Universidad de Valladolid

TRABAJO DE FIN DE GRADO

Grado en Matemáticas

MEJORA DE LAS MODELIZACIONES QUBO DEL PROBLEMA DEL VIAJANTE Y ENRUTAMIENTO DE VEHÍCULOS PARA SU RESOLUCIÓN MEDIANTE *QUANTUM ANNEALING*.

Autor:
Saúl González Bermejo

Tutor (UVa):
Iván Cabria Álvaro

Tutor (LDIG):
Parfait Atchade

24-09-2021

Índice general

Índice de figuras

Resumen

El objetivo de este TFG consiste en construir una función con ciertas propiedades, de manera que mediante la herramienta denominada *Quantum Annealing* (atemperamiento cuántico) se resuelva el siguiente problema propuesto por la empresa Lighthouse Disruptive Innovation Group LLC. Supongamos que tenemos un almacén donde están localizados un número N de pedidos. Contamos con Q robots que van a ser los encargados de coger los pedidos y llevarlos al punto desde donde van a ser distribuidos. En el artículo *qRobot: A Quantum computing approach in mobile robot order picking and batching problem solver optimization* [2], desarrollado entre otros por Parfait Atchadé (empresa Lighthouse e investigador de la universidad Ramon Llull) y Guillermo Alonso (UVa), se dan todos los detalles acerca del problema y su resolución cuando se busca minimizar la distancia total que recorren los robots.

El propósito de este TFG será resolver el problema consistente en minimizar el tiempo que tardan los robots en recoger todos los pedidos o, lo que es equivalente, minimizar el máximo de las distancias que recorre cada robot. Aparte de la modelización y del código (en Python) necesarios para resolver el problema, lo extenso del trabajo será un desarrollo sobre en qué consiste el algoritmo *Quantum Annealing*, centrándonos en exponer las matemáticas que ayudan a entender cómo realizar buenas modelizaciones.

Palabras claves: Computación cuántica, *Quantum Annealing*, Optimización combinatoria, *Quadratic Unconstruined Binary Optimization*, *Travel Salesman Problem.*

Capítulo 1

Introducción

Uno de los grandes y más frecuentes problemas de las matemáticas consiste en encontrar los puntos donde se alcanzan los valores máximos y mínimos de una función, con sus respectivos valores. Para funciones que cumplen ciertas propiedades, el cálculo diferencial nos proporciona técnicas que permiten calcular estos máximos y mínimos. Desgraciadamente, muchas de las funciones que aparecen al tratar de modelizar problemas no satisfacen las hipótesis de los teoremas del cálculo diferencial o poseen restricciones adicionales que anulan su utilidad. Uno de los mayores logros del siglo XX en esta rama de las matemáticas fue el desarrollo del algoritmo símplex por George Dantzig. Durante muchos años fue el algoritmo más utilizado a nivel mundial. Éste permite encontrar el óptimo global de una función lineal con restricciones lineales. Si tratamos de generalizar las condiciones bajo las cuales funciona el algoritmo símplex, pronto nos quedamos sin técnicas que resuelvan estos problemas. Existen herramientas de cálculo numérico para tratar de aproximar los máximos y mínimos de funciones bajo ciertas condiciones de regularidad, pero no son suficientes para tratar de resolver todos los problemas.

La mayor parte de las técnicas con las que contamos para resolver problemas de minimización o maximización de funciones (a partir de ahora hablaremos solo de minimización para abreviar) necesitan condiciones de continuidad y diferenciabilidad. Sin embargo, muchas de las funciones que aparecen en problemas a resolver no cumplen estas condiciones y en algunos casos ni siquiera tiene sentido hablar de estas propiedades por estar tratando con un conjunto discreto. La rama que trata estos últimos problemas se conoce como optimización combinatoria. Es por ejemplo el caso del problema del viajante, conocido como TSP por las siglas en inglés de *Travelling Salesman Problem*. Su enunciado es el siguiente: *Supongamos que un viajero debe recorrer N ciudades. ¿Qué ruta deberá seguir para recorrer la menor distancia posible sabiendo que tiene que acabar en la ciudad en la que empezó y pudiendo estar sólo una vez en cada ciudad?*. Para este problema podemos asociar cada posible solución p a una permutación $(x_1, ..., x_N)$ de los números $\{1, ..., N\}$ que afecte al orden en que se van a recorrer las ciudades, y tomando como función $f(p) = d_{x_1,x_2} + ... + d_{x_{n-1},x_n}$, donde $d_{x_i,x_{i+1}}$ corresponde a la distancia entre las ciudades x_i y x_{i+1}, tenemos que nuestro objetivo consistirá en encontrar el mínimo de f en el conjunto de las permutaciones de $\{1, ..., N\}$. Dejando a un lado los algoritmos de fuerza bruta (consistentes en probar todas las combinaciones posibles) debido a su impracticabilidad computacional, ya que el número de permutaciones crece de manera factorial en función de N, no existe un algoritmo o método de búsqueda eficiente de estos máximos y mínimos. Observemos que al estar tratando en un conjunto discreto no disponemos de propiedades como la continuidad, por lo que f podría representar por ejemplo una función aleatoria. Con esto en mente es sencillo apreciar la imposibilidad de encontrar un algoritmo eficaz de cálculo del mínimo global sin recorrer cada uno de los puntos del conjunto. Aparece entonces la necesidad de desarrollar algoritmos específicos para cada problema, siendo fundamental estudiar en profundidad este campo y cada una de las herramientas disponibles. Alguna de estas herramientas son lo que se denomina

algoritmos heurísticos. Los algoritmos heurísticos tratan de aproximarse al mínimo global de una función, pero sin garantizar alcanzarlo. Constituyen un método de búsqueda de buenas soluciones para cierta función. Como comentamos anteriormente, el mismo argumento que impide la imposibilidad de encontrar algoritmos de búsqueda de mínimos globales para cualquier función hace que tampoco sea sencillo elegir el algoritmo heurístico adecuado para cada problema. El funcionamiento de éste dependerá de la naturaleza de la función con la que estemos trabajando, haciendo necesaria su adaptación. Algunos ejemplos importantes de algoritmos heurísticos son los algoritmos genéticos, algoritmos basados en inteligencia de enjambre o el algoritmo *annealing* (atemperamiento o recocido simulado) que trataremos más en profundidad en el próximo capítulo.

En este trabajo vamos a presentar una de las tecnologías más potentes que tratan de resolver problemas de optimización combinatoria. Esta es la herramienta denominada *Quantum Annealing* (atemperamiento o recocido cuántico), que trata de lograr buenos resultados en la tarea de minimizar un tipo específico de funciones discretas. Veremos cómo es su funcionamiento y trataremos de aplicarlo a un problema que hasta ahora no había sido resuelto mediante *Quantum Annealing*, para apreciar sus ventajas. Al tratar de resolver este problema a través de recocido cuántico, ha sido necesario mejorar alguna de las soluciones existentes de problemas que ya habían sido abordados mediante *Quantum Annealing*. En particular, se ha logrado disminuir notablemente el número de variables requeridas de una de las modelizaciones QUBO (*Quadratic Unconstrained Binary Optimization*) conocidas del problema del viajante y que aparece en diversos artículos. Para aquellos que tengan dominio sobre el algoritmo *Quantum Annealing*, podrán encontrar información de su interés a partir del capítulo 5.

Capítulo 2

Historia de *El Problema del Viajante* y alguna de sus generalizaciones.

Se conocen estudios que mencionan El Problema del Viajante de principios del siglo XIX, pero hasta 1950 no surgieron los primeros resultados importantes abordándolo. En 1972 se demostró que el TSP era un problema NP-completo y se empezaron a estudiar distintas heurísticas que trataban de resolverlo. A partir del TSP aparecen gran cantidad de generalizaciones que resultan de gran interés. Estas generalizaciones aparecen al introducir nuevas condiciones. Veamos algunos ejemplos conocidos de ellas:

1 Las ciudades aparecen agrupadas en lo que comúnmente se denomina estados y el viajero debe encontrar la ruta óptima pasando por cada estado (por lo que es suficiente con pasar por una de las ciudades de cada estado).

2 Existe un ordenamiento secuencial de las ciudades, por lo que algunas ciudades deben visitarse antes que otras. La solución que desarrollamos en la sección 5.2 de este trabajo para el TSP como problema QUBO puede generalizarse de manera muy sencilla a este caso.

3 Existen unas restricciones de tiempo según las cuales el viajero solo puede estar en determinadas ciudades en momentos concretos. A esta generalización se la conoce como problema del viajante con ventanas de tiempo o con las siglas TSPTW, procedentes de *Travel Salesman Problem with Time Windows*. Daremos más detalles sobre este problema en la sección 5.3.5.

4 Existen varios viajantes que deben recorrer todas las ciudades. Esta es la generalización fundamental sobre la que hemos construido nuestro estudio. Veamos a continuación un desarrollo más amplio de este caso general de TSP.

Como acabamos de comentar, la generalización 4 es la que nos ocupa en este trabajo. A este problema se lo conoce como *Vehicle Routing Problem* (VRP). Este nombre procede del siguiente enfoque del problema: *Supongamos que se dispone de una flota de vehículos que deben recoger pedidos localizados en distintas ciudades. Si cada pedido tiene asociado un peso y cada vehículo posee un límite de peso que puede transportar, ¿cuál es la manera óptima de recoger los pedidos minimizando la distancia que recorren los coches?* Nuestro trabajo se enfoca en resolver esta generalización, por lo que vamos a presentarla en profundidad.

Uno de los primeros artículos donde se presenta el problema VRP data de 1974 y podemos encontrarlo en [4]. En este artículo aparecen distintos algoritmos para abordar el problema basándose en heurísticas que trataban de resolver otros problemas relacionados con la teoría de grafos. Uno de los primeras métodos que atacaban el VRP fue un modelo realizado a partir de variables binarias y continuas por G.Dantzig, R.Fulkerson y S.Johnson (modelo DFJ) [6], que posteriormente

fue mejorado por Miller, Tucker y Zemlin (modelo MTZ) [14]. Dado que estas modelizaciones son cercanas a las necesarias para resolver un problema mediante *Quantum Annealing*, serán analizadas en el apéndice exponiendo sus ventajas e inconvenientes. A partir de entonces se han probado y desarrollado distintos algoritmos heurísticos que tratan de conseguir buenos resultados. Actualmente, estos algoritmos siguen aplicándose y mejorándose para resolver los casos actuales. Incluso, como comentaremos posteriormente, están empezando a aparecer distintas propuestas de mejora de estos algoritmos clásicos a partir de la programación cuántica (una de dichas propuestas es este trabajo). Entre los distintos tipos de algoritmos heurísticos existentes, podemos hacer una clasificación en tres grandes grupos: las heurísticas constructivas, heurísticas de mejora y metaheurísticas. Las heurísticas constructivas tratan de ir construyendo las soluciones finales a partir de soluciones parciales del problema. Un ejemplo de este grupo para el TSP consiste en el algoritmo de vecinos cercanos. Las heurísticas de mejora tratan de mejorar soluciones factibles y las metaheurísticas pueden entenderse como estrategias y guías para realizar las mejoras. Las heurísticas de mejora a menudo requieren de heurísticas constructivas para partir de soluciones que después mejorar. En [8] encontramos un modelo que trata de aprovechar la programación cuántica proponiendo un algoritmo híbrido que aplica un algoritmo de mejora a soluciones localizadas mediante herramientas que usan hardware cuántico.

Tenemos entonces que en nuestro trabajo vamos a tratar de aplicar el algoritmo de *Quantum Annealing* al problema del viajante para el caso en el existen varios viajeros que parten del mismo punto central y que entre todos deben recorrer todas las ciudades. Si solo dispusiésemos de un solo viajero, el problema sería equivalente a un problema del viajante clásico. Notar que para este caso, y al contrario de lo que ocurre en otros problemas similares, es necesario o bien introducir un concepto relativo al tiempo que tardan en recorrerse todas las rutas de manera simultánea o bien establecer un máximo de ciudades que puede recorrer cada vehículo. Esta última condición aparece cuando los vehículos deben recoger distintos pedidos en las ciudades y tienen un límite de peso que pueden transportar. Esto se debe a que si solo realizamos un estudio sobre minimizar la distancia que recorren los viajeros o vehículos sin restricciones adicionales, debido a la desigualdad triangular, con un solo vehículo se resolvería el problema. Veamos en el siguiente capítulo en qué consiste el algoritmo de *Quantum Annealing* y cómo podemos aplicarlo al problema en cuestión.

Capítulo 3

Algoritmo *Annealing*

3.1. Annealing clásico

El algoritmo annealing (en inglés *Simulated Annealing*, por lo que a partir de ahora lo denominaremos SA) es un algoritmo de optimización perteneciente a la familia de los algoritmos heurísticos. Se caracteriza por partir de un estado inicial e ir recorriendo soluciones cercanas mejores tratando de localizar los mínimos globales. Podemos encontrar más detalles sobre el funcionamiento de este algoritmo en [20]. Este algoritmo toma su nombre del proceso metalúrgico denominado recocido. Este proceso consiste en aumentar la temperatura de cierto metal para modificar sus propiedades físicas y hacerlo más manejable. Debido a esto es común denominar estados de baja temperatura o baja energía a los mínimos globales de la función que estemos optimizando.

Su funcionamiento básico es el siguiente: supongamos que nuestro objetivo es minimizar una función $f : \mathbb{R}^k \to \mathbb{R}$. Para ello partimos de un estado inicial x_0 y, analizando un conjunto de valores cercanos a x_0 (valores vecinos), que serán calculados mediante la función $fvec$, el algoritmo estudiará si mantener la solución actual o moverse a una de las nuevas. Para llevar a cabo este último paso se realiza una evaluación mediante una función probabilística (función $fsel$), que nos permite decidir si seleccionar la nueva solución calculada o permanecer en el estado anterior. El algoritmo finaliza tras ejecutarse el número de iteraciones fijadas al inicio (*maxite*) mientras busca converger hacia el mínimo global.
Algorítmicamente, se puede describir de la siguiente manera:

1. Inicialización:
   $x \leftarrow x_0$
   $M \leftarrow$ maxite

2. Para i in $\{1, ..., M\}$:
   $y \leftarrow \text{mín}\, fvec(x)$
   $x \leftarrow fsel(x, y)$

Es fundamental modelizar de manera correcta el espacio en el que se van a buscar las soluciones. Trabajar en un conjunto en el que luego resulte difícil establecer distancias entre individuos provocará que el algoritmo carezca de sentido. Esto se debe a que no se dispondrá de una manera correcta de seleccionar vecinos cercanos a uno dado, por lo que el algoritmo se reducirá al estudio de un número determinado de posibles soluciones elegidas aleatoriamente. Si estamos trabajando en $\mathbb{R}^k$, es sencillo establecer la cercanía entre puntos a partir de la norma euclídea de este espacio. Sin embargo, en el espacio de las permutaciones de N elementos no existe una manera natural de establecer distancias entre distintos puntos.

A lo largo del tiempo se han hecho estudios sobre la función que hemos denominado $fsel$ y que selecciona si en cada paso se mantiene la solución actual o si se escoge una de las que se acaban de calcular. Un ejemplo de esta función es el propuesto por Kirkpatrick [11], que define la probabilidad de escoger la nueva solución o permanecer en la anterior de la siguiente manera:

$$P(x, y, i) = \begin{cases} 1 \text{ si } y < x, \\ \exp\left(\frac{-(y-x)}{i}\right) \text{ si } y \geq x \end{cases}$$

donde i se corresponde con la iteración en la que estemos. Esta función de probabilidad busca disminuir la probabilidad de escoger estados de mayor temperatura a medida que avanza el algoritmo.

La elección adecuada de las funciones $fvec$ y $fsel$ es el aspecto más importante de cara a implementar SA. Esto se debe a que el problema fundamental a evitar de este algoritmo es que converja hacia un mínimo local. Debido a esto, ciertas implementaciones del SA permiten que en algunos casos podamos movernos hacia estados de mayor temperatura de cara a tratar de escapar de mínimos locales. El algoritmo de atemperamiento cuántico (QA por sus siglas en inglés, *Quantum Annealing*) permite mejorar los resultados del SA en muchas situaciones. Veamos en la siguiente sección un ejemplo práctico donde apreciar las ventajas de esta tecnología.

3.2. ¿Por qué utilizar *Quantum Annealing?*

Supongamos que nos planteamos el siguiente problema: Dado un tablero de ajedrez queremos disponer sobre el tablero ocho reinas de manera que entre ellas no compartan fila, columna o diagonal. Recordemos que un tablero de ajedrez posee 8×8 casillas y durante el juego una reina se mueve recorriendo un número de casillas arbitrario entre las que se encuentran en su misma fila, columna o cualquiera de sus diagonales. La generalización de este problema para N arbitrario se conoce como problema de las N-reinas. Es conocido que la complejidad computacional de este problema es NP-completo. Podemos encontrar más información acerca de este problema en [18].

Este es uno de los problemas sobre los cuales se puede apreciar los buenos resultados de algoritmos basados en *Annealing*. En la imagen 7 que encontramos en el apéndice en la sección dedicada el problema de las $N-$reinas, tenemos un ejemplo de solución para el caso $N = 50$. Dicha solución se ha encontrado en apenas unos segundos mediante el código que encontramos en [1] y que trata de resolver el problema mediante un simulador de recocido cuántico. Existen pocos algoritmos eficientes para resolver el problema de las N-reinas. Mediante un algoritmo de *backtracking*, siendo éste una de las propuestas más comunes para resolver este problema, el tiempo estimado para encontrar alguna solución de este problema para $N = 50$ sería mucho mayor. Las soluciones proporcionadas para este problema mediante la herramienta del *Annealing* son un ejemplo de la potencia que tiene esta tecnología para resolver ciertos problemas específicos, e incluso se espera que su versión cuántica los mejore. Debido a esto, se hace de interés realizar un estudio sobre los resultados que proporciona el *Quantum Annealing* en distintos problemas. El problema que vamos a tratar de resolver en este trabajo corresponde con una generalización del TSP. En el apartado de conclusiones expondremos la calidad de los resultados obtenidos en nuestro problema. En la siguiente sección trataremos de exponer de manera sencilla el funcionamiento de los dispositivos que implementan esta herramienta sin entrar en mucho detalle, ya que se corresponde con aspectos físicos y no matemáticos.

