otra. Es decir, tomando como ciudad inicial C_0 para todos los tour, construir los caminos indicando el número de saltos entre las ciudades que forman cada uno de ellos. Gracias a esto, conforme se va avanzando, se van eliminando ciudades posibles a las que dirigirse, debido a la condición de solo pasar una vez por cada una de ellas, y es por eso que el espacio de Hilbert se reduce bastante con respecto a la codificación anterior.

Por ejemplo, si tenemos una distribución de N ciudades, para representar un camino necesitaríamos un total de $\sum_{i=N-1}^{2} \lceil \log_2 i \rceil$ qubits, los cuales podrían ser agrupados en qudits de diferente tamaño, cada uno de ellos asociado a cada salto de una ciudad a otra. Un tour quedaría expresado de la forma

$$|T_2\rangle = |Q_{0,1}\rangle \otimes |Q_{1,2}\rangle \otimes ... \otimes |Q_{N-3,N-2}\rangle, \tag{17}$$

donde $|Q_{i,j}\rangle$ es el qudit que codifica el número de saltos de la ciudad i a la ciudad j.

5.2. Escalabilidad

Como hemos visto en el apartado anterior, cada una de las dos codificaciones utiliza un número de qubits diferente, y a su vez, trabajan en espacios de Hilbert diferentes. Para decantarnos por una u

otra, debemos realizar previamente un análisis de todas ellas, y ver cual se ajusta mejor a nuestro problema.

En la tabla 2 podemos ver un resumen de las dos codificaciones contempladas en el trabajo, en la cual se indica el número de qubits necesarios para cada una de ellas.

Codificación	**Escalabilidad**	**Nº qubits para 5, 10 y 15 ciudades**
Basada en ciudades	$N * \lceil \log_2 N \rceil$	15, 40, 60 (qubits)
Basada en número de saltos	$\sum_{i=2}^{N-1} \lceil \log_2 i \rceil$	5, 21, 41 (qubits)

Tabla 2: Comparación de las distintas codificaciones contempladas en base a su escalabilidad (Nº de qubits).

Figura 6: Comparación de la escalabilidad de las distintas codificaciones. La recta azul representa la codificación basada en ciudades, y la recta naranja la codificación basada en número de saltos.

Como podemos observar tanto en la tabla 2 como en la figura 6, la segunda codificación utiliza un menor número de qubits con respecto a la primera, y es por eso por lo que se ha decidido utilizarla. Por lo que. para una distribución asimétrica de 5 ciudades, quedaría de la siguiente manera.

Salto 1	Salto 2	Salto 3	Salto 4	Salto 5
$\|00\rangle$	$\|00\rangle$	$\|0\rangle$	$\|0\rangle$	$\|0\rangle$
$\|01\rangle$	$\|01\rangle$	$\|1\rangle$		
$\|10\rangle$	$\|10\rangle$			
$\|11\rangle$				

Tabla 3: Codificación basada en saltos para una distribución asimétrica de 5 ciudades.

De la tabla 3 podemos destacar varias cosas. La primera sería que, como hemos dicho, tan solo se indican los saltos que se tienen que dar para pasar de una ciudad a otra, es decir, no se codifica la ciudad inicial, ya que siempre se va a tomar la misma. También, en el salto 3, vemos que nos falta el estado $|11\rangle$. Esto es debido a que en este salto solo tendríamos tres posibles ciudades a las que ir, por lo que este estado se correspondería con un estado no contemplado, el cual se verá más adelante para qué se utiliza. Por último, podemos ver que los saltos 4 y 5 solo tienen un estado. Esto significa que estos dos saltos tampoco va a ser necesario codificarlos porque, conociendo los 3 primeros, es suficiente para poder saber el camino completo a partir del estado cuántico codificado.

Para una mayor compresión de ambas codificaciones, se va a mostrar un ejemplo de cada una. Para ello, se va a utilizar el siguiente camino de ejemplo.

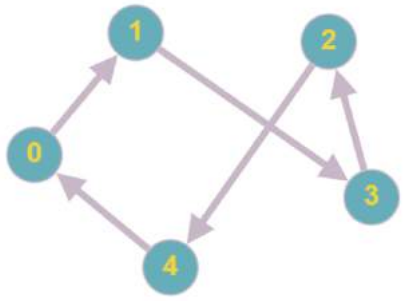

Figura 7: Camino arbitrario en distribución de cinco ciudades. Fuente: www.graphonline.es

Como podemos observar, el camino se corresponde con la secuencia de ciudades $0-1-3-2-4-0$. Como hemos visto, podemos representarlo utilizando dos codificaciones diferentes. Si utilizamos la primera de ellas, el camino quedaría definido por

$$|T_1\rangle = |000\rangle \otimes |001\rangle \otimes |011\rangle \otimes |010\rangle \otimes |100\rangle, \tag{18}$$

donde podemos ver que efectivamente es necesario utilizar 15 qubits para poder codificarlo. Estos

qubits se dividen en 5 qudits de dimensión 5 cada uno o en 5 grupos de 3 qubits, y estos hacen referencia al número de cada ciudad, la ciudad 0 equivaldría al qudit con estado $|000\rangle$, la ciudad 1 al $|001\rangle$, etc. Sin embargo, si utilizamos la segunda codificación, vamos a ver que el número de qubits se reduce notablemente. El camino quedaría definido por

$$|T_2\rangle = |00\rangle \otimes |01\rangle \otimes |1\rangle, \tag{19}$$

que a simple vista puede ser menos intuitivo de identificar, pero es mucho más eficiente ya que tan solo utiliza 5 qubits. El primer qudit indica que el primer paso se hace a la ciudad inmediatamente posterior a la inicial, en este caso, de la ciudad 0 a la ciudad 1. El siguiente qudit indica que en el siguiente paso es necesario dar 2 saltos, por lo que vamos de la ciudad 1 a la ciudad 3. Finalmente, en el último qudit vemos que para ir a la siguiente ciudad hay que dar también dos pasos, por lo que se iría a la ciudad 2. Y ahora podríamos pensar que nos faltan pasos por codificar pero, en realidad, solo queda una forma de terminar el tour, y esta es yendo a la ciudad 4 y posteriormente de vuelta a la 0, ya que es necesario cumplir las condiciones impuestas por el problema como hemos comentado anteriormente.

