eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
ZIENTZIA ETA TEKNOLOGIA FAKULTATEA
FACULTAD DE CIENCIA Y TECNOLOGÍA

Gradu Amaierako Lana / Trabajo Fin de Grado
Fisikako Gradua / Grado en Física

# Algoritmos cuánticos y Quantum Ant Colony Optimization

Egilea/Autor/a:
Mikel Garcia de Andoin Bolaño
Zuzendaria/Director/a:
Javier Echanobe Arias

Leioa, 2020ko maiatzaren 26a / Leioa, 26 de mayo de 2020

**Abstract**

Quantum computation is on the verge of becoming the new technology for solving time- and energy-consuming problems. To make use of the full potential of quantum computers one must implement algorithms that are significantly different from the ones used on digital computers. This work serves as an introduction to both quantum computing and algorithms. Starting from the basis of the quantum physics, I expose each of the steps needed to understand the mathematics beneath this topic. Using this basis, I develope one quantum algorithm from scratch. Based on a previously proposed Quantum Ant Colony Optimization (QACO) algorithm, I propose an improved algorithm that can be implemented on a real quantum computer with a new guided exploration strategy. The benchmarks made by simulating the quantum circuit shows that the time to solve a problem using QACO could outperform greatly other classical bio-inspired algorithms, in particular the one QACO is based on, the Ant Colony Optimization.

# Contenidos

# 1. Objetivo y desarrollo

La computación cuántica estudia los procesos por los cuales se computa la información en sistemas del ámbito de la mecánica cuántica. En la computación digital clásica los datos están codificados a través de un sistema binario, representados por diferencias de potencial en un circuito electrónico. Frente a esto, la computación cuántica presenta un paradigma en el que la la información se encuentra en el estado de un sistema físico sujeto a las reglas de la mecánica cuántica.

En un computador digital clásico el resultado del cálculo se obtiene tras del paso de la información a través de una serie de puertas lógicas bien conocidas. Estas puertas funcionan de una manera determinista, en el sentido de que, salvo "bugs", se conocen perfectamente los estados de los bits a la salida. Esto no ocurre con la computación cuántica. En lugar de bits, los computadores cuánticos codifican la información a través de qubits en un estado de superposición. A la hora de realizar los cálculos se usan puertas cuánticas, lo que permite realizar operaciones sobre cada uno de los estados superpuestos de manera simultánea. Esta característica da lugar a crear nuevas estrategias a la hora del diseño de algoritmos cuánticos, siendo algunos mucho más eficientes que cualquier algoritmo anterior.

Aunque a primera vista parezca una rama de computación todavía poco desarrollada, lo cierto es que se empezó a plantear en los 80. Antes del año 2000 ya se habían desarrollado dos de los algoritmos cuánticos más reconocidos, el algoritmo de descomposición en factores de Shor [1] y el algoritmo de búsqueda de Grover [2]. Hoy en día, el catálogo de algoritmos cuánticos ha crecido hasta al menos 63 [3]. El desarrollo de estos algoritmos requiere de un conocimiento previo de fundamentos de mecánica cuántica y matemáticas, pero sobre todo, de una capacidad de pensar más allá de los límites de la programación de algoritmos clásicos.

Viendo la historia de los computadores clásicos y la electrónica digital, parecería que en 40 años desde la presentación de las bases de la computación cuántica deberíamos haber pasado de los mainframes (como lo eran el ENIAC o la serie IBM 700/7000) a computadores cuánticos personales (por ejemplo, los equivalentes a los primeros modelos de Commodore Amiga o Macintosh, basados en motorola 68000). Esta evolución predicha por la ley de Moore, no parece tener una analogía con la computación cuántica. Esto se debe principalmente a que el objetivo de estos computadores por el momento no está siendo la de conseguir un procesador universal, sino la de tener circuitos específicos para resolver un tipo de problema. Para ello, y por la complejidad de los problemas tecnológicos a los que nos enfrentamos en la actualidad, es crucial el desarrollo de algoritmos cuánticos que mejoren el tiempo de cómputo de los problemas.

En este sentido, este trabajo se ha centrado en el desarrollo de algoritmos cuánticos, en concreto, en el estudio y adaptación de un algoritmo cuántico de colonia de hormigas (QACO). Primero, se han presentado las bases sobre las que se construyen estos algoritmos, dando una pequeña introducción también a los algoritmos basados en hormigas (Cap.2). Se ha estudiado en profundidad una propuesta de algoritmo cuántico, y sobre ella, se ha propuesto una mejora que amplía la cantidad de problemas resolubles con este algoritmo y permite una implementación real en un computador cuántico (Cap.3). Para mejorar el rendimiento del algoritmo propuesto, se ha hecho una optimización de los parámetros de entrada del algoritmo y se ha comparado el algoritmo cuántico propuesto con su versión clásica. Se han comentado algunos problemas que pueden ser resueltos usando QACO, mostrando la eficiencia del algoritmo propuesto (Cap.4). Al final de este trabajo se comentan los resultados obtenidos y posibles líneas de trabajo futuro (Cap.5).

# 2. Fundamentos teóricos

La computación cuántica se presenta en un paradigma completamente distinto al de la computación clásica. Una idea tan básica y aceptada de forma automática como la existencia de una memoria permanente a la que poder acceder, no existe en el mundo cuántico. Mientras que en computación clásica se puede conocer con exactitud el estado de un bit; no es posible hacer esto con un qubit, su análogo en computación cuántica. Además de esto, en un sistema compuesto por varios qubits el estado de cada uno de ellos no es independiente, sino que dependen de la superposición de estados entrelazados. Esto cambia por completo la forma de pensar a la hora de enfrentarse a este tipo de computación. Por ello, se presenta a continuación una serie de ideas básicas necesarias para introducirse en el tema.

En este trabajo se usará la notación de braket y se usarán resultados básicos de la mecánica cuántica y teoría de álgebra vectorial. En este capítulo se condensan todos los aspectos matemáticos necesarios para poder entender este trabajo. En concreto, se da una pequeña introducción a algunos conceptos básicos de la mecánica y computación cuántica, para luego introducir los algoritmos cuánticos. Al final del capítulo se da también una descripción general de los algoritmos de hormigas. Sin embargo, puede que haya algunas ideas no triviales sin comentar. Para resolver dudas que requieran una explicación técnica o en profundidad recomiendo el clásico libro de J. J. Sakurai "Modern Quantum Mechanics" [4] o "Quantum Computation and Quantum Information" de M. A. Nielsen e I. L. Chuang [5], los cuales han servido de referencia para escribir este capítulo.

## 2.1 Mecánica cuántica

La computación cuántica se basa en los principios y las leyes de la mecánica cuántica. En concreto, para realizar las operaciones sobre los datos, se deja evolucionar un sistema a partir de un estado conocido, para después medir el resultado. La evolución temporal de un sistema se describe a través de la archiconocida ecuación de Schrödinger dependiente del tiempo

$$i\hbar\frac{\partial\Psi(\vec{r},t)}{\partial t} = H(\vec{r},t)\Psi(\vec{r},t) \Leftrightarrow \; i\hbar\frac{\partial\,|\Psi(t)\rangle}{\partial t} = \hat{H}(t)\,|\Psi(t)\rangle\,, \tag{1}$$

donde el hamiltoniano $\hat{H}$ representa las interacciones con las fuerzas que actúan sobre el sistema.

Uno de los postulados de la mecánica cuántica permite representar un sistema a través de un vector de un espacio de Hilbert, donde los coeficientes son números complejos. Este vector tiene que cumplir la condición de normalización de la probabilidad

$$\int_{-\infty}^{\infty} |\Psi(x)|^2\, dx = 1 \;\Leftrightarrow\; \langle\Psi|\Psi\rangle = 1. \tag{2}$$

Los operadores en el espacio de Hilbert están representados por matrices, y los estados por vectores. De esta manera, la acción de un operador sobre un estado se hace a través de la multiplicación de ambos. Los operadores que corresponden a observables serán por definición matrices hermíticas (con valores propios reales). La evolución temporal del sistema también se puede representar a través de la aplicación de un operador de evolución temporal $\hat{U}$(t). Para

mantener la simetría temporal del sistema, $\hat{U}$ tiene que se un operador unitario. La definición de un operador unitario es que la matriz del operador sea una tal que su transpuesta conjugada sea igual que su inversa

$$U^{\dagger} = U^{-1} \Leftrightarrow U^{\dagger}U = \mathbb{1} \Leftrightarrow U \text{ es unitaria.} \tag{3}$$

Si se quiere modificar esta evolución, se tiene que hacer aplicando otra transformación $\hat{\mathcal{V}}$, tal que la nueva evolución sea $\hat{U}'=\hat{U}\hat{\mathcal{V}}$, donde $\hat{U}'$ tendrá que seguir siendo unitaria.

### 2.1.1 Sistemas compuestos de varias partículas

En computación cuántica es común usar sistemas en los que se tiene más de una partícula. El estado en un computador cuántico está dado por la composición de los estados de varias partículas distinguibles. Para representar correctamente estos estados es necesario primero obtener una base adecuada. En general, dado que cada partícula es independiente del estado de las otras, se puede escribir un estado genérico como el producto de Kronecker de todas las partículas

$$|\Psi\rangle = |\varphi^1\rangle \otimes |\varphi^2\rangle \otimes \cdots \otimes |\varphi^n\rangle \equiv |\varphi^1, \varphi^2, \ldots, \varphi^n\rangle\,, \tag{4}$$

donde el estado de cada partícula $|\varphi^i\rangle$ describe el estado del sistema en su respectivo espacio de Hilbert. Así, el estado completo del sistema estará formado por la unión de los estados de todas las partículas. La dimensión del espacio de Hilbert del sistema compuesto será

$$dim(H_\Psi) = \prod_{i=1}^{n} d_i, \tag{5}$$

con $d_i$ la dimensión del espacio de Hilbert de la partícula i.

De la misma manera, los operadores en sistemas de varias partículas se forman a través del producto de Kronecker del operador sobre cada una de las partículas. Sean operadores $A^{(i)}$ que actúan sobre la partícula $i$, su composición viene dada por

$$A = A^{(1)} \otimes A^{(2)} \otimes \cdots \otimes A^{(n)}. \tag{6}$$

La forma en la que actúan los operadores sobre cada una de las partículas es idéntica a la de sistemas de una sola partícula. Solo que en este caso la acción de un operador sobre una partícula tiene que dejar invariante el estado del resto. Por ejemplo, sea un sistema formado por 2 partículas

$$|\Psi\rangle = |\nu\rangle \otimes |\mu\rangle\,, \tag{7}$$

y un operador A que actúa sobre la primera partícula tal que

$$A\,|\varphi\rangle = A_\varphi\,|\varphi'\rangle\,. \tag{8}$$

La acción de este operador A sobre el sistema se puede escribir como

$$A\,|\Psi\rangle = (A \otimes \mathbb{1})(|\nu\rangle \otimes |\mu\rangle) = A\,|\nu\rangle \otimes \mathbb{1}\,|\mu\rangle = A_\nu\,|\nu'\rangle \otimes |\mu\rangle = A_\nu\,|\nu', \mu\rangle\,. \tag{9}$$

En el caso de los sistemas de los computadores cuánticos, normalmente se tienen solo 2 estados por cada partícula, el estado base $|0\rangle$ y un estado excitado $|1\rangle$. Así, cada qubit del

sistema tiene asociado un estado que se puede expresar como

$$|\Psi\rangle \equiv \begin{pmatrix}\alpha\\ \beta\end{pmatrix},\ \text{con}\ \alpha \equiv \langle 0|\Psi\rangle\ ^{a},\ \beta \equiv \langle 1|\Psi\rangle\,,\ \alpha,\beta \in \mathbb{C},\ \alpha^2+\beta^2 = 1. \tag{10}$$

Esto hace que la base que describa el sistema tenga una dimensión de $2^n$, siendo $n$ el número de qubits. El estado cuántico del sistema será una superposición de cada uno de los estados de una base. Esta peculiaridad permite elegir una base de los estados que resulta familiar, una base en forma de número binario natural. Por ejemplo, para un sistema de 2 qubits, la base se puede escribir como

$$\Big\{\, |\varphi_1\rangle \otimes |\varphi_2\rangle \,\Big\} = \Big\{|00\rangle\,, |01\rangle\,, |10\rangle\,, |11\rangle\Big\}. \tag{11}$$

En este trabajo los estados y las matrices de los operadores se darán usando la base binaria natural o base computacional, con el 0 escrito primero. A la hora de dibujar los circuitos cuánticos, el qubit inferior representa el bit más significativo y el superior el menos significativo.

## 2.2 Computación cuántica

La computación cuántica se basa en la aplicación de operadores $\hat{\mathcal{V}}$ a través de un circuito cuántico, tales que a la salida se tenga el estado cuántico buscado a partir de cierta entrada. Esta definición general es completamente análoga a la de la computación clásica. Sin embargo, la diferencia principal es que las transformaciones del estado inicial no se hacen a través de puertas lógicas, sino a través de puertas cuánticas.

Las puertas lógicas se pueden describir a través de una tabla de verdad, las puertas cuánticas se describen de forma análoga usando matrices. Por lo visto en la sección anterior, estas matrices tienen que representar operadores unitarios reversibles. Por eso, en computación cuántica no existe como tal la puerta AND o la OR, ya que para esto se necesitaría una puerta irreversible. Por este motivo, no es posible hacer operaciones de "fan-out" ni "fan-in". Dicho de otra manera, no se puede hacer una copia de un estado cuántico.

Estamos acostumbrados a que los circuitos electrónicos digitales trabajen entre dos estados 0 o 1, dependiendo del valor del potencial al que se encuentre cada punto del circuito. En computación cuántica la codificación de la información es más amplia. Aunque se puede trabajar con sistemas de más niveles de energía, la mayor parte de la computación cuántica se centra en sistemas de dos estados.

Los estados cuánticos también se pueden representar a través de una esfera de Bloch (Fig.7), en la que la intersección con los ejes principales representan los estados propios de los operadores de momento angular. Dicho de otra manera, estos puntos son los autovectores de las respectivas matrices de Pauli $\sigma_x$, $\sigma_y$ y $\sigma_z$.

Otra peculiaridad es que un estado cuántico se destruye en el momento en el que se mide. Esto se puede entender fácilmente recurriendo a la manida paradoja del Gato de Schrödinger, en la que una vez se abre la caja, el gato permanece en el estado en el que se encontró al abrir la caja. En un computador cuántico, esto hace que solo se pueda hacer una medición de cada qubit. Sabiendo que el estado del qubit se encuentra en la superficie de la esfera de Bloch, esto hace que la información que se puede extraer de un estado esté limitada a su proyección sobre uno de sus ejes.

[a] Notación braket para el producto escalar de vectores en el espacio de Hilbert

### 2.2.1 Puertas cuánticas

A pesar de estas restricciones a primera vista insalvables, se puede desarrollar una teoría de computación cuántica completa. Estas puertas cuánticas deberán ser reversibles y unitarias, por ello, cada una de ellas actúa a través de una rotación del estado cuántico sobre la esfera de Bloch. Los circuitos cuánticos que implementen los algoritmos estarán formados por la aplicación sucesiva de operaciones sobre los qubits del sistema. Se presentan a continuación algunas puertas cuánticas con las que poder construir los circuitos cuánticos:

- **Pauli X, Y, Z**
  Estas puertas sirven para proyectar el estado cuántico de un qubit sobre cada uno de ejes principales. Representan giros sobre cada uno de los ejes principales. Por la forma en la que actúa la puerta $\sigma_x$ (Eq.12), suele identificarse como la puerta NOT cuántica, ya que aplica las transformaciones $|0\rangle \rightarrow |1\rangle$ , $|1\rangle \rightarrow |0\rangle$.

$$\boxed{X} \quad \sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \boxed{Y} \quad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad \boxed{Z} \quad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{12}$$

- **Hadamard**
  La puerta Hadamard (Eq.13) representa la operación de mezcla de un estado. Dado un estado puro $|0\rangle$ o $|1\rangle$ a la entrada, a la salida se tendrá un estado mezcla en el que la probabilidad de obtener el estado base o el excitado es la misma.

$$\boxed{H} \quad H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{13}$$

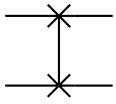
- **Desplazamiento de fase**
  Esta puerta (Eq.14) modifica la fase del estado de un qubit, que es la diferencia de fase entre los coeficientes complejos $\alpha$ y $\beta$ de la ecuación (Eq.10). Dado que la probabilidad viene dada por el módulo cuadrado del estado, esta operación no modifica la probabilidad de obtener los estados de la base computacional.

$$\boxed{\theta} \quad R_\theta = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}, \quad \boxed{\pi/8} \quad T = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}. \tag{14}$$

- **Puertas controladas**
  Las puertas controladas aplican una operación sobre un número dado de qubits objetivo en función del estado de los qubits de control. Estas puertas representan todas las posibles operaciones condicionales, sin las cuales no sería posible desarrollar una teoría computacional completa. Las puertas controladas más simples actúan sobre 2 qubits. Si el qubit de control está en un estado excitado, se aplicará una puerta U sobre el qubit objetivo. Esto se puede extender tanto a puertas U que actúen sobre varios qubits como a puertas con múltiples qubits de control, sin más que componiendo varias puertas controladas básicas. Una de las puertas controladas más útiles es la puerta cnot (Eq.15), que aplica una puerta $\sigma_x$ de forma condicional.

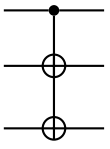
$$\text{C-U} = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & U_{11} & U_{12} \\ & & U_{21} & U_{22} \end{pmatrix}, \quad \text{cnot} = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 0 & 1 \\ & & 1 & 0 \end{pmatrix}. \tag{15}$$

- **Swap**
  La operación de Swap (Eq.16) intercambia los estados de los dos qubits sobre los que actúa.



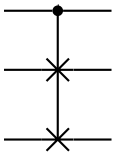
$$Swap = \begin{pmatrix} 1 & & & \\ & 0 & 1 & \\ & 1 & 0 & \\ & & & 1 \end{pmatrix}. \tag{16}$$

- **Toffoli**
  La puerta Toffoli (Eq.17) en computación clásica es una puerta universal, que además es reversible. En computación cuántica, esta puerta es idéntica a su versión clásica, salvo que en este caso tiene una utilidad importante. Dependiendo de la configuración de los qubits de entrada, esta puerta puede usarse para simular la puerta NAND o la operación de fan-out. De esta manera, se podrían simular circuitos digitales en uno cuántico, aunque la utilidad de esto sea prácticamente nula.



$$toff = \begin{pmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & & 1 & & & & & \\ & & & 1 & & & & \\ & & & & 1 & & & \\ & & & & & 1 & & \\ & & & & & & 0 & 1 \\ & & & & & & 1 & 0 \end{pmatrix}. \tag{17}$$

- **Fredkin**
  Al igual que con la puerta Toffoli, la puerta Fredkin (Eq.18) es también una puerta universal reversible. La puerta Fredkin aplica la operación de swap sobre dos qubits controlada por el tercero. Esta puerta será importante en el desarrollo posterior del algoritmo QACO.



$$fred = \begin{pmatrix} 1 & & & & & & & \\ & 1 & & & & & & \\ & & 1 & & & & & \\ & & & 1 & & & & \\ & & & & 1 & & & \\ & & & & & 0 & 1 & \\ & & & & & 1 & 0 & \\ & & & & & & & 1 \end{pmatrix}. \tag{18}$$

Siguiendo la comparación con la computación clásica, aquí también se busca un conjunto reducido de puertas cuánticas pequeño tal que puedan realizar todas las operaciones posibles. El conjunto más pequeño posible que lo consigue está formado por las puertas que actúan sobre un qubit y una puerta controlada, la más simple siendo la puerta cnot. Se puede reducir aún más este conjunto y reducirlo a 4 puertas: Hadamard, $R_{\pi/2}$, $R_{\pi/4}$ y cnot. Sin embargo, para realizar una operación arbitraria con precisión absoluta, habría que usar un número infinito de estas puertas. Dado que esto no es posible, las puertas que no se pueden obtener por combinación directa de estas puertas, se aproximan hasta cierta precisión que no afecte al resultado.

### 2.2.2 Soporte físico para la computación cuántica

Uno de los principales problemas que impide la extensión de la computación cuántica es la dificultad en la realización física de estos computadores. Al igual que en computación clásica, un circuito cuántico tiene que permitir preparar el estado de entrada, hacer que evolucione a través de operaciones controlables y medir el estado a la salida. Además, circuitos cuánticos se tienen que diseñar de tal manera que sean resistentes al ruido. En general, cuantas más puertas tenga un circuito cuántico, más se verá afectado por este.

Además de esto, el ruido térmico es un gran problema a la hora de mantener estable un estado. Se suele trabajar en sistemas con una diferencia entre niveles de energía del orden de unos pocos eV, o incluso con diferencias de energía dadas por el desdoblamiento causado por el efecto de la estructura fina e hiperfina. Esto hace que se necesite trabajar a una temperatura lo más cercana posible a 0K.

Una de las formas de construir un circuito cuántico es usando iones atrapados en una trampa magnética. Dado que los iones en una trampa magnética oscilan con una cierta frecuencia alrededor de un punto de equilibrio. Estas vibraciones (fonones) ocurren en los niveles de energía más bajos de un oscilador armónico. Bajando la temperatura de los átomos se consigue que la contribución de la energía cinética sea despreciable frente a la de los estados de spin, por lo que trabajando a nivel de estructura fina se puede tener un qubit con los electrones de las capas semillenas. La aplicación de las puertas cuánticas se hace a través de la incidencia de pulsos de láser monocromáticos de una duración controlada. Este pulso afecta a los niveles de energía de los fonones del átomo, que interactúa a su vez con los estados de espín al desexcitarse.

Otra manera de realizar un circuito cuántico es usando fotones atrapados en cavidades de un cristal. Dado que los fotones pueden interactuar entre sí, es posible realizar puertas cuánticas sobre un qubit haciendo incidir dos láseres sobre una de estas cavidades. Sin embargo, para realizar este tipo de operaciones controlables, es necesario que la cavidad del cristal tenga unas propiedades ópticas no lineales específicas. Este tipo de circuitos son uno de los mas prometedores, ya que al tratarse de fotones en cristales, permitirían crear computadores cuánticos a temperatura ambiente. Sin embargo, aunque se haya demostrado su viabilidad teórica, los métodos de fabricación de este tipo de cristales no cumplen todavía con las exigencias de precisión requeridas para tener unos resultados óptimos [6].