3.3. Introducción al *Quantum Annealing*.

El hardware que implementa el algoritmo *Quantum Annealing* corresponde a una red de estados de energía denominados cúbits, que poseen una corriente circulante y un campo magnético [22]. Los cúbits acaban tomando los valores 0 y 1, de manera que con una correcta modelización estas posiciones representan los posibles valores de la función que se desea minimizar. Tendremos entonces que los mínimos globales se corresponderán con los estados de menor energía del sistema de campos magnéticos que hayamos preparado. Por la naturaleza de los componentes físicos que forman el hardware, los cúbits tratarán de establecerse en las posiciones que supongan menor energía. De esta manera tendremos una relación entre las soluciones de nuestro problema y las posiciones finales de los cúbits.

A diferencia del annealing clásico, que simula la evolución de una función, el algoritmo de atemperamiento cuántico trata de recrear el problema de manera física. Al final del proceso los cúbits toman los valores 0 o 1, lo que permite construir la solución final. La diferencia con un bit y la computación clásica es que durante la ejecución del algoritmo los cúbits representan la probabilidad de que al final del algoritmo acaben colapsando a 0 y 1. Es decir, mientras que un bit clásico siempre está en posición 0 o 1, un cúbit contiene la probabilidad de acabar colapsando al 0 o al 1.
Lo que acabamos de explicar se debe a que el cúbit es un objeto cuántico. Debido a ello, aparte de tener la propiedad que hemos explicado antes y que en física cuántica se conoce como estado de superposición, los cúbits poseen otras propiedades cuánticas como son el entrelazamiento cuántico y el efecto túnel. No daremos detalles sobre las cuestiones físicas de estas dos características ya que nos centraremos en el estudio de la modelización que debe realizarse para poder aplicar esta tecnología. Sin embargo, es interesante destacar que el efecto túnel permite escapar de ciertos mínimos locales en los que SA se quedaría atascado, y el entrelazamiento cuántico, junto con los estados de superposición, permite al algoritmo estudiar valores de manera mucho más eficiente de lo que conseguiría un ordenador clásico.

En el párrafo anterior ha aparecido uno de los conceptos fundamentales de la programación cuántica: el cúbit. Es necesario aclarar que lo que se conoce como computación cuántica engloba dos paradigmas diferentes. Uno de ellos es el que hemos presentado en el párrafo anterior, consistente en aplicar el algoritmo *Quantum Annealing* a la minimización de cierta función y que está siendo desarrollado por la empresa estadounidense D-Wave. El otro paradigma abarca lo que se conoce como modelo de puertas (en inglés *gates*). El modelo de puertas está siendo desarrollado principalmente por las empresas IBM y Google. Este modelo maneja los cúbits de manera diferente. La principal diferencia entre ambos modelos es que en el modelo de puertas los algoritmos buscan poder controlar el valor de los cúbits en cada instante, mientras que en el modelo de *Quantum Annealing* se dispone una posición inicial y, tras la evolución del sistema tratando de encontrar los mejores estados de energía, es al final del proceso cuando se accede al valor de sus soluciones. Esta diferencia hace que sea muy complicada la construcción de hardware cuántico para el modelo de puertas. Debido a esto, los hardwares de modelos de puertas sólo han alcanzado un máximo de 65 cúbits y todavía no pueden hacer frente a problemas reales. A pesar de ello, ya cuentan con algoritmos poderosos que, cuando puedan ser implementados (Google e IBM lo estiman en el 2030), provocarán una revolución en la computación. Destaca entre ellos el algoritmo de Shor, que permite factorizar números en tiempo polinómico, permitiendo romper los modelos de criptografía basados en el sistema RSA [7]. Por otro lado, los hardware del modelo *Annealing* desarrollados principalmente por la compañía D-Wave ya cuentan con computadoras con 5000 cúbits y pueden tratar problemas reales. Uno de los trabajos actuales desarrollados con esta tecnología ha sido la mejora del tráfico de Lisboa, proyecto en el que se involucró la empresa Volkswagen y del que podemos encontrar más información en [5].

Las condiciones básicas en las que QA aporta grandes beneficios sobre SA aparecen cuando la

función a optimizar cuenta con mínimos locales muy "profundos" pero poco "anchos"; es decir, su valor es mucho menor al de sus vecinos cercanos (profundidad) pero existen puntos con valores menores no lejanos (anchura). Los algoritmos cuánticos que acabamos de mencionar deben implementarse en hardware especializado para resolver este tipo de problemas. Como comentamos antes, la primera empresa en desarrollar esta tecnología ha sido D-Wave y ya cuenta con clientes como Google, Amazon o Volkswagen, del que ya hemos hablado anteriormente. Para resolver un problema con los algoritmos cuánticos que soporta el hardware de D-Wave, debe expresarse dicho problema en lo que se conoce como modelización QUBO. En la siguiente sección explicamos en detalle en qué consisten esta familia de modelizaciones.

3.4. Modelizaciones tipo QUBO de cara a aplicar *Quantum Annealing.*

Los problemas que los ordenadores cuánticos de D-Wave están preparados para resolver son aquellos que consisten en encontrar el mínimo de una función de la forma

$$\sum_{i=1}^{n} b_i x_i + \sum_{i=1}^{n}\sum_{j=1}^{n} q_{i,j} x_i x_j$$

donde las variables $x_i \in \{0,1\}$ y los coeficientes $b_i, q_{i,j} \in \mathbb{R}$. Tenemos entonces que dado un problema debemos modelizarlo de tal manera que las variables que forman la solución solo tomen los valores 0 o 1. Observemos que, por tomar las variables x_i los valores 0 o 1, se cumple que $x_i^2 = x_i$. Por lo tanto, podemos agrupar los términos lineales con los cuadráticos y expresar la ecuación anterior en formato matricial como

$$x^t Q x$$

donde $x \in \{0,1\}^n$ y $Q \in \mathfrak{M}_{n\times n}$, siendo Q una matriz simétrica. Esta modelización que acabamos de definir se denomina QUBO. Su característica principal es que las variables implicadas toman únicamente los valores 0 y 1. En esta sección estudiaremos cómo podemos tomar un problema de optimización combinatoria y modelizarlo en formato QUBO. Tenemos entonces que dado un problema, nuestro **objetivo** será construir una función que corresponda con un polinomio de grado 2 donde las variables solo tomen los valores 0 o 1 y tal que su mínimo global coincida con la solución a nuestro problema. En numerosos problemas no suele ser inmediato encontrar una modelización que corresponda con una expresión de la forma $x^t Q x$. Veamoslo con el siguiente ejemplo.
Ejemplo: Supongamos que queremos encontrar el mínimo de la función

$$f(x_1, x_2, x_3) := -x_1 - 2x_2 - 3x_3. \tag{3.1}$$

Para este ejemplo es sencillo ver que la solución corresponde con $x_1 = 1, x_2 = 1, x_3 = 1$. Ahora supongamos que tenemos la restricción extra de que alguna de las variables x_i debe ser 0. Esta restricción extra puede representarse como $x_1x_2x_3 = 0$. Tenemos entonces que si tratamos de minimizar la función

$$g(x_1, x_2, x_3) := -x_1 - 2x_2 - 3x_3 + 10x_1x_2x_3, \tag{3.2}$$

nos encontramos con que debido a la introducción del término $10x_1x_2x_3$, estamos penalizando los casos en los cuales todas las variables x_i toman el valor 1. Tenemos entonces que el mínimo de la función g coincide con el mínimo de la función f cuando se aplica la restricción $x_1x_2x_3 = 0$. Esto se debe a que el coeficiente 10 del término $x_1x_2x_3$ es lo suficientemente grande para que si $x_1x_2x_3 = 1$, la función g tome valores grandes. Sin embargo, la función g posee términos cúbicos que deben convertirse en cuadráticos para que el problema pueda ser resuelto mediante la herramienta del *Quantum Annealing*. Veamos cómo podemos reducir g a una expresión $x^t Q x$.

3.4.1. Polinomios con grado mayor que 2

Como acabamos de ver en el ejemplo anterior, al tratar de modelizar un problema en variables binarias puede ocurrir que en la función a minimizar (o función de coste) aparezcan términos de grado mayor que 2. Como el *annealing* cuántico de D-Wave está implementado para manejar términos de grado menor o igual que 2, se debe reducir el grado de los monomios que no cumplen esta condición. Veamos distintas maneras de transformar un monomio cúbico de la función de coste en cuadrático. Una vez que contemos con esta técnica, aplicándola recurrentemente podremos disminuir el grado de cualquier monomio.

Una primera forma consiste en transformar el problema $\min xyz$ en el problema $\min wz$ sujeto a $w = xy$. Veremos más adelante que no podemos tener una restricción cuadrática, así que busquemos una manera de sustituir $w = xy$ por restricciones lineales. Tomando las restricciones $w \leq x$, $w \leq y$ y $w \geq x + y - 1$, tenemos que podemos transformar el problema de minimizar xyz en el problema

$$\text{minimizar } wz$$

$$\text{sujeto a } \begin{cases} w \leq x \\ w \leq y \\ w \geq x + y - 1 \end{cases}.$$

Esta remodelización involucra introducir tres restricciones de desigualdad. Las restricciones de desigualdad, si bien son muchas veces inevitables, resultan un poco ineficientes. Esto se debe a la necesidad de incluir un conjunto de variables de holgura auxiliares necesarias para transformar cada desigualdad en igualdad. Este procedimiento es similar al que ocurre cuando se modeliza un problema de programación lineal que vaya a resolverse mediante el algoritmo símplex. En las siguientes secciones daremos más detalles de esta técnica. Busquemos por lo tanto otras técnicas mejores que permitan reducir el grado de los monomios.

La primera técnica que veremos consiste en explotar la relación

$$xyz = \max_{w \in \{0,1\}} \{w(x + y + z - 2)\}$$

Esta relación se deduce de que, por un lado, si alguna de las variables x, y, z es igual a 0, tenemos que $(x + y + z - 2) \leq 0$ y por tanto el máximo de la derecha se alcanzará cuando w tome el valor 0, dando como resultado 0. Si por el contrario todas las variables valen 1, tendríamos el producto $w \cdot 1$, que toma el valor máximo 1 cuando $w = 1$. Ahora bien, si el monomio $axyz$ del que queremos reducir el grado tiene coeficiente $a < 0$ se cumple que

$$\min_{x,y,z} axyz = a \max_{x,y,z}(xyz) = a \max_{x,y,z}(\max_{w}(w(x + y + z - 2))) =$$

$$a \max_{x,y,z,w}(w(x + y + z - 2)) = \min_{w,x,y,z} aw(x + y + z - 2)$$

Por lo tanto, si $a < 0$ podemos simplemente sustituir monomios de la forma $axyz$ por $a(w(x + y + z - 2))$. Si $a > 0$ no podemos utilizar la misma técnica ya que, aunque

$$xyz = -\min_{w \in \{0,1\}} w(-x - y - z + 2),$$

el signo menos no permite introducir este término en la función de coste. Para solucionar este inconveniente volvamos al problema de expresar la ecuación $z = xy$ en términos lineales. Para ello vamos a tratar de construir una función de penalización que tome valores grandes cuando $z \neq xy$ y el valor 0 cuando $z = xy$.

3.4.2. Funciones de penalización

Supongamos que queremos minimizar una función f sujeta a cierta restricción. Podemos intentar transformar f en una función g de manera que los valores que no cumplan la restricción tomen valores "grandes" en g. Para ello vamos a introducir el concepto de función de penalización.

Definición 3.4.1. Sean P y f funciones de $\{0,1\}^N$ en $\mathbb{R}$, r una restricción y $y := argmin\{f(z) : z \text{ cumple la restricción } r\}$, es decir, y es el punto donde se alcanza el mínimo para los valores que cumplen r. Diremos que P es una *función de penalización* de la función f para la restricción r si para los valores x que no satisfacen r, se cumple que $f(x) + p(x) > f(y)$ y $p(z) = 0$ para los valores z que cumplen la restricción r.

De esta manera, añadiendo dicha función de penalización a la función de coste, tendremos que las mejores soluciones serán aquellas que cumplan la restricción. Con esta definición tenemos que la función $P(x_1, x_2, x_3) := 10x_1x_2x_3$ es una función de penalización para (3.1). Volviendo al caso anterior, vamos a tratar de encontrar una función de penalización cuadrática que cumpla que $P(x, y, z = f(x, y)) = 0$ y $P(x, y, z \neq f(x, y)) = p$ (en el caso anterior $f(x, y) = xy$, pero puede representar cualquier otra función), siendo p una constante que simboliza la penalización para los casos que no nos interesan. La **notación** $(x, y, z = f(x, y))$ corresponde a los puntos (x, y, z) tales que se cumple que $f(x, y) = z$. Como la función $P(x, y, z)$ debe ser cuadrática, podemos expresarla con la forma

$$P(x, y, z) = a_1x + a_2y + a_3z + a_4xy + a_5xz + a_6yz + a_7.$$

por lo que para el caso de tener n variables, estas restricciones se corresponden con un sistema de 2^n ecuaciones linealmente independientes, una por cada posible combinación de los valores de las variables, y $\binom{n}{2} + n + 1$ incógnitas, valor resultante de considerar la función de penalización como una función cuadrática y calcular el número de términos de la misma, siendo las incógnitas los correspondientes coeficientes (el siguiente ejemplo ayudará a entender estos valores). Como consecuencia, para $n > 2$ tenemos un sistema sin solución. La alternativa principal consiste en establecer como restricciones $P(x, y, z = f(x, y)) = 0$ y $P(x, y, z \neq f(x, y)) >= p$. Por la forma de la función de penalización podemos establecer $p = 1$ y, una vez calculada la función de penalización, multiplicar toda ella por el valor $p > 0$ que nos interese. En [23] se propone utilizar para el caso $z = xy$ la función de penalización $P(x, y, z) = xy - 2(x + y)z + 3z$. Veamos cómo obtener esta función. Como hemos comentado en el párrafo anterior, podemos expresar la función $P(x, y, z)$ como

$$P(x, y, z) = a_1x + a_2y + a_3z + a_4xy + a_5xz + a_6yz + a_7.$$

Es claro que si queremos que la función tome el mismo valor para todos los casos en los que $z = xy$, este valor debe ser 0, deducido de tomar $x = 0, y = 0$. Por lo tanto, para los cuatro casos en los que $z = xy$ se debe cumplir que $P(x, y, z) = 0$. Esto conduce a tomar $a_7 = 0$. Estudiando cada caso de posibles valores extraeremos unas ecuaciones para los coeficientes $a_1, ..., a_6$.

- De los valores $(1, 0, 0)$ y $(0, 1, 0)$ deducimos que $a_1 = 0$ y $a_2 = 0$.
- Del valor $(1, 1, 1)$ se deduce que $a_3 = -(a_4 + a_5 + a_6)$.

Veamos qué valor deben tomar los coeficientes para que en los puntos $(0, 0, 1)$, $(0, 1, 1)$, $(1, 0, 1)$, y $(1, 1, 0)$ la función alcance valores de penalización.

- Del punto $(0, 0, 1)$ deducimos que $a_4 + a_5 + a_6 < 0$.
- Del punto $(0, 1, 1)$ deducimos que $a_4 + a_5 < 0$.
- Del punto $(1, 0, 1)$ deducimos que $a_4 + a_6 < 0$.

- Del punto $(1,1,0)$ deducimos que $a_4 > 0$.

Podemos tomar valores cualesquiera que cumplan las cuatro restricciones. Si tomamos $a_4 = 1$, $a_5 = -2$, $a_6 = -2$, tenemos que se cumplen todas las restricciones, por lo que la función de penalización tendría la forma

$$P(x,y,z) = 3z + xy - 2xz - 2yz = 3z + xy - 2(x+y)z \tag{3.3}$$

coincidiendo con la función propuesta por D-Wave. Hemos desarrollado una manera de calcular funciones de penalización que nos garanticen que los mínimos de nuestra función a minimizar sean puntos donde se satisfagan las restricciones que impongamos. Sin embargo, existen casos para los cuales conseguir funciones de penalización de manera directa no suele ser tan sencillo. Veamos a continuación un ejemplo que presenta mayores dificultades.

Ejemplo. Supongamos ahora que queremos encontrar una función de penalización para la expresión $z = f(x,y)$, donde f corresponde con la puerta lógica XOR. Dichos valores buscados corresponden con $(0,0,0),(0,1,1),(1,0,1),(1,1,0)$. Es decir, $z = 0$ si $x = y$ y $z = 1$ para el resto de casos. Si tratamos de encontrar una función de penalización con la forma

$$P(x,y,z) = a_1x + a_2y + a_3z + a_4xy + a_5xz + a_6yz$$

nos damos cuenta de que el sistema de ecuaciones e inecuaciones que se genera no tiene soluciones factibles. Para probar esto podemos hacer uso del algoritmo símplex. Cuando se da este caso, podemos añadir una variable auxiliar w y tratar de encontrar una función de penalización cuadrática $P(x,y,z,w)$. Explicaremos a continuación el comportamiento de la variable w.

Funciones de penalización con variables auxiliares. Nuestro objetivo es conseguir que, en la función de coste, los casos en los que $z \neq f(x,y)$ tengan valores altos. Tenemos por lo tanto que se debe cumplir que $P(x,y,z \neq f(x,y),w) \geq 1$. Veamos qué ocurre en los casos donde $P(x,y,z = f(x,y),w)$. Parece intuitivo tratar de lograr que para todos estos casos la función $P(x,y,z,w)$ tenga valor 0. Sin embargo, al realizar las operaciones necesarias nos encontramos con que no existen soluciones que cumplan las restricciones. Nuestro propósito será entonces conseguir una función de penalización tal que cuando tome el valor 0 se cumpla la condición que deseamos, y no que tome el valor 0 cuando se dé la condición que buscamos. Es decir, será suficiente con garantizar que para algunos valores de $(x,y,z = f(x,y),w)$ (intentaremos que por lo menos uno para cada cada valor de $(x,y,z = f(x,y))$) se cumpla que $P(x,y,z,w) = 0$. Para los valores restantes simplemente impondremos que la función tome valores mayores que 0. De esta manera estamos relajando restricciones sobre los coeficientes a_i. Sea $A := \{(x,y,z,w) : z = f(x,y)\}$ y sea $B \subset A$ el conjunto $B := \{(x,y,z,w) : P(x,y,z,w) = 0\}$. Para el resto de puntos $C := A \setminus B$ impondremos que $P(x,y,z,w) \geq 0$. De esta manera garantizamos que para valores donde $P = 0$ se da la condición que buscamos y que no estamos dando preferencia a ninguna combinación. Observar que en función de cómo tomemos el conjunto B tendremos funciones de penalización distintas. Tenemos entonces que una vez introducida la variable auxiliar w podemos encontrar una función de penalización para la ecuación $z = f(x,y)$, siendo f la función XOR. La prueba de que no existe una función de penalización con las condiciones anteriores puede aplicarse para encontrar la función de penalización que buscamos ahora.