6. Implementación del Oráculo

En primer lugar, podemos dividir el funcionamiento del oráculo en dos partes. La primera es una conversión o escalado de las distancias proporcionadas a fases. La segunda parte y más importante consiste en cómo implementar una función que se encarga de asignar estas fases a cada uno de los estados cuánticos, los cuales equivalen a los caminos, como ya hemos visto en el apartado anterior.

6.1. Escalado de distancias

Esta parte consiste en un escalado de las distancias entre cada par de ciudades. Este escalado consiste en, a partir de una matriz que contiene las distancias entre cada par de ciudades, clasificarlas como fase 0 o fase $\frac{\pi}{2}$. Para obtener a partir de qué valor una distancia se considera de fase 0 o de fase $\frac{\pi}{2}$, se hace mediante el siguiente procedimiento.

1. Calcular valor medio en cada fila:

$$\alpha = \frac{MAX - MIN}{2}$$

2. Establecer rango de distancias asociadas a fase 0 y fase $\frac{\pi}{2}$:

$$[MIN, MIN + \alpha] = 0$$

$$(MIN + \alpha, MAX] = \frac{\pi}{2}$$

Como hemos dicho, este escalado no se hace una vez para la matriz completa, sino que se hace en cada una de las filas por separado, ya que se ha comprobado que se obtiene un mejor resultado de esta forma. Este proceso se ha implementado mediante la función *escalarMatriz(m)*, a la cual se le pasa como parámetro la matriz de distancias, y devuelve la matriz de fases.

6.2. Asignación de fases

Por último, una vez tenemos la matriz de fases, solo nos queda asignar dichas fases a los estados cuánticos. Para ello, se han implementado dos funciones.

La primera es la función *coste1q(a,b)*, la cual actúa sobre un único qubit y se encarga de dar las fases a y b, que se le pasan como parámetros, a los estados $|0\rangle$ y $|1\rangle$ respectivamente. Esta función crea un circuito cuántico de un solo qubit, y mediante dos puertas FASE y una puerta NOT, se les asigna dichas fases a los estados cuánticos. La primera puerta de fase actúa sobre el estado $|0\rangle$ y la segunda sobre el estado $|1\rangle$. El circuito que crea esta función tiene la siguiente forma:

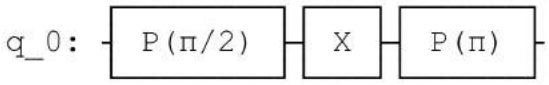

Figura 8: Representación del circuito generado por la función *coste1q(a,b)* para $a = \frac{\pi}{2}$ y $b = \pi$. En él se hacen uso de dos puertas FASE ($P(\pi/2)$ y $P(\pi)$) y una puerta NOT (X).

A partir de esta función, se ha creado la función *phasing()*, la cual es algo más compleja, y se encarga de dar fases a todos los estados que forman el espacio de Hilbert en cuestión. Esta función crea un circuito más grande, en el que, gracias a las puertas NOT y al uso de la función *coste1q(a,b)* como puerta condicionada, es posible asignar las fases correspondientes a cada estado.

Como ya sabemos, se han utilizado un total de cinco qubits en nuestro circuito, por lo que la función *coste1q(a,b)* como puerta condicionada tendría como qubit destino el qubit 0, y los qubits del 1 al 4 como qubits de control. Por eso, gracias a las puertas NOT, se van negando únicamente los qubits de cada estado que estén en $|0\rangle$, así, se aplicarán las fases a y b correspondientes a sus estados correspondientes. En la siguiente imagen podemos ver un ejemplo de como se aplicaría la función *coste1q(a,b)* condicionada a los estados $|10110\rangle$ y $|10111\rangle$.

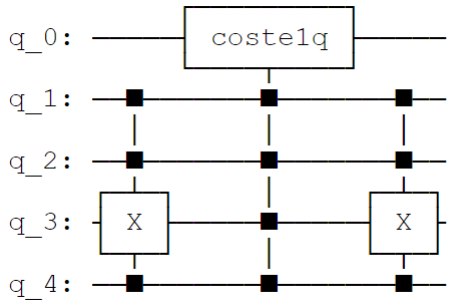

Figura 9: Representación del circuito generado por la función *phasing()* para la asignación de fases a los estados $|10110\rangle$ y $|10111\rangle$.

Como podemos ver, las puertas CNOT se encargan de negar el único qubit que está en estado $|0\rangle$, para que así la puerta condicionada generada por la función *coste1q(a,b)* aplique las fases solamente al estado en el que los qubits q_1, q_2, q_3 *y* q_4 están en $|1\rangle, |1\rangle, |0\rangle$ y $|1\rangle$.

Tal y como hemos dicho, las fases se le pasan como parámetro a la función *coste1q(a,b),* y estas se toman de la matriz de fases que nos da la función *escalarMatriz(m).* Como, en la matriz escalada, cada enlace entre cada par de ciudades tiene, o bien fase 0, o bien fase $\frac{\pi}{2}$, es necesario decidir de qué forma se van a aplicar dichas fases a los estados, para que la fase final de cada uno tenga uno de los valores de fase válidos, los cuales son $[0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}]$. Para conseguirlo, solo se han tenido en cuenta los tres primeros pasos de cada camino, ya que con cualquier combinación de fases de la matriz obtendría-

mos una fase final dentro del rango válido. En cambio, para tener en cuenta todo el camino completo, tendríamos que dividir los caminos en más grupos de fases.