Los computadores cuánticos basados en iones atrapados son una realidad a fecha de hoy, teniendo sistemas cuánticos entrelazados de hasta 20 qubits [7]. Otro tipo distinto de computadores cuánticos consiguen elevar el número de qubits programables hasta los 54, usando una implementación en superconductores [8]. Esto se encuentra en el límite en el cual la simulación de los circuitos cuánticos deja de ser viable, el llamado “Quantum Supremacy”, cuya cota inferior parece estar en 54 qubits [9].

Es importante mencionar también otro tipo de computadores cuánticos basados en “Quantum Annealing”, los cuales implementan algoritmos estocásticos para encontrar soluciones a problemas de optimización [10]. Este tipo de computadores permite diseñar sistemas con miles de qubits, llegando hasta la cifra de los 2000 qubits [11].

Dado que el objetivo de este trabajo es solamente estudiar los algoritmos cuánticos desde un punto de vista teórico, no se va a profundizar acerca del soporte físico de los computadores cuánticos. Todo el análisis hecho en este trabajo se ha hecho suponiendo un sistema ideal, en el que no se tiene en cuenta ni las restricciones de hardware ni el ruido del sistema.

## 2.3 Algoritmos cuánticos

Una de las aspiraciones principales de los algoritmos cuánticos es la de conseguir mejorar la complejidad computacional de los algoritmos clásicos o poder resolver problemas hoy en día inabordables. Muchos algoritmos que se usan en computación clásica se encuentran en el límite de la eficiencia, en el sentido de que existen problemas que hoy en día no se pueden resolver en un tiempo razonable. Poniendo el ejemplo de los algoritmos de ordenación, es imposible encontrar un algoritmo cuyo peor caso sea más rápido que $O(n \cdot \log n)$ [12, p. 167].

Hay algunos tipos de problemas que pueden ser demasiado costosos de resolver si se quiere obtener un resultado exacto. Frente a este problema, se tienen algoritmos estocásticos, que usan métodos en los que influye la aleatoriedad para obtener resultados aproximados. Estos algoritmos aceleran en gran medida cálculos que de otra manera no serían viables para su cómputo. Al final de este capítulo (Sec.2.5) se desarrolla uno de estos algoritmos, el algoritmo de colonia de hormigas. Aunque los algoritmos cuánticos no sean todos estocásticos, sí que es conveniente tenerlos en mente a la hora de entender como funcionan.

Debido a la propia naturaleza de los sistemas cuánticos, los resultados de un algoritmo cuántico exacto serán correctos hasta cierto margen de incertidumbre. Esto supone una diferencia crucial con los algoritmos estocásticos, donde no es posible asegurar que el resultado obtenido sea el correcto. Sin embargo, para obtener un resultado con una certeza del 100%, sería necesario lanzar el algoritmo cuántico un número infinito de veces. Esto hace que sea necesario un estudio del número de lanzamientos necesarios para llegar al margen de error deseado.

Como ya he mencionado antes, existe una amplia variedad de algoritmos cuánticos [3]. Por la forma en la que se implementan, se pueden separar en dos grandes grupos, los que necesitan un oráculo y los que no. Un oráculo es un operador ideal tal que dado un estado a la entrada, devuelve otro a la salida según unas reglas internas. Este tipo de concepto, también descrito como caja negra, permite realizar operaciones que no serían posibles en un circuito sin recurrir a mediciones intermedias u otro tipo de "trampas" [5, p. 221]. El resto de algoritmos, al no necesitar de oráculo, sí que se pueden implementar usando únicamente las puertas cuánticas disponibles, aunque casi siempre requieren de cierta parte de computación clásica que se tendrá que computar en un circuito externo.

Para conocer más en profundidad los algoritmos cuánticos, se describen a continuación algunos de los más importantes:

### 2.3.1 Algoritmo de búsqueda de Grover

Para buscar un elemento dado entre $N$ elementos desordenados, los algoritmos clásicos necesitan al menos $N - 1$ pasos para encontrar la solución. Con el algoritmo de Grover se demuestra que en $\sqrt{N}$ pasos haciendo uso de un oráculo se obtiene la solución [2] [5, ch.6].

El estado cuántico sobre el que se trabaja es uno en el que cada estado representa cada ele-mento del vector donde se está haciendo la búsqueda. Un vector de longitud $N$ podrá ser representado en un sistema cuántico por $\log_2(N) = n$ qubits.

Antes de hacer la primera consulta al oráculo se necesita preparar el estado inicial para que todos los estados tengan la misma probabilidad. Suponiendo que el estado inicial es $|0\rangle^{\otimes n}$, basta con aplicar una puerta Hadamard a cada uno de los qubits del sistema. El resto de operaciones

constituyen una sola iteración del algoritmo, por lo que se le suele llamar operador o iteración de Grover.

El oráculo que se usa en este algoritmo es uno tal que devuelve, en un qubit auxiliar, $|1\rangle$ si el estado de entrada representa al buscado y $|0\rangle$ para el resto de casos. Usando el resultado de esta oráculo, se aplica una operación controlada de giro sobre el eje z para cambiar el signo del estado. Así, si el elemento que se busca está representado por $|\Psi_{sol}\rangle$, el oráculo actuará sobre el estado de tal manera que

$$f(x) = \begin{cases} -x & \text{si } |x\rangle = |\Psi_{sol}\rangle\,, \\ x & \text{en caso contrario.} \end{cases} \tag{19}$$

Después de la aplicación del oráculo, se aplica una transformación de difusión D. Esta operación busca amplificar los estados con fase negativa a la vez que atenúa el resto de estados. la transformación se puede construir a partir de la composición de 2 puertas más simples, la puerta de rotación

$$\boxed{R} \quad R = \begin{pmatrix} 1 & & & \\ & -1 & & \\ & & \ddots & \\ & & & -1 \end{pmatrix}, \tag{20}$$

y una transformación Walsh-Hadamard

$$\boxed{W} \quad W_{ij} = 2^{-n/2}(-1)^{(i,j)},\ W = 2^{-n/2}\begin{pmatrix} 1 & -1 & -1 & 1 & & \\ -1 & 1 & 1 & -1 & & \\ -1 & 1 & 1 & -1 & & \\ 1 & -1 & -1 & 1 & & \\ & & & & \ddots & \\ & & & & & 1 \end{pmatrix}, \tag{21}$$

con (i,j) refiriéndose al producto escalar de la representación binaria de i y j [a]. Así, la puerta D se puede construir como

$$\boxed{D} \quad D = WRW = \frac{2}{n}J_n - \mathbb{1} = \begin{pmatrix} \frac{2}{n}-1 & \frac{2}{n} & & & \\ \frac{2}{n} & \frac{2}{n}-1 & & & \\ & & \ddots & & \\ & & & \frac{2}{n}-1 \end{pmatrix}, \tag{22}$$

donde $J_n$ es la matriz n×n con todos sus elementos igual a 1.

De esta manera, el circuito cuántico descrito por Grover se puede escribir como se muestra en la figura (Fig.1).

Para obtener el resultado, basta con aplicar el operador de Grover $\sqrt{N}$ veces. Después de esto, se mide el qubit auxiliar donde se coloca la respuesta del oráculo. Si el estado medido es $|1\rangle$, se tendrá una probabilidad que tiende a 1 de obtenerse el resultado correcto al medir el estado del resto de los qubits.

[a] Por ejemplo: (3,5)=(011,101)=$0 \cdot 1 + 1 \cdot 0 + 1 \cdot 1 = 1 + 1 + 0 = 0\ mod\ 2$.

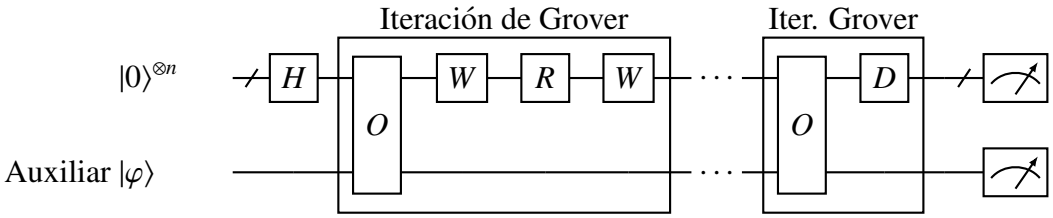



Fig. 1: Circuito cuántico del algoritmo de Grover. Los indicadores a la derecha representan la medición de los estados cuánticos.

Este desarrollo del algoritmo se corresponde con el original, tal y como lo describió Grover en su artículo original de 1996. Sin embargo, hay una forma más intuitiva de entender este algoritmo. Para ello se parte de las mismas dos primeras operaciones, el oráculo y la rotación sobre el eje z. La diferencia es que se va a usar una nueva estrategia de difusión. Ahora, se va a usar una puerta de cambio de fase condicional que actúa solo sobre los estados distintos de $|0\rangle^{\otimes n}$, tal que

$$f(x):\ |x\rangle\ \rightarrow\ \begin{cases} |0\rangle^{\otimes n} & \text{si } |x\rangle = |0\rangle^{\otimes n}\,, \\ -|x\rangle & \text{para el resto}\,. \end{cases} \tag{23}$$

Para completar la iteración, basta con añadir una puerta Hadamard antes y después de aplicar esta operación. Así, otra forma de escribir la iteración de Grover se da en siguiente figura (Fig.2).

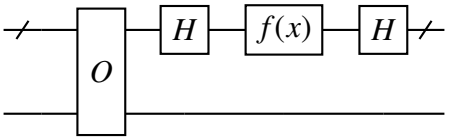



Fig. 2: Forma alternativa del circuito cuántico para la iteración de Grover.

De esta manera es sencillo visualizar la operación de difusión. El estado que corresponde a la solución cambiará de signo al aplicarle la puerta Hadamard. Esto hace que cuando se le aplique al estado el cambio de fase condicional, el estado solución cambie de signo y su módulo aumente en $2/\sqrt{N}$. Al aumentar la probabilidad de obtener el estado solución, el resto de estados verán disminuida su probabilidad. Este hecho se puede representar gráficamente con la siguiente figura (Fig.3). Repitiendo esta iteración $\sqrt{N}$ veces, se llegará a que el estado solución tenga una probabilidad muy por encima de la media.

### 2.3.2 Transformada cuántica de Fourier (QFT)

En procesado de señal, y en general cualquier procesado de información, la transformada de Fourier es una herramienta imprescindible. Una de las formas más comunes en la que se aplica es a través de la transformada rápida de Fourier (FFT). Esta operación requiere de $O(n \log n)$ pasos. En computación cuántica, se puede aplicar esta operación en tan solo $O(\log n)$ pasos [1] [5, ch.5.1].

Este algoritmo cuántico aplica una transformación a un estado inicial, de tal manera que las amplitudes del estado a la salida se correspondan con los coeficientes de la DFT. Para represen-

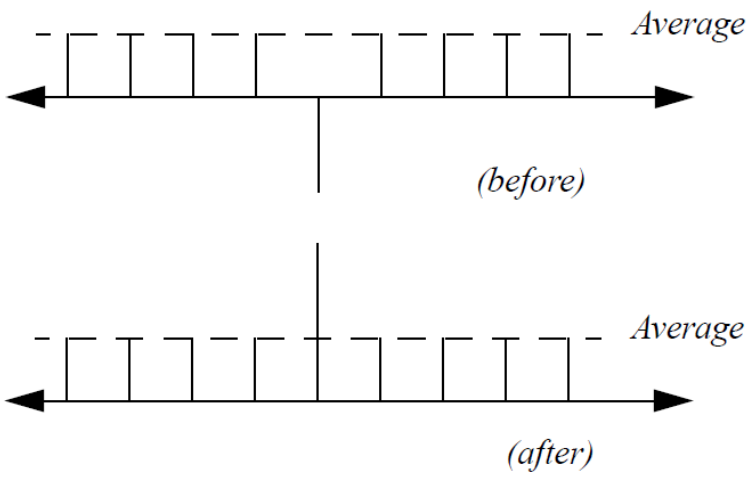



Fig. 3: Efecto de la operación de difusión [2, Fig.2]

tar cada número se usa la base computacional. Así, para un vector $x$ de longitud $n$, su DFT ($y$) se puede escribir como

$$\text{Clásico: } y_k = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} x_j e^{2\pi i jk/n}, \quad \text{Cuántico: } |x^{(k)}\rangle \rightarrow \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} e^{2\pi i jk/n} |y^{(j)}\rangle, \tag{24}$$

de tal manera que

$$\sum_{j=0}^{n-1} x_j |x^{(j)}\rangle = \sum_{k=0}^{n-1} y_k |y^{(k)}\rangle. \tag{25}$$

De esta manera, $y_k$ serán los coeficientes de la transformada de fourier del vector $x_k$.

Otra forma de dar la expresión de la QFT es a través del producto de Kronecker. Usando una representación a través de la base computacional $|x\rangle = |x_n x_{n-1} \dots x_2 x_1\rangle$, su QFT está dada por

$$|x\rangle \rightarrow \frac{1}{2^{n/2}} \left(|0\rangle + e^{2\pi i 0.x_1} |1\rangle\right) \otimes \left(|0\rangle + e^{2\pi i 0.x_2 x_1} |1\rangle\right) \otimes \cdots \otimes \left(|0\rangle + e^{2\pi i 0.x_n \dots x_1} |1\rangle\right), \tag{26}$$

donde $0.x_n \dots x_1$ representa la fracción binaria[a]. Con esta representación se puede llegar a un circuito cuántico que implementa la QFT usando solamente 2 tipos de puertas, la Hadamard y una de desplazamiento de fase $R_k$ definida como

$$-\boxed{R_k}- \quad R_k = \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{pmatrix}. \tag{27}$$

A falta de puertas swap para invertir el orden de los qubits, el circuito que implementa la QFT se muestra en la figura (Fig.4).

Sin embargo, es difícil preparar un estado cuántico con unos coeficientes concretos, por lo menos hasta cierta precisión, que representen una señal. Además, los coeficientes de la QFT

[a] Por ejemplo: la fracción binaria de 0.625 se escribe como 0.101, $0.625 = 1 \cdot 1/2 + 0 \cdot 1/4 + 1 \cdot 1/8$

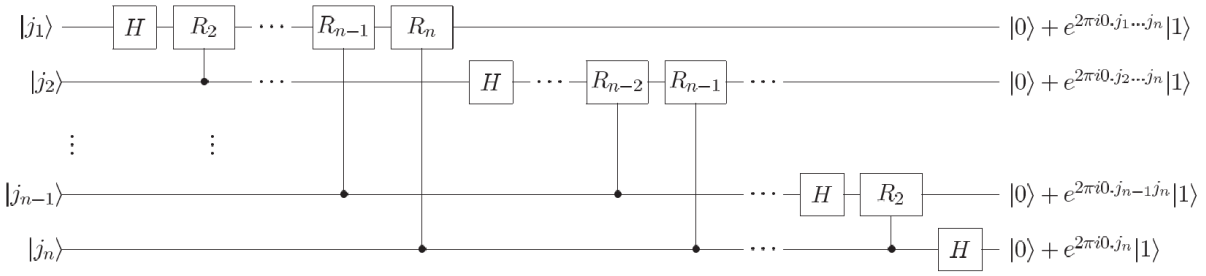



Fig. 4: Circuito cuántico que implementa QFT [5, Fig.5.1]. $j_k$ de la figura es equivalente a $x_k$.

están codificados en las amplitudes de los estados cuánticos de la salida, los cuales no son accesibles. Esto hace que, aunque la QFT sea más rápida que la FFT, no se pueda aplicar en los mismos casos que esta. No por ello esta operación es inútil. Algunos algoritmos sí que hacen uso de esta operación para resolver problemas mucho más rápido que cualquier algoritmo clásico. Todos estos algoritmos se basan en solucionar el problema del subgrupo oculto (HSP), que se puede describir de la siguiente manera.

**Definición.** *Sea una aplicación f definida en un grupo G a un conjunto X (f: $G \rightarrow X$), tal que f(K) es constante para el subgrupo K, $K \subset G$. El problema HSP consiste en encontrar el conjunto generador A del subgrupo K, $K = \langle A \rangle$.*

Aunque este problema parezca un artilugio meramente matemático, resolver este problema en algunos grupos concretos tiene aplicaciones interesantes. Entre todos los ejemplos, Shor mostró una forma de resolver este problema para el grupo del producto directo de dos grupos cíclicos de orden r ($\mathbb{Z}_r \otimes \mathbb{Z}_r, +$). Este problema es equivalente a resolver el problema del logaritmo discreto. Algunos métodos de criptografía, como el cifrado ElGamal [13] o el protocolo Diffie-Hellman [14], basan su seguridad en la dificultad para resolver el problema, que crece de manera exponencial con el número de bits necesarios para representar el número. Los algoritmos cuánticos suponen un nuevo paradigma para los sistemas de criptografía, ya que podrían romper dichos sistemas en un tiempo polinómico, haciéndolos potencialmente poco seguros.

### 2.3.3 Algoritmo de factorización de Shor

El problema de factorización es el de encontrar todos los factores primos de un número. Un método naif necesita $O(n/2)$ pasos por cada candidato para comprobar la divisibilidad con todos los números menores que $\lceil n/2 \rceil + 1$. El algoritmo de Lenstra es (salvo un factor $\alpha$) el más eficiente que se conoce, sin embargo, el tiempo necesario para llegar al resultado es exponencial, $O(\exp(\alpha(\log n)^{1/3}(\log\log n)^{2/3})$. El algoritmo de factorización de Shor utiliza la QFT para reducir el coste del algoritmo a $O((\log n)^2(\log\log n)\,(\log\log\log n))$ en un computador cuántico.

Este resultado es harto complicado de demostrar, ya que usa, entre otros aspectos, resultados de teoría de números. Además, para llegar al resultado final es necesario explicar una serie de pasos y transformaciones previas. Se puede encontrar una explicación más detallada de esto en los textos usados como referencia [1] [5, ch.5.2-5.3]. El siguiente desarrollo, resumido en el diagrama de la figura (Fig.A.1), pretende dar una idea general de los pasos necesarios hasta llegar al algoritmo de Shor:

**Paso 1: Estimación de fase**

Una de las aplicaciones de la QFT es la aproximación de la fase de los autoestados de un operador unitario. El problema se puede plantear como

$$U\left|u_i\right\rangle = e^{2\pi i\varphi_i}\left|u_i\right\rangle,\ i = 1, 2, \ldots, 2^n, \tag{28}$$

con $\varphi_i$ la fase desconocida, $|u_i\rangle$ los autoestados y $n$ el número de qubits del sistema.

Al ser U unitario, la aplicación de U sobre un autoestado no lo modifica salvo por la fase introducida. Esto permite "extraer" la información de esa fase a través de la aplicación de de la puerta U como una puerta U-controlada.

Sea un qubit de control que se inicializa aplicándole una puerta Hadamard, su estado será $2^{-1/2}\left(|0\rangle + |1\rangle\right)$. Teniendo en un registro auxiliar el autoestado $|u\rangle$, se le aplica la puerta U-controlada. De forma explícita, los estados se transformarán a lo largo del circuito de la figura (Fig.5).

$$|0\rangle\otimes|u\rangle \ \rightarrow\ H\left|0\right\rangle\otimes|u\rangle = \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)\otimes|u\rangle \ \rightarrow\ C\text{–}U\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)\otimes|u\rangle = \left(\frac{|0\rangle + e^{2\pi i\varphi}\left|1\right\rangle}{\sqrt{2}}\right)\otimes|u\rangle\,. \tag{29}$$

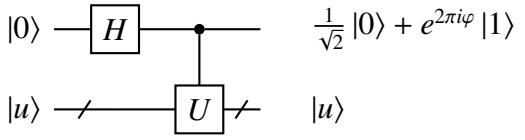



Fig. 5: Primer paso del algoritmo de aproximación de fase con 1 qubit de control.

Si en vez de un qubit se usan $n$, y cada uno de ellos se usa como qubit de control de una puerta controlada que aplica U $2^i$ veces ($i = 0, 1, ..., n-1$), se llega a una expresión muy similar a la que se obtenía como resultado de la QFT. En concreto, el qubit de control $i$ a la salida estará en un estado $2^{-1/2}(|0\rangle + \exp(2\pi i 2^i\varphi)\left|1\right\rangle$. Aplicando la QFT inversa a los qubits de control se puede obtener de forma explícita el valor de $\varphi$, con un una precisión de n decimales binarios.

**Paso 2: Búsqueda de orden**

El orden multiplicativo de un número es un concepto usado en teoría de números, en concreto en aritmética modular.

**Definición.** *Sea a un número entero $a \in \mathbb{Z}$ y N un natural, $N \in \mathbb{N}$, tales que sean coprimos, $mcd(a, N) = 1$. El orden de a módulo N es el menor número natural r que cumpla $a^r = 1\ mod\ N$.*

Para resolver este problema se usa un operador unitario V que actúa como $V\left|y\right\rangle = |ay\ mod\ N\rangle$, donde el estado $|y\rangle$ está compuesto por $n$ qubits y $n$ es el número de bits necesarios para expresar $N$, $n = \log_2(N)$. Los estados propios de este operador V están definidos por

$$|v_s\rangle = \frac{1}{\sqrt{n}}\sum_{k=0}^{n-1} exp\left(\frac{-2\pi isk}{r}\right)|a^n \bmod N\rangle\,, 0 < s < r-1 \tag{30}$$

los cuales, al aplicarles V se transforman en

$$V\left|v_s\right\rangle = \frac{1}{\sqrt{n}}\sum_{k=0}^{n-1} exp\left(\frac{-2\pi isk}{r}\right)|a^{n+1} \bmod N\rangle = \exp\left(\frac{2\pi is}{n}\right)\left|v_s\right\rangle. \tag{31}$$

Aplicando el algoritmo de aproximación de fase, se puede calcular el valor de la fracción $s/r$. En este punto, la teoría de números permite obtener cuál es el número entero desconocido $r$ a través del algoritmo de fracciones continuas en un número polinómico de pasos. El desarrollo de este último paso es suficientemente complejo como para incluirlo en los objetivos de este trabajo.