Tratar de manera directa con funciones de penalización no es una práctica habitual. Sin embargo, en la mayoría de problemas que tratan de resolverse mediante el algoritmo *Quantum Annealing* aparecen restricciones de igualdad y de desigualdad que deben satisfacer las variables. Veamos en las siguientes secciones cómo podemos construir de manera sencilla funciones de penalización a partir de este tipo de restricciones.

3.4.3. Restricciones de igualdad

Supongamos que tenemos una restricción de igualdad de la forma $a = p(x_1, ..., x_n)$, siendo p un polinomio de grado 1 en las variables $x_1, ..., x_n$. Se cumple entonces que la función $f := (a - p(x_1, ..., x_n))^2$ toma su valor mínimo 0 en los puntos para los cuales $p = a$. De esta manera, encontrar $x_1, ..., x_n$ tales que $p(x_1, ..., x_n) = a$ coincide con encontrar los valores $x_1, ..., x_n$ donde f alcanza su mínimo. Tenemos entonces que la función f anterior se corresponde con una función de penalización para la restricción lineal $a = p(x_1, ..., x_n)$. Una vez llegados a este punto, queda multiplicar la función de penalización f por un valor λ, al que denominamos coeficiente de Lagrange, de manera que se garantice que las mejores soluciones son aquellas que satisfacen la restricción lineal.

Detallemos en este párrafo lo referido al coeficiente de Lagrange que hemos nombrado antes. Supongamos que queremos minimizar la función $g(x_1, ..., x_n)$ sujeto a $f(x_1, ..., x_n) = 0$, siendo f un polinomio de grado 1. Podemos entonces tomar como función de coste

$$h(x_1, ..., x_n) = g(x_1, ..., x_n) + \lambda f^2(x_1, ..., x_n),$$

donde λ es un coeficiente de Lagrange. Si λ tiene un valor suficientemente alto, tendremos entonces que cuando no se satisfaga la restricción de la función f, la función de coste tomará valores más altos que en aquellos puntos donde sí se cumpla la restricción correspondiente a la función f. Daremos más detalles sobre un correcto ajuste de los coeficientes de Lagrange en la sección 3.5. Una vez que hemos explicado cómo construir funciones de penalización que garanticen que se cumplen restricciones lineales de igualdad, explicaremos como garantizar que se cumplen las restricciones de desigualdad.

3.4.4. Restricciones de desigualdad

Otras de las posibles restricciones que pueden aparecer modelizando un problema en su formato QUBO son las restricciones de desigualdad. Estas restricciones son tratadas de forma similar a lo que ocurría en el algoritmo símplex. Tenemos entonces que debemos convertirlas en restricciones de igualdad introduciendo variables de holgura. Como en las modelizaciones QUBO las variables deben ser binarias, estas variables deben introducirse expresándose de manera binaria. Veámoslo con el siguiente ejemplo. Supongamos que tenemos la restricción $f(x_1, .., x_n) \leq b$ y sea $m := \min f$. Calculemos entonces el número $na := \lfloor log_2(b - m) \rfloor + 1$. Podemos convertir la restricción anterior en $f(x_1, ..., x_n) + \sum_{j=0}^{na} 2^j a_j = b$. Este método tiene el inconveniente de que aumenta el número de variables necesarias en el modelo por cada restricción de desigualdad, añadiendo gran cantidad de cúbits de cara a su programación. La solución óptima consiste en encontrar una función de penalización P que tome el mismo valor cuando se cumpla la restricción, y valores grandes cuando no. De esta manera evitaríamos introducir variables auxiliares extra. Sin embargo, encontrar estas funciones de penalización suele ser un trabajo muy complicado.

3.5. Coeficientes de Lagrange

Una de las tareas más importantes para realizar una buena modelización QUBO es el ajuste de los coeficientes de Lagrange. En muchos artículos que abordan problemas mediante el algoritmo *Quantum Annealing* se proponen diversas modelizaciones sin hacer referencia sobre los valores que deben tener los coeficientes de Lagrange. Una buena modelización con unos malos coeficientes de Lagrange provocará que el algoritmo de atemperamiento cuántico no arroje buenos resultados. En muchas modelizaciones el ajuste de estos coeficientes resulta ser una tarea complicada y delicada. Si los coeficientes de Lagrange no son lo suficientemente grandes, aparecerán soluciones que no cumplen ciertas restricciones. Si por el contrario toman valores demasiado grandes, el algoritmo se atascará en

soluciones que garanticen que se cumplen ciertas restricciones y no conseguirá buenos valores de la función objetivo o en otras restricciones. Podemos apreciar este hecho con el código correspondiente al problema de las N-reinas que encontramos en [1]. Si tomamos en el código $lagrange = size * *2$ (es decir, la variable de nombre *lagrange* se obtiene al elevar al cuadrado la variable de nombre *size*), tendremos que el algoritmo deja de encontrar soluciones para el problema a partir de $N \geq 11$. Si por el contrario tomamos en el código $lagrange = 2$, el algoritmo proporciona soluciones para valores de N tan grandes como 100. El caso óptimo en el que el algoritmo de *Quantum Annealing* proporciona resultados realmente buenos se produce cuando el óptimo está suficientemente alejado del resto de valores y no existen puntos para los cuales la función toma valores grandes muy alejados del resto.

Vamos a tratar de mostrar algunos resultados que nos ayuden a estimar los coeficientes de Lagrange. El problema de estimar los coeficientes de Lagrange no es un problema novedoso del *Quantum Annealing*. En el artículo [21] se realiza un estudio sobre cómo estimar estos coeficientes cuando se está realizando una modelización QUBO previamente a aplicar *annealing* clásico. Dicho artículo propone unas condiciones que ha de cumplir el coeficiente de Lagrange cuando el algoritmo heurístico con el que se va a resolver el problema es de tipo 1-flip. Los algoritmos heurísticos de tipo 1-flip son aquellos en los que, para pasar de una posible solución a una nueva, simplemente se produce un cambio en una de las variables que forma el modelo. Esta condición conduce a un caso más sencillo para el cual el coeficiente de Lagrange debe cumplir condiciones menos restrictivas. Sin embargo, algunos de los resultados de los que se habla son similares a los que vamos a manejar ahora.

Tratemos de generalizar lo expuesto en el artículo citado a estimar el valor que debe tomar λ para cada penalización en el caso del *Quantum Annealing*. Supongamos que tenemos como función de coste $h(x_1, ..., x_n) = g(x_1, ..., x_n) + \lambda f(x_1, ..., x_n)$, donde g es la función que queremos minimizar y f representa una restricción que queremos que se satisfaga. Sea $y \in \{0,1\}^n$ y $x \in \{y : f(y) = 0\}$. Supongamos que x_0 es el valor que minimiza g y cumple $f(x) = 0$ (la solución de nuestro problema). Necesitaremos que se cumpla que para todo y:

$$g(y) + \lambda f(y) > g(x_0) + \lambda f(x_0) = g(x_0).$$

Si tomamos como $p := \min\{f(y) : f(y) > 0, y \in \{0,1\}^n\}$, tenemos que para todo y

$$g(y) + \lambda f(y) \geq g(y) + p\lambda \geq \min g(y) + \lambda p, \tag{3.4}$$

por lo que si garantizamos que $\min g(y) + \lambda p > g(x_0)$, tendremos que se cumplirá lo que buscamos. Como en muchos problemas calcular $g(x_0)$ será a equivalente a resolver el problema, es suficiente con tomar

$$\lambda > \frac{1}{p}\left(g(x) - \min_y g(y)\right),$$

siendo x uno de los puntos donde se cumpla que $f(x) = 0$.
Todos los mínimos que aparecen en el párrafo anterior se corresponden con el mínimo de cierta función en un conjunto de valores finito, por lo que existen, son finitos y se cumple que $p \neq 0$. Si no fuese sencillo estimar un valor de $g(x)$, podría sustituirse por $\max_y g(y)$. Estimar de manera exacta el valor p se correspondería con resolver el denominado problema de suma 0, cuya complejidad computacional es 2^n (como se indica en el artículo citado anteriormente). Sin embargo, si la precisión de los coeficientes de f se corresponde con 10^{-m}, podría simplemente aproximarse p por el valor $a10^{-m}$ con $a \in (0,1)$. A pesar de esto, como comentamos anteriormente, λ no debe ser excesivamente grande ya que ocasionará que la función a minimizar presente mínimos locales con valores mucho menores que los de su alrededor. Por lo tanto, trataremos de ajustar λ de manera que sea lo menor posible tratando de satisfacer la desigualdad

$$\lambda > \frac{1}{p}\left(g(x) - \min_y g(y)\right) \tag{3.5}$$

Apliquemos lo que acabamos de exponer al problema de las N-reinas. De [1] tenemos que nuestra función de coste se corresponde con

$$f(x) = -\sum_{i=1}^{n}\sum_{j=1}^{n} x_{i,j} + \lambda \sum_{i_1=1}^{n}\sum_{i_2=1}^{n}\sum_{i_3=1}^{n}\sum_{i_4=1}^{n} J_{i_1,i_2,i_3,i_4} x_{i_1,i_2} x_{i_3,i_4}, \tag{3.6}$$

donde las variables $x_{i,j}$ toman el valor 1 si hay una reina en la posición (i,j) y el valor 0 si no la hay. Además, los coeficientes J_{i_1,i_2,i_3,i_4} cumplen que toman el valor 1 si la reina que se encuentra en la posición (i_1,i_2) ataca a la que se encuentra en la posición (i_3,i_4), y toman el valor 0 en caso contrario. Bajo estas condiciones, si la reina de la casilla (a,b) está atacando a la reina de la casilla (c,d), entonces la función de penalización toma un valor superior a $J_{a,b,c,d} + J_{c,d,a,b} = 2$, por lo que $p = 2$. Para este problema conocemos el valor $g(x_0) := -N$ y el valor $\min g(y) = -N^2$. Por tanto, será suficiente tomar $\lambda > \frac{1}{2}(-N - (-N^2)) = \frac{1}{2}(N^2 - N)$. Sin embargo, podemos mejorar el valor de λ para este problema. Basta observar que si en la solución óptima introducimos una reina nueva, dicha reina estará atacándose como mínimo con otra reina. Tenemos entonces que la función de coste tomará el valor $-(N+1) + 2\lambda = -N - 1 + 2\lambda$. Por lo que si garantizamos que $-N < -N-1+2\lambda \Leftrightarrow \frac{1}{2} < \lambda$ tendremos garantizado que el óptimo cumple las restricciones. Por tanto podemos tomar como λ cualquier valor mayor que $\frac{1}{2}$, lo que hace que funcione el valor $lagrange = 2$ que comentamos en la introducción de esta sección. Muy relacionado con los valores de Lagrange aparece el concepto de valor de restricción. Veamos cómo se define y sus consecuencias.

3.5.1. Valores de restricción

Vamos a definir como valor de restricción al intervalo de valores que puede tomar la función de penalización correspondiente a una restricción. El valor mínimo de estas funciones de penalización suele ser sencillo de calcular, ya que si la función de penalización la hemos construido a partir de una igualdad, dicho mínimo coincidirá con el término independiente que hemos eliminado al elevar al cuadrado. Es decir, si teníamos la condición de igualdad $h(x_1,...,x_n) - a = 0$, la función de penalización corresponderá con $P(x_1,...,x_n) := h^2(x_1,...,x_n) - 2ah(x_1,...,x_n)$, donde hemos eliminado el término a^2, por lo que el mínimo de la función P corresponderá con $-a^2$. Sin embargo, calcular el máximo de una función de penalización suele ser más complicado, aunque puede estimarse de manera muy sencilla con el propio algoritmo de D-Wave.

Volviendo a tomar como ejemplo el problema de las N-reinas, tenemos como función de penalización

$$\sum_{i_1=1}^{n}\sum_{i_2=1}^{n}\sum_{i_3=1}^{n}\sum_{i_4=1}^{n} J_{i_1,i_2,i_3,i_4} x_{i_1,i_2} x_{i_3,i_4}. \tag{3.7}$$

Esta función toma como valor mínimo 0. Calcular el valor máximo para esta restricción no es muy complicado. Este valor corresponde con tomar $x_{i,j} = 1$ para todo i,j; es decir, colocar una reina en cada casilla del tablero. Con esta disposición cada reina ataca a todas las reinas de su misma fila, columna y las de las diagonales. Tenemos entonces que el máximo se corresponde con

$$\frac{1}{3}(N-1)(10N-2)N. \tag{3.8}$$

Podemos encontrar las operaciones realizadas para llegar a este resultado en la sección del apéndice dedicada al problema de las N-reinas (7.9). Tenemos entonces que para obtener una modelización óptima, se debe cumplir que el intervalo de valor de restricción para cada restricción no debe ser muy amplio. Por lo que si tomamos un coeficiente de Lagrange λ con valor muy alto, el valor de restricción asociado será muy grande y provocará que el *Quantum Annealing* se atasque en mínimos locales. Si por el contrario el coeficiente de Lagrange λ de una restricción no es lo suficientemente grande,

puede ocurrir que se llega soluciones que no cumplan dicha restricción. Una vez que conocemos los aspectos necesarios para realizar la modelización de un problema, veremos en el siguiente capítulo algunos ejemplos sencillos de ellas.

Capítulo 4

Problemas resueltos utilizando el algoritmo *Quantum Annealing*

Antes de tratar el problema en el que se centra el trabajo, veamos dos modelizaciones de distintos problemas que podemos resolver aplicando el algoritmo *Quantum Annealing*.

4.1. El problema Max Cut

Uno de los problemas más sencillos para resolver a través del algoritmo de atemperamiento cuántico es el problema conocido como Max Cut. Su enunciado es el siguiente: *Dado un grafo G con N vértices, se debe separar dicho grafo en dos subgrafos G_0 y $G_1 := G \setminus G_0$ de manera que la suma del número de aristas de G_0 y el número de aristas de G_1 sea la menor posible.* Notar que las aristas que no formarán parte de esta suma son aquellas tales que uno de sus vértices pertenece a G_0 y el otro a G_1. Definamos las variables que formarán nuestra solución:

- Sean las variables binarias $\{x_i\}_{i=1}^N$, de manera que $x_i = 0$ significa que el vértice i pertenece a G_0 y $x_i = 1$ significa que el vértice i pertenece a G_1.
- Sea $J_{i,j}$ la matriz que representa el grafo. De esta manera, $J_{i,j} = 1$ significa que existe una arista entre los vértices i y j. Observar que la matriz J es simétrica. Introducimos la notación $deg(i) := \sum_{j=1}^N J_{i,j}$; es decir, $deg(i)$ denota el grado del vértice i. Recordemos que en teoría de grafos el grado de un vértice denota el número de aristas que confluyen en él.

En estas condiciones, la función a minimizar corresponde con la suma de las aristas de G_0 y las aristas de G_1. Esta función se puede expresar como

$$\text{mín} \sum_{i=1}^{N} \sum_{j=1}^{N} J_{ij}(x_i x_j + (1 - x_i)(1 - x_j)). \tag{4.1}$$

Notar que el factor $(x_i x_j + (1 - x_i)(1 - x_j))$ toma el valor 1 si los vértices i y j pertenecen al mismo subgrafo y 0 si no. Desarrollando el factor anterior y eliminando los términos independientes, llegamos a que minimizar dicha función es equivalente a minimizar

$$\sum_{i=1}^{N} \sum_{j=1}^{N} J_{i,j} x_i x_j - \sum_{i=1}^{N} deg(i) x_i. \tag{4.2}$$

En este problema que acabamos de resolver no ha sido necesario introducir restricciones. Veamos ahora un ejemplo donde sí aparecen restricciones.

4.2. El problema Máximum Clique

El problema Máximum Clique se define de la siguiente manera: *Dado un grafo G con N vértices, hallar el subgrafo G_1 con mayor número de vértices de manera que todos los vértices del subgrafo estén conectados entre ellos.* Veamos la modelización de este problema:

- Sean las variables $\{x_1, ..., x_N\}$ tales que $x_i = 1$ si el vértice i pertenece a G_1 y 0 si no.
  La matriz $J_{N\times N}$ contiene la información de los vértices que están conectados. Tendremos que $J_{i,j} = 1$ si existe un arista entre los vértices i y j, y 0 si no. Definamos también $J_{i,i} = 0$ para todo i.

- Se debe cumplir que para toda pareja de vértices que formen parte de G_1 exista un arista que los una. Por lo que si $x_i = 1$ y $x_j = 1$, entonces $J_{i,j}$ debe valer 1. Esta restricción se puede expresar con la función de penalización para cada i, j:

  $$P_{i,j}(x_i, x_j) := x_i x_j (1 - J_{i,j}). \tag{4.3}$$

  La función de penalización $P_{i,j}$ toma el valor 1 únicamente en el caso de que $x_i = 1$, $x_j = 1$ y $J_{i,j} = 0$, y 0 en el resto de casos, por lo que en efecto cumple nuestras condiciones de función de penalización. Una vez que conozcamos cuál es la función a minimizar, ajustaremos su coeficiente de Lagrange.

- La función objetivo consiste en maximizar el número de vértices del grafo G_1, lo cual se puede expresar como

  $$\text{mín} \sum_{i=1}^{N} -x_i. \tag{4.4}$$

Tenemos entonces que la función a minimizar se corresponde con

$$\sum_{i=1}^{N} -x_i + \sum_{i=1}^{N}\sum_{j=1}^{N} \lambda x_i x_j (1 - J_{i,j}). \tag{4.5}$$

Veamos ahora cómo ajustar el coeficiente λ para completar nuestro modelo. Se debe conseguir que las soluciones que no cumplan la restricción tomen valores más altos que aquellas que sí la cumplan. Para ello observamos que si los vértices i y j no están conectados, entonces su aportación a la función de coste se corresponderá con

$$-x_i - x_j + \lambda x_i x_j + \lambda x_j x_i = -2 + 2\lambda. \tag{4.6}$$

Para que nuestro modelo descarte las soluciones no factibles, se deberá tomar λ tal que $-2 + 2\lambda > 0$; es decir, $\lambda > 1$. Para que exista una diferencia suficientemente grande entre las soluciones factibles y no factibles podemos tomar $\lambda := \frac{3}{2}$. Una vez explicado cómo modelizar estos dos problemas, vamos a tratar de resolver el problema en el que se centra este trabajo.

Capítulo 5

Solución del problema Q-Robots

Llegados a este punto, recordemos el problema que vamos tratar de resolver: *Supongamos que en un almacén hay N objetos distribuidos que deben ser llevados al punto de salida. En el mismo punto de salida tenemos Q robots que van a ser los encargados de ir a por los N objetos y volverlos a traer al punto de salida. ¿Cómo deben los robots ir a por los N objetos, de manera que se minimice el tiempo que estos tardan?* Observar que minimizar el tiempo que los robots tardan en completar los viajes es equivalente a minimizar el máximo de las distancias que recorren los robots. En [2] encontramos este problema resuelto cuando se busca minimizar la distancia que recorren los robots.