Por último, en el apartado de codificaciones se ha hablado de que, a pesar de que nuestra codificación se ajusta bastante al espacio de Hilbert que abarca todos los caminos posibles, existen ciertos estados no contemplados, estos son los que tienen el segundo qudit en estado $|11\rangle$. A estos estados, en un primer momento se decidió darles directamente la fase más alta, ya que de esta forma penalizábamos más estos caminos. Pero, si observamos bien el funcionamiento de la variación del algoritmo de Grover utilizada, vemos que este algoritmo funciona bien siempre y cuando se cumplieran ciertas condiciones en la distribución de los caminos, es decir, el algoritmo era óptimo cuando el número de caminos en fase π era un número cercano al total de caminos, es decir $f_1 \approx 1$. Por lo que, aprovechando que tenemos ciertos estados no contemplados, se decidió darles fase π para un mejor funcionamiento del algoritmo.

7. Implementación del algoritmo y análisis de resultados

7.1. Implementación del algoritmo

Una vez tenemos clara la implementación del oráculo, el algoritmo consiste, como hemos visto en la ecuación (13), en aplicar dicho oráculo en conjunto con lo que llamamos difusor o amplificador. El circuito quedaría de la siguiente forma.

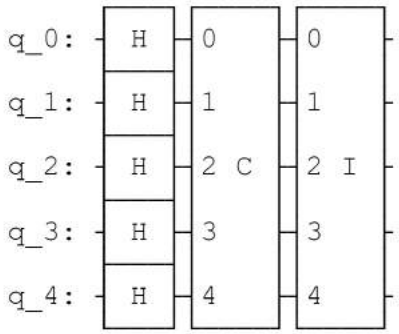

Figura 10: Representación del circuito correspondiente a la inicialización del circuito mediante las puertas de Hadamard junto con una iteración del algoritmo.

Donde el bloque C se corresponde con el oráculo de fase y el bloque I se corresponde con el difusor o amplificador.

7.2. Resultados obtenidos

Como hemos comentado en el apartado de la implementación del oráculo, dependiendo de qué fase tomaran los estados no contemplados, podríamos obtener un resultado u otro. La diferencia estaba en si, a los estados no contemplados, es decir, a los estados en los que el segundo qudit tenia valor $|11\rangle$, se les daba la fase más alta, para que así estos tuvieran una mayor penalización, o se les daba fase π, para que la distribución de fases se ajustara más al escenario ideal en el cual el algoritmo funciona con alta probabilidad.

Para obtener dichos resultados, se ha utilizado una matriz de distancias arbitraria, la cual es

$$M = \begin{bmatrix} 0 & 1 & 9 & 7 & 3 \\ 4 & 0 & 5 & 9 & 9 \\ 6 & 7 & 0 & 8 & 6 \\ 6 & 5 & 7 & 0 & 6 \\ 8 & 8 & 1 & 4 & 0 \end{bmatrix}$$

Que si la escalamos a fases utilizando la función implementada, nos quedaría

$$M = \begin{bmatrix} 0 & 0 & \frac{\pi}{2} & \frac{\pi}{2} & 0 \\ 0 & 0 & 0 & \frac{\pi}{2} & \frac{\pi}{2} \\ 0 & 0 & 0 & \frac{\pi}{2} & 0 \\ 0 & 0 & \frac{\pi}{2} & 0 & 0 \\ \frac{\pi}{2} & \frac{\pi}{2} & 0 & 0 & 0 \end{bmatrix}$$

Una vez tenemos la matriz escalada, es hora de ejecutar el algoritmo. Primero, obtendremos el resultado dándole a los estados no contemplados la fase de $\frac{3\pi}{2}$, y nos devuelve lo siguiente:

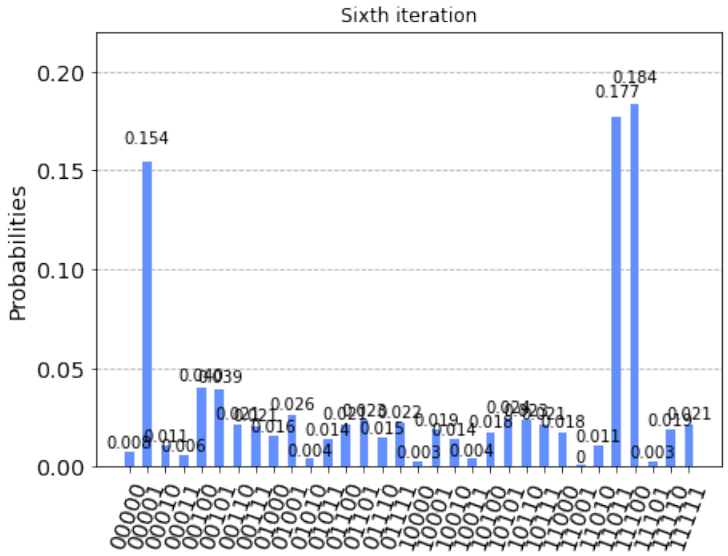

Figura 11: Solución generada por el algoritmo para el primer escenario.

Con la siguiente distribución de caminos

	M
Caminos en fase 0	3
Caminos en fase $\frac{\pi}{2}$	10
Caminos en fase π	11
Caminos en fase $\frac{3\pi}{2}$	8

Tabla 4: Distribuciones de grupos de fase para la primera matriz, dando la fase $3\pi/2$ a los estados no contemplados

Como podemos ver, los estados con una probabilidad notablemente mayor son los estados $|00001\rangle$, $|11011\rangle$ y $|11100\rangle$, y estos los obtenemos en la iteración número seis del operador Grover modificado. Ahora, vamos a obtener el resultado generado por el segundo escenario, en el cual los estados no contemplados obtienen una fase de π.