**Paso 3: Reducción del problema de factorización a la búsqueda de orden**
Dado un número $N$, la descomposición factorial de $N$ trata de encontrar los divisores no triviales tales que su producto sea $N$. Sin perder generalidad, se puede suponer que $N$ solo tiene 2 divisores no triviales, tales que $N = A \cdot B$ con $A$ y $B$ primos [15].

Aplicando teoría de números, se llega a que se puede obtener uno de estos factores si se encuentra una solución no trivial ($X \neq \pm 1$) a la ecuación $X^2 = 1 \mod N$. Otro resultado útil es que si se escoge un número aleatorio $Y$ coprimo a $N$, con $0 < Y < N$, la probabilidad de que el orden de $Y$ módulo $N = L$ sea par y que $Y^{L/2} \neq -1 \mod N$ es mayor o igual que 1/2. Para calcular $L$ se usará el algoritmo cuántico del paso anterior.

Con esto, se tiene que, con una probabilidad del orden del 40%, uno de los divisores de $N$ es $A$, obteniendo $A$ de la ecuación

$$mcd(Y^{L/2} \pm 1, N) = A. \tag{32}$$

Si el $A$ calculado de esta manera no es un divisor de $N$, es probable que el divisor que se busca sea un número muy próximo a $A$.

El algoritmo de Shor es un buen ejemplo de como se pueden usar los algoritmos cuánticos, no para computar un problema entero, sino para resolver de manera eficiente alguno de sus pasos. La mayoría de algoritmos cuánticos, como el QACO de este trabajo, necesitan un procesado clásico en alguno de sus pasos.

## 2.4 Simulación del circuito

Uno de los problemas de la computación cuántica es la forma de probar los circuitos cuánticos. La evolución de un sistema cuántico viene dada por la ecuación de Schrödinger dependiente del tiempo (Eq.1).

En general, para resolver esta ecuación hay que usar técnicas de resolución de ecuaciones en derivadas parciales. En sistemas de dimensión infinita hay que cuantizar el espacio de Hilbert. En sistemas con partículas en un potencial dado, esto se puede hacer discretizando el sistema a través de un enrejado. Sin embargo, esto presenta un problema grave en la frontera, por lo que hay que encontrar una solución para determinar el comportamiento del sistema en estos puntos. Este problema se suele soslayar imponiendo condiciones de frontera compatibles con el sistema, siendo las más comunes imponer condiciones de periodicidad o colocar el sistema en una caja de potencial.

Además, la dimensión de espacio de Hilbert asociado al problema aumenta exponencialmente con el número de partículas (o qubits) del problema. En concreto, para un sistema de $n$ qubits de 2 niveles de energía, se tiene un espacio de Hilbert de dimensión $2^n$. Esto hace que las simulaciones de sistema cuánticos sean rápidamente inviables para su ejecución en un ordenador de sobremesa corriente.

Por suerte, los postulados de la mecánica cuántica ofrecen una forma fácil para simular

sistemas cuánticos con un un espacio de Hilbert de dimensión finita. Como ya se ha visto, los operadores pueden ser representados a través de una matriz cuadrada y los estados a través de un vector con componentes complejas. A partir de un estado cuántico bien definido, por ejemplo $|0\rangle^{\otimes n}$, el sistema evoluciona a través de multiplicaciones con las matrices de los operadores. Esta estrategia de simulación cuántica se puede aplicar al caso de un computador cuántico de forma directa.

En un computador cuántico de $n$ qubits el sistema siempre empezará en el estado $|0\rangle^{\otimes n}$. El paso del estado a través de las distintas puertas se hace multiplicando el estado por cada una de las matrices que definen las puertas cuánticas. Esto hace que, para un circuito cuántico dado, el estado al final del circuito sea siempre el mismo. El punto del algoritmo de simulación en el que se tiene en cuenta la naturaleza cuántica del circuito es en la medición. La medición de un qubit se hace siempre a través de la medición del estado de energía del qubit, es decir, haciendo una medición sobre el eje $z$ de la esfera de Bloch del estado. Por lo tanto, para simular la medición del estado final es necesario escoger uno de los estados de la base de forma aleatoria, teniendo en cuenta para ello la distribución de probabilidad del estado cuántico. De esta manera se ha desarrollado el siguiente algoritmo para evaluar los resultados de un circuito cuántico (Alg.1).

**Algoritmo 1** Simulación de un circuito cuántico

**Require:** Sea el circuito de n qubits, las operaciones tienen que ser todas matrices de tamaño n×n

1: Generar el estado inicial, estado=[1 0 0 ... 0]. ▷ La longitud del vector del estado es $2^n$
2: **for** operación=conjuntoDeOperaciones **do**
3: estado=estado·operación
4: prob=|estado ∘ estado| ▷ Producto Hadamard
5: Elegir aleatoriamente un estado de la base según la distribución dada por el vector prob

Un circuito cuántico de 10 qubits necesita que cada operación esté representada por una matriz de tamaño 1024×1024. Si cada elemento de la matriz está representado por un número de coma flotante, cuyo espacio en memoria es de 8 bytes, se necesitan 8MB de memoria para almacenarlo. Para simular un circuito de 16 qubits, cada matriz necesitaría 32GB de memoria, lo que hace imposible su simulación de forma práctica fuera de los superordenadores o sistemas especiales. Por suerte, la mayoría de matrices asociadas a operaciones de puertas cuánticas tienen una gran cantidad de ceros, por lo que es posible ahorrar espacio simplemente al usar una representación de matriz dispersa. Esto, además de ahorrar espacio en memoria, hará las operaciones de multiplicación más rápidas.

El programa que simula los circuitos cuánticos en este trabajo se ha escrito en la versión de Matlab R2019b. Se ha elegido este programa por la facilidad que ofrece en el cálculo con matrices y por la posibilidad de paralelizar la ejecución fácilmente.

### 2.4.1 Matrices de las puertas cuánticas

Una forma legítima de definir las matrices de las puertas cuánticas es escribir a mano sus valores. Esto puede ser la opción más conveniente si se tienen circuitos de pocos qubits, o si las matrices tienen una representación sencilla. Sin embargo, cuando las operaciones son complejas escribirlas a mano deja de ser una opción. Para hacerlo, se va a hacer uso de 2 propiedades de los circuitos cuánticos. La primera es que cualquier operación se puede expresar a través de un conjunto de puertas unitarias de 1 qubit, y la puerta cnot [5, Ch.4.5]. La segunda tiene que ver

con la forma en la se construyen las matrices de las operaciones. De la misma manera que los estados cuánticos de partículas distintas se componen a través del producto de Kronecker, las puertas cuánticas se pueden componer de la misma manera.

En un sistema de $n$ qubits, aplicar una operación A sobre el qubit $i$ deja invariante el resto de qubits. Por lo tanto, si se quiere definir la matriz de la operación que actúa sobre todo el sistema, bastará con componer la operación identidad de cada qubit con la operación A sobre el qubit $i$, tal que

$$A = \mathbb{1}^{(1)} \otimes \mathbb{1}^{(2)} \otimes \cdots \otimes \mathbb{1}^{(i-1)} \otimes A^{(i)} \otimes \mathbb{1}^{(i+1)} \otimes \cdots \otimes \mathbb{1}^{(n)}. \tag{33}$$

Para generar la puerta cnot no se puede usar una misma expresión, sin embargo, sí que se puede usar un razonamiento similar. Para generar la matriz asociada a la puerta cnot se identifican primero los qubits de control ($c$) y objetivo ($t$). La puerta cnot se podrá construir con la suma de dos matrices tales que

$$cnot(c,t) = \left( \mathbb{1}^{(1)} \otimes \mathbb{1}^{(2)} \otimes \cdots \otimes \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}^{(c)} \otimes \cdots \otimes \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}^{(t)} \otimes \cdots \otimes \mathbb{1}^{(n)} \right) + \mathbb{1}. \tag{34}$$

Esta idea de sumar una matriz generada a partir de productos de Kronecker a la matriz identidad se puede extender a las puertas cnot con más qubits de control u objetivo. Solo habría que repetir la expresión anterior a todos los qubits correspondientes. En este sentido, se puede definir la puerta Toffoli con sus qubits de control ($c_1$, $c_2$) y su qubit objetivo ($t$) como

$$toff(c_1,c_2,t) = \left( \mathbb{1}^{(1)} \otimes \mathbb{1}^{(2)} \otimes \cdots \otimes \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}^{(c_1)} \otimes \cdots \otimes \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}^{(t)} \otimes \cdots \otimes \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}^{(c_2)} \otimes \cdots \otimes \mathbb{1}^{(n)} \right) + \mathbb{1}. \tag{35}$$

Cualquier puerta cuántica podrá ser generada a partir del producto de las 3 puertas presentadas. Esta sencilla forma de expresar las operaciones permite generar simulaciones de circuitos de forma rápida. Para ilustrar esto, se detalla la forma en la que se ha generado el paso de la exploración para los problemas CBQP (Sec.3.2.3). En esta parte del circuito cuántico se propone la aplicación de una serie de puertas Fredkin. Haciendo el ejercicio 4.25 propuesto en el libro de A. Nielsen & L. Chuang [5], se demuestra que una puerta Fredkin se puede hacer usando 2 puertas cnot y una puerta Toffoli. Así, la puerta Fredkin que actúa intercambiando los estados de $t_1$ y $t_2$ controlado por el qubit $c$ se puede generar como

$$fred(c,t_1,t_2) = cnot(t_1,t_2) \cdot toff(c,t_2,t_1) \cdot cnot(t_1,t_2) = cnot(t_2,t_1) \cdot toff(c,t_1,t_2) \cdot cnot(t_2,t_1). \tag{36}$$

En el ejemplo 1 se muestra la simulación de la acción una puerta de un circuito cuántico sobre un estado.

### 2.4.2 Puertas controladas

A la hora de calcular el efecto de las puertas controladas sobre los estados, los autores de [17] proponen "medir" el estado del qubit de control. Para ello, simplemente generan un número aleatorio y lo comparan con la probabilidad de que el qubit de control sea 1. Si el qubit de control resulta ser 1, aplican la puerta controlada, y en caso contrario, saltan a la siguiente operación.

**Ejemplo 1:** Simulación de la puerta D (Eq.22)

Se quiere simular una puerta D para usarla en un programa que simula el algoritmo de Grover. Para hacer la prueba se va a usar el problema no trivial de menor tamaño, un problema de tamaño 4 codificado en 2 qubits en la base computacional $\{00, 01, 10, 11\}$. En el ejemplo usado, la solución será el estado 01.

Lanzamos una simulación del circuito, y tras pasar por primera vez a través del oráculo, simulamos que se introduce ruido en el sistema. En este punto se tiene el sistema en el estado $x_0 = [0.48, -0.51, \sqrt{0.2886}, 0.47]$. Para aplicar la puerta D se multiplica este estado por la matriz del operador D, dando como resultado $x_0 \cdot D = x = [0.0086, 0.9986, -0.0486, 0.0186]$. El vector $x$ contiene toda la información del estado cuántico. Si queremos calcular la probabilidad de obtener cada uno de los estados cuánticos a la salida basta con calcular el módulo cuadrado, $P(x) = |x|^2 = [0.0001, 0.9972, 0.0024, 0.0003]$. Esto nos da como resultado que se medirá el estado correcto con una probabilidad del 99.72%.

Si se quisiera conocer el comportamiento del circuito frente a distintas configuraciones de ruido, bastaría con repetir esta estrategia el número de veces que se considere suficiente como para hacer un estudio estadístico adecuado.

Esta opción intenta simular de manera directa la acción de las puertas controladas en un computador cuántico real, en la que la aplicación se puede dar o no dependiendo del estado del qubit de control. Por lo tanto, dado que no se puede saber con exactitud las puertas aplicadas en la ejecución del programa, la solución propuesta parece ser la más coherente con la realidad.

Sin embargo, esta opción no es la mejor en cuanto al tiempo de cómputo de la simulación. Usar matrices para representar la aplicación de puertas controladas es una opción equivalente, en la que se habrá eliminado parte de código que puede ralentizar el cómputo de la simulación.

*Demostración.* Sea un circuito cuántico en el que se tiene una puerta controlada. Para simular este circuito, se eligen hacer $n$ iteraciones idénticas para obtener una estadística del circuito. En cada iteración se genera un número aleatorio para aplicar o no dicha puerta controlada. Por lo tanto, para obtener una estadística fiel del funcionamiento del circuito, $n$ tendrá que ser un número suficientemente grande como para dar cuenta de todas las posibilidades de aplicación o no de la puerta controlada. Para obtener una información correcta del resultado del circuito, $n$ tiene que ser un número tal que $1/n <<$ que la probabilidad de aplicación o no de la puerta controlada (el menor de ambos). Además, para seguir obteniendo un resultado de la simulación idéntico al que se obtendría si no se tuviera puerta controlada, este nuevo número $1/n$ tiene que ser del orden de $1/(n' \cdot p)$, siendo $1/n'$ la exactitud que se quiere conseguir para el estado final. Se elige ahora aplicar una puerta que tenga en cuenta la distribución de probabilidades de exploración, haciendo el paso a través de la puerta controlada usando el método propuesto. Dado que la probabilidad o no de aplicación de la puerta controlada se ha mezclado con la probabilidad de obtener el resto de resultados, $n$ tiene que ser capaz ahora de distinguir esta diferencia. Si el circuito sin puerta controlada necesita $k$ iteraciones para obtener un resultado bueno, ahora se necesitarían $k \cdot p$ iteraciones, siendo $1/p <<$ que la probabilidad de aplicación o no de la puerta controlada (el menor de ambos).
Se ve por lo tanto que ambas estrategias son equivalentes en tanto que ambas requieren un número similar de iteraciones para conseguir una precisión idéntica. □

## 2.5 Algoritmos de hormigas

Cuando se plantea un problema computacionalmente demasiado grande como para resolverse de manera exacta, suele ser suficiente una solución subóptima. Sin embargo, encontrar esta solución puede requerir algoritmos complejos en los que la toma de decisiones, aunque aleatoria, implique un coste computacional o una cantidad de memoria elevada. Para encontrar algoritmos sencillos que resuelvan estos problemas algunos investigadores se inspiraron en sistemas biológicos que de alguna forma implementan estos algoritmos, como lo son los enjambres, sistemas evolutivos o a las hormigas [16].

Para buscar comida, los hormigueros envían hormigas exploradoras para investigar el entorno. Al principio, estas hormigas viajan de forma aleatoria alejándose del nido hasta encontrar una fuente de comida. Cuando lo encuentran, esta hormiga vuelve al nido para avisar de su hallazgo y así que el resto del hormiguero pueda explotar esta fuente de alimento. Comunicar esta posición entre individuos es una acción compleja, por lo que las hormigas tienen desarrollado un sistema de feromonas por el cual comunican mensajes simples al resto. Cada hormiga deja un rastro allá por donde camina, y la fuerza de este rastro dependerá de un conjunto reducido de variables. Por ejemplo, de la calidad del alimento, de su cantidad y de la distancia al hormiguero. Una sola hormiga es capaz de hacer este cálculo y dejar esta información a través de lo fuerte que sea el rastro que haya dejado. Esta idea de reforzar el camino óptimo a una fuente de alimento buena se puede modelizar como un feedback positivo.

Una vez se encuentra un alimento, el hormiguero empieza a explotarlo rápidamente. Sin embargo, el camino recorrido por la hormiga exploradora que lo encontró puede no se el más corto. Las primeras hormigas recolectoras seguirán el camino marcado por la exploradora. Pero dado que el rastro de feromonas no es una línea ideal, las recolectoras se pueden desviar alrededor del camino fijado. Estas desviaciones alrededor del camino inicial pueden llevar a encontrar un camino más corto a la comida. En ese caso, la hormiga indicará este descubrimiento al resto haciendo que el rastro dejado sea más fuerte que el anterior, guiándolas por este camino más corto. Este comportamiento se puede entender como la búsqueda de mínimos locales a través de fluctuaciones aleatorias alrededor de la mejor solución hasta el momento.

En el momento en el que se encuentra un camino mejor a la comida, tiene que haber un mecanismo que permita abandonar el peor camino. El rastro de feromonas no es una línea pintada en el suelo, sino que es volátil. Según pasa el tiempo, las hormonas depositadas en el camino se degradan o se pierden a causa del viento. Esto hace que en un camino malo por el que pasen pocas hormigas el rastro de feromonas sea más débil, haciendo que finalmente todas las hormigas elijan el mejor camino. Este mecanismo se conoce como evaporación de las feromonas, y permite reforzar la búsqueda del camino óptimo solamente alrededor del mejor camino encontrado hasta el momento.

Existen otros mecanismos que hacen aún más eficiente la búsqueda del camino óptimo en menos tiempo, por ejemplo la aparición de hormigas élite para reforzar con más fuerza el rastro de feromonas o el refuerzo negativo de caminos peores. Una condición imprescindible de todos estos mecanismos es que se basen en unos pocos principios básicos:

- Una hormiga solo tiene acceso a la información de su entorno inmediato.
- No puede existir comunicación a distancia entre hormigas.
- La información que puede almacenar una hormiga está limitada, entre otros, al camino que ha recorrido.

– La capacidad de cómputo de una hormiga es limitada, por lo que las operaciones que realice no pueden ser demasiado complejas.

Expuestas las herramientas básicas disponibles, se pueden desarrollar los algoritmos basados en hormigas más simples. En concreto, en este trabajo se ha usado el algoritmo de colonia de hormigas ACO (Ant Colony Optimization) [16].

La idea principal de ACO consiste en agrupar las 3 ideas comentadas: el refuerzo positivo de las mejores soluciones, fluctuaciones aleatorias alrededor del mismo, y la disminución de la probabilidad de escoger las soluciones abandonadas. El problema de encontrar el camino óptimo entre 2 puntos se puede codificar a través de un grafo ponderado. El peso de cada arista será su longitud, por lo que el camino óptimo será el mínimo de la suma de los pesos de sus aristas. Además del peso, cada arista contendrá información acerca de la cantidad de feromonas en ella. En el algoritmo (Alg.2) se describe con más detalle este algoritmo.

**Algoritmo 2** ACO

**Require:** Grafo ponderado $W_{ij}$, máximo de iteraciones maxIter, número de hormigas lanzadas por iteración antIter, parámetro de exploración b, parámetro de evaporación $\rho$, condición de convergencia converCondition

```
1:  Inicializar parámetros y las feromonas
2:  for i=1:maxIter do
3:      for j=1:antIter do
4:          Lanzar una hormiga paso a paso del nodo inicial al final (Alg.3)
5:      Encontrar la mejor hormiga de la iteración, bestAntIter
6:      if f(bestAntIter,M)>f(bestAnt,M) then      ▷ Actualizar la mejor hormiga encontrada
7:          bestAnt=bestAntIter
8:          mismoResultado=0
9:      else
10:         mismoResultado=mismoResultado+1
11:         if mismoResultado==converCondition then
12:             break   ▷ Salir cuando se obtiene el mismo resultado un número dado de veces
13:     Actualizar las fermonas (Alg.4)
```

En ACO, las hormigas se mueven paso a paso entre los nodos del grafo. Para elegir el siguiente movimiento, las hormigas toman su decisión en base a 2 parámetros, las feromonas de las aristas incidentes al nodo en el que se encuentran y la probabilidad que tienen de explorar de manera aleatoria. Así, la elección del siguiente paso de una hormiga se hace siguiendo el algoritmo (Alg.3). En cada paso, cada hormiga realizará esta operación hasta que llegue al nodo final.

Una vez todas las hormigas lleguen al final, se actualizará el rastro de feromonas. En ese momento, cada hormiga realiza el cálculo de lo óptimo que haya sido su recorrido, depositando feromonas a través del mismo según una regla de actualización. Suponiendo que solo nos interesa obtener el camino más corto recorrido, la regla de actualización de feromonas puede consistir en depositar un número dado de feromonas a lo largo de todo el recorrido. Suponiendo que no hay hormigas élite, todas las hormigas tienen la misma cantidad de feromonas, por lo que una hormiga que haya encontrado un camino corto dejará una densidad de feromonas mayor a lo largo de todo el recorrido que otra que haya encontrado un camino más largo. Después de depositar todas las feromonas, se aplica la evaporación. Dado un parámetro de evaporación, las

**Algoritmo 3** Elección del siguiente paso para una hormiga.

**Require:** Grafo ponderado $W_{ij}$, feromonas de las aristas $\tau_{ij}$, nodo actual $n_{act}$, nodo previo $n_{pre}$, probabilidad de elegir un camino aleatorio b

1: Lanzar 2 números aleatorios, $0 \leq \text{explor} \leq 1$, $0 \leq \text{step} \leq 1$
2: Ordenar las k aristas incidentes a $n_{act}$, excepto la que lleva a $n_{pre}$
3: **if** explor $\leq$ b **then** ▷ La hormiga se moverá de manera aleatoria
4: Moverse a través de la arista i, tal que $(i-1)/k \leq \text{step} \leq i/k$
5: **else**
6: Moverse a través de la arista i, tal que $S_i/S \leq \text{step} \leq S_{i+1}/S$

$$S = \sum_{i=1}^{k} \tau_i,\ S_i = \sum_{j=1}^{i-1} \tau_j,\ S_1 = 0,\ S_{k+1} = 1. \tag{37}$$

feromonas en cada arista de disminuyen en un factor. Aunque este factor puede ser variable y aumentar con el tiempo, la versión más simple del algoritmo usa un factor constante. Se presenta este paso de actualización de feromonas en el algoritmo (Alg.4).