Una vez que hemos dado en el capítulo anterior una primera idea sobre cómo realizar la modelización de un problema para que pueda resolverse mediante el algoritmo de *Quantum Annealing*, tratemos de resolver nuestro problema en cuestión. La modelización de este problema no es tan sencilla como las anteriores. De hecho, ni siquiera aparecen de manera directa las variables que representan las posibles soluciones. Los estudios realizados sobre problemas similares al que nos enfrentamos y que requieren modelizaciones QUBO, desde el propio TSP a generalizaciones como las propuestas en [15] y [10], no terminan de lograr buenos resultados para el caso de *Quantum Annealing*. Esto se debe a que presentan ciertos aspectos a mejorar que considerábamos necesarios, el más importante de ellos corresponde con el número de variables que emplean. Como explicaremos en detalle más adelante, para nuestro problema no era aplicable de manera eficiente la modelización QUBO más frecuente del TSP, debido a que la función que calcula la distancia es cuadrática. Como para nuestro problema es necesario establecer relaciones entre distancias, es necesario manejar una modelización para la cual la función de cálculo de las distancias sea lineal. Existe una modelización del TSP que cumple que el cálculo de las distancias es lineal, y que es utilizada en [15]. Sin embargo, requiere de N^3 variables para aplicarse, lo que en nuestro problema se convierte en QN^3 variables necesarias. Esto la hacía computacionalmente impracticable para nuestro caso debido a que los ordenadores cuánticos que se manejan a día de hoy permiten trabajar con un número de cúbits muy reducido (en torno a 2000). Era por tanto fundamental tratar de desarrollar una modelización nueva del TSP, para la cual el cálculo de la distancia fuese lineal y que requiriese de menos de N^3 variables. La modelización realizada se presenta en 5.2. Posteriormente veremos cómo podemos generalizar nuestra modelización del TSP a nuestro problema Q-Robots. Esta generalización tampoco es inmediata y requiere hacerse de manera delicada. El último inconveniente a salvar para terminar de resolver el problema consistía en tratar de conseguir minimizar el máximo de las distancias que recorren los robots. Recordemos que en nuestra modelización solo podemos manejar polinomios cuadráticos, por lo que no puede aparecer una expresión de la forma $\text{máx}(d_1, ..., d_q)$. En la última sección de este capítulo veremos cómo solventar esto mediante una restricción.

5.1. Primera modelización del problema Q-Robots

Como explicamos en la introducción de este capítulo, no es eficiente tratar de generalizar la modelización fundamental del TSP a nuestro problema. Para verlo vamos a tratar de desarrollar una modelización que resuelva el problema Q-robots manejando las variables que aparecen en [2]. Tomemos las variables $x_{t,i,q}$ de manera que $x_{t,i,q} = 1$ si el robot q recoge el pedido i en el instante t. Tenemos entonces que, con las restricciones correctas, estas variables nos indicarán los recorridos que puede realizar cada robot. En [2] podemos encontrar cómo, a partir de estas variables, ya se puede plantear una función de coste que busque minimizar la distancia total recorrida por los robots. El concepto de instante puede dar lugar a confusión. Aclaramos que no está representando instantes de tiempo como pueden ser segundos o similar. El instante t representa el momento en el que se han recogido t objetos. Por lo tanto, el instante $t = 0$ representa el momento inicial, $t = N$ representa el instante en el que se han recogido N objetos, y $t = N + 1$ representa el instante en el que todos los robots han regresado con los objetos al punto inicial.

Llegados a este punto veamos nuestra primera modelización. Para ello, supongamos que tenemos N objetos o pedidos (podemos referirnos a ellos de ambas maneras) y Q robots. Recordemos qué representan las variables que van a aparecer en nuestro modelo:

- Nuestras soluciones van a estar representadas por las variables $x_{t,i,q}$, de manera que $x_{t,i,q} = 1$ si el robot q recoge el pedido i en el instante t.
- Las distancias entre los distintos objetos vienen representadas en la matriz D, de manera que $d_{i,j}$ corresponde a la distancia existente entre el objeto i y el j.

Veamos entonces las restricciones que han de verificar las variables $x_{t,i,q}$.

(1) Los robots deben recoger todos los pedidos. Por lo tanto, es necesario que para cada objeto i uno de los robots haya pasado por la localización del objeto i en algún instante. Esta restricción se puede expresar como

$$\text{Para todo } i \in \{1, ..., N\}\text{: } \sum_{t=0}^{N+1} \sum_{q=1}^{Q} x_{t,i,q} = 1. \tag{5.1}$$

(2) Se debe cumplir que en cada instante cada robot solo puede estar en un lugar.

$$\text{Para todo } q \in \{1, ..., Q\}, t \in \{1, ..., N+1\}\text{: } \sum_{i=0}^{N} x_{t,i,q} = 1. \tag{5.2}$$

(3) Los robots deben comenzar todos en la posición inicial.

$$\text{Para todo } q \in \{1, ..., Q\}\text{: } x_{0,0,q} = 1 \tag{5.3}$$

(4) Los robots deben acabar todos en la posición inicial.

$$\text{Para todo } q \in \{1, ..., Q\}\text{: } x_{N+1,0,q} = 1 \tag{5.4}$$

Una vez que tenemos planteadas estas restricciones, para minimizar la distancia que recorren los robots podríamos tomar como función objetivo

$$\sum_{q=1}^{Q} \sum_{t=1}^{N} \sum_{i=0}^{N} \sum_{j=0}^{N} x_{t-1,i,q} x_{t,j,q} d_{i,j}. \tag{5.5}$$

Sin embargo, tratar de minimizar el tiempo resulta más complicado. Como las variables que debemos manejar en nuestro modelo deben ser binarias, planteamos la siguiente idea para realizar una primera modelización. Esta no es la idea **definitiva** que se aplica en la modelización final, si bien resulta del interés suficiente como para explicarla, no es necesaria para entender la modelización QUBO que finalmente se plantea. Esta idea consiste en introducir un conjunto de variables $y_{s,q}$, de manera que representen lo siguiente: $y_{s,q} = 1$ si el robot q ha completado su ruta recorriendo una distancia menor a t_s, y 0 si no. Esto es, se va a elegir un número S de posibles tiempos $\{t_1, ..., t_S\}$ en los que los robots pueden haber completado sus rutas. Y vamos a tratar de conseguir que todos los robots acaben sus rutas en el menor de los t_j posible. De esta manera se puede tratar de minimizar el tiempo consiguiendo que para cada q, $y_{s,q}$ tome el valor 1 para el menor s posible. Más adelante se proporciona un ejemplo que ayuda a entender el funcionamiento de estas variables $y_{s,q}$.

Una vez explicado lo anterior, veamos cómo se pueden incorporar al modelo. Para ello es necesario elegir unos valores t_s adecuados, de forma que los robots traten de completar sus rutas recorriendo una distancia menor a t_s. Las variables t_s se pueden entender que representan tanto tiempo como distancia, ya que el tiempo que tarde un robot en terminar sus ruta viene únicamente determinado por la distancia que este tenga que recorrer. Para introducir las variables $y_{s,q}$ vamos a seguir unos pasos:

- Primero definiremos el número de distancias intermedias S que vamos a requerir. Intentaremos que el número total de variables no sea desmesurado. Por ejemplo, $S = 10$ o $S = 100$ posibles distancias (esto dependerá del resto de variables Q, N). Veremos más adelante cómo podemos mejorar nuestra solución a pesar de contar con un número S bajo. De hecho, en el ejemplo propuesto tomaremos $S = 2$.
- Tratamos de estimar S distancias (o tiempos) en las que los robots pueden haber acabado sus rutas. S_0 será una cota inferior del tiempo en el que pueden haber acabado los robots, que puede coincidir por ejemplo con la distancia que se ha de recorrer para recoger el objeto que se encuentra más alejado de la posición inicial (ya que es el mínimo tiempo necesario para completar todos los recorridos).

  S_f será una cota superior del tiempo mínimo necesario para realizar una ruta. Puede por ejemplo coincidir con el tiempo necesario para realizar una ruta cualquiera en la que se recojan todos los pedidos.
- Tendremos entonces que el modelo va a tratar de conseguir que todos los robots completen sus caminos recorriendo una distancia lo menor posible.

Planteamos entonces las restricciones que deben satisfacer las variables $y_{s,q}$. Para facilitar la comprensión del modelo vamos a tomar para cada $s \in \{1, ..., S\}$, $t_s := S_0 + s\frac{S_f - S_0}{S}$.

(5) Para cada robot q, $y_{s,q}$ debe tomar el valor 1 en algún momento. Por lo que

$$\text{Para cada } q,\ \sum_{s=0}^{S} y_{s,q} = 1. \tag{5.6}$$

(6) Si se cumple que $y_{s,q} = 1$, entonces el robot q debe haber terminado su ruta recorriendo una distancia menor que t_s. Lo cual conduce a la siguiente restricción:

$$\text{Para cada } s, q\text{: } S_0 + s\frac{S_f - S_0}{S} \geq y_{s,q} \sum_{t=1}^{N} \sum_{i=0}^{N} \sum_{j=0}^{n} x_{t-1,i,q} x_{t,j,q} d_{i,j}. \tag{5.7}$$

En la ecuación anterior podemos evitar la multiplicación por $y_{s,q}$ de la siguiente manera. Tomemos $D_{max} := \sum_{i=0}^{n} \text{máx}_j\{d_{i,j}\}$; es decir, una cota de la distancia máxima que puede tener cualquier camino. Tenemos entonces que la ecuación (5.1) es equivalente a

$$S_0 + s\frac{S_f - S_0}{S} + (1 - y_{s,q})D_{max} \geq \sum_{t=1}^{N}\sum_{i=0}^{N}\sum_{j=0}^{N} x_{t-1,i,q}x_{t,j,q}d_{i,j}. \tag{5.8}$$

Si tomamos $t_s := s_0 + s\frac{s_f - s_0}{S}$ y las variables auxiliares $z_{t,i,j,q} := x_{t-1,i,q}x_{t,j,q}$, podemos transformar la restricción anterior en

$$t_s + (1 - y_{s,q})D_{max} \geq \sum_{t=1}^{N}\sum_{i=0}^{N}\sum_{j=0}^{N} z_{t,i,j,q}d_{i,j}. \tag{5.9}$$

Tenemos ahora que las variables $y_{s,q}$ marcan la distancia que recorre cada robot. Para completar el modelo necesitamos una función de coste que tome el mínimo cuando todas las variables $y_{s,q}$ sean 1 para el menor s posible. Para conseguir esto mediante una combinación lineal de los $y_{s,q}$, necesitamos que, por ejemplo, si tenemos 2 robots, nuestra función a optimizar tome valores menores si ambos robots recorren una distancia k que si uno de los robots recorre una distancia 0 y el otro $k+1$. Para ello necesitamos una expresión de la forma exponencial, por lo que la función de coste puede ser

$$\sum_{s=0}^{S} Q^s \sum_{q=1}^{Q} y_{s,q}. \tag{5.10}$$

En efecto se cumple la condición requerida debido a la desigualdad

$$Q^s \sum_{q=1}^{Q} y_{s,q} \leq Q^s Q = Q^{s+1}. \tag{5.11}$$

Volveremos a aclarar este punto con el ejemplo siguiente. Notar que esta función puede llegar a tomar valores de la forma Q^S, que pueden ser demasiado altos y causar problemas computacionales. Esto restringe el valor máximo que puede tomar S. Aunque parezca restrictivo para nuestra solución manejar un tamaño de S pequeño, no lo es. Esto se debe a que podemos mejorar nuestra solución aplicando lo siguiente. Con el modelo desarrollado hasta ahora, ajustamos que los robots recorran una distancia máxima perteneciente a $[s_0 + s_p(s_f - s_0), s_0 + (s_p + 1)(s_f - s_0)]$ para algún valor $s_p \in \{0, ..., S\}$. Podemos seguir refinando la distancia máxima todo lo que queramos sin más que repetir el algoritmo tomando ahora $S_0 \leftarrow S_0 + s_p\frac{S_f - S_0}{S}$ y $S_f \leftarrow S_0 + (s_p + 1)\frac{S_f - S_0}{S}$.

Ejemplo. Veamos un caso práctico sobre el funcionamiento del algoritmo explicado anteriormente. Supongamos que tenemos las siguientes condiciones: el número de robots de nuestro problema será $Q = 2$ y el número de pedidos será $N = 2$. Los pedidos están localizados en las posiciones $(0, 1)$ y $(1, 1)$. Es claro que la solución que minimiza el tiempo consistirá en que el primer robot recoja el pedido 1 localizado en $(0, 1)$ y el segundo robot recoja el pedido 2 localizado en $(1, 1)$. De esta manera tendremos una distancia recorrida total de $2 + 2\sqrt{2}$ con una distancia máxima recorrida $2\sqrt{2}$. Sin embargo, la solución que minimiza la distancia que recorren los robots consiste en que uno de los robots recoja los dos pedidos, llegando así a una distancia de $2 + \sqrt{2}$, que coincide en este caso con la distancia máxima recorrida. Veamos cómo nuestra modelización asigna un valor menor a la primera solución. Para ello vamos a tomar $S = 2$, $s_0 = 2\sqrt{2}$, $s_f = 2 + \sqrt{2}$. Bajo estas condiciones comparemos las dos posibles estrategias (que uno de los robots recoja todos los pedidos o que cada robot vaya a por uno de los pedidos) a través de la función de coste que hemos propuesto.

- *Caso 1.* Que uno de los robots recoja todos los pedidos. En estas condiciones las variables $x_{t,i,q}$ distintas de 0 serán:

  - Para el primer robot,
  $$1 = x_{0,0,1} = x_{1,1,1} = x_{2,2,1} = x_{3,0,1},$$
  correspondiente con que en el instante 0 esté en la posición inicial, en el instante 1 recoja el primer pedido, en el instante 2 recoja el segundo pedido, y en el instante 3 regrese a la posición inicial.
  - Para el segundo robot, las variables distintas de 0 serán
  $$1 = x_{0,0,2} = x_{1,0,2} = x_{2,0,2} = x_{3,0,2},$$
  correspondiente con que el robot 2 permanezca en la posición inicial en todo instante.

  Tomando $s = 2$, tenemos que las posibles distancias que se pueden tomar serán $\{2\sqrt{2}, \frac{2+3\sqrt{2}}{2}, 2+\sqrt{2}\}$. En estas condiciones tenemos que la disposición de las variables $y_{s,q}$ que minimiza la función de coste cumpliéndose la restricción 7 consistirá en tomar $y_{2,1} = 1$, $y_{0,2} = 1$. Lo anterior se deduce de que el robot 1 recorre una distancia menor o igual a S_f y el robot 2 recorre una distancia menor o igual a S_0. En estas condiciones tenemos que la función de coste toma un valor de $2^0 + 2^2 = 5$.

- *Caso 2.* Cada robot recoge uno de los pedidos. Para este caso tendremos que las variables $x_{t,i,q}$ distintas de 0 serán
  $$1 = x_{0,0,1} = x_{1,1,1} = x_{2,0,1} = x_{3,0,1},$$
  $$1 = x_{0,0,2} = x_{1,2,2} = x_{2,0,2} = x_{3,0,2}.$$
  Para este caso, las variables $y_{s,q}$ distintas de 0 son $1 = y_{1,1} = y_{1,2}$, debido a que ambos robots recorren una distancia menor o igual que $\frac{2+3\sqrt{2}}{2}$, lo cual se puede deducir a partir de las desigualdades $2 \leq \frac{2+3\sqrt{2}}{2}$ y $2\sqrt{2} \leq \frac{2+3\sqrt{2}}{2}$. Bajo estas condiciones, la función de coste tomará como valor $2^1(1+1) = 4$. Tenemos por lo tanto que, en efecto, esta modelización encuentra la mejor de las soluciones.

Tras aplicar este procedimiento una vez, hemos encontrado una solución para la cual la distancia máxima que recorrerá cada robot pertenece al intervalo $\left[2\sqrt{2}, \frac{2+3\sqrt{2}}{2}\right]$. Si hubiéramos tomado $S_0 = 0$, no habríamos encontrado esta solución, ya que el algoritmo concluiría que la distancia máxima que recorre cada robot pertenece al intervalo $\left[\frac{2+\sqrt{2}}{2}, 2+\sqrt{2}\right]$. De estar en esta situación se habría hecho necesario volver a aplicar el algoritmo tomando ahora $S_0 = \frac{2+\sqrt{2}}{2}$. Iterando este proceso en sucesivas ocasiones vamos calculando rutas para las cuales el tiempo que se necesita para recorrerlas se va acotando hasta alcanzar la precisión que se desee.

A pesar de que esta modelización resuelve el problema, tenemos el obstáculo de que en la restricción [7] aparecían coeficientes de grado 2. Esto fue resuelto introduciendo las variables auxiliares $z_{t,i,j,q} := x_{t-1,i,q}x_{t,i,j,q}$, lo cual se realiza de manera sencilla tomando como funciones de penalización polinomios de grado 2. Sin embargo, este cambio supone introducir en nuestra modelización N^3Q variables auxiliares extra. Esto provoca una pérdida de la eficiencia de esta modelización al necesitar un excesivo número de variables y por tanto de cúbits. Llegados a este punto, y como comentamos en la introducción de este capítulo, se hizo necesario desarrollar una mejor modelización del problema TSP y tratar de aplicarla a nuestro problema. Veamos en la siguiente sección esta modelización.

5.2. Modelización del problema TSP

En la documentación encontrada sobre modelizaciones QUBO del problema TSP, se hace referencia fundamentalmente a dos. La primera de ellas es la que se utiliza en [2] para posteriormente resolver su problema. Cuando tratamos de resolver nuestro problema aplicando esta modelización, nos encontramos con que la función de cálculo de las distancias que recorre cada robot es cuadrática. Esto no presenta ningún inconveniente si estamos tratando de resolver el problema TSP, pero tiene mala generalización si necesitamos establecer relaciones entre las distancias que recorre cada robot, como se explica en la sección anterior.