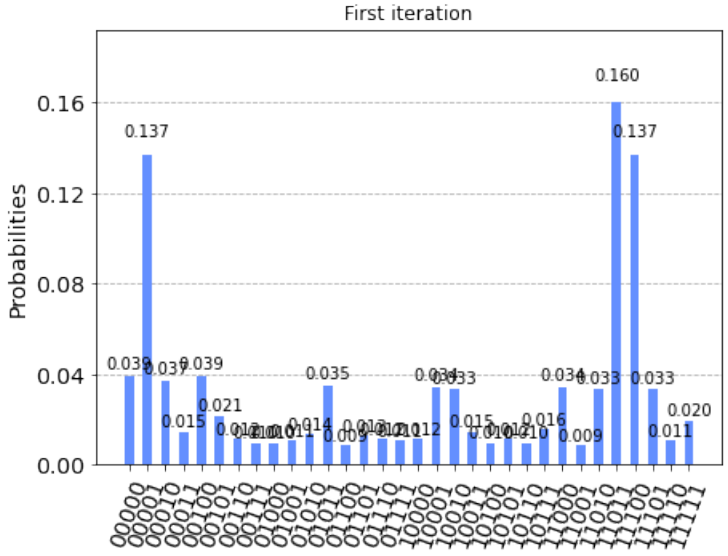

Figura 12: Solución generada por el algoritmo para el segundo escenario.

Con la siguiente distribución de caminos

	M
Caminos en fase 0	3
Caminos en fase $\frac{\pi}{2}$	10
Caminos en fase π	19
Caminos en fase $\frac{3\pi}{2}$	0

Tabla 5: Distribuciones de grupos de fase para la primera matriz, dando la fase π a los estados no contemplados

Al igual que en el escenario anterior, obtenemos los mismos estados como solución, pero esta vez, en la primera iteración, por lo que comprobamos que con este escenario se necesita un menor número de iteraciones del operador.

Ahora vamos a analizar los resultados obtenidos calculando las distancias totales de cada uno. Los caminos correspondientes a los estados $|00001\rangle$, $|11011\rangle$ y $|11100\rangle$ tienen una distancia de 22, 26 y 23 respectivamente. Si observamos todos los caminos posibles, vemos que el camino más corto es el camino asociado al estado $|11010\rangle$, cuyo coste es tan solo de 21, y sin embargo no nos lo da como solución. El motivo por el cual el algoritmo no nos da este estado como solución es porque este camino utiliza un enlace que en la matriz ha sido escalado con fase π, ya que se ha considerado como enlace largo con respecto a los demás de su misma fila. Es por ello que este es un algoritmo heurístico, ya que las soluciones van a ser buenas pero no siempre van a ser las óptimas.

Si obtenemos el número de iteraciones R a partir de la ecuación 15 vemos que no se corresponde con lo obtenido, ya que para ambos casos el número de iteraciones vendría dado por

$$R = \left[\frac{\frac{\pi}{2} - \sqrt{\frac{3}{32}}}{2\sqrt{\frac{3}{32}}} \right],$$

obteniendo un total de 2 iteraciones. Sin embargo, vemos que este parámetro no solo depende del número de caminos solución, sino que depende de la distribución total de caminos, ya que con la primera distribución obtenemos el mejor resultado en la sexta iteración, y en la segunda distribución lo obtenemos en la primera iteración.

Para una mejor demostración del funcionamiento del algoritmo, se ha implementado una función que genera matrices aleatoriamente, con costes comprendidos entre 1 y 9, para comparar cómo varía el resultado dependiendo de la distribución de caminos.

Se han utilizado las siguientes matrices

$$M_1 = \begin{bmatrix} 0 & 6 & 4 & 3 & 2 \\ 3 & 0 & 8 & 3 & 2 \\ 6 & 6 & 0 & 1 & 9 \\ 6 & 9 & 1 & 0 & 3 \\ 1 & 8 & 3 & 4 & 0 \end{bmatrix} \quad M_2 = \begin{bmatrix} 0 & 3 & 7 & 6 & 3 \\ 7 & 0 & 4 & 3 & 2 \\ 8 & 7 & 0 & 2 & 1 \\ 5 & 1 & 2 & 0 & 5 \\ 3 & 4 & 4 & 9 & 0 \end{bmatrix}$$

$$M_3 = \begin{bmatrix} 0 & 1 & 1 & 9 & 9 \\ 3 & 0 & 5 & 4 & 1 \\ 8 & 7 & 0 & 4 & 8 \\ 6 & 4 & 7 & 0 & 3 \\ 6 & 9 & 8 & 8 & 0 \end{bmatrix} \quad M_4 = \begin{bmatrix} 0 & 7 & 9 & 9 & 4 \\ 6 & 0 & 3 & 7 & 7 \\ 1 & 4 & 0 & 7 & 4 \\ 2 & 8 & 8 & 0 & 5 \\ 7 & 8 & 4 & 9 & 0 \end{bmatrix}$$

para las cuales, teniendo en cuenta los estados no contemplados, se ha observado que tienen las siguientes distribuciones de grupos de fase:

	M_1	M_2	M_3	M_4
Caminos en fase 0	4	7	2	1
Caminos en fase $\frac{\pi}{2}$	15	10	6	4
Caminos en fase π	12	15	18	23
Caminos en fase $\frac{3\pi}{2}$	1	0	6	4

Tabla 6: Distribuciones de grupos de fase para las cuatro matrices de ejemplo.