**Algoritmo 4** Actualización de feromonas.

**Require:** Grafo ponderado $W_{ij}$, feromonas de las aristas $\tau_{ij}$, camino recorrido por cada hormiga $X^{(i)}$ (con $X^{(i)}_{jk}$=1 para las aristas jk visitadas, 0 para el resto), parámetro de evaporación $\rho$

1: Inicializar a 0 las feromonas depositadas en cada arista, $\Delta\tau_{ij} = 0$
2: **for** i=1:número total de hormigas **do**
3: Calcular la longitud del camino recorrido por la hormiga, $D^{(i)} = \sum_{j,k} W \cdot X^{(i)}$
4: Actualizar las feromonas depositadas, $\Delta\tau_{ij} = \Delta\tau_{ij} + 1/D^{(i)}$
5: Actualizar las feromonas, $\tau_{ij} = (1-\rho)\tau_{ij} + \rho\Delta\tau_{ij}$

Esta iteración de lanzar hormigas y actualizar las feromonas se repite hasta que se llega a una situación de convergencia, en el que ya no se encuentra un camino mejor. Empezando desde una situación en la que todas las aristas tienen la misma cantidad de feromonas, el tiempo que tarda en converger depende de los valores de los parámetros de entrada. Un problema de estos algoritmos de hormigas es que para cada problema particular los parámetros óptimos pueden ser distintos. Además, estos parámetros también afectarán a la probabilidad de llegar a la solución óptima del problema. Sin embargo, para unos parámetros fijos, es bastante probable llegar a una solución buena en un número razonable de pasos para un amplio rango de problemas.

El algoritmo usado en este trabajo se ha modificado para adecuarse al problema que se busca resolver. Sin embargo, el desarrollo es prácticamente idéntico al que se usaría para describirlo de manera genérica (Alg.2). Más adelante en el apartado 3.7 se explicará en detalle la forma concreta en la que se ha implementado.

# 3. Quantum Ant Colony Optimization (QACO)

En este trabajo se ha estudiado un algoritmo cuántico basado en el ACO (Ant Colony Optimization), con sus siglas QACO. Este algoritmo se publicó por primera vez en 2007 por L. Wang, Q. Niu y M. Fei [17]. En dicho trabajo, se propone un algoritmo cuántico análogo al algoritmo ACO en el que cada camino está representado por un estado del sistema. Cada qubit codifica la información acerca de las feromonas depositadas, que se irá actualizando en cada iteración a través de una puerta cuántica de giro.

A partir de este trabajo previo (Sec.3.1), en este capítulo se desarrolla en detalle una propuesta de mejora del algoritmo (Sec.3.2). Tras analizar esta primera versión del algoritmo y encontrar algunos problemas (Secs.3.4-3.3), se propone una variante para resolverlos (Sec.3.5). Para mejorar el rendimiento del algoritmo, se hace una búsqueda de los parámetros de entrada óptimos al algoritmo (Sec.3.6), para luego compararlo con un algoritmo clásico de ACO (Sec.3.7) y con el algoritmo QACO anterior (Sec.3.8).

## 3.1 Contexto previo

Como ya se ha adelantado, la principal contribución de ese trabajo previo es la representación directa de las feromonas en un computador cuántico a través de los estados de los qubits. Para que haya una relación directa entre las probabilidades de obtener cada uno de los estados cuánticos y que pase la hormiga por un camino, se trabaja en el Hyper-Cube Framework para ACO, siendo un camino una arista de este hipercubo [18][a]. Así, el mecanismo de las feromonas está relacionado con el estado de ese qubit. Dado que los estados se preparan inicialmente en el estado base, la información acerca de las feromonas se introduce en el sistema a través de una puerta de rotación. En la primera generación, y dado que no se tiene información previa, esta puerta será una puerta Hadamard. Sea el qubit $i$

$$\tau_i \equiv |\Psi_i\rangle = \alpha_i |0\rangle + \beta_i |1\rangle , \tag{38}$$

la probabilidad de que una hormiga escoja pasar por la arista con su elemento en la posición $i$ a 1 es la misma que la de medir el estado excitado, es decir, $\beta_i$. De esta manera, el estado de las feromonas del problema viene dado por

$$\tau \equiv |\Psi\rangle = |\Psi_1\rangle \otimes |\Psi_2\rangle \otimes \cdots \otimes |\Psi_n\rangle . \tag{39}$$

La principal ventaja de QACO es que la actualización de las feromonas de cada una de las aristas se hace a través de una operación cuántica. Para modificar las probabilidades de obtener cada estado, se aplica una operación de giro sobre cada uno de los qubits $\tau_i$. El ángulo de giro se escoge a través de una tabla de valores (Tab.1) que refuerce positivamente el paso a través de las aristas que son parte de la solución que se está buscando.

Para evitar converger a un máximo local, los algoritmos ACO implementan una estrategia de exploración. Esta estrategia se puede implementar definiendo un parámetro de exploración $pe : 0 \leq pe \leq 1$, que al compararlo con un número aleatorio, hará que la hormiga elija una arista haciendo uso o no del mecanismo de las feromonas.

[a] Por ejemplo, en un sistema con 2 qubits, los posibles caminos que puede recorrer la hormiga son las aristas del hipercubo, en este caso un cuadrado de aristas (0,0), (0,1), (1,0) y (1,1).

Tab. 1: Tabla de valores propuesta para el ángulo de rotación en [17, Tab.2]. $f$ es la función objetivo, $x$ es el estado de la iteración actual, $b$ es el estado de la mejor hormiga encontrada y $S$ es el signo del ángulo de rotación $\Delta\theta$.

| $x_i$ | $b_i$ | $f(x)>f(b)$ | $\Delta\theta_i$ | $S(\alpha_i,\beta_i)$ | | | |
|---|---|---|---|---|---|---|---|
| | | | | $\alpha_i\beta_i > 0$ | $\alpha_i\beta_i < 0$ | $\alpha_i = 0$ | $\beta_i = 0$ |
| 0 | 0 | False | 0.01 π | −1 | +1 | ±1 | ±1 |
| 0 | 0 | True | 0.01 π | −1 | +1 | ±1 | ±1 |
| 0 | 1 | False | 0.025 π | −1 | +1 | ±1 | ±1 |
| 0 | 1 | True | 0.025 π | +1 | −1 | ±1 | ±1 |
| 1 | 0 | False | 0.025 π | +1 | −1 | ±1 | ±1 |
| 1 | 0 | True | 0.025 π | −1 | +1 | ±1 | ±1 |
| 1 | 1 | False | 0.01 π | +1 | −1 | ±1 | ±1 |
| 1 | 1 | True | 0.01 π | +1 | −1 | ±1 | ±1 |

Al igual que con ACO, en este algoritmo se implementa una estrategia de selección de la mejor hormiga en cada generación, siendo una hormiga el estado medido a la salida del circuito cuántico. Para cada configuración de feromonas, se lanza un número fijo de hormigas. Calculando el valor de la función objetivo para cada una de ellas, se escogerá la mejor de todas ellas para actualizar las feromonas y pasar a la siguiente generación. Este algoritmo se presenta en el algoritmo (Alg.5).

**Algoritmo 5** Implementación de L. Wang, Q. Niu y M. Fei de QACO

**Require:** Parámetro de exploración *pe*

1: Inicializar los qubits para que la probabilidad de pasar por una arista sea 1/2.
2: **repeat**
3: Lanzar las hormigas. Si *pe* es mayor que un número generado aleatoriamente, las hormigas exploran como si no hubiera feromonas; sino, las hormigas siguen los rastros de feromonas como en ACO.
4: Medir el resultado de las hormigas.
5: Actualizar las feromonas de cada arista de acuerdo con la tabla valores del ángulo de giro para la mejor hormiga de la iteración.
6: **until** se llega a la condición de salida

## 3.2 Implementación propuesta de QACO

La solución propuesta por L. Wang, Q. Niu y M. Fei, mientras que es fácilmente simulable, no permite una implementación directa en un computador cuántico. En concreto, la tabla de ángulos de rotación requiere conocer con gran detalle los estados cuánticos de cada qubit, tanto en fase como en amplitud. Además, la ramificación del algoritmo en el paso 3, para decidir si explorar o seguir las feromonas, implica un problema a la hora de realizar un circuito cuántico sin aumentar drásticamente el número de qubits necesarios.

La propuesta de QACO en este trabajo que se desarrolla en los siguientes apartados tiene los mismos principios que el algoritmo anterior. De la misma manera, la información de las feromonas que guían a las hormigas se introducen en el estado cuántico a través de una operación cuántica de giro (Sec.3.2.1). Dependiendo de este ángulo, las hormigas tendrán una

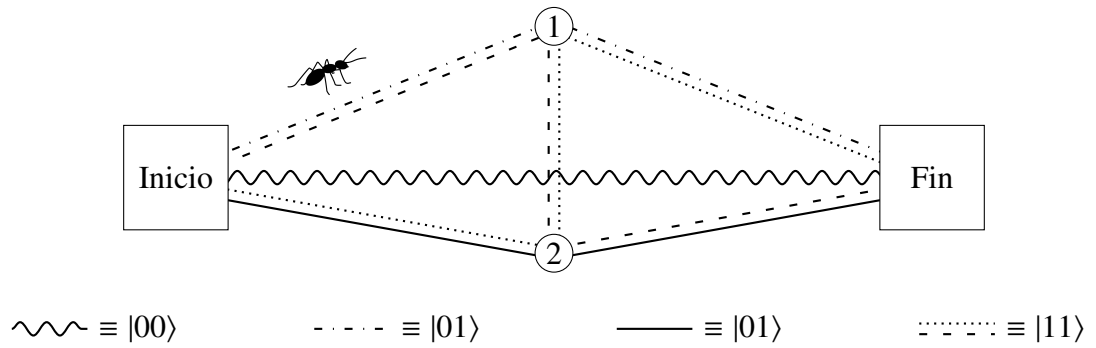



Fig. 6: Visualización del recorrido de una hormiga en los algoritmos ACO y QACO con un ejemplo sobre un grafo de 2 nodos. En ACO, la hormiga llegará al final eligiendo atravesar una de las aristas incidentes al nodo en el que se encuentra sin repetir nodo ni volver hacia atrás. Así, la hormiga llegará a su meta en un máximo de 3 pasos. En total, se tienen 4 posibles caminos de distinta longitud representados por cada uno de los tipos de línea. En este ejemplo, el camino recorrido al seguir la línea rayada o punteada es el mismo, aunque el orden en el que se visitan los nodos cambie. En QACO, la hormiga escoge directamente recorrer uno de los caminos en el momento en el que se mide su estado. Para representar estos caminos en estados cuánticos se pueden usar 2 qubits, en el que cada uno representa haber pasado $|1\rangle$ o no $|0\rangle$ por un nodo.

probabilidad de tomar el estado $|0\rangle$ o $|1\rangle$ para cada uno de sus qubits. Cada uno de los posibles estados del sistema representará un posible camino que pueden seguir las hormigas (Fig.6). Sin embargo, en este trabajo se ha usado una estrategia de giro que permite una implementación real del algoritmo, respetando las limitaciones impuestas al trabajar con estados cuánticos.

El espacio de trabajo del algoritmo se divide en dos partes. Una, los qubits de las hormigas, donde se tiene un número de qubits tal que se puedan representar todos las posibles soluciones del problema. La mayoría de operaciones del algoritmo se realizan sobre estos qubits, como la exploración o la aplicación de las feromonas. Serán solamente estos de los que se extraiga la información al terminar cada iteración. La parte restante corresponde al qubit de exploración, que sirve para codificar la información del parámetro de exploración clásico en un estado cuántico. Se usará como qubit de control para las puertas controladas usadas en el paso de exploración.

En algunos problemas no todos los posibles estados representan una solución válida. Es posible que se obtenga un estado no válido para el problema. A pesar de ello, esta solución se habrá generado a partir de información acerca de una solución correcta. Para estos casos, se propone un nuevo paso que filtre las soluciones válidas y permita extraer información útil para resolver el problema (Sec.3.2.2).

Para la exploración de nuevas soluciones se propone una estrategia de exploración que mantenga la validez de los estados filtrados como soluciones al problema. En problemas con restricciones, se usarán puertas Fredkin para obtener estados que representen soluciones al problema, haciendo que la exploración sea más dirigida que la hecha con puertas cnot (Sec.3.2.3).

Finalmente se propone una estrategia de actualización de feromonas que no depende del conocimiento completo del estado a la salida del circuito (Sec.3.2.4). Aunque en una primera versión del algoritmo los parámetros se han escogido por comparación con el trabajo anterior, más adelante se buscarán los valores óptimos de los ángulos de giro (Sec.3.6).

En el (Alg.6) se muestra una descripción general del algoritmo propuesto. En la figura

(Fig.A.2) se muestra un diagrama del mismo. Se da la versión final del algoritmo en la sección 3.5.

**Algoritmo 6** Propuesta inicial de QACO

1: Inicialización de las feromonas
2: **repeat**
3: Inicializar todos los estados cuánticos al estado base ($|\Psi_i\rangle = |0\rangle \; \forall i$)
4: Aplicar las feromonas y el parámetro de exploración
5: **if** el problema tiene restricciones **then**
6: Operar los qubits de las hormigas, de tal manera que a la salida los únicos estados posibles sean aquellos que representan una solución
7: Usando el qubit de exploración como qubit de control, aplicar la estrategia de exploración para el problema
8: Medir los qubits de las hormigas y se actualizan las feromonas
9: **until** se llega a la condición de salida

### 3.2.1 Aplicación de las feromonas

En el texto de L. Wang, Q. Niu y M. Fei, la aplicación de las feromonas se hace a través de una rotación simple en un plano real. Esta visión acerca de la distribución de estados limita en gran medida la realidad de los mismos. El espacio de los estados posibles de un qubit es en realidad la superficie de la esfera de Bloch.

En la implementación propuesta de QACO, se propone limitar los estados posibles de cada una de las hormigas a la semicircunferencia definida sobre el plano $xz$ en el semiplano de $x$ positivo. Las feromonas en este caso, vienen dadas por rotaciones alrededor del eje $y$, cuyo ángulo dependerá de la cantidad de feromonas depositada en cada arista. En la figura (Fig.7) se muestra una representación sobre la esfera de Bloch de estos estados.

En la primera iteración, todos los caminos tienen la misma cantidad de feromonas, por lo que elegir o no un qubit tendrá una probabilidad de 1/2. Así, la operación de rotación en la primera iteración ($j = 0$) para cada qubit $i$ de las hormigas es

$$\text{—}\boxed{R_y(\theta_{i,0})}\text{—} \quad R_y(\theta_{i,0}) = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}. \tag{40}$$

La actualización del ángulo $\theta_{i,j}$ se desarrollará en el apartado 3.2.4.

### 3.2.2 Generación de soluciones (GenS)

El paso 6 del algoritmo (Alg.6) representado por GenS es un paso condicional, que se aplicará solo en los casos en los que la solución tenga restricciones. En estos casos, tras el paso del sistema por las puertas que aplican las feromonas, se pueden tener soluciones no compatibles con el problema. Sin embargo, tal como pasa en los algoritmos evolutivos, estas soluciones pueden estar generadas a partir de la exploración alrededor de soluciones correctas. Se puede, por lo tanto, extraer información acerca de los resultados correctos en estas situaciones.

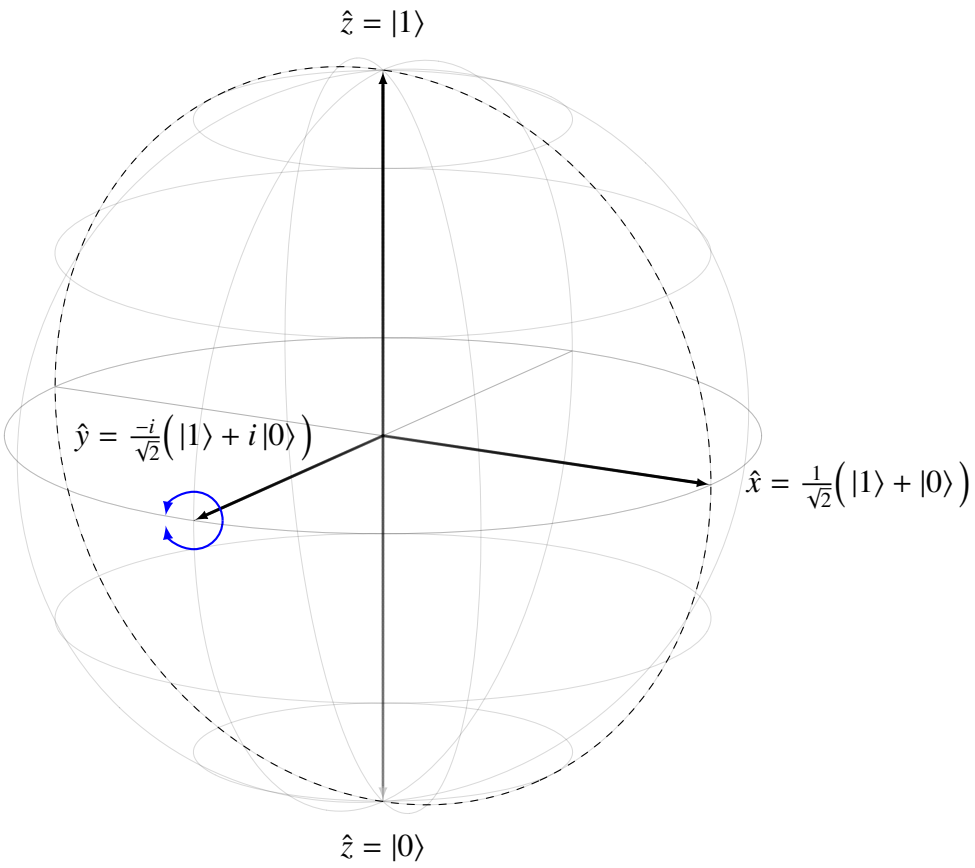



Fig. 7: Esfera de Bloch de un qubit de una hormiga. La línea rayada muestra los estados que puede tomar el qubit, aunque se intenta trabajar solamente en la región de estados con $\hat{x}$ positivo. Cualquiera de estos estados se obtiene a partir de una rotación sobre el eje $\hat{y}$.

Para ello, se propone añadir un paso extra que filtre los estados después de aplicar las feromonas, y que filtre las soluciones válidas. Para ello, se ha diseñado una puerta que actúa sobre los $n$ bits asignados a las hormigas. Los estados cuánticos que a la entrada correspondan a una solución válida se dejan inalterados. Para los que no, se descompone en una suma de estados válidos, con una probabilidad asignada a cada uno de ellos proporcional a lo parecido que sea el estado inicial a cada uno de los estados solución.

El criterio para calcular estos valores ha sido el siguiente. Primero, se cuentan para cada estado diferente a la entrada el número de qubits iguales con respecto de cada posible solución, asignando este número a cada uno de los estados finales. Después, se divide por la suma de las coincidencias para un estado inicial. Para tener una combinación normalizada de estados a la salida, hay que aplicar la raíz cuadrada a todos los valores. En el ejemplo 2 se desarrolla en detalle la forma en la que se puede hacer este cálculo.

Este paso GenS se puede visualizar fácilmente usando la forma matricial de las puertas cuánticas. La matriz asociada depende por lo tanto del tamaño del problema y del número de 1's que deban contener las soluciones. Otro tipo de problemas podrán tener unas restricciones de distinto tipo, pero en todas ellas se podrá aplicar esta estrategia de descomposición. Se muestra en el ejemplo 3 la aplicación numérica a un problema de tamaño 4 y las soluciones restringidas a tener solamente dos 1's en la solución.

**Ejemplo 2:** Obtención de los elementos de GenS

Sea un problema de tamaño 3 con las soluciones restringidas a tener un solo 1, es decir, 001, 010 o 100. Los posibles estados a la entrada se corresponden con los estados de la base computacional.

Empezando con el estado 000, que comparte 2 qubits con cada una de las soluciones, la probabilidad de obtener cada uno de ellas a partir de este estado tiene que ser igual. Teniendo un total de 6 coincidencias, los elementos de la matriz para esta entrada serán todos $\sqrt{2/6} = 1/\sqrt{3}$. Este caso será el mismo que para el estado 111.

Para los estados con dos 1's se tiene que aplicar el cálculo propuesto, por ejemplo, con el estado 011. Este estado comparte 2 qubits iguales con las soluciones 001 y 010, pero no comparte ningún qubit con 100. Teniendo un total de 4 coincidencias, la probabilidad de obtener 001 a partir de 011 será $\sqrt{2/4} = 1/\sqrt{2}$, al igual que para 010.

Así, la matriz GenS para este caso se construye como

$$\mathrm{GenS}(3,1) = \begin{pmatrix} 0 & \sqrt{1/3} & \sqrt{1/3} & & \sqrt{1/3} & & & \\ & 1 & & & & & & \\ & & 1 & & & & & \\ & \sqrt{1/2} & \sqrt{1/2} & 0 & & & & \\ & & & & 1 & & & \\ & \sqrt{1/2} & & & \sqrt{1/2} & 0 & & \\ & & \sqrt{1/2} & & \sqrt{1/2} & & 0 & \\ & \sqrt{1/3} & \sqrt{1/3} & & \sqrt{1/3} & & & 0 \end{pmatrix}.$$

Hay que notar que la aplicación de esta operación GenS sobre un estado normalizado, no da en general un estado normalizado a la salida. Esta operación solo da como resultado un estado normalizado si a la entrada se tiene un estado de la base canónica. Tras aplicar esta operación, el resultado se tiene que normalizar para mantener la coherencia con la condición de norma-lización de la probabilidad.

### 3.2.3 Estrategia de exploración (Explor)

El problema de implementar una estrategia de exploración se basa en encontrar una herramienta que permita cambiar el estado de los qubits, y así que se escoja un camino distinto al dado por las feromonas. Algunos autores usan solamente puertas cnot [19] para realizar estas operaciones. Esta estrategia es útil en los casos en los que el problema no está restringido, ya que permite la exploración qubit a qubit. La implementación de esta estrategia es sencilla. Basta con aplicar de manera sucesiva una puerta cnot sobre cada uno de los qubits de las hormigas, usando un qubit de control auxiliar, que llamaré qubit de exploración. Colocando el qubit de exploración en primera posición, la matriz que representa esta operación (Explor) se calcula como

$$Explor = \prod_{i=2}^{n+1} cnot(1,i), \tag{41}$$

donde se ha usado la misma notación que en la ecuación (Eq.34).