La otra modelización más común del problema TSP, y que aparece por ejemplo en [15], consiste en tomar las variables $x_{i,j,t}$ de manera que $x_{i,j,t} = 1$ si en el instante t el viajero va de la ciudad i a la ciudad j. Bajo esta modelización la función de cálculo de la distancia es lineal, como podemos observar en 5.4. En el *paper* que acabamos de mencionar aplican esta modelización debido a su necesidad de conseguir una función de distancia que sea lineal para así poder establecer relaciones de igualdad con ella. Citan además [10] como otro ejemplo de generalización del problema TSP donde se utiliza esta modelización. La modelización de la que estamos hablando resuelve el problema de tener como función de distancia un polinomio de grado 2, ya que ahora dicha función corresponde con

$$\sum_{i=0}^{N}\sum_{j=0}^{N}\sum_{t=0}^{N+1} d_{i,j}x_{i,j,t}, \tag{5.12}$$

siendo $d_{i,j}$ la distancia entre las ciudades i y j. Sin embargo, esta modelización requiere de N^3 variables, por lo que al tratar de aplicarla a nuestro problema de Q-Robots contaríamos con QN^3 variables. Esto provoca que apenas se mejore la solución que nos apareció en la sección anterior. Tenemos entonces que para encontrar una buena modelización QUBO para nuestro problema, resulta conveniente desarrollar una modelización del problema TSP que requiera menos de N^3 variables. Esta sección presenta la modelización del problema TSP que hemos tenido que desarrollar para tratar de conseguir que la función de cálculo de la distancia sea lineal y que el número de variables que aparezca sea inferior a N^3.

Modelización del problema TSP. Supongamos que estamos resolviendo el problema del viajero para N ciudades. Vamos a tomar el conjunto de variables

$$x_{i,j,r} \text{ con } i,j \in \{0,...,N+1\} \text{ y } r \in \{0,1,2\}. \tag{5.13}$$

El nodo $N+1$ corresponde con el nodo final, que en el problema del viajante que estamos tratando corresponde con el punto inicial (aunque podría representar cualquier otro). Tenemos entonces que el número de variables que vamos a necesitar es $3(N+2)^2$. Veamos que representa cada variable:

- $x_{i,j,0} = 1$. La arista (i,j) no aparece en el recorrido y se llega antes al nodo i que al j.
- $x_{i,j,1} = 1$. La arista (i,j) aparece en el recorrido, por lo que se llega antes al nodo i que al j.
- $x_{i,j,2} = 1$. La arista (i,j) no aparece en el recorrido, y se llega antes al nodo j que al i.

El índice r nos va a permitir dotar al problema de un orden en el cual se recorren las ciudades. Esto es necesario ya que si no, como veremos más adelante, podemos llegar a soluciones que no representen un recorrido. Se darán más detalles sobre esta observación en la restricción 8 (5.24). En la modelización del problema TSP de la que hablamos en la introducción de esta sección y que utilizaba las variables $x_{i,j,t}$, el índice t permite establecer el orden del que estamos hablando. Esta modelización sustituye el índice t por r, permitiendo pasar de N^3 a $3N^2$ variables, lo que disminuye notablemente el número variables utilizadas. Sin embargo, se hace necesario introducir una restricción que garantice que la

solución final representa un ciclo. Veamos entonces las restricciones que deben cumplir las variables $x_{i,j,r}$.

(1) Para cada i, j se debe dar uno y sólo uno de los 3 casos de r, por lo que

$$\text{Para todo } i,j\text{: } \sum_{r=0}^{2} x_{i,j,r} = 1. \tag{5.14}$$

(2) Se debe salir una vez de cada nodo.

$$\text{Para cada } i \in \{0,...,N\}\text{: } \sum_{j=0}^{N+1} x_{i,j,1} = 1. \tag{5.15}$$

(3) Se debe llegar una vez a cada nodo.

$$\text{Para cada } j \in \{1,...,N+1\}\text{: } \sum_{i=0}^{N} x_{i,j,1} = 1. \tag{5.16}$$

(4) Se debe salir de la posición inicial; es decir, el nodo 0 debe alcanzarse antes que cualquier otro.

$$\text{Para todo } j \in \{0,...,N+1\}\text{: } x_{0,j,0} + x_{0,j,1} = 1. \tag{5.17}$$

Notar que esta restricción se puede expresar de manera más sencilla como

$$\text{Para todo } j \in \{0,...,N+1\}\text{: } x_{0,j,2} = 0.$$

(5) Se debe acabar en la posición inicial, por lo que el nodo $N+1$ debe alcanzarse después de cualquier otro.

$$\text{Para todo } i \in \{0,...,N+1\}\text{: } x_{i,N+1,0} + x_{i,N+1,1} = 1. \tag{5.18}$$

Al igual que en el caso anterior, es posible expresar esta restricción de manera más sencilla como

$$\text{Para todo } i \in \{0,...,N+1\}\text{: } x_{i,N+1,2} = 0.$$

(6) En el recorrido no pueden aparecer aristas (i, i). Para ello vamos a fijar lo siguiente:

$$\text{Para todo } i \in \{0,...,N+1\}\text{: } x_{i,i,0} = 1. \tag{5.19}$$

Las restricciones que acabamos de presentar no son suficientes para garantizar que nuestra solución forme un ciclo. Podemos apreciar ésto con el siguiente ejemplo.

Ejemplo: Sea $N = 4$ y tomemos la siguiente lista de aristas

$$(0,1),(1,2),(2,0),(3,4),(4,3)$$

Podemos observar que este conjunto de aristas cumple todas las restricciones anteriores. Sin embargo, no generan una solución factible. Esto se debe a que con este conjunto de aristas no existe manera de recorrer todos los nodos antes de regresar al 0 y, por tanto, no se puede construir un ciclo a partir de ellas. Las siguientes dos restricciones impiden que se genere la situación anterior:

(7) Si el nodo i se alcanza antes que el j, entonces el nodo j se alcanza después que el i, por lo que

$$\text{Para todo } i,j \in \{0,...,N+1\} \text{ tal que } i \neq j\text{: } x_{i,j,2} = 1 - x_{j,i,2}. \tag{5.20}$$

Volviendo al ejemplo anterior, al incluir esta restricción se evita la situación de que en la solución aparezcan las aristas $(3,4)$ y $(4,3)$. Sin embargo, esta restricción no es suficiente para evitar que se genere un ciclo de la forma $(3,4),(4,5),(5,3)$, el cual conduciría a una solución no factible. Para acabar de solucionar el problema es necesaria la siguiente restricción, que al actuar junto con la 7 permite evitar la situación que acabamos de describir.

(8) Si el nodo i se alcanza antes que el nodo j y el nodo j se alcanza antes que el nodo k, entonces el nodo i se debe alcanzar antes que el k. Esta condición va a evitar que el recorrido vuelva a un nodo al que ya había llegado anteriormente. Para introducir esta restricción vamos a calcular directamente una función de penalización que tome el valor 0 en los casos correctos y 1 cuando no se cumpla nuestra condición. Para facilitar la comprensión de la función de penalización, vamos a tomar para i,j,k las siguientes variables auxiliares:

$$a_{i,j} = x_{i,j,0} + x_{i,j,1},\ b_{j,k} = x_{j,k,0} + x_{j,k,1} \text{ y } c_{i,k} = x_{i,k,0} + x_{i,k,1}.$$

Tenemos entonces que si $a_{i,j} = 1$, entonces el nodo i se alcanza antes que el nodo j, y lo mismo ocurre para las variables $b_{j,k}$ y $c_{i,k}$. Para facilitar la notación vamos a fijar i,j,k de manera que solo manejemos las variables a,b,c. Recordemos que a,b,c solo toman los valores 0 o 1. Los casos que conducen a valores de las variables para los cuales llegamos a soluciones no factibles y debemos penalizar son $(a,b,c) = (0,0,1)$ y $(a,b,c) = (1,1,0)$.

- El caso $(0,0,1)$ representa que el nodo i se alcanza antes que el j, el j antes que el k y sin embargo el nodo k antes que el i. Por lo tanto esta situación debe penalizarse.
- El caso $(1,1,0)$ también debe ser evitado, ya que representa que el nodo i se alcanza después que el j, el nodo j después que el k y sin embargo el nodo i se alcanza antes que el k, lo cual conduce a una situación absurda.

Tenemos pues que debemos construir una función de penalización de manera que para $f(a,b,c)$ se cumpla que $f(0,0,1) > 0$, $f(1,1,0) > 0$ y $f(a,b,c) = 0$ para el resto de casos. Una función que cumple estas condiciones es

$$f(a,b,c) := ab - ac - bc + c^2 \tag{5.21}$$

Volviendo a las variables $x_{i,j,r}$, tendremos que si

$$\begin{aligned} g_{i,j,k} := (x_{i,j,0} + x_{i,j,1})(x_{j,k,0} + x_{j,k,1}) - (x_{i,j,0} + x_{i,j,1})(x_{i,k,0} + x_{i,k,1}) - \\ (x_{j,k,0} + x_{j,k,1})(x_{i,k,0} + x_{i,k,1}) + (x_{i,k,0} + x_{i,k,1})(x_{i,k,0} + x_{i,k,1}), \end{aligned} \tag{5.22}$$

entonces, añadiendo a la función de coste

$$\lambda \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{N} g_{i,j,k},$$

las mejores soluciones serán aquellas que cumplan esta restricción. **Nota:** Para que esta restricción se aplique de manera correcta, es necesario que se cumpla la restricción 1 (5.14). De no ser así, puede ocurrir que las variables $a_{i,j}$ tomen valores mayores que 1, provocando que en la función de penalización aparezcan valores negativos para situaciones no deseadas. Debido a esto, el coeficiente de Lagrange de la restricción 1 y el coeficiente de Lagrange de la restricción

8 deben estar relacionados. Podemos evitar esta relación entre los coeficientes de Lagrange de las restricciones 1 (5.14) y 8 (5.2) con la siguiente observación. Como acabamos de comentar, si no se cumpliese la restricción 1, los valores de $a_{i,j}$ podrían no pertenecer a $\{0,1\}$, por lo que la función de coste anterior podría tomar valores negativos y podría ocurrir que una solución que no cumpliese la restricción 1 tuviera un valor que el mejor óptimo. Debemos entonces tratar de garantizar que $a_{i,j} \in \{0,1\}$. Para ello podemos tomar $a_{i,j} = x_{j,i,2}$, lo cual se deduce de la siguiente igualdad:

$$x_{i,j,0} + x_{i,j,1} = 1 - x_{i,j,2} = 1 - (1 - x_{j,i,2}) = x_{j,i,2}. \tag{5.23}$$

Como ahora las variables $x_{j,i,2}$ cumplen siempre que $x_{j,i,2} \in \{0,1\}$, tenemos lo que buscamos. Tenemos entonces que la expresión de la función de penalización para la restricción 8 se corresponde con

$$\lambda_8 \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N}(x_{j,i,2}x_{k,j,2} - x_{j,i,2}x_{k,i,2} - x_{k,j,2}x_{k,i,2} + x_{k,i,2}) \tag{5.24}$$

Con esta modelización, la función a minimizar correspondiente a la distancia recorrida es

$$\sum_{i=0}^{N+1}\sum_{j=0}^{N+1} d_{i,j}x_{i,j,1}. \tag{5.25}$$

Una vez que tenemos la modelización, debemos transformar las restricciones de igualdad en funciones de penalización. Para ello es necesario expresarlas en la forma $p(x_1, ..., x_n) - a = 0$ y elevar el término de la izquierda al cuadrado. Una vez que tengamos esto, el siguiente paso es construir la función de coste e introducirla en el software de D-Wave. Calculemos entonces las funciones de penalización correspondientes a cada restricción.

(1) De la restricción 1 tomamos como función de penalización

$$\sum_{i=0}^{N+1}\sum_{j=0}^{N+1}\left(\sum_{r_1=0}^{2}\sum_{r_2=0}^{2}\lambda_1 x_{i,j,r_1}x_{i,j,r_2} + \sum_{r=0}^{2}-2\lambda_1 x_{i,j,r}\right).$$

(2) De la restricción 2 tomamos como función de penalización

$$\sum_{i=0}^{N}\left(\sum_{j_1=0}^{N+1}\sum_{j_2=0}^{N+1}\lambda_2 x_{i,j_1,1}x_{i,j_2,1} + \sum_{j=0}^{N}-2\lambda_2 x_{i,j,1}\right).$$

(3) De la restricción 3 tomamos como función de penalización

$$\sum_{j=1}^{N+1}\left(\sum_{i_1=1}^{N}\sum_{i_2=1}^{N}\lambda_3 x_{i_1,j,1}x_{i_2,j,1} + \sum_{i=1}^{N}-2\lambda_3 x_{i,j,1}\right).$$

(4) De la restricción 4 tomamos como función de penalización

$$\sum_{j=0}^{N+1}\lambda_4 x_{0,j,2}.$$

(5) De la restricción 5 tomamos como función de penalización

$$\sum_{i=0}^{N+1}\lambda_5 x_{i,N+1,2}.$$

(6) De la restricción 6 tomamos como función de penalización

$$\sum_{i=0}^{N+1} -\lambda_6 x_{i,i}.$$

(7) De la restricción 7 tomamos como función de penalización

$$\sum_{i\neq j}(2\lambda_7 x_{i,j,2}x_{j,i,2} - \lambda_7 x_{i,j,2} - \lambda_7 x_{j,i,2}).$$

(8) La restricción 8 ya estaba planteada como función de penalización, por lo que tomamos

$$\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N} \lambda_8(x_{j,i,2}x_{k,j,2} - x_{j,i,2}x_{k,i,2} - x_{k,j,2}x_{k,i,2} + x_{k,i,2}).$$

Para que la restricción 8 no pierda sentido, debemos retirar de la suma los casos en los que $i = j$ o $j = k$ o $i = k$.

5.2.1. Coeficientes de Lagrange en el problema TSP

Como mencionamos anteriormente, uno de los aspectos más importantes para realizar una correcta modelización consiste en el ajuste de los coeficientes de Lagrange. En esta sección vamos a tratar de estudiar cómo afecta cada coeficiente de Lagrange a la función de coste global. Sin embargo, en la práctica este estudio no resulta suficiente. Esto se debe a que, a pesar de que tengamos garantizado que en nuestra función el mínimo global coincida con el mínimo de la función que estamos tratando de resolver, para un correcto funcionamiento del algoritmo *Quantum Annealing* se hace necesario realizar distintas simulaciones de nuestro problema para acabar de ajustar estos coeficientes. Aun así, vamos a proporcionar algunos datos que deben cumplir los coeficientes de Lagrange que ajustamos. Veamos qué valores deben tener los coeficientes de Lagrange de las restricciones con las que estamos trabajando.

- *Primera restricción.* Tenemos que si alguna más de las variables $x_{i,j,r}$ toma el valor 1, esto no influirá de manera positiva en nuestra función a optimizar. Sin embargo, sí podemos generar una mejora en la función objetivo si todas nuestras variables tomaran el valor 0. De esta manera, la mejora conseguida coincidiría con la distancia existente entre el objeto i y el j. Tenemos entonces que, como $p_1 = 0$, se deberá cumplir que

  $$\lambda_{1,i,j} > dist_{i,j}.$$

  Como comentamos en la introducción, algunas de las restricciones están enlazadas entre ellas. Tenemos, por ejemplo, que aunque no influye de manera negativa en la función objetivo que alguna más de las variables $x_{i,j,r}$ tome el valor 1, sí que influye en la restricción 7. Esto se debe a que alterar estos valores puede conducir a una mejora en el resultado de la función de penalización de la restricción 7.

- *Segunda restricción.* Que en nuestro recorrido deje de cumplirse que salgamos de uno de los nodos tendrá el beneficio en la función objetivo de que puede tomar valores sin tener en cuenta el coste de una de las aristas que salen de dicho nodo. De esta manera, el beneficio máximo de incumplir la restricción 2 corresponde con la mayor de las distancias entre el nodo y el resto de puntos. Si estas distancias las tenemos almacenadas en la matriz *dist*, tendremos que $\lambda_2 p_2 > \text{máx}_{j=1,\dots,N}\, dist[0,j]$. Como $p_2 = 1$, podemos tomar

  $$\lambda_2 > \max_{j=1,\dots,N} dist[0,j].$$

- *Tercera restricción.* Podemos aplicar el razonamiento de la restricción anterior para deducir la ventaja en la función objetivo de no llegar a un nodo. Por lo tanto,

$$\lambda_{3,j} > \max_{i \in \{0,\ldots,N+1\}} dist_{i,j}.$$

- *Cuarta restricción.* Que no se salga del nodo inicial permite ahorrar en la función objetivo el máximo de las distancias de las aristas que salen del nodo 0. Tenemos entonces que podemos tomar

$$\lambda_{2,i} > \max_{j \in \{0,1,\ldots,N+1\}} dist_{i,j}.$$

- *Quinta restricción.* Que un robot no acabe en la posición final permite no incluir en la función a minimizar una de las aristas que terminan en $N+1$. Como $p_3 = 1$, tenemos que

$$\lambda_5 > \max_{j \in \{1,\ldots,N\}} dist_{j,N+1}.$$

- *Sexta restricción.* La ventaja en la función objetivo de introducir aristas $x_{i,i,1,q}$ hace que no sea necesario ni volver a la arista i ni salir de la arista j. Por lo tanto, tenemos que

$$\lambda_{6,i} > \max\left(\max_{j \in \{0,\ldots,N+1\}} dist_{i,j}, \max_{j \in \{0,\ldots,N+1\}} dist_{j,i}\right).$$

- *Séptima restricción.* Si hemos alcanzado el i antes que el j, entonces incumpliendo esta restricción conseguimos que ir de j a cualquier otro punto sea mejor que volver de nuevo a i. Tenemos entonces que

$$\lambda_{7,i,j} > \frac{1}{dist_{i,j}} \max_k dist_{i,k}, distj, k.$$

- *Octava restricción.* Podemos aplicar el mismo razonamiento de la restricción anterior y concluir que

$$\lambda_{8,i,j} > \frac{1}{dist_{i,k}} \max_k dist_{i,k}, distj, k.$$

Una vez que tenemos una modelización del problema del viajante que necesita menos de N^3 variables, vamos a tratar de generalizarla al problema de los Q-Robots.

5.3. Modelización del problema de los Q-Robots

En esta sección vamos a presentar una modelización del problema de los Q-Robots a partir de la modelización del problema TSP que acabamos de desarrollar. Recordar que lo que denominamos Q-Robots aparece en la literatura como VRP (*Vehicle Routing Problem*). Sin embargo dicho problema se suele asociar a la existencia de unas restricciones de peso en vez de a minimizar el tiempo que tardan los vehículos en completar sus rutas. Conseguir esta modelización a partir de la del problema TSP anterior no es una tarea directa. Ésto se debe a que requiere de un paso delicado para cuadrar la restricción número 10. Daremos más detalles de ésto más adelante en esta sección. En un primer momento planteamos otra modelización similar a esta, aparentemente más directa y que necesita menos variables. Sin embargo, dicha modelización presenta un gran obstáculo al tratar de aplicar la décima restricción, por lo que no termina de ser correcta. Una vez explicado lo anterior, comencemos la modelización.