Si observamos las distribuciones de probabilidad que nos da como salida el algoritmo vemos que, efectivamente, el número de estados en fase 0 en cada escenario se corresponde con el número de estados con mayor probabilidad, y a su vez, estos son los que el algoritmo considera de menor coste. Sin embargo, en la matriz 1, podemos observar en la figura 13 que obtenemos como resultado un total de cinco caminos, mientras que solo tenemos cuatro en fase 0, por lo que hay un camino que no debería tener tanta probabilidad. Este es el camino perteneciente al grupo de fase de $3\pi/2$, y es que, cuando el número de caminos en este grupo de fase es pequeño, ya sean uno o dos caminos, estos también salen con una probabilidad elevada. Es por ello que la distribución de grupos de fase es tan importante en este tipo de algoritmo. Si nos fijamos en la figura 14, correspondiente a la matriz 2, podemos ver que al no tener ningún camino perteneciente al grupo de fase $3\pi/2$, solo obtenemos con probabilidad más alta los caminos solución, al igual que en las figuras 15 y 16, correspondientes a las matrices 3 y 4.

El algoritmo estaría funcionando de forma correcta con variaciones en los resultados dependiendo de cada distribución, incluso con matrices generadas aleatoriamente, aunque estas no sigan una distribución de fases perfecta en la que la mayoría de estados pertenezcan al grupo de fase π.

En la tabla 7 podemos ver los costes de cada camino solución, comparados con el coste del camino más corto de cada distribución. Vemos que en todas las pruebas, el camino más corto se encuentra siempre

dentro del conjunto de caminos solución.

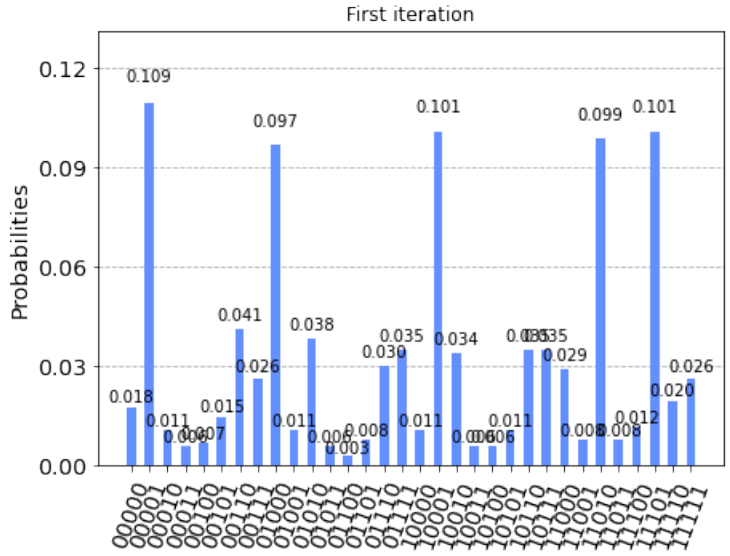

Figura 13: Distribución de probabilidades para la primera matriz.

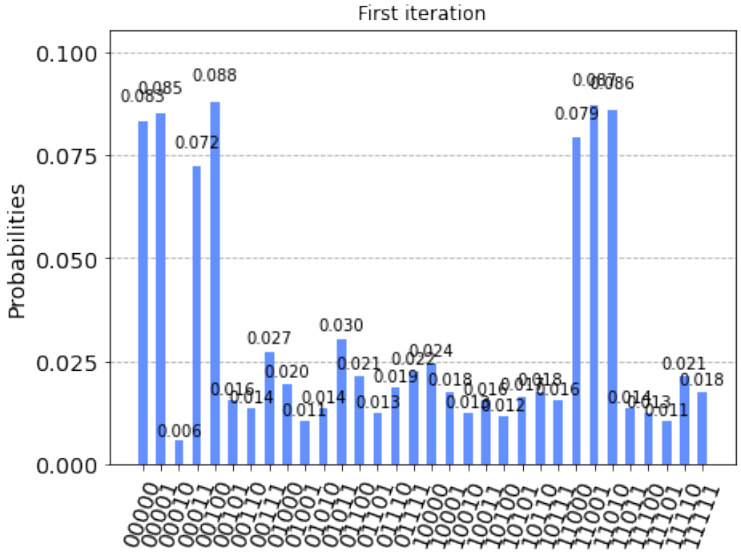

Figura 14: Distribución de probabilidades para la segunda matriz.

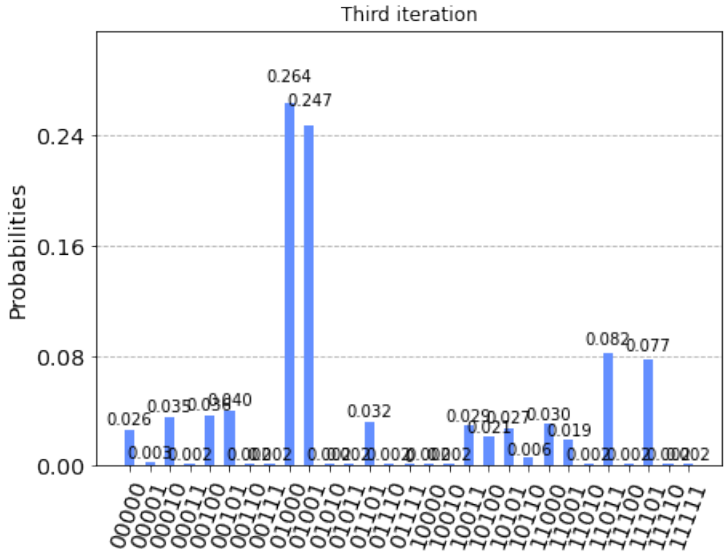

Figura 15: Distribución de probabilidades para la tercera matriz.