**Ejemplo 3:** Aplicación de GenS

Sea un problema de tamaño 4 con las soluciones restringidas a tener 2 unos. La matriz GenS en este caso será

$$\text{GenS}(4,2) = \begin{pmatrix}
0 & & \sqrt{1/6} & & \sqrt{1/6} & \sqrt{1/6} & & \sqrt{1/6} & \sqrt{1/6} & & \sqrt{1/6} & & \\
 & 0 & 1/2 & & 1/2 & \sqrt{1/12} & & 1/2 & \sqrt{1/12} & & \sqrt{1/12} & & \\
 & & 0 & 1/2 & & \sqrt{1/12} & 1/2 & & \sqrt{1/12} & 1/2 & & \sqrt{1/12} & \\
 & & & 1 & & & & & & & & & \\
 & & & \sqrt{1/12} & 0 & 1/2 & 1/2 & & \sqrt{1/12} & \sqrt{1/12} & & 1/2 & \\
 & & & & & 1 & & & & & & & \\
 & & & & & & 1 & & & & & & \\
 & & & 1/2 & & 1/2 & 1/2 & 0 & \sqrt{1/12} & \sqrt{1/12} & & \sqrt{1/12} & \\
 & & & \sqrt{1/12} & & \sqrt{1/12} & \sqrt{1/12} & & 0 & 1/2 & & 1/2 & \\
 & & & & & & & & & 1 & & & \\
 & & & & & & & & & & 1 & & \\
 & & & 1/2 & & \sqrt{1/12} & \sqrt{1/12} & & 1/2 & 1/2 & 0 & \sqrt{1/12} & \\
 & & & & & & & & & & & 1 & \\
 & & & \sqrt{1/12} & & 1/2 & \sqrt{1/12} & & 1/2 & \sqrt{1/12} & & 1/2 & 0 \\
 & & & \sqrt{1/12} & & \sqrt{1/12} & 1/2 & & \sqrt{1/12} & 1/2 & & 1/2 & \\
 & & & \sqrt{1/6} & & \sqrt{1/6} & \sqrt{1/6} & & \sqrt{1/6} & \sqrt{1/6} & & \sqrt{1/6} & \\
 & & & & & & & & & & & & 0
\end{pmatrix}.$$

En el caso de que la solución obtenida no sea válida por no tener el número correcto de unos, la aplicación de este paso permite obtener información válida.

Supongamos que se ha obtenido el estado 1011. Dado que tiene 3 unos, será una solución incorrecta. Sin embargo, sí que es un estado cercano a soluciones que son válidas. En concreto, está a un solo bit de distancia de las soluciones 1010, 1001 y 0011, y a 3 del resto de soluciones. Esto quiere decir que el estado obtenido representará con una probabilidad 3 veces mayor a los estados 1010, 1001 y 0011 que al resto. De esta manera, si el estado 1011 se representa en la base computacional ($|s\rangle$=[0000100000000000]) y se multiplica por la matriz GenS, se obtendrá

$$|x\rangle = \hat{GenS}\,|s\rangle \equiv x = s \cdot GenS = [0,0,0,\sqrt{\frac{1}{12}},0,\frac{1}{2},\frac{1}{2},0,0,\sqrt{\frac{1}{12}},\sqrt{\frac{1}{12}},0,\frac{1}{2},0,0,0].$$

Dado que x representa la amplitud del estado, para obtener la probabilidad de obtener uno de los estados de la base hay que calcular el módulo cuadrado de cada elemento

$$P(x) = |x|^2 = [0,0,0,1/12,0,1/4,1/4,0,0,1/12,1/12,0,1/4,0,0,0].$$

Viendo la probabilidad, ahora se ve claramente la relación de 1 a 3 de obtener uno de los estados 1010, 1001 o 0011 frente al resto. Se puede comprobar que el estado $|x\rangle$ verifica la condición de normalización de la probabilidad (Eq.2).

En los algoritmos clásicos se suele definir un parámetro que indica la probabilidad de que una hormiga explore en cada uno de los pasos. Esta información se puede contener de forma directa en un qubit. Definiendo este qubit como

$$|\Psi_e\rangle = \alpha_e\,|0\rangle + \beta_e\,|1\rangle\,, \tag{42}$$

y teniendo la puerta cnot típica, es fácil ver que la probabilidad de cambiar el estado del qubit objetivo es $\beta_e^2$.

**Ejemplo 4:** Exploración con puertas Fredkin vs puertas cnot

Sea un problema de tamaño 3 con las soluciones restringidas a estados con un solo 1. Supongamos que el qubit de exploración está preparado de tal manera que la probabilidad de explorar sea 1/2. Supongamos también que tras aplicar las feromonas el estado de los qubits de las hormigas es 001. De esta manera, el estado del sistema completo estará dado por

$$|\Psi\rangle = \frac{1}{\sqrt{2}}\left(|0001\rangle + |1001\rangle\right).$$

Si se explora usando puertas cnot, el estado a la salida vendrá dado por

$$|\Psi_{cnot}\rangle = cnot(1,2)\cdot cnot(1,3)\cdot cnot(1,4)\,|\Psi\rangle = \frac{1}{\sqrt{2}}\left(|0001\rangle + |1110\rangle\right).$$

Si en cambio se usa la estrategia de exploración propuesta, el estado final será

$$|\Psi_{fred}\rangle = \begin{cases} fred(1,2,3)\cdot fred(1,2,4)\cdot fred(1,3,4)\,|\Psi\rangle = \frac{1}{\sqrt{2}}\left(|0001\rangle + |1010\rangle\right), \\ fred(1,2,3)\cdot fred(1,3,4)\cdot fred(1,2,4)\,|\Psi\rangle = \frac{1}{\sqrt{2}}\left(|0001\rangle + |1100\rangle\right), \\ fred(1,2,4)\cdot fred(1,2,3)\cdot fred(1,3,4)\,|\Psi\rangle = \frac{1}{\sqrt{2}}\left(|0001\rangle + |1001\rangle\right), \\ fred(1,2,4)\cdot fred(1,3,4)\cdot fred(1,2,3)\,|\Psi\rangle = \frac{1}{\sqrt{2}}\left(|0001\rangle + |1100\rangle\right), \\ fred(1,3,4)\cdot fred(1,2,3)\cdot fred(1,2,4)\,|\Psi\rangle = \frac{1}{\sqrt{2}}\left(|0001\rangle + |1001\rangle\right), \\ fred(1,3,4)\cdot fred(1,2,4)\cdot fred(1,2,3)\,|\Psi\rangle = \frac{1}{\sqrt{2}}\left(|0001\rangle + |1010\rangle\right). \end{cases}$$

Se puede ver que en el caso de la exploración con puertas cnot el estado a la salida no representa una solución válida al problema. Por el contrario, al usar la estrategia de exploración con puertas Fredkin, cualquier orden de exploración da soluciones válidas a la salida.

Para preparar el qubit de exploración en este estado, basta con aplicar una puerta de rotación sobre el eje y al estado inicial

$$\boxed{R_y(\theta_e)} \quad R_y(\theta_e) = \begin{pmatrix} \sqrt{1-\beta_e^2} & -\beta_e \\ \beta_e & \sqrt{1-\beta_e^2} \end{pmatrix} \rightarrow \theta_e = 2\mathrm{atan}(\beta_e). \tag{43}$$

Para otro tipo de problemas, en los que las soluciones tienen restricciones en el número de estados excitados, esta opción puede no ser válida. Cuando cambiar el estado de un qubit implicaría afectar al resto de qubits al mismo tiempo, es necesario encontrar otra forma para explorar. La puerta Fredkin puede ser útil para conseguir ese objetivo, ya que intercambia de forma efectiva los estados de dos qubits si el qubit de control está activado. Esto permite mantener constante el número de estados excitados, y por lo tanto, mantener también la validez del estado como solución. En el ejemplo 4 se muestra esta diferencia para un caso concreto.

La estrategia de exploración es similar a la anterior. Para cada par de qubits de hormigas posibles, se coloca una puerta Fredkin controlada por el qubit de exploración. Ahora, el número de puertas controladas necesarias aumenta de $n$ a $n(n-1)/2$. A cambio, se consigue una exploración dirigida de las hormigas.

Hay que resaltar que siempre es posible alterar el orden en el que se realicen cada una de las operaciones con las puertas Fredkin, ya que la composición de estas puertas siempre darán un operador unitario. Esto se puede demostrar por aplicación directa del siguiente resultado:

**Lema 1.** *El producto de operadores unitarios, es un operador unitario.*

### 3.2.4 Estrategia de actualización de las feromonas

Después de obtener el resultado de la hormiga de la iteración $i$ ($x_i$), esta se evalúa con el valor de la función objetivo ($f(x)$). Para ello, se modifican los valores de las puertas que actúan como feromonas según la siguiente tabla (Tab.2).

Tab. 2: Tabla de valores del ángulo de giro propuesta. Los valores con * se multiplican por -1 si $\cos(\theta_i/2)<0$.

| $x_i$ | $b_i$ | f($x_i$) mejor que f($b_i$)? | $\Delta\theta_i$ |
|---|---|---|---|
| 0 | 0 | True | -0.025 $\pi$* |
| 0 | 0 | False | 0.005 $\pi$ |
| 0 | 1 | True | -0.05 $\pi$* |
| 0 | 1 | False | 0.01 $\pi$ |
| 1 | 0 | True | 0.05 $\pi$* |
| 1 | 0 | False | -0.01 $\pi$ |
| 1 | 1 | True | 0.025 $\pi$* |
| 1 | 1 | False | -0.005 $\pi$ |

Para escoger los valores de los ángulos de giro, se han tomado como referencia los valores de la tabla 1 de [17], en concreto la relación 5/1 entre resultados buenos/malos. Pero mientras que en ese trabajo algunos valores de salida no modificaban el ángulo de giro, en este caso se ha elegido reforzar positivamente el hecho de haber obtenido un resultado bueno, y reforzar aún más el caso en el que se haya encontrado un resultado mejor tras la exploración de un nuevo camino. La filosofía detrás de los valores elegidos está en limitar los posibles estados de los qubits de las hormigas a una semicircunferencia de la superficie de la esfera de Bloch. Además, cuando se llega a una situación de convergencia, el ángulo de giro oscila alrededor del resultado. Cuando se sabe con cierta certeza que el estado de un qubit tiene que ser $|0\rangle$ o $|1\rangle$, el ángulo de giro cambia de signo cuando se pasa al semiplano izquierdo y se ha encontrado el mismo estado, haciendo que el resultado oscile alrededor de la solución encontrada. Si el estado encontrado es el mismo un número dado de veces (*converCondition*), se tendrá el resultado del problema.

## 3.3 Problema de reversibilidad y solución

Como se ha comentado en la introducción, las puertas lógicas cuánticas tienen que estar representadas por matrices unitarias. Inmediatamente salta a la vista que la matriz de la operación de generación de soluciones GenS, aunque conserve la probabilidad, no es unitaria.

En un computador cuántico, para implementar una operación no unitaria hace falta medir el estado del sistema. Esto atenta contra el principio de los computadores cuánticos, que buscan evolucionar un estado sin destruirlo. En ciertos casos, para resolver este problema, se puede reorganizar el orden de las operaciones de tal manera que todas las mediciones se hagan al final de cada programa. Esto suele ser posible en problemas en los que se tienen qubits auxiliares, que no necesariamente tienen que ser medidos para obtener la solución. Sin embargo, en este algoritmo esto no ocurre. Solo se tiene un qubit auxiliar, por lo que no se tiene espacio suficiente como para contener la información necesaria para aplicar la operación GenS.

La operación GenS realiza un filtrado de los estados del sistema, dejando solamente aquellos que proporcionan una solución válida al sistema. Pero este filtrado no influye en la capacidad de exploración de las hormigas. Ambas operaciones conmutan, es decir, son intercambiables. Para demostrarlo, se usan las definiciones vistas de las operaciones GenS y Explor, y la siguiente propiedad de los conjuntos completos de operadores que conmutan.

**Teorema 1** (Teorema de compatibilidad [4, pp. 23-34])**.** *Sean 2 operadores A y B, las siguientes afirmaciones son equivalentes:*

1. *A y B son operadores compatibles.*
2. *A y B tienen una base común de estados propios.*
3. *A y B conmutan, es decir,* $[A,B]=AB-BA=0$.

*Demostración.* Es fácil ver que los valores propios de GenS son 1 para los estados dentro del espacio de las soluciones y 0 para los estados fuera, nada más que atendiendo a los valores de su diagonal y sus respectivas columnas. Se tiene además que los estados propios son vectores de la base canónica, por lo que es fácil ver que

$$\mathrm{GenS} \cdot \mathrm{D}_G = \mathrm{D}_G, \text{ en la base canónica,} \tag{44}$$

con $\mathrm{D}_G$ la matriz cuya diagonal está formada por los valores propios de GenS.
Dado que Explor es el producto de una serie de puertas Fredkin, basta con demostrar que cualquiera de ellas conmuta con $\mathrm{D}_G$ para demostrar que Explor conmuta con $\mathrm{D}_G$. La matriz asociada a una puerta Fredkin (F) es una matriz identidad salvo 2 elementos que se intercambian. Con esto se puede separar el espacio en 2 conjuntos, uno asociado a la parte de la identidad ($\mathrm{F}_i$) y otro de dimensión 2 asociado a la parte del intercambio ($\mathrm{F}_c$), con $\mathrm{F}_i \cup \mathrm{F}_c = \mathbb{R}^n$ y $\mathrm{F}_i \cap \mathrm{F}_c = \emptyset$. Por definición de la puerta Fredkin, $\mathrm{F}_i$ es un subespacio con elementos que corresponden a estados con el mismo número de 1's.
GenS se puede dividir en dos subespacios, uno asociado a los estados solución $\mathrm{GenS}_s$ y el otro al resto $\mathrm{GenS}_r$. Dado el teorema (Teo.1), para que GenS y Fredkin conmuten, basta con demostrar que se da alguna de las dos $\mathrm{F}_c \subset \mathrm{GenS}_s$ o $\mathrm{F}_c \subset \mathrm{GenS}_r$. Esto es fácil de demostrar viendo que $\mathrm{GenS}_s$ está asociado a un espacio con elementos que corresponden a estados con un mismo número de 1's. Si el número de 1's de los estados asociados a $\mathrm{F}_c$ y $\mathrm{GenS}_s$ es el mismo, entonces $\mathrm{F}_c \subset \mathrm{GenS}_s$, y en el caso contrario $\mathrm{F}_c \subset \mathrm{GenS}_r$. Por lo tanto, se tiene que

$$[Explor, F] = 0 \;\Rightarrow\; [\mathrm{Explor}, \mathrm{D}_G] = 0. \tag{45}$$

Se escribe ahora la propiedad que queremos demostrar

$$[\mathrm{GenS}, \mathrm{Explor}] = 0 \;\Leftrightarrow\; \mathrm{GenS} \cdot \mathrm{Explor} = \mathrm{Explor} \cdot \mathrm{GenS}. \tag{46}$$

Se multiplica la segunda expresión por la matriz $\mathrm{D}_G$ a ambos lados

$$\mathrm{GenS} \cdot \mathrm{Explor} \cdot \mathrm{D}_G = \mathrm{Explor} \cdot \mathrm{GenS} \cdot \mathrm{D}_G. \tag{47}$$

A la izquierda se intercambian Explor y $\mathrm{D}_G$ usando el resultado de (45), y a la derecha se introduce (44)

$$\mathrm{GenS} \cdot \mathrm{D}_G \cdot \mathrm{Explor} = \mathrm{Explor} \cdot \mathrm{D}_G. \tag{48}$$

Finalmente, se vuelve a aplicar (44) a la izquierda

$$\mathrm{D}_G \cdot \mathrm{Explor} = \mathrm{Explor} \cdot \mathrm{D}_G, \tag{49}$$

que es cierta por (45), por lo que (46) también es cierta. □

Teniendo la operación GenS justo antes de la medición de los estados cuánticos, uno puede plantearse si realizar esta operación en el dominio cuántico aporta alguna ventaja frente a realizar el cálculo de manera ordinaria. Esta operación de filtrado solo es útil para sumar la probabilidad de obtener estados no válidos a la de obtener estados válidos. Por lo tanto, la cantidad de información antes y después de aplicar la operación es la misma. Si se realiza esta operación sobre un estado ya medido y luego se aplica la operación GenS, no se estará perdiendo información. Basta con elegir uno de los nuevos estados en función de la probabilidad de cada uno de ellos.

Después de este análisis, es posible afirmar que se puede llevar esta operación al final de cada iteración del algoritmo, permitiendo implementar el algoritmo de forma cuántica a través de puertas reversibles.

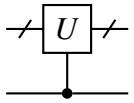
## 3.4 Posible problema del orden de la exploración y solución

Usando un argumento totalmente similar al usado en el apartado anterior, se puede demostrar que las puertas Fredkin controladas por un mismo qubit solo conmutan consigo mismas o con otras puertas si todos los qubits objetivo son distintos. Así, se tiene la siguiente regla de conmutación

$$[Fred(n,a,b), Fred(n,c,b)] \begin{cases} = 0 & \text{si } (a \neq b \neq c \neq d) \text{ o } (a = b \, y \, b = d), \\ \neq 0 & \text{en el resto de casos.} \end{cases} \tag{50}$$

Esto indica que el orden en el que se apliquen las puertas afectará al resultado de la exploración. De hecho, dependiendo del orden en el que se multipliquen las matrices asociadas, la matriz correspondiente a todo el bloque de exploración será distinto. Debido a esto, se ha decidido buscar una nueva estrategia de exploración. Siendo la principal característica de la estrategia propuesta en este trabajo, se deberá mantener la paridad de los estados de entrada. Sobre esta idea se han pensado varias soluciones.

La primera consiste en generar una puerta que distribuya uniformemente la probabilidad de un estado sobre todos los estados con el mismo número de 1's a una distancia de una sola puerta Fredkin[a]. Sin embargo, estas matrices no son unitarias, por lo que tienen el mismo problema que presentaba la propuesta inicial de GenS. Esto se muestra fácilmente para un problema de dimensión 3




$$U = \begin{pmatrix} 1 & & & & & & & \\ & 0 & \sqrt{1/2} & & \sqrt{1/2} & & & \\ & \sqrt{1/2} & 0 & & \sqrt{1/2} & & & \\ & & & 0 & & \sqrt{1/2} & \sqrt{1/2} & \\ & \sqrt{1/2} & \sqrt{1/2} & & 0 & & & \\ & & & \sqrt{1/2} & & 0 & \sqrt{1/2} & \\ & & & \sqrt{1/2} & & \sqrt{1/2} & 0 & \\ & & & & & & & 1 \end{pmatrix}, \; U^{\dagger} \neq U^{-1}. \tag{51}$$

[a] Esta distancia se refiere al número puertas Fredkin que se necesitan para cambiar de un estado a otro. Por ejemplo, 1100 y 1010 están a una distancia 1 ya que solo se intercambian los elementos 2-3. En cambio, 1100 y 0011 están a una distancia 2 ya que se tienen que intercambiar los elementos 1-3 y 2-4.

Otra posible solución se basa en una estrategia útil a la hora de diseñar circuitos cuánticos. Ya que las puertas cuánticas son autoadjuntas y unitarias, dos aplicaciones sucesivas de una misma puerta es igual a la identidad. Así, para cualesquiera operadores unitarios y simétricos $U^{(k)}$ del mismo tamaño se tiene que

$$U^{(1)} \cdot U^{(2)} \cdots U^{(n)} \cdot U^{(n)} \cdots U^{(2)} \cdot U^{(1)} = \mathbb{1}. \tag{52}$$

Se puede pensar que la aplicación o no de una puerta controlada depende del qubit de control. Por lo tanto, si después de aplicar la estrategia de exploración se aplica nuevamente pero con el orden de las puertas invertidas, se tendría una mejor distribución de las exploraciones. Sin embargo, las puertas controladas no miden el estado del qubit de control para aplicarse, sino que actúan sobre la superposición de todos los estados. Esto hace que esta nueva manera de explorar resulte en una matriz identidad, lo que haría inútil toda esta parte del circuito.

Una última posible solución consiste en reordenar de manera aleatoria las distintas puertas Fredkin que componen Explor. Al dejar explorar en un orden aleatorio en cada iteración, el promedio de todas las exploraciones darían lugar a una exploración homogénea, manteniendo la paridad de los estados. Esta fórmula parece la solución adecuada, por lo que se han hecho pruebas comparándolo con la estrategia inicial. Los resultados se muestran en la tabla (Tab.3).

Los errores medios cometidos usando los dos métodos de exploración son suficientemente similares, con una diferencia máxima de 0.386%. Además, la diferencia media entre los errores cometidos, aunque es apreciable, no excede el 3% en ningún caso. Analizando las curvas de dispersión de convergencia obtenidas a partir de los cálculos realizados para (Tab.3) con ambos métodos, no se observa ninguna diferencia a simple vista, así que se puede deducir que las dos estrategias son equivalentes o suficientemente parecidas como para considerarlas así.

Salvo en el caso en el que cambiar el circuito en cada iteración no sea una opción viable en un computador cuántico real, sería conveniente aplicar la estrategia de ordenar aleatoriamente las puertas Fredkin en cada iteración para hacer una exploración uniforme de las soluciones.

## 3.5 Algoritmo implementado

Tras haber discutido los posibles problemas de la propuesta inicial, se han incorporado las soluciones encontradas a cada uno de ellos. A la hora de implementar el circuito cuántico para simularlo, se ha usado la versión del algoritmo mostrada en el algoritmo (Alg.7) y en forma de circuito cuántico en la figura (Fig.8). El diagrama del circuito para una iteración del algoritmo se muestra en la figura (Fig.A.3) del anexo.

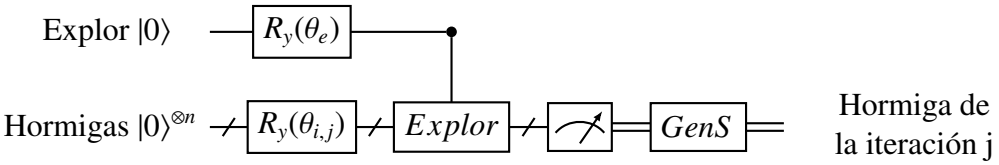



Fig. 8: Circuito para el algoritmo QACO propuesto en la iteración j. El paso de la actualización de las feromonas es un paso que se realiza una vez se tiene el resultado de la hormiga de la iteración.