Para este problema Q-Robots tendremos en cuenta que N es el número de objetos localizados en el almacén y Q es el número de robots. Presentamos primero las variables que van a formar el problema. Tomemos el conjunto de variables

$$x_{i,j,r,q} \text{ con } i,j \in \{0,...,N+1\}, r \in \{0,1,2,3,4\} \text{ y } q \in \{1,...,Q\}$$

Las variables i, j hacen referencia a los objetos que tienen que ser recogidos por los robots (también pueden ser llamados nodos o pedidos) y la variable q hace referencia al robot al que nos referimos. Los nodos 0 y $N+1$ corresponden con los puntos inicial y final. Los valores $d_{i,j}$ con $i,j \in \{0,...,N+1\}$ corresponden a la distancia entre el nodo i y el j. Veamos bajo estas condiciones la interpretación de cada variable:

- $x_{i,j,0,q} = 1$. El robot q recoge los objetos i y j, no recorre la arista (i,j) y llega antes al nodo i que al j.
- $x_{i,j,1,q} = 1$. El robot q recoge los objetos i y j, recorre la arista (i,j) (es decir, una vez que recoge el objeto i el siguiente objeto que recoge es el j) y por tanto se llega antes al objeto i que al j.
- $x_{i,j,2,q} = 1$. El robot q recoge los objetos i y j y llega antes al objeto j que al i.
- $x_{i,j,3,q} = 1$. El robot q no recoge los objetos i y j, y se llega antes al nodo i que al j. Observar que $x_{i,j,3,q}$ puede tomar el valor 1 si el robot q recoge el objeto i y no el j.
- $x_{i,j,4,q} = 1$. El robot q no recoge los objetos i y j, y se llega antes al objeto j que al i.

Nota: Aunque no haya ningún robot que pase por los objetos i y j, es necesario para la modelización establecer un orden entre ellos. Esta restricción no hace que la modelización pierda sentido, ya que podemos suponer que si los robots están ordenados siguiendo el orden de $\{1,...,Q\}$, entonces i se alcanzará antes que j si el robot que pasa por el nodo i tiene un número menor que el robot que pasa por el nodo j.
De esta manera tendremos que, aunque no haya un robot que pase por los nodos i,j,k, se tienen que seguir cumpliendo las restricciones que teníamos para el problema TSP normal. Esto se debe a que, si tenemos que el objeto i se alcanza antes que el objeto j y el j se alcanza antes que el k, entonces obligatoriamente el objeto i se debe alcanzar antes que el objeto k, sea porque hay un robot que está recorriendo ambas aristas correctamente o porque el número del robot que pasa por i es menor que el numero del robot que pasa por j y, al cumplirse también para j y k, obligatoriamente el robot que pase por el nodo i tendrá que tener un número menor que el robot que pase por k. Esto es necesario para poder aplicar correctamente la restricción 10.
Observación. Esta modelización podría mejorarse introduciendo un elemento más sobre el índice r y manejando solamente los índices i,j tales que $i < j$. De esta manera la modelización pasaría de $5N^2Q$ a $3N^2Q$ variables. Sin embargo, para facilitar la comprensión del modelo preferimos presentar la modelización con $5N^2Q$ variables.

Restricciones. Veamos las restricciones que se deben cumplir una vez que tenemos las variables que van a formar el modelo. Como cada restricción de igualdad debe transformarse en una función de penalización, iremos calculando las funciones de penalización correspondientes según presentemos las distintas restricciones. Para cada restricción se indica el valor de restricción asociado. Este valor corresponde con el mínimo que toma la función de penalización y se alcanza al cumplirse la restricción. Constituye una manera muy eficiente de conocer si en una solución propuesta se está cumpliendo la restricción correspondiente.

(1) Para cada i, j, q se debe cumplir una y solo una de las posibilidades para r, por lo que

$$\text{Para todo } i,j,q\text{: } \sum_{r=0}^{4} x_{i,j,r,q} = 1, \tag{5.26}$$

de donde se deduce la función de penalización

$$\sum_{i=0}^{N+1}\sum_{j=0}^{N+1}\sum_{q=1}^{Q}\sum_{r=0}^{4} -2\lambda_1 x_{i,j,r,q} + \sum_{i=0}^{N+1}\sum_{j=0}^{N+1}\sum_{q=1}^{Q}\sum_{r_1=0}^{4}\sum_{r_2=0}^{4} \lambda_1 x_{i,j,r_1,q}x_{i,j,r_2,q}$$

Esta restricción tiene un valor asociado de $-\lambda_1(N+2)^2Q$.

(2) Cada robot tiene que cumplir que salga de la posición inicial. Para ello vamos a imponer que

$$\text{Para todo } q\text{: } \sum_{j=1}^{N+1} x_{0,j,1,q} = 1, \tag{5.27}$$

de donde se deduce la siguiente función de penalización:

$$\sum_{q=1}^{Q}\sum_{j=1}^{N+1} -2\lambda_2 x_{0,j,1,q} + \sum_{q=1}^{Q}\sum_{j_1}^{N+1}\sum_{j_2=1}^{N+1} \lambda_2 x_{0,j_1,1,q}x_{0,j_2,1,q}.$$

Ningún robot puede volver a la posición inicial desde un objeto, por lo que

$$\text{Para todo } q\text{: } \sum_{i=0}^{N+1} x_{i,0,1,q} = 0, \tag{5.28}$$

obteniendo la función de penalización

$$\sum_{q=1}^{Q}\sum_{i=0}^{N+1} \lambda_{2_1} x_{i,0,1,q},$$

cuyo valor de restricción asociado es $-\lambda_2 Q$.

(3) Todo robot debe acabar en la posición final. Para ello se debe cumplir que

$$\text{Para todo } q\text{: } \sum_{i=0}^{N} x_{i,N+1,1,q} = 1, \tag{5.29}$$

que conduce a la función de penalización

$$\sum_{q=1}^{Q}\sum_{i=0}^{N} -2\lambda_3 x_{i,N+1,1,q} + \sum_{q=1}^{Q}\sum_{i_1=0}^{N}\sum_{i_2=0}^{N} \lambda_3 x_{i_1,N+1,1,q}x_{i_2,N+1,1,q},$$

con valor de restricción asociado $-\lambda_3 Q$.

Ningún robot puede salir de la posición final. Tenemos entonces que

$$\text{Para todo } q\text{: } \sum_{j=0}^{N+1} x_{N+1,j,1,q} = 0. \tag{5.30}$$

La función de penalización asociada a esta restricción es

$$\sum_{q=1}^{Q}\sum_{j=0}^{N+1}\lambda_{3_1}x_{N+1,j,1,q},$$

con valor de restricción asociado 0.

Los robots que no recorran ningún camino cumplirán todas las restricciones al tomar

$$x_{0,N+1,1,q}=1.$$

(4) Se debe salir una vez y solo una vez de cada objeto, luego

$$\text{Para cada } i\in\{1,...,N\}\text{: }\sum_{q=1}^{Q}\sum_{j=1}^{N+1}x_{i,j,1,q}=1. \tag{5.31}$$

Tenemos entonces la función de penalización

$$\sum_{i=1}^{N}\sum_{q=1}^{Q}\sum_{j=1}^{N+1}-2\lambda_4x_{i,j,1,q}+\sum_{i=1}^{N}\sum_{q_1=1}^{Q}\sum_{j_1=1}^{N+1}\sum_{q_2=1}^{Q}\sum_{j_2=1}^{N+1}\lambda_4x_{i,j_1,1,q_1}x_{i,j_2,1,q_2},$$

con valor de restricción asociado $-\lambda_4N$.

(5) Se debe llegar una y solo una vez a cada objeto, luego

$$\text{Para cada } j\in\{1,...,N\}\text{: }\sum_{q=1}^{Q}\sum_{i=0}^{N}x_{i,j,1,q}=1. \tag{5.32}$$

La función de penalización asociada a esta restricción es

$$\sum_{j=1}^{N}\sum_{q=1}^{Q}\sum_{i=0}^{N}-2\lambda_5x_{i,j,1,q}+\sum_{j=1}^{N}\sum_{q_1=1}^{Q}\sum_{i_1=0}^{N}\sum_{q_2=1}^{Q}\sum_{i_2=0}^{N}\lambda_5x_{i_1,j,1,q_1}x_{i_2,j,1,q_2},$$

con valor de restricción asociado $-\lambda_5N$.

(6) En el recorrido no pueden aparecer aristas (i,i) para ningún i y para todo q.

$$\text{Para todo } q \text{ y para todo } i\in\{0,...,N+1\}\text{: } x_{i,i,1,q}=0, \tag{5.33}$$

lo que corresponde con la función de penalización

$$\sum_{q=1}^{Q}\sum_{i=0}^{N+1}\lambda_6x_{i,i,1,q},$$

cuyo valor de restricción asociado es 0.

(7) Si al nodo i se llega antes que al j para algún robot, esto debe cumplirse para todos los robots, por lo que introduciendo las variables auxiliares $a_{i,j}$ tenemos que

$$\text{Para todo } i,j\in\{1,...,N\}\text{: }\sum_{q=1}^{Q}x_{i,j,0,q}+x_{i,j,1,q}+x_{i,j,3,q}=a_{i,j}Q. \tag{5.34}$$

Esto quiere decir que o para todos los robots tenemos que se llega antes al objeto i que al objeto j y por lo tanto la suma anterior toma el valor Q y $a_{i,j}=1$, o para todos los robots se

llega antes al objeto j que al objeto i y esa suma toma el valor 0, por lo que $a_{i,j} = 0$.

De esta restricción tomamos como función de penalización

$$\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{q=1}^{Q} -2\lambda_7 Q(x_{i,j,0,q} + x_{i,j,1,q} + x_{i,j,3,q})a_{i,j} +$$

$$+\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{q_1=1}^{Q}\sum_{q_2=1}^{Q} \lambda_7(x_{i,j,0,q_1} + x_{i,j,1,q_1} + x_{i,j,3,q_1})(x_{i,j,0,q_2} + x_{i,j,1,q_2} + x_{i,j,3,q_2}) +$$

$$+\sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_7 a_{i,j}a_{i,j}Q^2,$$

con valor de restricción asociado 0.

(8) Si el robot q llega al nodo j, entonces el robot q debe salir del nodo j. Para ello imponemos la restricción

$$\text{Para } i \in \{0, ..., N\},\ j \in \{1, ..., N\} \text{ y } q \in \{1, ..., Q\}\text{: } x_{i,j,1,q}(1 - \sum_{k=1}^{N+1} x_{j,k,1,q}) = 0. \tag{5.35}$$

Notar que aunque esta restricción parezca cuadrática, al tener una restricción igualada a 0 y no tomar dicha restricción valores negativos, debido a que la restricción 4 obliga a que dicho sumatorio tome los valores 0 o 1, no es necesario elevarla al cuadrado para introducirla en la función de coste. **Sin embargo**, para que en efecto se cumplan ambas restricciones, el coeficiente de Lagrange de la restricción 4 debe ser mucho mayor que el de esta restricción. De no ser así, podría darse el caso de que, al tomarse valores negativos en esta restricción, dichos valores compensaran que no se cumpla la restricción 4.
Por lo tanto, la penalización correspondiente es

$$\sum_{i=0}^{N}\sum_{j=1}^{N}\sum_{q=1}^{Q} \lambda_8 x_{i,j,1,q} + \sum_{i=0}^{N}\sum_{j=1}^{N}\sum_{q=1}^{Q}\sum_{k=1}^{N+1} -\lambda_8 x_{i,j,1,q}x_{j,k,1,q},$$

con valor de restricción asociado 0.

Impongamos ahora las condiciones que hacen que los robots, en efecto, recorran un tour.

(9) Se tiene que cumplir que o se pasa antes por el nodo i que por el j o se llega antes al nodo j que al i. Por lo tanto, se debe verificar que para todo $i, j \in \{1, ..., N\}$ y $q \in \{1, ..., Q\}$:

$$x_{i,j,0,q} + x_{i,j,1,q} + x_{i,j,3,q} = 1 - (x_{j,i,0,q} + x_{j,i,1,q} + x_{j,i,3,q}). \tag{5.36}$$

Esta restricción se puede añadir a la función de coste directamente como

$$\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{q=1}^{Q} \lambda_9(x_{i,j,0,q} + x_{i,j,1,q} + x_{i,j,3,q})(x_{j,i,0,q} + x_{j,i,1,q} + x_{j,i,3,q}),$$

con valor de restricción asociado 0.

(10) Si el nodo i se alcanza antes que el j y el nodo j se alcanza antes que el k, entonces el nodo i debe alcanzarse antes que el k. Esta condición va a impedir que se vuelva a un nodo por el que ya se ha pasado y por lo tanto se forme un ciclo.

Para introducir esta restricción vamos a calcular directamente una función de penalización que valga 0 en los casos correctos y 1 en los que no lo son. Para facilitar la comprensión de la función de penalización vamos a tomar, para i, j, k, q, las siguientes variables:

$a_{i,j} := x_{i,j,0,1}+x_{i,j,1,1}+x_{i,j,3,1}$, $b_{j,k} := x_{j,k,0,1}+x_{j,k,1,1}+x_{j,k,3,1}$ y $c_{i,k} := x_{i,k,0,1}+x_{i,k,1,1}+x_{i,k,3,1}$.

Por tanto, $a_{i,j} = 1$ significa que el nodo i se alcanza antes que el nodo j y lo mismo con b y c.
Nota: Debido a la séptima restricción, podemos tomar como referencia cualquiera de los robots. En este caso hemos tomado como referencia el robot 1.
De esta manera, fijado i, j, k, tenemos las 3 variables a, b, c. Recordemos que a, b, c solo toman los valores 0 o 1.
Los casos que conducen a valores de las variables para los cuales pueden formarse ciclos y que debemos descartar son $(a, b, c) = (0, 0, 1)$ y $(a, b, c) = (1, 1, 0)$.

En el caso $(0, 0, 1)$ tendríamos que el nodo j se alcanza después del i, el k después del j, y sin embargo el nodo k antes que el i, lo cual es absurdo. El caso $(1, 1, 0)$ tampoco se puede dar, ya que en él se alcanza i antes que j y j antes que k, por lo que no puede darse que alcancemos también k antes que i. Tenemos pues que debemos construir una función de penalización de manera que para $f(a, b, c)$ se cumpla que $f(0, 0, 1) > 0$, $f(1, 1, 0) > 0$ y $f(a, b, c) = 0$ para el resto de casos. Una función que cumple estas condiciones es

$$f(a, b, c) := ab - ac - bc + c^2.$$

Por lo tanto, volviendo a los subíndices i, j, k tendremos que si

$$g_{i,j,k} := a_{i,j}b_{j,k} - a_{i,j}c_{i,k} - b_{j,k}c_{i,k} + c_{i,k}^2,$$

entonces, añadiendo a la función de coste

$$\lambda \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{N} g_{i,j,k}, \tag{5.37}$$

tendremos que las mejores soluciones serán aquellas que cumplan esta restricción.

En este caso, el valor de restricción asociado es 0.

Llegados a este punto, tenemos que la función para calcular la distancia que recorre cada robot se corresponde con

$$\sum_{i=0}^{N+1} \sum_{j=0}^{N+1} d_{i,j}x_{i,j,1,q}.$$

Debemos ahora desarrollar una función de manera que tratemos de minimizar la distancia máxima que recorren los robots. Veamos tres maneras de conseguir esto en las siguientes subsecciones. La mejor modelización que hemos encontrado para resolver nuestro problema coincide con la tercera de las tres propuestas.

5.3.1. Primera minimización del tiempo mediante las variables $y_{s,q}$.

La primera de estas tres formas de tratar de minimizar el tiempo que tardan los robots coincide con la desarrollada en la sección 5.1. Al tener ahora una expresión lineal para la fórmula de la distancia, nos encontramos con que se vuelve practicable desde el punto de vista de la programación. Para

esta modelización vamos a tomar, con la notación de la sección 5.1, $S = 1$ y tratar de minimizar la distancia máxima que recorren los robots a partir de las variables y_q. Como en la mencionada sección ya entramos en detalle sobre el funcionamiento de esta modelización, simplemente recordaremos qué representan las variables y_q y las restricciones que corresponderían para este caso.

Variables y_q. Vamos a tratar ahora de calcular la función a minimizar correspondiente. Para conseguir esto, lo vamos a realizar de la siguiente manera:

- Primero estimamos las distancias S_0 y S_f, correspondientes a cotas de las distancias máximas y mínimas para las cuales todos los robots pueden haber acabado sus rutas. S_0 puede coincidir, por ejemplo, con la distancia que se ha de recorrer para recoger el objeto que se encuentra más alejado de la posición inicial. S_f coincidirá con la distancia que hay que recorrer para realizar una ruta cualquiera de manera que se recojan todos los pedidos.
- Tomamos $S_m = \frac{S_f+S_0}{2}$ y vamos a comprobar con nuestro modelo si los robots pueden recoger todos los pedidos recorriendo todos ellos una distancia menor que S_m. Si esto es afirmativo, volveremos a aplicar el procedimiento tomando ahora $S_f = S_m$ y $S_0 = S_0$. Si la respuesta fuese negativa, volveríamos a repetir el procedimiento tomando $S_f = S_f$ y $S_0 = S_m$. De esta manera, repitiendo el algoritmo tantas veces como se requiera, podremos conseguir la precisión deseada sobre la mejor manera de organizar los robots y los viajes que deberán hacer.

Para conseguir esto introducimos las variables y_q. Tendremos que $y_q = 0$ si el robot q puede haber acabado el recorrido antes de la distancia intermedia que estemos comparando, y $y_q = 1$ si no.