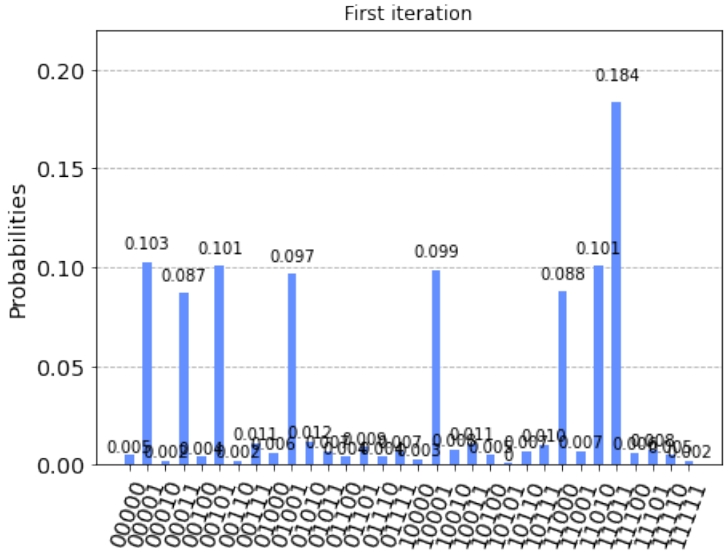

Figura 16: Distribución de probabilidades para la cuarta matriz.

	Costes de caminos solución	Coste de mejor camino
Matriz 1	19, 18, 18, 16	16
Matriz 2	17, 22, 12, 16, 18, 20, 17	12
Matriz 3	20, 16	16
Matriz 4	21	21

Tabla 7: Comparación de caminos solución con mejor camino.

7.3. Escalabilidad del algoritmo

En cuanto a la escalabilidad del algoritmo, vemos que para una distribución de cinco ciudades, tan solo nos han hecho falta utilizar cinco qubits. La parte más costosa es el oráculo, ya que en él se le van dando las fases individualmente a cada par de estados, por lo que a medida que va aumentando el número de ciudades, la implementación de este se va haciendo cada vez más costosa, aunque una vez implementado, es válido para cualquier distribución, y solo habría que modificar la matriz origen de la cual se toman los datos, por lo que solo sería necesario implementarlo una vez.

Ahora, vamos a medir el éxito del algoritmo calculando la distancia entre los resultados obtenidos y los resultados objetivo para los cuatro últimos ejemplos. Para ello, vamos a calcular la distancia euclidiana d entre los vectores de probabilidades, obteniendo así un valor que nos indicará cómo de cerca está la solución obtenida de la solución objetivo. Se calcula mediante la fórmula

$$d = \sqrt{\sum_{k=1}^{n} (x_{k1} - x_{k2})^2}\,, \tag{20}$$

siendo n la longitud del vector y $x_{1,}\, x_2$ los vectores de probabilidades. Una vez calculado, obtenemos los siguientes resultados para cada distribución.

	Distancia euclidiana	Iteración	Iteración calculada (R)
Matriz 1	0.3398	1	2
Matriz 2	0.1824	1	1
Matriz 3	0.3762	3	3
Matriz 4	0.8559	1	4

Tabla 8: Distancia euclidiana entre los vectores de probabilidad obtenidos y los objetivo.

Podemos observar en la tabla 8 que la solución con distancia euclidiana más baja, o solución con más probabilidad de éxito, es la matriz número 2 y esta se obtiene en la primera iteración. Si nos fijamos

en la tabla 6, podemos ver que para esta distribución no hay ningún camino perteneciente al grupo de fase $3\pi/2$, y la mayoría de caminos están repartidos entre los grupos de fase $\pi/2$ y π. En la matriz 1 pasa algo similar, ya que aunque tenemos ese único camino en el grupo de fase $3\pi/2$ que causa el error comentado anteriormente, tenemos la mayoría de estados repartidos en los grupos de fase $\pi/2$ y π. En la matriz 3 vemos que tenemos un mayor número de caminos en fase $3\pi/2$, es por ello que obtenemos la mejor solución en la iteración número 3. Por último, en la matriz 4, vemos que la distancia es mucho mayor, y si nos fijamos en su distribución en la tabla 6, observamos que tan solo tiene un estado solución.

Teniendo todos estos resultados, podemos llegar a la conclusión de que la mayor probabilidad de éxito se da cuando la mayoría de caminos pertenecen a los grupos de fase $\pi/2$ y π y el número de caminos pertenecientes al grupo de fase $3\pi/2$ es cercano a 0. A su vez, si tenemos tan solo un estado solución, la probabilidad de éxito será menor, como hemos visto en la matriz 4. En cuanto al número de iteraciones, vemos que solo coincide en dos ocasiones con respecto al valor de R calculado en la ecuación 15, ya que este valor se calcula con respecto al número de caminos solución, teniendo en cuenta una distribución de ciudades perfecta.

8. Conclusión

8.1. Conclusión

En este trabajo, se ha desarrollado un algoritmo cuántico heurístico para la resolución del problema del vendedor ambulante, para una distribución arbitraria asimétrica de cinco ciudades. Para ello, se ha hecho uso de una variación del algoritmo de Grover. Se han obtenido diversos resultados para ejemplos aleatorios, y tras su análisis, se han obtenido varias conclusiones.

Teniendo en cuenta los resultados que se han obtenido, tanto las distribuciones de probabilidad como la distancia euclidiana de cada ejemplo, podemos darnos cuenta de varios detalles. Una de ellas es que estos dependen mucho de la distribución de caminos, ya que, dependiendo de esta, los estados solución se darán con una mayor o menor probabilidad con respecto a los demás estados, y se necesitará un numero de iteraciones distinto. Además, podemos observar que no solo se obtienen buenas soluciones cuando la distribución de ciudades es la óptima, sino que, aún variando la distribución, el algoritmo es capaz de obtener los caminos solución con una mayor probabilidad que el resto.

A modo de conclusión personal, la realización de este trabajo me ha servido para iniciarme en el mundo de la computación cuántica, desarrollar mi primer algoritmo, y aprender un sinfín de utilidades y aplicaciones de este paradigma. Además, se han conseguido buenos resultados, los cuales se corresponden con lo esperado, e incluso superan lo que en un primer momento se planteó.