Tab. 3: Porcentajes de errores cometidos al usar la estrategia de exploración inicial y la nueva propuesta con reordenación aleatoria. Dif. abs. media indica la media del valor absoluto de la diferencia en cada prueba. Las pruebas se han hecho usando 100 matrices aleatorias por cada tipo de problema, con *maxIter* = 80, *converCondition* = 10 y *nRepeticiones* = 500.

| | n=4 | n=5 | n=6 | | n=7 | |
|---|---|---|---|---|---|---|
| | m=2 | m=2 | m=2 | m=3 | m=2 | m=3 |
| Error medio: | | | | | | |
| b=0.25 | 9.262 | 22.812 | 35.460 | 43.81 | 45.596 | 61.746 |
| b=0.5 | 9.882 | 22.418 | 35.372 | 44.628 | 46.466 | 61.288 |
| b=0.75 | 9.396 | 22.122 | 34.650 | 43.966 | 46.004 | 60.876 |
| Error medio nuevo: | | | | | | |
| b=0.25 | 9.282 | 22.800 | 35.354 | 44.154 | 45.526 | 61.972 |
| b=0.5 | 10.252 | 22.578 | 35.400 | 44.806 | 46.424 | 61.464 |
| b=0.75 | 9.474 | 22.212 | 34.930 | 44.054 | 45.816 | 61.262 |
| Dif. abs. media: | | | | | | |
| b=0.25 | 1.424 | 2.196 | 2.338 | 2.228 | 2.846 | 2.074 |
| b=0.5 | 1.346 | 2.108 | 2.408 | 2.678 | 2.426 | 2.572 |
| b=0.75 | 1.338 | 1.902 | 2.468 | 2.516 | 2.544 | 2.426 |

**Algoritmo 7** Implementación final de QACO

1: Inicializar feromonas.
2: **repeat**
3: Inicializar todos los estados cuánticos al estado base ($|\Psi_i\rangle = |0\rangle \; \forall i$).
4: Apliacr las feromonas y el parámetro de exploración.
5: Usando el qubit de exploración como qubit de control, aplicar la estrategia de exploración para el problema.
6: Medir los qubits de las hormigas.
7: Evaluar el resultado medido.
8: **if** el resultado obtenido no es válido **then**
9: Repartir la probabilidad de haber un estado válido según la matriz GenS.
10: Usando esta distribución de probabilidades, elegir un resultado válido.
11: Actualizar las feromonas.
12: **until** se llega a la condición de salida

## 3.6 Optimización de parámetros de QACO

En el trabajo donde se propone el algoritmo de QACO [17] se plantea un análisis sobre cuál es el mejor parámetro de exploración. Sin embargo, no se profundiza en cuál es la mejor condición de salida del algoritmo ni se desarrolla el proceso por el cual se han elegido los valores de la tabla de actualización de las feromonas. Para resolver estas incógnitas, en este apartado se desarrolla una estrategia para obtener estos parámetros.

El objetivo de esta búsqueda de parámetros es minimizar el número medio de iteraciones (hormigas) para llegar al resultado. Se fija además una restricción, el error cometido tiene

que ser menor que 1.5%, lo que daría unos resultados suficientemente fiables. Las variables de decisión usadas para llegar al objetivo serán: el parámetro de exploración $\beta_e$ (Sec.3.2.3), el número de veces que se tiene que repetir un resultado para salir del ciclo *converCondition* (Sec.3.2.4) y todos los valores de la tabla de actualización de las feromonas (Tab.2). Dado que los valores de la tabla son simétricos, tan solo hay que ajustar 4 de estos valores. Esto hace que se tenga un problema de optimización en 6 dimensiones con una restricción.

Para resolver este tipo de problemas se pueden usar varios algoritmos, y entre todos ellos se ha escogido el más simple, el método de descenso de coordenadas [20]. Se ha implementado este algoritmo usando una propiedad útil de este problema: el error de los resultados decrece monótonamente con *converCondition*, mientras que aumenta de la misma manera el número medio de hormigas lanzadas. Sin embargo, el programa desarrollado no conseguía llegar al resultado en un tiempo razonable, por lo que se ha tenido que buscar otra forma de hacer estos cálculos.

La librería de Matlab "Global Optimization Toolbox" proporciona diferentes funciones para la búsqueda de mínimos globales en funciones n-dimensionales. Entre estas funciones, "surrogateopt" implementa un algoritmo de optimización basado en la búsqueda, no sobre la función objetivo, sino sobre una aproximación de esta. "Surrogate optimization" está pensado para optimizar parámetros en funciones objetivo que requieren de demasiado tiempo de cálculo para obtener el resultado de la función. Aunque otros algoritmos, como los genéticos, pueden dar una mejor solución, se ha elegido "surrogateopt" dado el coste exponencial de simular el circuito que implementa QACO.

Con intención de encontrar una regla general para escoger los parámetros, se ha lanzado "surrogateopt" para cada tipo distinto de problema desde tamaño 4 hasta 7. Dado que este tipo de algoritmo de optimización puede caer en un mínimo local, y estando la posibilidad abierta de que varios conjuntos de parámetros den una misma solución, se ha lanzado varias veces este programa. Los resultados obtenidos se muestran en la tabla (Tab.A.1) del anexo.

Viendo los resultados obtenidos, no parece que los parámetros de entrada guarden ninguna relación entre ellos. Esto parece indicar que, o bien la elección de estos parámetros es poco relevante en cuanto a la optimización, o bien que el número medio de iteraciones es una función que fluctúa fuertemente. Cualquiera de las dos opciones impide encontrar ninguna regla general para escoger estos parámetros. Por ello, se ha decidido seguir con los valores para los ángulos de giro propuestos en la tabla (Tab.2) y un parámetro de exploración $\beta_e = 0.95$.

En los resultados de la tabla sí que se observa una progresión clara en el parámetro de *converCondition* y en número de iteraciones medias. Si se representan estos valores frente al tamaño del espacio de soluciones a cada problema (Fig.9), se observa una progresión aproximadamente proporcional a la raíz cuadrada. Haciendo un ajuste del parámetro *converCondition*, se obtiene que este parámetro crece con el número de combinaciones posibles (*nComb*) como

$$\text{converCondition(nComb)} = (9 \pm 2)\ \text{nComb}^{0.66\pm0.04} - (12 \pm 5). \tag{53}$$

Se observa también que el número medio de iteraciones hasta la obtener el resultado crece de manera proporcional a *converCondition* como

$$\text{iterM} = (1.347 \pm 0.002)\ \text{converCondition}. \tag{54}$$

Para dar una idea de la utilidad de estos resultados, uno puede pensar en la diferencia entre las iteraciones medias y *converCondition*. Este dato da cuenta del la media de las iteraciones

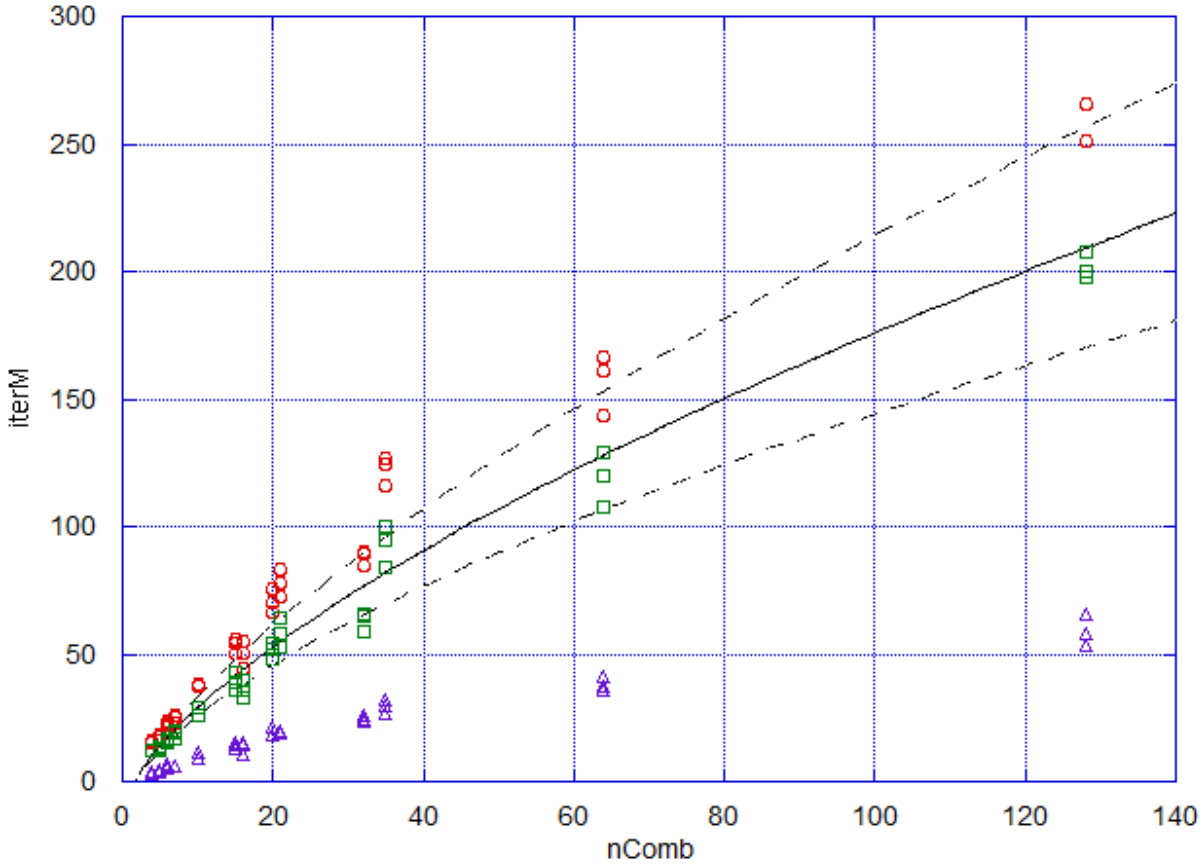



Fig. 9: Iteraciones medias (círculo), *converCondition* (cuadrado) e iteraciones medias - *converCondition* (triángulo) frente al número total de soluciones a un problema. La curva muestra el ajuste de la ecuación (Eq.53) para *converCondition* (línea sólida), con la desviación dada por el error en el exponente (línea rayada).

necesarias para llegar a la solución del problema por primera vez. Como se hace en algunos algoritmos de optimización, en vez de imponer un criterio de salida al programa, se puede usar un número máximo de iteraciones tras las cuales se da el resultado (*maxIter*). Sin poder llegar a demostrar este resultado, parece claro que en caso de querer poner como condición de salida de QACO un número dado de iteraciones, este valor estaría dado por *iterMedia-converCondition*,

$$\text{maxIter} = \text{iterM} - \text{converCondition} = (12 \pm 3)\ \text{nComb}^{0.66\pm0.04} - (16 \pm 7). \tag{55}$$

## 3.7 Comparación de QACO con ACO

Para comprobar la eficiencia del algoritmo propuesto, se tiene que comparar con los algoritmos de hormigas clásicos. Dado que la forma en la que trabajan ambos tipos de algoritmos es totalmente diferente, para comprobar la eficiencia no se pueden usar benchmarks clásicos. Una forma usual de comparar los algoritmos cuánticos con los clásicos es analizar el número de iteraciones necesarias para llegar al resultado con cierta precisión. Por ejemplo, el algoritmo de búsqueda clásico tiene una complejidad de $O(n)$, mientras que la complejidad algoritmo de búsqueda de Grover es solamente $O\left(\sqrt{n}\right)$. Para hacer este cálculo, se tiene en cuenta el número de veces que se tiene que hacer pasar el estado cuántico a través de las puertas cuánticas que aplican la operación de búsqueda [2].

Usando una idea similar, para calcular la velocidad de convergencia de QACO se han contado el número de iteraciones necesarias de media para llegar al resultado. Por analogía con el algoritmo clásico de ACO, este número de iteraciones se puede comparar con el número medio de hormigas lanzadas desde el principio del algoritmo[a]. Para hacer una comparación válida, el error obtenido al usar ambos algoritmos tiene que ser equivalente, por lo que además del ajuste de parámetros hecho para QACO, también se necesita hacer el mismo trabajo para cualquier algoritmo ACO que se use.

Dado que existe un gran número de variaciones de los algoritmos ACO, se ha decidido restringir la búsqueda a uno de ellos: un algoritmo ACO puro [16, Alg.2.6] usando el framework HCF [18, Alg.2]. Esta elección busca comparar los resultados de los dos algoritmos que más se parecen por la forma en la que se trata cada una de las hormigas generadas y la actualización de las feromonas. Esta comparación puede dar como resultado la magnitud de la mejora de la implementación cuántica respecto a la clásica. QACO no implementa ninguna estrategia de optimización de hormigas elitistas (típico de algoritmos de sistemas de hormigas AS) ni de búsqueda de extremos locales alrededor de cada una de las hormigas generadas (aplicados en algoritmos de sistema de colonia de hormigas ACS) [16]. Por lo tanto, para poder hacer una comparación entre dos algoritmos equivalentes, no se van a usar ninguno de estos algoritmos mejorados. El algoritmo usado es una adaptación directa de la versión de ACO desarrollada anteriormente (Alg.2), salvo que el objetivo será maximizar una función.

Se ha implementado este algoritmo de tal manera que sea equivalente en sus entradas y salidas al programa que implementa QACO. De esta manera, las comparaciones que se harán usando los mismos problemas y tan solo se lanzará una hormiga por generación. Los parámetros elegidos para ACO se han obtenido de la referencia usada, con el parámetro de exploración $b = 0.9$ y el de evaporación $\rho = 0.1$ [16, Alg.2.2].

Se ha hecho una prueba de ambos algoritmos en la que se compara la velocidad de convergencia en término de número de iteraciones y la eficacia para llegar al resultado del problema. Los resultados obtenidos se muestran en la tabla (Tab.A.2) del anexo. De estos resultados uno puede deducir que ambos algoritmos tienen un rendimiento parecido a la hora de dar un resultado correcto al problema. Por lo general, en problemas con restricciones QACO llega a la solución en un número ligeramente menor comparado con ACO. Sin embargo, en problemas libres ACO muestra una velocidad de convergencia significativamente mayor.

Sin embargo, no debe olvidarse que una iteración de un algoritmo cuántico puede ser más rápida que una iteración de un algoritmo clásico, sobre todo para problemas grandes.

## 3.8 Comparación de QACO con QACO previo

De manera similar a la que se ha comparado el algoritmo desarrollado con el algoritmo ACO, se va a comparar con el algoritmo del trabajo en el que se propuso inicialmente este algoritmo [17].

En ese trabajo proponen el problema de encontrar el extremo global de una función real de 2 variables en una región dada. Para codificar los puntos en el plano utilizan una representación en 30 qubits, dando a cada uno de los ejes 15 de los qubits. Dado que la solución de los problemas puede caer sobre los ejes, es necesario usar una codificación adecuada.

[a] Notar que el número de hormigas lanzadas no representa la complejidad de ACO, ya que para generar cada una de ellas se necesitan del orden de $O(n)$ pasos.

Para asegurar una búsqueda adecuada de los extremos, se ha decidido usar el código Gray. Este código tiene la ventaja de que la distancia entre valores contiguos es de un solo bit. Dado que la región de búsqueda contiene valores positivos y negativos, se ha decidido usar una representación de signo-magnitud, en el que la magnitud de los signos positivos está dada por un número binario natural y la de los negativos está desfasada en -1[a]. Para mantener la simetría, se elimina un número de la codificación, dejando $2^n - 1$ valores distintos. Siendo los límites del dominio cuadrados, se tendrá un enrejado en el que la distancia entre puntos contiguos es $\max(x)\cdot(2^{-15} - 1)$.

Dada la forma en la que se ha hecho la simulación del circuito en este trabajo es difícil poder hacer un benchmarking similar, ya que cada puerta cuántica necesitaría al menos 8GB de memoria para poder representarlas en Matlab. Sin embargo, dado que se trata de un problema sin restricciones, el estado del sistema se puede dividir fácilmente en distintos subespacios de un tamaño computable.

Aplicando esta estrategia para la simulación, se han buscado los extremos de las siguientes funciones con sus respectivos dominios y soluciones:

$$F_1 = 100(x^2 - y)^2 + (1 - x)^2, \qquad |x, y| \leq 2.048, \qquad F_1^{min} = 0, \tag{56}$$

$$F_2 = (x^2 + y^2)^{1/4}(\text{sen}^2(50(x^2 + y^2)^{1/10}) + 1), \qquad |x, y| \leq 100, \qquad F_2^{min} = 0, \tag{57}$$

$$F_3 = \frac{1}{2} - \frac{\text{sen}^2((x^2 + y^2)^{1/2}) - 1/2}{(1 + 10^{-3}(x^2 + y^2))^4}, \qquad |x, y| \leq 100, \qquad F_3^{max} = 1, \tag{58}$$

$$F_4 = (x^2 + y - 11)^2 + (x + y^2 - 7)^2, \qquad |x, y| \leq 10, \qquad F_4^{min} = 0. \tag{59}$$

Para obtener unos resultados comprables a los mostrados en el trabajo se han lanzado el algoritmo 1000 veces por función. Se ha igualado el número de hormigas totales usadas entre los dos algoritmos (40000 hormigas), por lo que el criterio de salida del algoritmo se ha cambiado a tener un número fijo de iteraciones. Así, los resultados obtenidos se muestran en la tabla (Tab.4). Los resultados muestran claramente que el algoritmo propuesto por L. Wang, Q. Niu y M. Fei es mejor en cuanto a la eficiencia y la velocidad para llegar al resultado correcto. Sin embargo, salvo para la primera función en la que el mínimo de la función está en una región muy plana de la superficie y la cuarta en la que hay 4 soluciones distintas, el algoritmo QACO propuesto llega al resultado correcto con suficiente frecuencia.

Tab. 4: Resultados de las pruebas de optimización de las funciones $F_{1-4}$ (Eqs.56-59). Se han repetido las pruebas 1000 veces para cada función. "% opt" muestra el número de veces que se ha obtenido el resultado correcto. "Media" es el valor medio de estas funciones.

| | $F_1$ | | $F_2$ | | $F_3$ | | $F_4$ | |
|---|---|---|---|---|---|---|---|---|
| | % opt | Media | % opt | Media | % opt | Media | % opt | Media |
| Novel QACO [17] | 100.0 | 0 | 100.0 | 0 | 100.0 | 1 | 100.0 | 0 |
| TFG QACO | 0.1 | 0.0125 | 99.1 | 0.001 | 98.4 | 1 | 0.0 | 0.1807 |

[a] Por ejemplo, los números enteros de -3 a 3 están codificados como 111, 101, 100, 000, 001, 011, 010.

# 4. Aplicaciones de QACO

Una aplicación de los algoritmos de hormigas es la resolución aproximada de problemas NP-comnpletos [21, pp. 463-465].

**Definición.** *Los problemas NP-completos (problemas de tiempo polinomial no deterministas) son aquellos que se pueden resolver en un número de pasos polinómico ($n^k$, $k< \infty$) en una máquina de Turing no determinista.*

**Teorema 2** (Equivalencia de los problemas NP-complejos [22])**.** *El conjunto de problemas NP-complejos es equivalente, es decir, cualquier problema puede ser reformulado para resolverse en términos de otro problema NP-complejo a través de una transformación de coste polinómico.*

Suponiendo que la conjetura de que P[a]≠NP fuera cierta , la complejidad del algoritmo para resolver estos problemas sería mayor que una cota polinómica [23]. Dado que existe una amplia discusión alrededor de los problemas P-NP (siendo además uno de los problemas del milenio [24]), este trabajo solo se centrará en plantear uno de ellos dando como cierta la afirmación anterior. Por suerte, se ha demostrado que todos lo problemas NP-complejos, y por lo tanto tambiénlos NP-completos, son equivalentes. Resolver uno de ellos será suficiente para comprobar la eficacia del algoritmo para resolver este tipo de problemas.

Los algoritmos de colonias de hormigas se pueden usar para resolver de forma aproximada este tipo de problemas. Dado que son algoritmos estocásticos, no pueden asegurar que la solución a la que lleguen sea la correcta. Sin embargo, las soluciones que obtienen suelen ser bastante buenas, encontrando al menos una solución localmente óptima [25] en un número de pasos menor que los algoritmos exactos. En relación a los algoritmos que resuelven estos problemas, es conveniente comentar el "No Free Lunch Theorem".

**Teorema 3** (No Free Lunch Theorem [26])**.** *No existe una estrategia de optimización global que reduzca el coste de llegar a la solución sobre el conjunto completo de problemas. Solo se podrá conseguir un algoritmo optimizado frente a otro más general si este se limita a resolver un tipo de problema concreto.*

Tal y como nos dice este teorema, no existe una estrategia de optimización general. Esto hace que haya que añadir mejoras o modificaciones según el tipo de problema que se aborde para llegar a los resultados deseados. Los algoritmos de colonias de hormigas no se salvan de este problema. En la tabla 2.8 de [16] se tiene un resumen de algunas de las estrategias de optimización que se aplican a los algoritmos ACO para resolver distintos tipos de problemas NP-completos. En concreto, esto se puede ver al analizar las formas en la que se optimiza un mismo algoritmo Min-Max Ant System[b] (MMAS) para el problema del viajante y para el problema de asignación cuadrática, las cuales no se pueden aplicar a otros problemas. El resto del capítulo tratará de la aplicación de QACO para resolver el problema de asignación cuadrática, o más concretamente, dos variaciones del mismo.

[a] Los problemas P son los problemas de tiempo polinomial deterministas.

[b] MMAS es una versión de un algoritmo de sistema de hormigas (AS) en el que se busca un camino mejor alrededor de la mejor hormiga de cada iteración.