(11) Sea $D_{max} := \sum_{i=0}^{n} \text{máx}_j\{d_{i,j}\}$; es decir, una cota de la distancia máxima que puede tener cualquier camino. Tendremos entonces la restricción de desigualdad

$$\text{Para cada } q\text{: } \sum_{i=0}^{N+1}\sum_{j=0}^{N+1} d_{i,j}x_{i,j,1,q} \leq S_m + y_q D_{max}. \tag{5.38}$$

Transformamos la desigualdad anterior en igualdad añadiendo las variables $b_{i,q}$ con $i \in \{0, ..., b_{max}\}$ y $q \in \{1, ..., Q\}$, donde estamos tomando $b_{max} := \lfloor log_2(D_{max}) \rfloor + 1$. De esta manera, la ecuación anterior se transforma en

$$\text{Para cada } q\text{: } \sum_{i=0}^{N+1}\sum_{j=0}^{N+1} d_{i,j}x_{i,j,1,q} + \sum_{h=0}^{h_{max}} 2^h b_{h,q} = S_m + y_q D_{max}, \tag{5.39}$$

donde podemos tomar la función de penalización

$$\sum_{i_1=0}^{N+1}\sum_{j_1=0}^{N+1}\sum_{i_2=0}^{N+1}\sum_{j_2=0}^{N+1} d_{i_1,j_1}d_{i_2,j_2}x_{i_1,j_1,1,q}x_{i_2,j_2,1,q} + \sum_{i=0}^{N+1}\sum_{j=0}^{N+1}\sum_{h=0}^{h_{max}} 2\cdot 2^h b_{h,q}x_{i,j,1,q}d_{i,j} +$$

$$+ \sum_{i=0}^{N+1}\sum_{j=0}^{N+1} -2\cdot x_{i,j,1,q}d_{i,j}(S_m + y_q D_{max}) + \sum_{h=0}^{h_{max}} -2\cdot 2^h b_{h,q}(S_m + y_q D_{max}) +$$

$$+ \sum_{h_1=0}^{h_{max}}\sum_{h_2=0}^{h_{max}} 2^{h_1}b_{h_1,q}2^{h_2}b_{h_2,q} + 2S_m y_q D_{max} + y_q D_{max}^2$$

Esta restricción tiene un valor asociado de $-\lambda_{11}QS_m$.

La función correspondiente a conocer si todos los robots pueden haber acabado sus recorridos en menos tiempo que S_m será

$$\sum_{q=1}^{Q} y_q.$$

Esta función tomará el valor 0 cuando todos los robots pueden haber completado sus recorridos recorriendo una distancia menor que S_m, y tomará valores ≥ 1 si alguno de los robots recorre una distancia mayor que S_m.

Para esta modelización necesitaremos $5N^2Q$ variables $x_{i,j,r,q}$, Q variables y_q, N^2 variables auxiliares requeridas por la restricción 7 y $Qlog_2(D_{max})$ variables de holgura necesarias para transformar la última restricción de desigualdad en igualdad (D_{max} es una cota de la distancia máxima que hay que cubrir). Tenemos entonces que el **número total de variables necesarias** es

$$(5Q+1)N^2 + Q(log_2(D_{max})+1) \approx 6QN^2. \tag{5.40}$$

Esta modelización que acabamos de presentar fue utilizada como un primer modelo que trataba de minimizar el tiempo que tardaban los robots en completar sus recorridos.

5.3.2. Minimización de la distancia máxima que recorren los robots

La modelización de la que acabamos de hablar permite, mediante un proceso recursivo, ir calculando rutas de los robots que van mejorando el tiempo máximo que tardan estos en realizar sus viajes. Sin embargo, las llamadas a un ordenador cuántico resultan costosas, haciendo que la modelización anterior, si bien tiene interés desde un punto de vista teórico, no acabe de resolver nuestro problema de manera eficiente. Tratemos de construir una modelización que nos permita resolver el problema de minimizar el tiempo que tardan los robots en completar todos los viajes con una sola simulación del algoritmo *Quantum Annealing*. Para ello vamos a disponer de las mismas variables $x_{i,j,r,q}$ que aplicamos en el caso anterior, y volvemos a aplicar las restricciones [1] a [10]. Una vez que tenemos construidas las variables $x_{i,j,r,q}$ que permiten trazar las rutas que van a recorrer los robots, vamos a tratar de encontrar una función que minimice el máximo de las distancias que recorren los mismos. Para ello vamos a llevar a cabo los dos siguientes pasos:

(11) Para este paso vamos a introducir las variables auxiliares p_{q_1,q_2}, cuya idea es tratar de conseguir que $p_{q_1,q_2} = 1$ si el robot q_1 recorre más distancia que el robot q_2, y $p_{q_1,q_2} = 0$ en caso contrario. Para ello vamos a añadir como penalización ara cada $q_1, q_2 \in \{1, ..., Q\}$:

$$\lambda_{11}(1-2p_{q_1,q_2})\left(\sum_{i=0}^{N+1}\sum_{j=0}^{N+1} d_{i,j}x_{i,j,1,q_1} - \sum_{i=0}^{N+1}\sum_{j=0}^{N+1} d_{i,j}x_{i,j,1,q_2}\right). \tag{5.41}$$

De esta manera, cuando el robot q_i recorra más distancia que el robot q_j el mínimo se alcanzará cuando $(1-2p_{q_i,q_j}) = -1$ y por lo tanto $p_{q_i,q_j} = 1$.

(12) Una vez que tenemos las variables p_{q_1,q_2} podemos construir las variables z_q de manera que $z_q = 1$ si el robot q recorre más distancia que todos los demás. Para ello introducimos los coeficientes de penalización

$$\text{Para cada } q\text{: } \lambda_{12}\left(2Q - 2\left(\sum_{q_1=1}^{Q} p_{q,q_1} + z_q\right)\right)^2. \tag{5.42}$$

De esta manera, existirá q_i tal que $p_{q_i,q_j} = 1$ para todo q_j, por lo que para ese q_i, $z_{q_i} = 0$, mientras que para el resto de los q_j, $z_{q_j} = 1$.

En estas condiciones tendremos que existirá un único z_{q_i} tal que $z_{q_i} = 0$, correspondiente con aquel que recorra mayor distancia que todos los demás. Por lo tanto, la función a minimizar se correspondería con

$$\sum_{q=1}^{Q} \sum_{i=0}^{N+1} \sum_{j=0}^{N+1} d_{i,j}(1 - z_q) x_{i,j,1,q}. \tag{5.43}$$

Esta modelización permite minimizar directamente el tiempo que tardan los robots en completar su camino. Sin embargo, ajustar los coeficientes de Lagrange de esta modelización resulta muy complicado, por lo que la simulación mediante *Quantum Annealing* se vuelve ineficiente. Veamos por lo tanto una modelización que mejora a esta última, permitiendo obtener una buena solución de nuestro problema.

5.3.3. Modelización final

Una vez que contábamos con la modelización anterior, que trataba de calcular cuál de los Q robots recorría mayor distancia, surgió la idea de obligar a todos los robots a que recorriesen menos distancia que el primero, y entonces minimizar la distancia que recorre el primer robot. En efecto, esto puede modelizarse de manera sencilla sin más que introducir una restricción que fuerce a que todos los robots recorran menos distancia que la recorrida por el primer robot. Como en el resto de esta sección, las variables que estamos manejando se corresponden con las introducidas al inicio de la sección 5.3.

- Esta restricción de la que acabamos de hablar puede expresarse como

$$\text{Para todo } q \in \{2, ..., Q\}\text{: } \sum_{i=0}^{N+1} \sum_{j=0}^{N+1} d_{i,j} x_{i,j,1,q} \leq \sum_{i=0}^{N+1} \sum_{j=0}^{N+1} d_{i,j} x_{i,j,1,1}. \tag{5.44}$$

  Transformamos esta desigualdad en igualdad tomando como en ocasiones anteriores $D_{max} := \sum_{i=0}^{n} \text{máx}_j\{d_{i,j}\}$ y las variables $b_{h,q}$ como auxiliares:

$$\sum_{i=0}^{N+1} \sum_{j=0}^{N+1} d_{i,j} x_{i,j,1,q} + \sum_{h=0}^{h_{max}} 2^h b_{h,q} - \sum_{i=0}^{N+1} \sum_{j=0}^{N+1} d_{i,j} x_{i,j,1,1} = 0. \tag{5.45}$$

  A partir de esta igualdad llegamos a la función de penalización:

$$\text{Para todo } q \in \{2, ..., Q\}\text{:}$$

$$\begin{aligned}
&\sum_{i_1=0}^{N+1} \sum_{j_1=0}^{N+1} \sum_{i_2=0}^{N+1} \sum_{j_2=0}^{N+1} d_{i_1,j_1} d_{i_2,j_2} x_{i_1,j_1,1,q} x_{i_2,j_2,1,q} + \\
&\quad + \sum_{i=0}^{N+1} \sum_{j=0}^{N+1} \sum_{h=0}^{h_{max}} 2^{h+1} d_{i,j} x_{i,j,1,q} b_{h,q} + \\
&+ \sum_{i_1=0}^{N+1} \sum_{j_1=0}^{N+1} \sum_{i_2=0}^{N+1} \sum_{j_2=0}^{N+1} -2 d_{i_1,j_1} d_{i_2,j_2} x_{i_1,j_1,1,1} x_{i_2,j_2,1,q} + \\
&+ \sum_{h_1=0}^{h_{max}} \sum_{h_2=0}^{h_{max}} 2^{h_1} 2^{h_2} b_{h_2,q} b_{h_1,q} + \sum_{h=0}^{h_{max}} \sum_{i=0}^{N+1} \sum_{j=0}^{N+1} -2^{h+1} d_{i,j} x_{i,j,1,1} b_{h,q} + \\
&\quad + \sum_{i_1=0}^{N+1} \sum_{j_1=0}^{N+1} \sum_{i_2=0}^{N+1} \sum_{j_2=0}^{N+1} d_{i_1,j_1} d_{i_2,j_2} x_{i_1,j_1,1,1} x_{i_2,j_2,1,1}.
\end{aligned}$$

Llegados a este punto, la función a minimizar se corresponde con

$$\sum_{i=0}^{N+1}\sum_{j=0}^{N+1} d_{i,j}x_{i,j,1,1}. \tag{5.46}$$

A partir de la modelización que acabamos de presentar se consiguieron los resultados mostrados en el siguiente capítulo. Una vez que tenemos este modelo, la parte de programación es sencilla. Simplemente se debe ir construyendo la matriz de coeficientes de todas las variables que aparecen en el modelo, a partir de la cual el algoritmo *Quantum Annealing* va a construir la función $f(x) = x^tQx$ sobre la que se va a tratar de calcular el mínimo global.

5.3.4. Estudio de los coeficientes de Lagrange de las restricciones

Una vez que tenemos construido el modelo, podemos hacer un estudio sobre los valores de los coeficientes de Lagrange. Realizar un estudio exhaustivo de los coeficientes de Lagrange se queda fuera de este trabajo. Esto se debe a que para cada restricción debemos manejar los resultados de dicha restricción con todos los coeficientes anteriores ajustados, lo que lo convierte en un trabajo muy tedioso. En nuestro caso simplemente vamos a hacer una comparación de cada coeficiente de Lagrange con su aportación a la función objetivo. Vamos a ir trabajando con ellos de uno en uno en función de la restricción. Como vimos en la sección de coeficientes de Lagrange, es importante contar con una primera solución de nuestro problema. Supongamos que dicha solución es D y en nuestro problema podemos estimarla, por ejemplo, mediante el algoritmo de vecinos cercanos. Vamos a suponer también que la información acerca de las distancias entre los distintos objetos está recogida en la matriz *dist*. Tenemos entonces que la distancia entre el objeto i y el j corresponde con el coeficiente $dist_{i,j}$.

Estudiemos entonces los valores de los coeficientes de Lagrange por restricción. El estudio necesario para estimar los coeficientes de las restricciones 1 a 6 es similar al realizado en la sección 5.2.1 para el caso del problema TSP. Debido a esto, solo analizaremos las restricciones 7 a 10.

- *Séptima restricción.* Que no se cumpla esta restricción provoca que la restricción 10 carezca de sentido. Sin embargo, no tiene consecuencias directas sobre la función objetivo. Debido a esto, tomamos
$$\lambda_7 > D.$$

- *Octava restricción.* El hecho de no tener que volver a salir del nodo j para el robot q consigue que el robot q no tenga que volver a su lugar inicial, por lo que será suficiente con tomar
$$\lambda_{8,j} > dist_{j,N+1}.$$

- *Novena y décima restricciones.* Que no se cumplan estas restricciones tiene la consecuencia de que, directamente, la solución no sea factible. Para ello vamos a tomar como penalización que
$$\lambda_9 > D$$
y también que
$$\lambda_{10} > D.$$

Una vez realizado el estudio de los coeficientes de Lagrange, hemos completado nuestra modelización. En la siguiente sección vamos a ver un par de ejemplos de problemas que pueden ser resueltos aplicando conceptos que hemos desarrollado durante este capítulo.

5.3.5. Aplicaciones del problema Q-Robots

La modelización que acabamos de construir puede aplicarse en otros problemas para mejorar sus resultados. Uno de estos problemas es el siguiente:

Problema del ciclo hamiltoniano

Se conoce como problema del ciclo hamiltoniano al consistente en tratar de encontrar sobre un grafo el mayor ciclo posible. Es decir, dado un grafo, encontrar un ciclo que incluya el mayor número de vértices posible. Este problema puede modelizarse de manera similar al problema TSP desarrollado. Sin embargo, no mostraremos dicha modelización por ser muy similar a la que aparece en la sección 5.2, evitando repetir ecuaciones.

Problema del viajante con ventanas de tiempo

Este problema se ha tratado de resolver mediante el algoritmo *Quantum Annealing*, formulándolo como un problema QUBO [15]. Sin embargo, una vez que contamos con nuestra modelización vamos a ver cómo podemos plantear una solución alternativa que disminuya el número de variables utilizadas. El problema del viajante con ventanas de tiempo consiste en una generalización del problema del viajante para el que se tiene la restricción adicional de que a cada ciudad se debe llegar en un intervalo de tiempo determinado. Veamos cómo podemos modelizarlo utilizando variables binarias.

Sea N el número de ciudades que hay que recorrer a las que añadimos la ciudad 0 que corresponde con el origen y la ciudad $N+1$ que corresponde con el final de la ruta. Sea $d_{i,j}$ el tiempo necesario para viajar de la ciudad i a la j y supongamos que los intervalos de tiempo para cada ciudad se corresponden con los intervalos $[a_n, l_n]$ para $n \in \{1, ..., N\}$, y M se corresponde con el valor máximo que podemos tardar en completar el recorrido. Las variables de este modelo serán las siguientes:

- Sean las variables $x_{i,j,r}$, con $i, j \in \{0, ..., N+1\}$ y $r \in \{0, 1, 2\}$ que representan lo explicado en la sección 6.2. En el modelo están eliminadas las variables $x_{i,i,r}$ con $i \in \{0, ..., N+1\}$ y $r \in \{0, 1, 2\}$.
- Por otro lado, sea T_i el instante en el que pasamos por la ciudad i. Tenemos entonces que para cada i:

$$T_i = \sum_{h=0}^{\lfloor log_2(M) \rfloor + 1} 2^h t_{i,h}.$$

  Para facilitar la notación, tomaremos en las ecuaciones siguientes las variables T_i en lugar del sumatorio de las variables $t_{i,h}$.

Planteando las mismas restricciones de la sección 6.2, nos encontramos con que simplemente es necesario introducir las restricciones necesarias para garantizar que se cumplen las condiciones temporales. Esto se puede realizar con las siguientes 2 restricciones:

- *Restricción 3*: Cada valor T_i debe pertenecer al intervalo $[a_i, l_i]$, por lo que

$$\text{Para cada } i \in \{1, ..., N\}\text{: } a_i \leq T_i \leq l_i. \tag{5.47}$$

- *Restricción 4*: Se debe cumplir que si de la ciudad i viajamos a la ciudad j, entonces $T_i + d_{i,j} \leq T_j$. Podemos expresar esta restricción con la siguiente ecuación (recordemos que no se tienen en cuenta los casos en los que $i = j$):

$$\text{Para cada } i, j \in \{0, ..., N+1\}\text{: } T_i + d_{i,j} \leq T_j + M(1 - x_{i,j,1}). \tag{5.48}$$

Una vez que se cumplen estas condiciones, la función a minimizar coincide con T_{N+1}. Sin embargo, no es necesario introducir el índice r. Esto se debe a que la restricción 4 está dotando implícitamente de un orden en el recorrido de los nodos. Por lo tanto podemos eliminar dicho índice y tomar la modelización realizada en el apéndice (7.5).

La modelización que encontramos en [15] requiere de $N^3 + MN$ variables. Como el valor M puede ser muy alto, dicha modelización es muy pobre. Sin embargo, nuestra modelización mejora el valor de N^3 y M, ya que requiere del siguiente número de variables: Por un lado, $(N+2)^2$ variables $x_{i,j}$ y $Nlog_2(M)$ variables $t_{i,h}$. Por otro lado, son necesarias las variables auxiliares requeridas para transformar las desigualdades de las restricciones 3 y 4 en igualdades, lo que se corresponde con $Nlog_2(M)$ de la tercera restricción y $(N+2)^2log_2(2M)$ de la cuarta, lo que hace un total de

$$(N+2)^2 + Nlog_2(M) + Nlog_2(M) + ((N+2)^2 + 1)log_2(M).$$

Tomando $N + 2 \approx N$ llegamos a que el número de variables utilizadas es

$$N^2 + log_2(M)(2N + N^2). \tag{5.49}$$

Aplicando la notación de Landau utilizada para estudiar la complejidad computacional de un algoritmo, hemos mejorado un modelo que requiere $N^3 + MN$ variables en uno que consta de $log_2(M)N^2$.

Capítulo 6

Resultados

En este capítulo vamos a mostrar los resultados de distintas simulaciones realizadas mediante el algoritmo *Quantum Annealing* a partir de nuestra modelización. Estos resultados corresponden a tomar parámetros de N y Q pequeños, siendo N el número de objetos y Q el número de robots. Como hemos comentado anteriormente, para conseguir buenos resultados en cierto problema es necesario estimar los coeficientes de Lagrange de manera correcta. Tenemos entonces que para conseguir buenas soluciones para valores de N y Q grandes es conveniente hacer un estudio de estos coeficientes para cada caso. Sin embargo, ya existen métodos desarrollados por D-Wave y que pueden ser implementados que tratan de calcular los coeficientes de Lagrange convenientes para cada problema. Como nuestro trabajo tiene un carácter matemático, simplemente vamos a mostrar los resultados para ejemplos sencillos con el objetivo de apreciar cómo, en efecto, las funciones calculadas en el capítulo cinco son correctas.