Como puntos a destacar, considero que deben estar la codificación utilizada y el oráculo de fase, ya que son dos partes muy importantes del algoritmo, y hacen que este funcione de forma correcta.

8.2. Líneas futuras

En cuanto a líneas futuras, en este trabajo se ha resuelto mediante un algoritmo de computación cuántica el problema del vendedor ambulante para una distribución arbitraria de cinco ciudades, para la cual no haría falta ningún algoritmo, ya que es un número de ciudades bastante pequeño. Lo ideal sería llegar a poder resolver este tipo de problemas para distribuciones más grandes, que no puedan ser tratadas por un ordenador clásico.

Quizá la clave esté en utilizar otro tipo de codificación, ya que hemos visto que esto es esencial en la resolución del problema, o incluso construir un oráculo más eficiente. Otra cuestión importante a resolver sería crear un algoritmo que tenga en cuenta todo el camino completo a la hora de aplicar las fases a los estados, ya que en este trabajo tan solo se tienen en cuenta los tres primeros pasos. Para ello, tendríamos que aplicar el algoritmo teniendo en cuenta un mayor número de grupos de fase, ya que con cuatro no nos bastaría. Esto es el principio de lo que puede llegar a ser una gran revolución, por lo que a base de trabajo, estudio y esfuerzo, a lo largo del tiempo llegarán mejores resultados.

9. Anexos

9.1. Código en Python del algoritmo

```
# importing Qiskit
from qiskit import IBMQ, Aer, assemble, transpile
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister

# import basic tools
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import random as rd
from qiskit.visualization import plot_histogram
from math import pi, sqrt

#Función que asigna una fase a cada estado posible con 1 qubit
#Los valores de entrada se obtienen de la matriz de fases
def coste1q(a,b):
        qc = QuantumCircuit(1)
        qc.p(a,0)
        qc.x(0)
        qc.p(b,0)
        return qc

# Función que genera matriz de costes aleatoria
def generarMatriz():
        m1 = np.random.rand(5,5)
        m1 = m1*10
        m1 = np.round(m1)
        for i in range (5):
                m1[i,i] = np.nan
        for i in range (5):
                for j in range (5):
                        if(m1[i,j]==0):
                                m1[i,j] = rd.randint(1, 9)
                        elif(m1[i,j]==10):
                                m1[i,j] = rd.randint(1, 9)
        return m1


# Función para pasar la matriz de costes a fases
def escalarMatriz(m2):
        matrix = pd.DataFrame(m2)
        maxValues = matrix.max(axis = 1)
        minValues = matrix.min(axis = 1)
```

```
        for i in range(5):
                a = maxValues[i]
                b = minValues[i]
                c = (a-b)/2
                for j in range(5):
                        d = m2[i][j]
                        if(d>=b) and (d<=b+c):
                                m2[i][j]=0
                        elif (d>=b+c+1) and (d<=a):
                                m2[i][j]=pi/2
        return m2

#Definimos función que da las fases a los diferentes estados
def phasing():
        qc = QuantumCircuit(5)

        cPhase0000=coste1q(m[0][1]+m[1][2]+m[2][3],m[0][1]+m[1][2]+m
            [2][4]).to_gate().control(4)
        qc.x([1,2,3,4])
        qc.append(cPhase0000,[4,3,2,1,0])
        qc.x([1,2,3,4])

        cPhase0001=coste1q(m[0][1]+m[1][3]+m[3][4],m[0][1]+m[1][3]+m
            [3][2]).to_gate().control(4)
        qc.cx(1,[2,3,4])
        qc.append(cPhase0001,[4,3,2,1,0])
        qc.cx(1,[2,3,4])

        cPhase0010=coste1q(m[0][1]+m[1][4]+m[4][2],m[0][1]+m[1][4]+m
            [4][3]).to_gate().control(4)
        qc.cx(2,[1,3,4])
        qc.append(cPhase0010,[4,3,2,1,0])
        qc.cx(2,[1,3,4])

        cPhase0011=coste1q(pi,pi).to_gate().control(4)
        qc.mcx([1,2],3)
        qc.mcx([1,2],4)
        qc.append(cPhase0011,[4,3,2,1,0])
        qc.mcx([1,2],3)
        qc.mcx([1,2],4)

        cPhase0100=coste1q(m[0][2]+m[2][3]+m[3][4],m[0][2]+m[2][3]+m
            [3][1]).to_gate().control(4)
        qc.cx(3,4)
        qc.cx(3,1)
```

```
qc.cx(3,2)
qc.append(cPhase0100,[4,3,2,1,0])
qc.cx(3,4)
qc.cx(3,1)
qc.cx(3,2)

cPhase0101=coste1q(m[0][2]+m[2][4]+m[4][1],m[0][2]+m[2][4]+m
    [4][3]).to_gate().control(4)
qc.mcx([1,3],2)
qc.mcx([1,3],4)
qc.append(cPhase0101,[4,3,2,1,0])
qc.mcx([1,3],2)
qc.mcx([1,3],4)

cPhase0110=coste1q(m[0][2]+m[2][1]+m[1][3],m[0][2]+m[2][1]+m
    [1][4]).to_gate().control(4)
qc.mcx([2,3],1)
qc.mcx([2,3],4)
qc.append(cPhase0110,[4,3,2,1,0])
qc.mcx([2,3],1)
qc.mcx([2,3],4)

cPhase0111=coste1q(pi,pi).to_gate().control(4)
qc.mcx([1,2,3],4)
qc.append(cPhase0111,[4,3,2,1,0])
qc.mcx([1,2,3],4)

cPhase1000=coste1q(m[0][3]+m[3][4]+m[4][1],m[0][3]+m[3][4]+m
    [4][2]).to_gate().control(4)
qc.cx(4,[3,2,1])
qc.append(cPhase1000,[4,3,2,1,0])
qc.cx(4,[3,2,1])

cPhase1001=coste1q(m[0][3]+m[3][1]+m[1][2],m[0][3]+m[3][1]+m
    [1][4]).to_gate().control(4)
qc.mcx([4,1],3)
qc.mcx([4,1],2)
qc.append(cPhase1001,[4,3,2,1,0])
qc.mcx([4,1],3)
qc.mcx([4,1],2)

cPhase1010=coste1q(m[0][3]+m[3][2]+m[2][4],m[0][3]+m[3][2]+m
    [2][1]).to_gate().control(4)
qc.mcx([4,2],3)
qc.mcx([4,2],1)
qc.append(cPhase1010,[4,3,2,1,0])
```

```
        qc.mcx([4,2],3)
        qc.mcx([4,2],1)

        cPhase1011=coste1q(pi,pi).to_gate().control(4)
        qc.mcx([4,2,1],3)
        qc.append(cPhase1011,[4,3,2,1,0])
        qc.mcx([4,2,1],3)

        cPhase1100=coste1q(m[0][4]+m[4][1]+m[1][2],m[0][4]+m[4][1]+m
            [1][3]).to_gate().control(4)
        qc.mcx([4,3],2)
        qc.mcx([4,3],1)
        qc.append(cPhase1100,[4,3,2,1,0])
        qc.mcx([4,3],2)
        qc.mcx([4,3],1)

        cPhase1101=coste1q(m[0][4]+m[4][2]+m[2][3],m[0][4]+m[4][2]+m
            [2][1]).to_gate().control(4)
        qc.mcx([4,3,1],2)
        qc.append(cPhase1101,[4,3,2,1,0])
        qc.mcx([4,3,1],2)

        cPhase1110=coste1q(m[0][4]+m[4][3]+m[3][1],m[0][4]+m[4][3]+m
            [3][2]).to_gate().control(4)
        qc.mcx([4,3,2],1)
        qc.append(cPhase1110,[4,3,2,1,0])
        qc.mcx([4,3,2],1)

        cPhase1111=coste1q(pi,pi).to_gate().control(4)
        qc.append(cPhase1111,[4,3,2,1,0])
        return qc

#Definimos nuestra función que hará de amplificador o difusor (Fuente:
    qiskit.org)
def diffuser(nqubits):
        qc = QuantumCircuit(nqubits)
        # Apply transformation |s> -> |00..0> (H-gates)
        for qubit in range(nqubits):
        qc.h(qubit)
        # Apply transformation |00..0> -> |11..1> (X-gates)
        for qubit in range(nqubits):
        qc.x(qubit)
        # Do multi-controlled-Z gate
        qc.h(nqubits-1)
        qc.mct(list(range(nqubits-1)), nqubits-1)
        # multi-controlled-toffoli
```

```
        qc.h(nqubits-1)
        # Apply transformation |11..1> -> |00..0>
        for qubit in range(nqubits):
        qc.x(qubit)
        # Apply transformation |00..0> -> |s>
        for qubit in range(nqubits):
        qc.h(qubit)
        # We will return the diffuser as a gate
        U_s = qc.to_gate()
        U_s.name = "I"
        return U_s

#Por último, realizamos la simulación

#Matriz de costes introducida a mano, o generada por la función
    aleatoriamente
#costes = [[np.nan,1,9,7,3],[4,np.nan,5,9,9],[6,7,np.nan,8,6],[6,5,7,np.
    nan,6],[8,8,1,4,np.nan]]
costes = generarMatriz()
costes1 = np.copy(costes)

#La pasamos por la función escalarMatriz
m = escalarMatriz(costes1)

tsp5 = QuantumCircuit(5)
tsp5.h([0,1,2,3,4])

phase = phasing().to_gate()
phase.name="C"

for i in range (1):
        tsp5.append(phase,[0,1,2,3,4])
        tsp5.append(diffuser(5),[0,1,2,3,4])

tsp5.measure_all()
aer_sim = Aer.get_backend('aer_simulator')
transpiled_grover_circuit = transpile(tsp5, aer_sim)
qobj = assemble(transpiled_grover_circuit)
results = aer_sim.run(qobj).result()
counts = results.get_counts()
plot_histogram(counts,title='First␣iteration')
```