## 4.1 Unconstrained Binary Quadratic Programming (UBQP)

Para probar la implementación propuesta se ha usado como referencia el problema de UBQP [27], otras veces llamado como QUBO (Quadratic Unconstrained Binary Optimization). El problema consiste en maximizar la función dada por

$$f(\mathbf{X}, \mathcal{M}) = \sum_{i=1}^{n} \sum_{j=1}^{i} X_i \mathcal{M}_{ij} X_j, \text{ con } \mathcal{M}_{ij} \in \mathbb{R}, \tag{60}$$

o escrito de forma matricial

$$f(\mathbf{X}, \mathcal{M}) = \mathbf{X}^t \mathcal{M} \mathbf{X}, \tag{61}$$

con $\mathbf{X}$ un vector columna de 1's y 0's y $\mathcal{M}$ una matriz simétrica o triangular. Ya que las soluciones a este problema pueden tener cualquier número de 1's, para que el problema sea no trivial la matriz $\mathcal{M}$ tiene que tener elementos tanto positivos como negativos.

Es fácil ver que el conjunto de soluciones al problema tiene exactamente un tamaño $2^n$, que corresponde justamente al mayor número entero sin signo representable con un número binario de longitud $n$. Cualquier algoritmo que encuentre la solución exacta a este problema tendrá una complejidad de $O(2^n)$. Debido a este coste exponencial, la aplicación de un algoritmo heurístico puede ser conveniente en problemas con un tamaño demasiado grande como para aplicar estrategias exactas. Sin embargo, y dado que no se dispone del equipo ni del tiempo necesario como para comprobar estos algoritmos de hormigas sobre problemas tan grandes, se ha aplicado el algoritmo propuesto de QACO sobre problemas pequeños.

En concreto, se muestran en la figura (Fig.10) los resultados de resolver el problema propuesto en el ejemplo 5, con un total de 16 soluciones posibles.

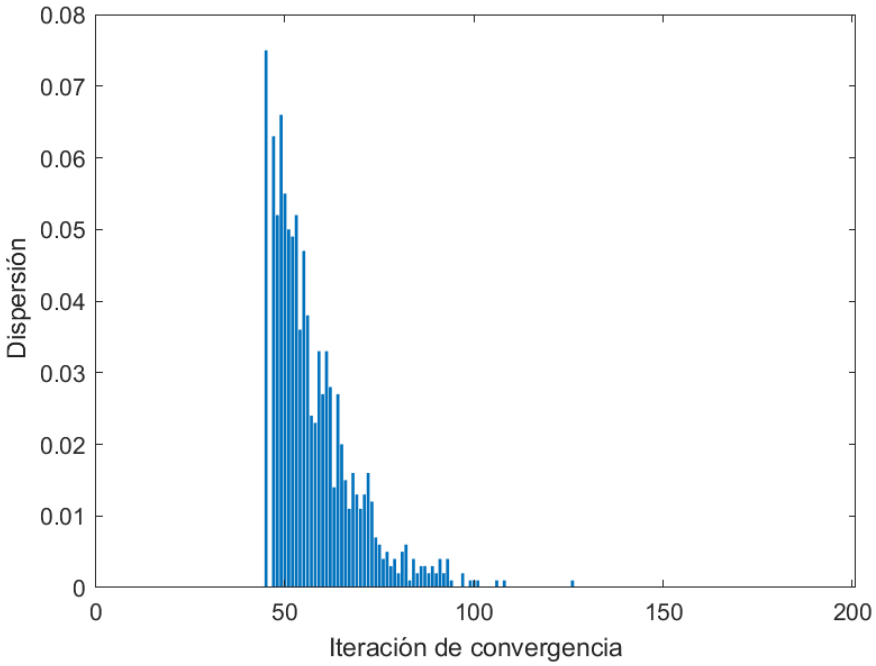
**Ejemplo 5:** UBQP en agricultura

Normalmente, las plantaciones a gran escala son monocultivo, es decir, solo tienen un tipo de planta. Sin embargo, algunas combinaciones de especies distintas permiten, además de un cultivo variado, la mejora de las condiciones de ambas especies. De la misma manera que hay combinaciones beneficiosas, algunas asociaciones pueden perjudicar el rendimiento del cultivo.

Supongamos que tenemos una huerta en la queremos plantar patatas, vainas, puerros o calabazas de tal manera que la asociación de plantas dé un rendimiento máximo. La experiencia nos dice que el cultivo conjunto de patata-vaina, patata-puerro y calabaza-vaina dan buenos resultados, pero la asociación de patata-calabaza, vaina-puerro y puerro-calabaza reducen la cosecha. Nuestros cálculos nos dan una relación de rendimiento dada por la siguiente matriz de tamaño 4x4 en el orden {patata, vaina, puerro, calabaza},

$$\mathcal{M}_{\text{rendimiento}} = \begin{pmatrix} 2 & 0.5 & 1 & -1.5 \\ & 1 & -1 & 1 \\ & & 1 & -3 \\ & & & 2.5 \end{pmatrix}.$$

De esta manera, si plantáramos todas las verduras a la vez (**X**=1111), aplicando la ecuación (Eq.61) obtendríamos un rendimiento de 3.5. Pero dado que existen muchas posibilidades (15 en este caso), usamos el algoritmo QACO para calcular la mejor combinación. El resultado que obtenemos es que si se plantan patatas, vainas y calabazas (**X**=1101) se obtendrá un rendimiento máximo de 5.5.




Soluciones obtenidas:

- 5.5 (1101), 976 veces
- 4.5 (1110), 23 veces
- 4.5 (0101), 1 vez

Fig. 10: Dispersión del número de iteraciones hasta la convergencia y soluciones obtenidas para el problema del ejemplo 5. Se ha repetido el problema 1000 veces, con converCondition=45. Se han obtenido un 2.4% de soluciones incorrectas.

## 4.2 Constrained Binary Quadratic Programming (CBQP)

El problema CBQP tiene exactamente la misma formulación que la usada en UBQP, salvo por que en este caso no todos los posibles valores del vector $\mathbf{X}$ son válidos. Tal y como se ha definido el problema CBQP en este trabajo, las soluciones pueden estar restringidas a cualquier conjunto de vectores de la base canónica.

En la referencia de CBQP consultada [27], se demuestra que los problemas con restricciones pueden ser convertidos a problemas sin restricciones sin más que añadiendo una penalización a los resultados no válidos obtenidos. Frente a esta solución, la propuesta de este trabajo ha sido tratar a los problemas restringidos de igual manera que los problemas sin restricciones.

Para ello, es necesario modificar la matriz GenS siguiendo las reglas descritas en la sección 3.2.2 para que se adecúe al problema. La forma en la que se construye esta matriz permite que los problemas que se resuelvan tengan un número arbitrario de soluciones posibles.

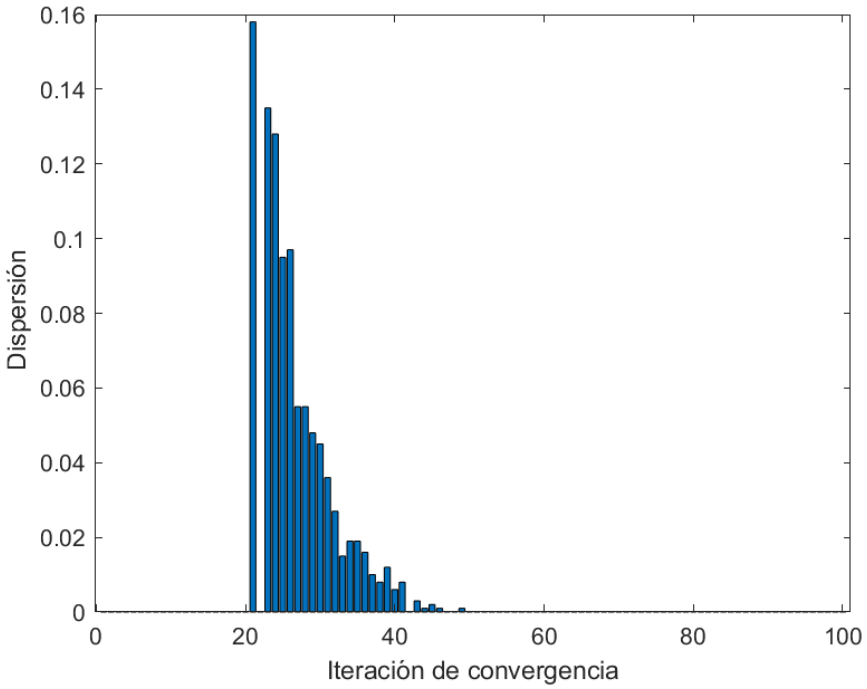
De esta manera, se han resuelto los dos ejemplos planteados en esta sección (Ejs.6-7) usando QACO. Los resultados de cada problema se muestran en las figuras (Fig.11) y (Fig.12) respectivamente. Aunque la aplicación de QACO sobre problemas tan pequeños no aporta ninguna ventaja, sí que sirven para mostrar que el algoritmo resuelve problemas de este tipo. Siguiendo los pasos descritos, se pueden resolver problemas CBQP de tamaño superior sin modificar el algoritmo.

**Ejemplo 6:** CBQP en agricultura

Se plantea el mismo problema que en el ejemplo anterior (Ej.5). Sin embargo, tenemos ahora un problema añadido. La tienda de suministros agrícolas tiene una oferta 2x1 en la compra de dos tipos distintos de semillas. Queremos aprovechar esa oferta maximizando al mismo tiempo el rendimiento del cultivo.

Dado que el problema ahora tiene restricciones, el espacio de las soluciones pasa de tener 16 elementos, a solamente 7. Esto podría hacernos pensar que podríamos obtener una eficiencia mayor en el algoritmo si usamos una representación de 3 qubits para codificar el problema. Sin embargo, la mejor opción es mantener el tamaño de la matriz, ya que permite una representación directa de las relaciones entre los distintos elementos de la base (las distintas configuraciones de verduras que queremos plantar). En este caso, la matriz GenS que se usa es la misma que la descrita en el ejemplo 3.

Lanzando nuevamente el algoritmo QACO se obtiene que la solución a este nuevo problema es hacer un cultivo de vainas y calabazas (**X**=0101), con un rendimiento de 4.5.




Soluciones obtenidas:

- 4.5 (0101), 994 veces
- 4 (1110), 6 veces

Fig. 11: Dispersión del número de iteraciones hasta la convergencia y soluciones obtenidas para el problema del ejemplo 6. Se ha repetido el problema 1000 veces, con converCondition=21. Se han obtenido un 0.6% de soluciones incorrectas.

**Ejemplo 7:** CBQP en el problema del barquero

Se plantea una modificación del clásico problema del barquero. Queremos llevar a la orilla opuesta del río una lechuga, una cabra o un lobo. Por cada uno de ellos nos darán una recompensa de 100, 250 y 200 monedas respectivamente. Si los llevamos juntos, la cabra se comerá la mitad de la lechuga y el lobo la mitad de la cabra. La barca es suficientemente grande como para llevar 2 mercancías, pero solo podemos hacer un viaje. Esto nos deja un espacio de 6 posibilidades {001,010,100,011,101,110}.

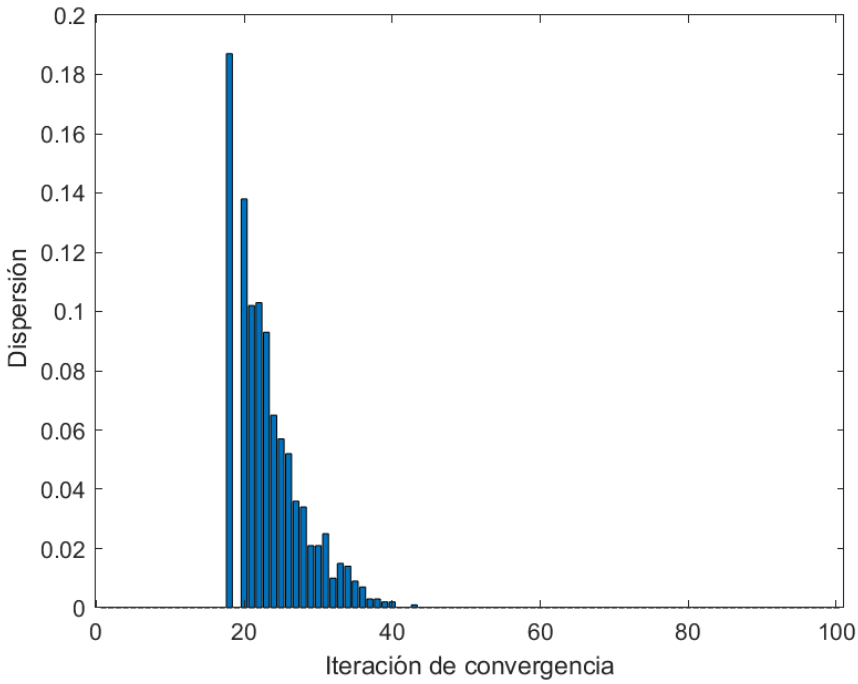
La matriz GenS que usaremos para este problema viene dada por

$$\text{GenS} = \begin{pmatrix} 0 & \sqrt{2}/3 & \sqrt{2}/3 & 1/3 & \sqrt{2}/3 & 1/3 & 1/3 \\ & 1 & & & & & \\ & & 1 & & & & \\ & & & 1 & & & \\ & & & & 1 & & \\ & & & & & 1 & \\ & & & & & & 1 \\ & 1/3 & 1/3 & \sqrt{2}/3 & 1/3 & \sqrt{2}/3 & \sqrt{2}/3 & 0 \end{pmatrix},$$

y la matriz del problema será

$$\mathcal{M}_{\text{monedas}} = \begin{pmatrix} 100 & -50 & \\ & 250 & -125 \\ & & 200 \end{pmatrix}.$$

Aplicando QACO a este problema, se obtiene que obtendremos un rendimiento máximo del viaje cargando el lobo y la cabra (**X**=011), obteniendo 325 monedas.




Soluciones obtenidas:

- 325 (011), 974 veces
- 300 (110), 13 veces
- 300 (101), 8 veces

Fig. 12: Dispersión del número de iteraciones hasta la convergencia y soluciones obtenidas para el problema del ejemplo 7. Se ha repetido el problema 1000 veces, con converCondition=18. Se han obtenido un 2.1% de soluciones incorrectas.

# 5. Conclusiones y trabajo futuro

El algoritmo QACO desarrollado en este trabajo permite una implementación en un computador real. Para ello, basándose en una versión anterior del algoritmo, se han planteado una serie de mejoras: **(1)** La aplicación de las feromonas se ha hecho teniendo en cuenta un giro sobre la esfera de Bloch. **(2)** La exploración de nuevos caminos ahora se hace a través de una serie de puertas controladas, lo que permite que el circuito cuántico se pueda construir usando solamente puertas cuánticas unitarias. **(3)** A través de un filtro de soluciones, QACO es capaz resolver problemas con restricciones de forma eficiente, en los que el número de soluciones posibles no es el mismo que el número total de estados posibles.

Se ha probado el algoritmo sobre problemas UBQP, CBQP y de optimización de 4 funciones de dos variables. Los resultados muestran que el algoritmo es capaz de llegar a la solución en un número de pasos proporcional a $nComb^{0.66\pm0.04}$, donde $nComb$ es el tamaño del espacio de las soluciones.

De los resultados obtenidos de comparar QACO con ACO, se tiene que el algoritmo desarrollado en este trabajo es razonablemente equivalente al algoritmo clásico en el que se basa. Esto refuerza el hecho de que QACO es útil para resolver problemas de tamaño considerable. En los problemas en los que por su tamaño el tiempo que se tarda en generar cada hormiga en ACO es demasiado grande, QACO puede ofrecer una ventaja significativa, al necesitar cada hormiga un tiempo aproximadamente constante independientemente del tamaño del problema. El algoritmo desarrollado no logra mejorar los resultados obtenidos por el algoritmo QACO en el que se basa. Sin embargo, se recuerda que el este último no es implementable en un computador cuántico, minetras que el aquí propuesto sí lo es.

Los resultados obtenidos de la velocidad de convergencia para QACO muestran una relación proporcional cercana a la raíz cuadrada del tamaño del espacio de las soluciones al problema. El algoritmo de búsqueda de Grover necesita $\sqrt{n}$ iteraciones para llegar al resultado (Sec.2.3.1). Siendo ambos algoritmos de búsqueda, parece que el algoritmo desarrollado aprovecha de manera adecuada las ventajas que ofrece la computación cuántica. Comparando ambos algoritmos se puede encontrar una relación entre ellos. En el algoritmo de Grover se tienen 2 pasos claros en cada iteración, una consulta con el oráculo para obtener información a cerca del resultado que se busca y una actualización del estado cuántico que haga más probable obtener el resultado correcto en la siguiente iteración. En el algoritmo de QACO propuesto se pueden identificar estos dos pasos de forma sutil. La consulta con el oráculo correspondería a los pasos entre la medición del estado cuántico y la selección de la hormiga de la iteración (pasos 5-10 Alg.7). Esta consulta extrae información acerca del resultado al problema, y la usa para aumentar la probabilidad de medir el estado solución. Esto se haría en el paso de aplicación de las feromonas a los qubits de las hormigas (paso 3 Alg.7). Queda abierta la posibilidad de estudiar más en profundidad esta relación y comprobar su eficiencia frente a otros algoritmos de optimización global basados en el algoritmo de Grover [28].

Además de esto, el algoritmo propuesto todavía tiene opciones de mejora. En el artículo en el que se planteó QACO proponen un parámetro de exploración decreciente con el número de iteraciones. No se ha comprobado si esta estrategia sería capaz de mejorar la velocidad de convergencia del algoritmo.

Para mejorar la estrategia de exploración, podría ser conveniente combinar los dos tipos de estrategias planteadas. Por la forma de la exploración con puertas Fredkin, se podría usar hacia

el final del algoritmo para hacer una búsqueda local del extremo de la función objetivo.

Una estrategia de la mayoría de los algoritmos de hormigas consiste en lanzar varias hormigas usando una misma configuración de feromonas. Entre todas las hormigas de la misma generación se escoge la mejor, usando esta para actualizar el rastro de las feromonas. Unos cálculos preliminares indican que usar esta estrategia en QACO podría ser equivalente a la estrategia actual en cuanto al número de hormigas totales (referido al lanzamiento de una iteración del circuito cuántico), con la ventaja de que se reduciría el número de veces en el que hay que actualizar los ángulos de giro de las feromonas.

La realización de este trabajo ha servido de introducción a la computación cuántica y, sobre todo, a los algoritmos cuánticos. Aunque se hayan dejado algunos aspectos importantes sin desarrollar en profundidad, como la implementación real, el control de errores y el ruido, este trabajo abre las puertas a seguir estudiando esta rama de conocimiento.

Una de las peculiaridades de la computación cuántica es la conexión directa entre la física y la ingeniería electrónica, lo cual encaja a la perfección con el perfil de la doble titulación. A la hora de abordar este trabajo se ha necesitado un conocimiento exhaustivo previo de conceptos de álgebra lineal, y sobre todo de física y mecánica cuántica. Por otra parte, se han aplicado competencias transversales adquiridas en todas las asignaturas en las que se han trabajado métodos matemáticos y técnicas computacionales, lo que ha permitido implementar de forma fácil y eficiente los algoritmos descritos en el trabajo. Se confirma además la capacidad de adaptación a un lenguaje de programación nuevo con facilidad, en este caso Matlab, añadiéndolo así a la lista de los lenguajes usados a lo largo de la doble titulación.

# Bibliografía

[1] P. W. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," in SIAM J. Comput., vol. 26, no. 5, 1997, pp. 1484-1509.

[2] L. K. Grover, "A fast quantum mechanical algorithm for database search," in Proc. STOC, 1996, pp. 212-219.

[3] S. Jordan. "Quantum Algorithm Zoo." https://quantumalgorithmzoo.org/ (accessed 4/4/2020).

[4] J. J. Sakurai, and J. Napolitano, "Modern Quantum Mechanics," 2nd ed., Pearson Educ., 1994.

[5] M. A. Nielsen, and I. L. Chuang, "Quantum Computation and Quantum Information," 10th anniversary ed., Cambridge Univ. Press, 2010.

[6] M. Heuck, K. Jacobs, and D. R. Englund, "Controlled-phase Gate using Dynamically Coupled Cavities and Optical Nonlinearities," in Phys. Rev. Lett., vol. 124, no. 16, 2020, p. 160501.

[7] N Friis *et al.*, "Observation of Entangled States of a Fully Controlled 20-Qubit System," in Phys. Rev. X, vol. 8, no. 2, 2018, p. 021012.

[8] F. Arute *et al.*, "Quantum supremacy using a programmable superconducting processor," in Nature, vol. 574, 2019, pp. 505-511

[9] E. Pednault *et al.*, "Leveraging Secondary Storage to Simulate Deep 54-qubit Sycamore Circuits", 2019, arXiv:1910.09534.

[10] S. Morita, and H. Nishimori, "Mathematical Foundation of Quantum Annealing," in J. Math. Phys., vol. 49, no. 12, 2008, pp. 125-210.

[11] C. C. MacGeoch *et al.*, "Practical Annealing-Based Quantum Computing," in IEEE Comput. Mag., vol. 52, 2019, pp. 38-46.

[12] T. H. Cormen *et al.*, "Introduction To Algorithms," 2rd ed., MIT Press, 2001.

[13] T. Elgamal, "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," in IEEE Trans. Inf. Theory, vol. 31, no. 4, 1985, pp. 469-472.

[14] W. Diffie, and M. E. Hellman, "New Directions in Cryptography," in IEEE Trans. Inf. Theory, vol. 22, no. 6, 1976, pp. 644-654.

[15] Ryan O'Donnell. (2018). Lecture videos for Quantum Computation and Information, Shor's Factoring Algorithm: Lecture 16. Accessed: 15/4/2020. [Online video]. Available: https://www.youtube.com/watch?v=P1gW_SPxjlU

[16] E. Bonabeau, M. Dorigo, and G. Theraulaz, "Swarn Intelligence: From Natural to Artificial Systems", Oxford Univ. Press, 1999.

[17] L. Wang, Q. Niu, and M. Fei "A Novel Quantum Ant Colony Optimization Algorithm," in Life Syst. Model. Simul. Int. Conf., 2007, pp. 277-286.

[18] C. Blum, and M. Dorigo, "The Hyper-Cube Framework for Ant Colony Optimization," in IEEE Trans. Syst. Man Cybern., part. B, vol. 34, no. 2, 2004, pp. 1161-1172.

[19] P. Ronah, B. Woods, and E. Iranmanesh, “Solving Constrained Quadratic Binary Problems via Quantum Adiabatic Evolution,” in Quantum Inf. Comput., vol. 16, no. 11-12, 2018, pp. 1029-1046.

[20] E. M. Zakharova, and I. K. Minashina, “Review of Multidimensional Optimization Methods,” in Informatsionnye Protsessy, vol. 14, no. 3, 2014, pp. 256-274 (Transl.: in J. Commun. Technol. Electron., vol. 60, no. 6, 2015, pp. 625-636).

[21] J. Kleinberg, and E. Tardos “Algorithm Design,” 2nd ed., Addison-Wesley, 2006.

[22] D. E. Knuth, “Postscript About NP-Hard Problems,” in SIGACT News, Apr. 1974, pp. 15-16.

[23] S. Cook, “The P Versus NP Problem,” partialy published in J. ACM, vol. 50, No. 1, 2003, pp. 27–29. [Online]. Avalilable: www.claymath.org/sites/default/files/pvsnp.pdf

[24] A. M. Jaffe, “The Millennium Grand Challenge in Mathematics,” in Notices Am. Math. Soc., vol. 53, no. 6, 2006, pp. 652-660.

[25] R. E. Burkard *et al.*, “The Quadratic Assignement Problem,” in Handbook of Combinatorial Optim., vol. 3, Springer, 1998, pp. 241-337.

[26] Y.-C. Ho, and D.L. Pepyne, “Simple Explanation of the No Free Lunch Theorem of Optimization,” in Cybern. Syst. Anal., vol. 38, no. 2, 2002, pp. 292-298.

[27] G. Kochenberger *et al.*, “The Unconstrained Binary Quadratic Programming Problem: A Survey,”in J. Combinatorial Optim., vol. 28, no. 1, 2014, pp. 58-81.

[28] W. P. Baritompa, D. W. Bulger, and G. R. Wood, “Grover’s Quantum Algorithm Applied to Global Optimization,” in SIAM J. Optim., vol. 15, no. 4, 2005, pp. 1170–1184.

# A. Diagramas y tablas

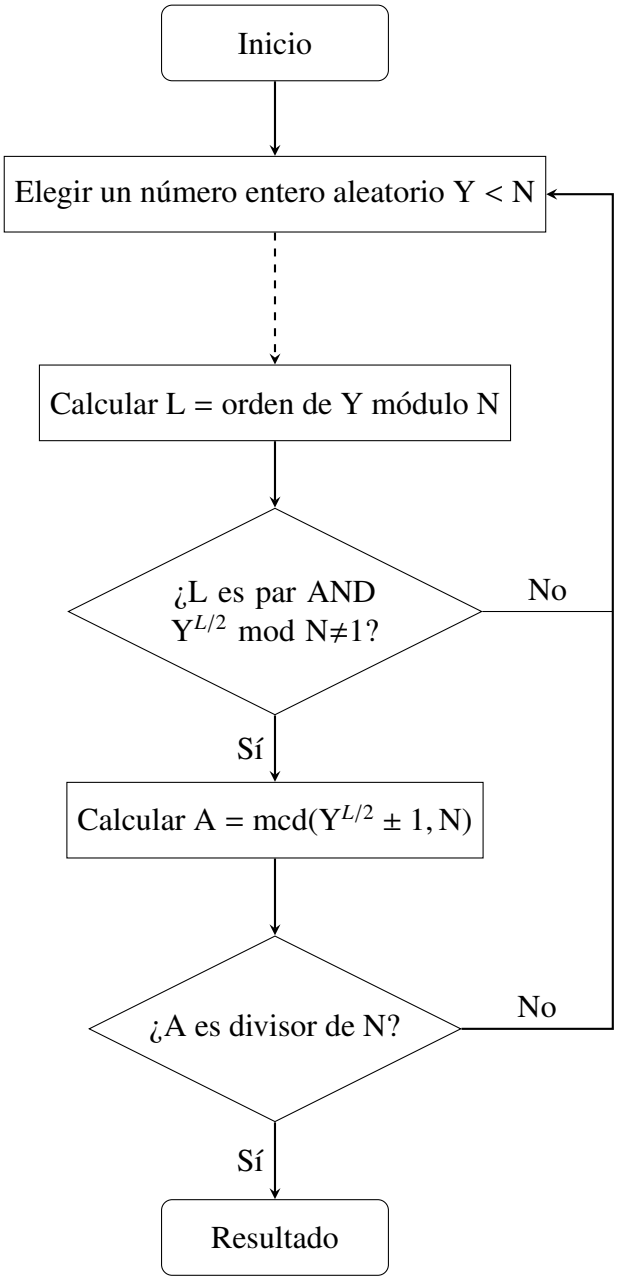



Fig. A.1: Diagrama de flujo simplificado del algoritmo de factorización de Shor. La línea a rayas indica la aplicación del algoritmo cuántico de búsqueda de orden.

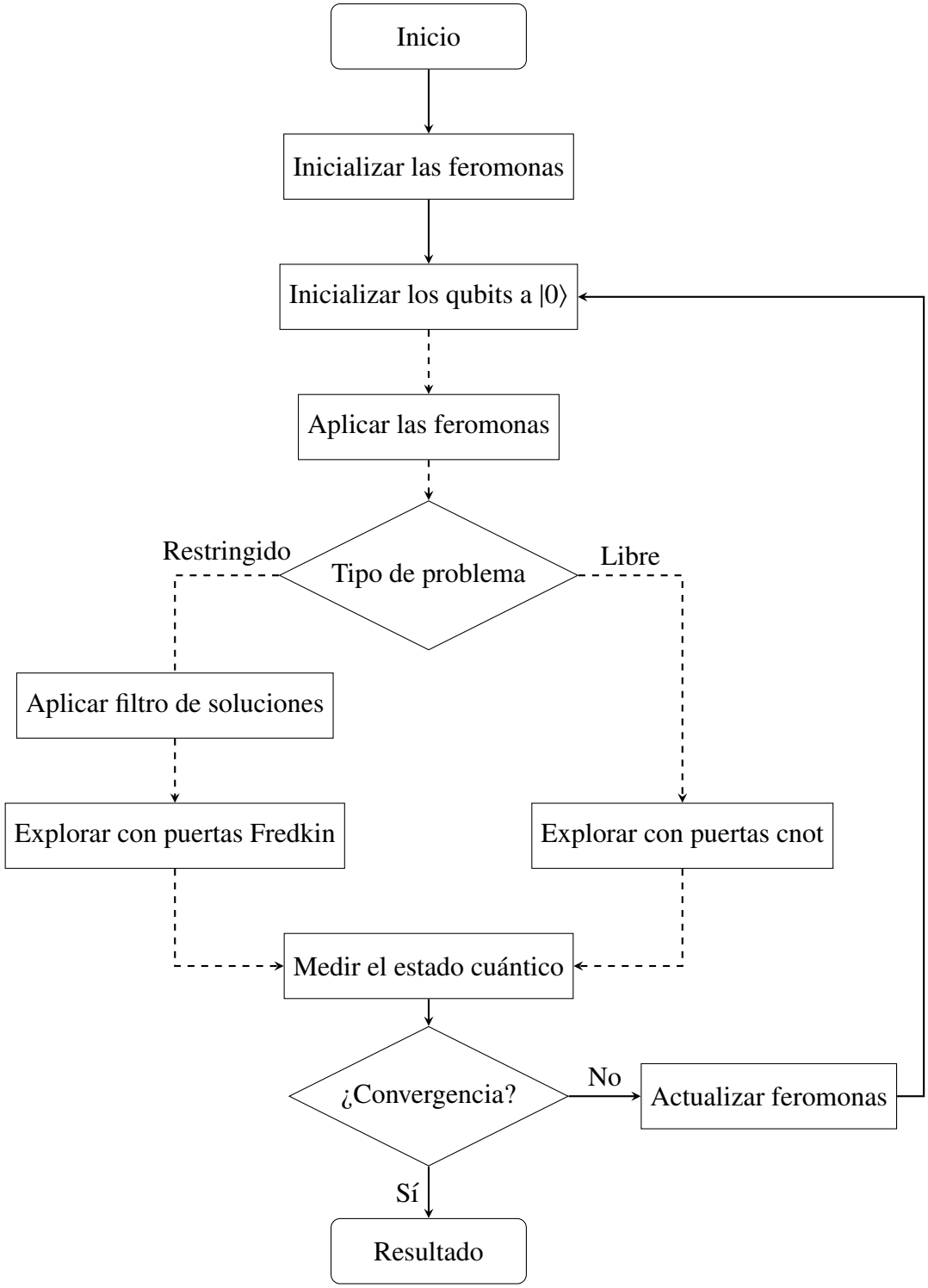



Fig. A.2: Diagrama de flujo de la propuesta inicial de QACO. La línea a rayas indica que la información entre pasos se encuentra en un estado cuántico.

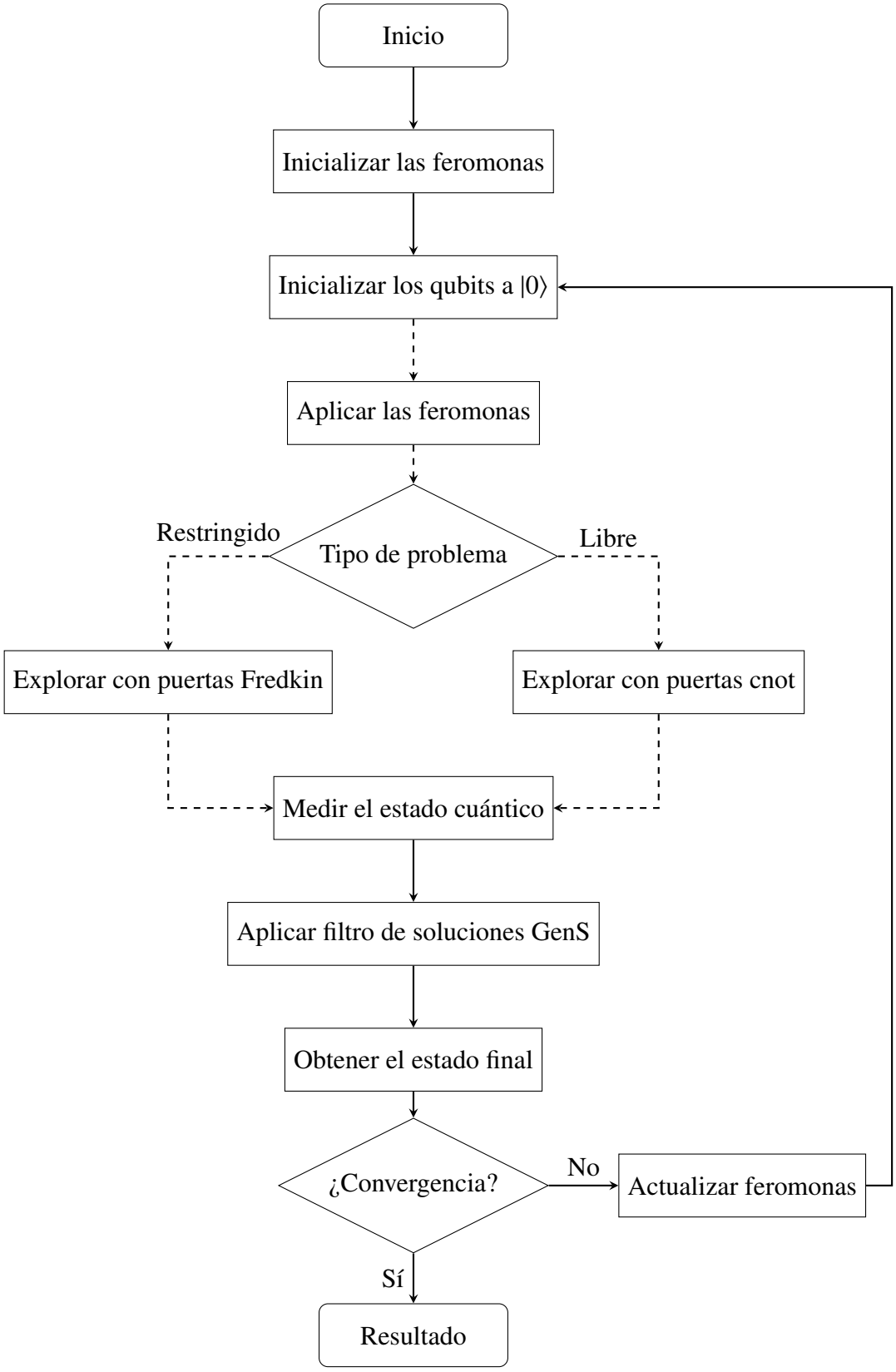



Fig. A.3: Diagrama de flujo de la versión final del algoritmo QACO propuesto. La línea a rayas indica que la información entre pasos se encuentra en un estado cuántico.

Tab. A.1: Valores de los parámetros de QACO que optimizan el número medio de iteraciones para encontrar un resultado con un error de menos del 1.5% sobre 200 problemas distintos. *Comb* es el tamaño del espacio de las soluciones, *IterM* el número medio de iteraciones hasta la convergencia, *CC* es el parámetro converCondition, $\beta_e$ el parámetro de exploración y $00T$, $00F$, $01T$ y $01F$ son los ángulos de giro siguiendo el mismo orden que la tabla (Tab.2). Cada fila muestra los resultados de cada una de las distintas pruebas para un mismo tamaño de problema $n, m$.

| n | m | Comb | IterM | CC | $\beta_e$ | 00T | 00F | 01T | 01F |
|---|---|---|---|---|---|---|---|---|---|
| 4 | 1 | 4 | 15.4 | 12 | 0.953009 | 0.038639 | 0.035875 | 0.049706 | 0.037544 |
| | | | 16.165 | 12 | 0.689292 | 0.098464 | 0.081189 | 0.037504 | 0.060391 |
| | | | 16.125 | 12 | 0.688237 | 0.056588 | 0.071359 | 0.07472 | 0.060201 |
| | 2 | 6 | 21.125 | 15 | 0.458468 | 0.088851 | 0.089771 | 0.020437 | 0.07773 |
| | | | 22.145 | 17 | 0.867914 | 0.091033 | 0.071923 | 0.043544 | 0.000491 |
| | | | 24.01 | 17 | 0.361777 | 0.087734 | 0.08948 | 0.018788 | 0.077993 |
| | 4 | 16 | 44.1 | 33 | 0.999898 | 0.005287 | 0.072619 | 0.052704 | 0.099261 |
| | | | 55.42 | 40 | 0.40846 | 0.007356 | 0.066019 | 0.040187 | 0.09063 |
| | | | 50.78 | 36 | 0.339328 | 0.013507 | 0.066593 | 0.045534 | 0.093298 |
| 5 | 1 | 5 | 18.41 | 14 | 0.965143 | 0.038032 | 0.038773 | 0.051775 | 0.040162 |
| | | | 17.55 | 13 | 0.951873 | 0.038768 | 0.035804 | 0.051377 | 0.038946 |
| | | | 16.095 | 12 | 0.72252 | 0.086325 | 0 | 0.064524 | 0.00904 |
| | 2 | 10 | 38.46 | 29 | 0.929688 | 0.078906 | 0.050781 | 0.097656 | 0.085156 |
| | | | 37.41 | 26 | 0.034456 | 0.012473 | 0.093842 | 0.094785 | 0.067533 |
| | | | 37.91 | 29 | 0.869506 | 0.00799 | 0.076792 | 0.065586 | 0.088161 |
| | 5 | 32 | 89.065 | 65 | 0.006319 | 0.075952 | 0.052835 | 0.011011 | 0.099941 |
| | | | 84.96 | 59 | 0.119121 | 0.08213 | 0.028222 | 0.003218 | 0.060292 |
| | | | 90.325 | 66 | 0.136788 | 0.050926 | 0.04533 | 0.023518 | 0.079709 |
| 6 | 1 | 6 | 22.28 | 17 | 0.949397 | 0.078846 | 0.05133 | 0.097475 | 0.081298 |
| | | | 21.265 | 16 | 0.841697 | 0.095242 | 0.022297 | 0.042618 | 0.008732 |
| | | | 22.86 | 17 | 0.582784 | 0.052969 | 0.071334 | 0.078576 | 0.068152 |
| | 2 | 15 | 54.475 | 39 | 0.141882 | 0.036098 | 0.053164 | 0.017303 | 0.060645 |
| | | | 55.94 | 43 | 0.67635 | 0.059263 | 0 | 0.084508 | 0.027658 |
| | | | 50.395 | 36 | 0.357873 | 0.003086 | 0.075364 | 0.067072 | 0.092852 |
| | 3 | 20 | 75.795 | 54 | 0.110509 | 0.031398 | 0.040715 | 0.016573 | 0.062948 |
| | | | 70.13 | 52 | 0.837704 | 0.022173 | 0.084125 | 0.065932 | 0.096963 |
| | | | 66.37 | 48 | 0.376463 | 0.008573 | 0.069437 | 0.040114 | 0.090536 |
| | 6 | 64 | 161.175 | 120 | 0.408018 | 0.072831 | 0.075271 | 0.000587 | 0.1 |
| | | | 166.385 | 129 | 0.287918 | 0.086737 | 0.030619 | 0.005601 | 0.067897 |
| | | | 143.785 | 108 | 0.087707 | 0.059202 | 0.04371 | 0.011008 | 0.078932 |
| 7 | 1 | 7 | 25.45 | 19 | 0.117016 | 0.091626 | 0.034342 | 0.032884 | 0.010498 |
| | | | 26.405 | 20 | 0.584878 | 0.056341 | 0.072995 | 0.077314 | 0.067019 |
| | | | 23.16 | 17 | 0.934189 | 0.074743 | 0.049347 | 0.088834 | 0.077299 |
| | 2 | 21 | 77.95 | 58 | 0.482929 | 0.084914 | 0.059125 | 0.065658 | 0.0318 |
| | | | 83.255 | 64 | 0.049252 | 0.072939 | 0.095816 | 0.038314 | 0.028845 |
| | | | 72.735 | 53 | 0.679251 | 0.048593 | 0.085696 | 0.096911 | 0.052201 |
| | 3 | 35 | 116.46 | 84 | 0.297709 | 0.059653 | 0.035636 | 0.03253 | 0.082611 |
| | | | 124.695 | 95 | 0.188057 | 0.06571 | 0.048471 | 0.029981 | 0.080064 |
| | | | 127.06 | 100 | 0.976380 | 0.084200 | 0.1 | 0 | 0.001187 |
| | 7 | 128 | 266.085 | 200 | 0.219669 | 0.06254 | 0.035393 | 0.012578 | 0.054803 |
| | | | 265.88 | 208 | 0.267739 | 0.043565 | 0.028528 | 0.002585 | 0.051883 |
| | | | 251.405 | 198 | 0.207422 | 0.066647 | 0.019659 | 0.046286 | 0.047514 |

Tab. A.2: Resultados de las pruebas de comparación de ACO y QACO. Se han realizado 1000 pruebas por tamaño de problema $(n, m)$, siendo *Comb* el tamaño del espacio de las soluciones. La condición de convergencia se ha calculado usando la ecuación (Eq.53). Las columnas bajo "Mejor resultado?" muestran el número de veces en las que se ha obtenido el mejor resultado con cada algoritmo. Las columnas bajo "Solución?" indican el número de veces que se ha llegado a la solución correcta al problema, calculada con un algoritmo exacto. "Iteraciones medias" muestra el número de iteraciones que se han necesitado con cada algoritmo para llegar a la condición de convergencia. Las celdas coloreadas muestran el mejor resultado para cada tamaño de problema.

| | | | Mejor resultado? | | | Solución? | | Iteraciones medias | |
|---|---|---|---|---|---|---|---|---|---|
| n | m | Comb | ACO | QACO | Iguales | ACO | QACO | ACO | QACO |
| 4 | 1 | 4 | 28 | 22 | 950 | 976 | 970 | 14.645 | 14.188 |
| 4 | 2 | 6 | 31 | 18 | 951 | 981 | 968 | 23.280 | 23.432 |
| 4 | 4 | 16 | 4 | 0 | 996 | 1000 | 996 | 51.250 | 56.262 |
| 5 | 1 | 5 | 25 | 16 | 959 | 984 | 975 | 19.511 | 19.393 |
| 5 | 2 | 10 | 14 | 21 | 965 | 979 | 986 | 39.507 | 38.855 |
| 5 | 5 | 32 | 1 | 0 | 999 | 1000 | 999 | 85.244 | 93.847 |
| 6 | 1 | 6 | 16 | 14 | 970 | 986 | 984 | 23.647 | 23.073 |
| 6 | 2 | 15 | 32 | 22 | 946 | 977 | 967 | 55.560 | 55.441 |
| 6 | 3 | 20 | 27 | 35 | 938 | 965 | 973 | 71.553 | 70.485 |
| 6 | 6 | 64 | 0 | 0 | 1000 | 1000 | 1000 | 139.243 | 152.893 |
| 7 | 1 | 7 | 20 | 23 | 957 | 977 | 980 | 27.455 | 27.225 |
| 7 | 2 | 21 | 39 | 37 | 924 | 962 | 960 | 73.899 | 73.560 |
| 7 | 3 | 35 | 40 | 50 | 910 | 948 | 958 | 112.150 | 113.098 |
| 7 | 7 | 128 | 0 | 0 | 1000 | 1000 | 1000 | 222.127 | 241.199 |
| 8 | 1 | 8 | 17 | 34 | 949 | 966 | 983 | 31.411 | 30.959 |
| 8 | 2 | 28 | 34 | 43 | 923 | 955 | 964 | 94.005 | 93.356 |
| 8 | 3 | 56 | 74 | 77 | 849 | 921 | 923 | 161.903 | 162.184 |
| 8 | 4 | 70 | 88 | 83 | 829 | 910 | 905 | 192.136 | 190.147 |
| 8 | 8 | 256 | 0 | 0 | 1000 | 1000 | 1000 | 353.738 | 380.114 |