Referencias

[1] Grover, L. K. (1997). Quantum mechanics helps in searching for a needle in a haystack. Physical review letters, 79(2), 325.

[2] Bang, J., Ryu, J., Lee, C., Yoo, S., Lim, J., & Lee, J. (2012). A quantum heuristic algorithm for the traveling salesman problem. Journal of the Korean Physical Society, 61(12), 1944-1949.

[3] Nielsen, M. A., & Chuang, I. (2002). Quantum computation and quantum information.

[4] Shor's Algorithm. Qiskit Textbook. https://qiskit.org/textbook/chalgorithms/shor.html

[5] Grover's Algorithm. Qiskit Textbook. https://qiskit.org/textbook/chalgorithms/grover.html

[6] López, A. G., & López, J. G. (2005). Criptografía cuántica. Departamento Matemática Aplicada. Escuela Universitaria de Informática. Universidad Politécnica de Madrid. España.

[7] Benioff, Paul. (1980). The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines. Journal of Statistical Physics. 22. 563-591. 10.1007/BF01011339.

[8] Feynman, R. P. (2018). Simulating physics with computers. In Feynman and computation (pp. 133-153). CRC Press.

[9] Mermin, D. (2006). Breaking rsa encryption with a quantum computer: Shor's factoring algorithm. Lecture notes on Quantum computation, 481-681.

[10] Problema del viajante. (2022). Wikipedia, La enciclopedia libre. https://es.wikipedia.org/w/index.php?title=Problema_del_viajante&oldid=143744283.