Las simulaciones realizadas para obtener los resultados de este capítulo han sido llevadas a cabo a través de un simulador desarrollado por D-Wave en Python y no mediante el uso de un ordenador cuántico. Tenemos entonces que los resultados que aparecen han sido conseguidos mediante **computación clásica**. Como comentamos en el párrafo anterior, el objetivo de este capítulo es comprobar que la teoría desarrollada en el capítulo anterior es correcta y no tratar de probar el buen funcionamiento del *Quantum Annealing*. En la descripción de estos ejemplos aparece el concepto de número de ***shots*** empleados en las distintas simulaciones necesarias para tratar de resolver los problemas. Hasta este momento no había aparecido este término y, si bien su definición es algo más complicada, daremos una idea intuitiva sobre este concepto. Como se explica en el capítulo 3, el Quantum Annealing es un algoritmo heurístico. Tenemos entonces que cuando se ejecuta el simulador de recocido cúantico, éste tratará de encontrar las mejores soluciones. Como no se garantiza encontrar el mínimo absoluto en cada ejecución, si elevamos el número de simulaciones que van a realizarse, mejoraremos las soluciones obtenidas y aumentaremos la probabilidad de localizar el mínimo absoluto. El número de *shots* puede entonces interpretarse como el número de veces que ejecutamos el algoritmo *Quantum Annealing* en el computador cuántico que estamos utilizando (en este caso en el simulador), para tratar de localizar el mínimo de nuestro problema. El resultado final corresponderá con la mejor solución encontrada entre esas simulaciones.

6.1. Resultados del problema TSP

Veamos en esta sección dos ejemplos de soluciones obtenidas mediante el algoritmo de atemperamiento cuántico a través de la modelización que hemos desarrollado.

Ejemplo 1. El primer problema que mostramos se corresponde con una resolución del problema TSP aplicando la modelización desarrollada en este trabajo. Es sencillo apreciar que el camino encontrado corresponde a la solución óptima. El tiempo empleado por el algoritmo *Quantum Annealing* para encontrar esta solución fue aproximadamente de unos dos minutos, ya que se configuraron 2000 *shots*. Para este ejemplo tomamos $N = 6$. De los seis puntos que forman el problema, cinco se escogieron manualmente, de tal manera que el algoritmo de vecinos cercanos proporcionara una mala solución a esta distribución. El sexto punto se escogió de manera aleatoria (de ahí las coordenadas reales que aparecen en la tabla). La localización de todos estos puntos la podemos encontrar en la tabla de la imagen 6.1.

```
*Los puntos que forman el problema son*
[[0.         0.        ]
 [0.         1.        ]
 [3.         1.        ]
 [3.         0.        ]
 [6.         0.        ]
 [6.         1.        ]
 [0.92942229 0.87632291]]
```

Figura 6.1: Coordenadas de los puntos que forman los nodos del ejemplo 1.

La solución a este problema consiste en un camino que pasa por todos los puntos y que podemos visualizar en la imagen (6.2). Como podemos apreciar, la solución proporcionada por el simulador

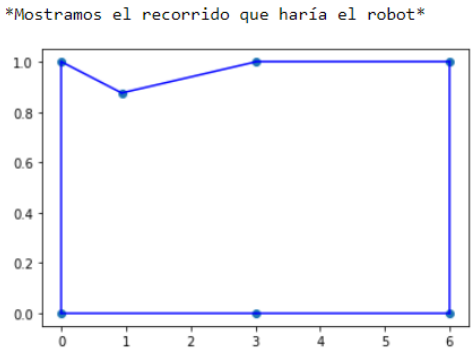


Figura 6.2: Imagen del recorrido realizado por el robot minimizando la distancia para el ejemplo 1.

de D-Wave se corresponde con el óptimo.

Las condiciones bajo las cuales el algoritmo de atemperamiento cuántico funciona de manera óptima se dan cuando el mínimo global está lo suficientemente alejado del resto de soluciones. Sin embargo, esta condición no es suficiente y a menudo entra en conflicto con la otra condición básica, y es que no existan soluciones para las cuales la función de coste tome valores muy alejados del resto, ya sean máximos o mínimos. Esto viene ligado a la idea de valor de restricción. Si existen restricciones cuyo intervalo de valor de restricción es muy amplio, el algoritmo de *Quantum Annealing* se atascará

en soluciones que cumplan esa restricción y no evolucionará a encontrar soluciones que traten de mejorar todas las demás. Si por el contrario los valores de restricción se mantienen pequeños debido a estar manejando coeficientes de Lagrange con valores bajos, será probable que el mínimo global no se encuentre muy alejado de otros puntos. Cuando la función a minimizar cumple las condiciones óptimas, se requiere un número de *shots* pequeño para encontrar el resultado. Para algunos problemas, un solo *shot* es suficiente para alcanzar el óptimo. Sin embargo, este no es el caso para el problema TSP, ya que existen soluciones muy cercanas al óptimo. Veámoslo con el siguiente ejemplo.

Ejemplo 2. Para este ejemplo se calculó el valor del mínimo global y posteriormente se realizó una simulación para tratar de resolver el problema con un número de *shots* bajo. Como podemos

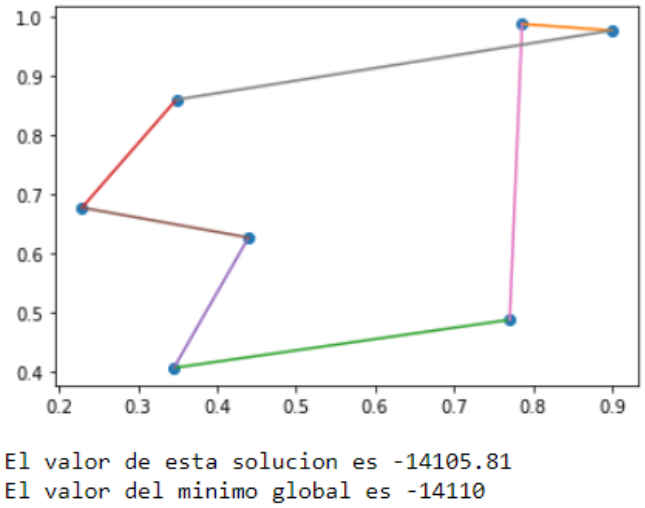


Figura 6.3: Imagen del recorrido propuesto como solución al resolver el ejemplo 2.

apreciar en la figura 6.3, tenemos que el valor de la solución propuesta es muy cercano al valor del mínimo global. Podemos llegar a encontrar la solución óptima elevando el número de *shots* empleados en la simulación. El valor -14110 que aparece en la imagen corresponde con la suma de los valores mínimos tomados por las funciones de penalización, más el valor de la distancia necesaria para realizar el recorrido (en este caso corresponde con 90).

6.2. Resultados del problema Q-Robots

Pasemos ahora al caso del problema Q-Robots. Para estos ejemplos vamos a tomar valores de N y Q pequeños. Al igual que en el ejemplo 1, servirá para mostrar que en efecto la modelización que hemos desarrollado es correcta. Veamos un par de ejemplos para apreciar los buenos resultados que proporciona el algoritmo *Quantum Annealing*. Para construir los ejemplos siguientes se han realizado algunas modificaciones en la modelización de cara a obtener mejores resultados. Tanto en la restricción 8 (5.35) como en la 10 (5.35) existe la posibilidad de que, debido a que no se cumplan otras restricciones, estas restricciones puedan tomar valores negativos en puntos donde se incumplan las otras restricciones. Existen dos formas de evitar esto: La primera consiste en realizar un ajuste delicado de los coeficientes de Lagrange de las restricciones pertinentes, de manera que los valores negativos de las restricciones 8 y 10 no proporcionen mejores resultados que ajustar las otras dos restricciones. La segunda manera consiste en introducir variables auxiliares que, al poder tomar solo los valores 0 y 1, garanticen que no existan dichos valores negativos en las restricciones 8 y 10. La

dificultad de ajustar correctamente los coeficientes de Lagrange, de la que hemos hablado en otras ocasiones, conduce a que hayamos optado por la segunda opción a la hora de realizar la programación de nuestra solución.

Ejemplo 3. Para este ejemplo se ha tomado $N = 5$ y $Q = 2$. A pesar de ser unos coeficientes pequeños, ilustrarán cómo, en efecto, la modelización desarrollada es correcta. Los N puntos que formarán parte del problema, y cuyas coordenadas se encuentran en la imagen 6.4, se escogieron de manera aleatoria. El número de *shots* escogidos para resolver este problema fue 2000.

```
*Las coordenadas de los puntos que forman el problema son*
[[0.59999828 0.24046079]
 [0.01054829 0.80124871]
 [0.36972145 0.33198302]
 [0.72997095 0.40888692]
 [0.05655322 0.04643113]
 [0.74156675 0.15060033]]
```

Figura 6.4: Coordenadas de los puntos del ejemplo 3.

Bajo estas condiciones, la solución propuesta mediante el simulador del algoritmo *Quantum Annealing* se puede observar en la imagen 6.5. En esta imagen tenemos que la ruta correspondiente al recorrido del robot 1 aparece dibujada con líneas rojas, y la ruta correspondiente al recorrido del robot 2 aparece dibujada con líneas verdes. Es sencillo ver cómo, en efecto, la solución obtenida

```
*Las distancias que recorren los robots son*
El robot  1  recorre  1.739
El robot  2  recorre  1.65
```

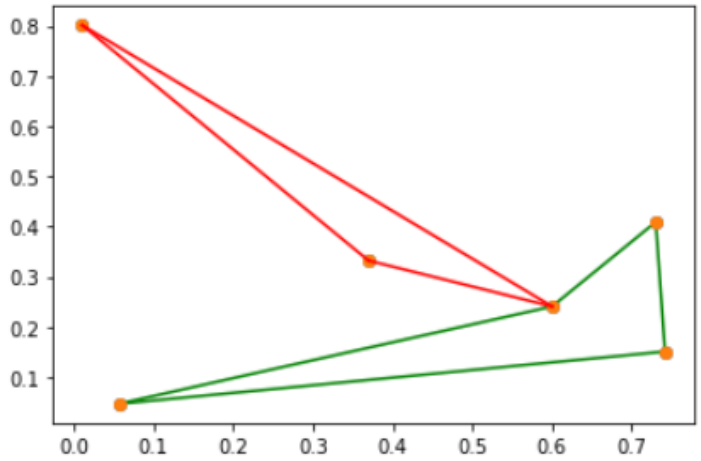


Figura 6.5: Recorridos de los dos robots del ejemplo 3 al minimizar el tiempo de sus rutas.

coincide con la óptima. Veamos un último ejemplo que vamos a resolver.

Ejemplo 4. En el siguiente ejemplo hemos aumentado el número de robots a $Q = 3$. Para que el algoritmo siga encontrando la solución óptima vamos a mantener un número de objetos bajo. Para este ejemplo fue necesario elevar el número de *shots* a 5000. El número de objetos empleados en este ejemplo fue seis. Las localizaciones se escogieron manualmente, de forma que cada robot tuviera que ir a por dos de esos objetos. En la figura 6.6 vemos cómo efectivamente los robots se reparten los

```
El robot  1  recorre  3299.0
El robot  2  recorre  2976.0
El robot  3  recorre  3191.0
```

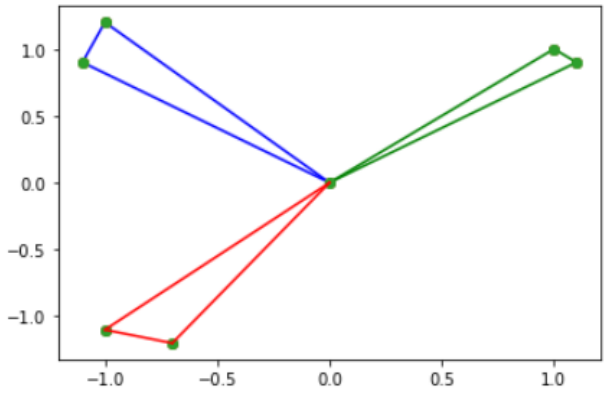


Figura 6.6: Imagen de los 3 recorridos trazados por los 3 distintos robots del ejemplo 4.

objetos de manera óptima. En el siguiente capítulo expondremos las conclusiones extraídas sobre este trabajo.

Capítulo 7

Conclusiones y líneas futuras

El objetivo de este trabajo consistía en realizar un estudio de las modelizaciones QUBO del problema del viajante y tratar de construir a partir de ellas una buena modelización del problema de enrutamiento de vehículos. Sin embargo, una vez examinadas las soluciones planteadas del problema del viajante se optó por realizar una nueva modelización de éste que mejoraba las modelizaciones con función objetivo lineal del TSP, convirtiéndose en la modelización que mejor generaliza al VRP en términos de número de variables utilizado. El interés por el estudio de las modelizaciones QUBO radica en que existen distintos proyectos en los cuales está involucrada principalmente la empresa D-Wave y que cuenta con grandes clientes como Airbus y Volkswagen, que apuesta por el desarrollo de ordenadores cuánticos especializados en resolver de manera eficiente este tipo de problemas. A esta herramienta se le denomina *Quantum Annealing* y trata de mejorar los métodos heurísticos clásicos de optimización de funciones en espacios discretos.

Como comentamos durante la memoria, aunque la programación cuántica de puertas difiera del *Quantum Annealing* es posible aprovechar nuestro trabajo en la computación basada en puertas cuánticas (desarrollada por IBM, Google o Xanadu entre otros). Para ello simplemente se debe aplicar el cambio de variable $z_i = 2x_i - 1$, donde z_i es una nueva variable que puede tomar los valores -1 o 1,. De esta manera transformamos la formulación QUBO en el modelo Ising [13], soportado por este paradigma de programación que acabamos de mencionar.

En proyectos futuros se seguirá estudiando otras características que existan entre distintas modelizaciones de cara a conseguir mejores resultados cuando tratan de ser resueltas mediante recocido cuántico. Una de estas propiedades es la forma de la función a minimizar. Una función objetivo con muchos valles aumentará la probabilidad de que el algoritmo se quede atascado en mínimos locales y no evolucione hasta alcanzar el mínimo global. De esta manera nos encontramos con que es interesante tratar de construir funciones con el menor número de mínimos locales posible y que estos mínimos no sean muy profundos, de forma que al ejecutar los algoritmos de *Annealing* se eviten soluciones alejadas del óptimo. Otra posible línea de futuro contemplada consiste en tratar de aplicar toda las mejoras conseguidas en el trabajo a la herramienta *Docplex* de IBMQ [9]. También profundizáremos como nuestro modelo QUBO puede contribuir en las aplicaciones[16, 3] de la era del *Quantum Machine Learning* (QML) [19].

Bibliografía

[1] Alonso Linaje, G., «https://github.com/KetpuntoG/Notebooks-del-canal/blob/master/N-Queens%20Problem/N-Reinas%202.0.ipynb », *GitHub - Código Problema de las N-reinas*, 2021.

[2] Atchade-Adelomou, P. y Alonso-Linaje, G., «qRobot: A Quantum Computing Approach in Mobile Robot Order Picking and Batching Problem Solver Optimization», *Algorithms*, 2021.

[3] Atchade, P. y Alonso-Linaje, G., «https://doi.org/10.21203/rs.3.rs-405334/v1», *Quantum Enhanced Filter: QFilter*, *Research Square Platform LLC*, 2021.

[4] Beltrami, E. y Bodin, L.D., «Networks and vehicle routing for municipal waste collection», *Networks*, 4:65-94, 1974.

[5] Castellanos, S., «Volkswagen to Test Quantum Navigation App in Real Traffic», *The Wall Street Journal*, 2019.

[6] Dantzig, G.B., Fulkerson, D.R. y Johnson, S.M., *Solution of a large-scale traveling-salesman problem.*, *Journal of the Operations Research Society, vol. 2, no. 4, pp. 393–410*, 1954.

[7] Ekertand, A. y Josza, R., «Quantum computation and Shor's factoring algorithm», *Rev. Mod. Phys. 68*, pp 733–753, 1996.

[8] Feld, S.; Roch, C.; Gabor, T.; Seidel, C.; Neukart, F.; Galter, I.; Mauerer, W. y Linnhoff-Popien, C., «A Hybrid Solution Method for the Capacitated Vehicle Routing Problem Using a Quantum Annealer», *Frontiers in ICT*, 6:1-13, 2019.

[9] IBM,«https://www.ibm.com/docs/en/icos/12.9.0?topic=docplex-python-modeling-api», *DOcplex Python Modeling API*, 2021.

[10] Irie, H.; Wongpaisarnsin, G.; Terabe, M.; Miki, A. y Taguchi, S., «Quantum annealing of vehicle routing problem with time, state and capacity», *Quantum Technology and Optimization Problems*, Springer, pp 145-156, 2019.

[11] Kirkpatrick, S.; Gelatt, C. D. y Vecchi, M. P., «Optimization by Simulated Annealing», *American Association for the Advancement of Science*, 1983.

[12] Lin, S. y Kernighan, B.W., «An effective heuristic algorithm for the traveling salesman problem», *Oper. Res.*, Vol.21, No.2, pp. 498-516, 1973.

[13] McGeoch Herine C., *Adiabatic Quantum Computation and Quantum Annealing Theory and Practice*, *Synthesis Lectures computing*, 2014.

[14] Miller C., *"Integer programming formulation of traveling salesman problems. jacm 1960; 7 (4): 326–329. desrochers m, laporte g. improvements and extensions to the miller-tucker-zemlin subtour elimination constraints"*, *Operations Research Letters, vol. 10, pp. 27–36*, 1954.

[15] Papalitsas, C. et al., «A QUBO Model for the Traveling Salesman Problem with Time Windows», *Algorithms*, 2019.

[16] Prasanna Date, D. y Pusey-Nazzaro, L., «https://doi.org/10.1038/s41598-021-89461-4», *QUBO formulations for training machine learning models*, *Springer Science and Business Media LLC*, 2021.

[17] Preskill, J., «Quantum Computing in the NISQ era and beyond.», *Verein zur Forderung des Open Access Publizierens in den Quantenwissenschaften*, 2018.

[18] Sáenz de Cabezón, E. [Derivando], «https://www.youtube.com/watch?v=WOZ4wDtiYA», *El problema de las 1000 reinas. ¡Un millón de dólares en juego!*, Youtube, 2017.

[19] Schuld, M. y Petruccione, F., *An introduction to quantum machine learning*, *Contemporary Physics*, 2015.

[20] van Laarhoven, P.J. y Aarts, E.H., *Simulated Annealing: Theory and Applications (Mathematics and Its Applications)*, vol. 37, Springer, 1987.

[21] Verma, A. y Lewis, M., «Penalty and partitioning techniques to improve performance of QUBO solvers», *Discrete Optimization*, 2020.

[22] «https://docs.dwavesys.com/docs/latest/c_gs_2.html», *What is Quantum Annealing?*, D-Wave System Documentation.

[23] «https://docs.dwavesys.com/docs/latest/doc_handbook.html», *Problem-Solving Handbook*, D-Wave System Documentation.

Apéndice

Modelizaciones DFJ y MTZ

Como comentamos en el capítulo 2, existen dos modelizaciones conocidas del TSP bastante cercanas a una modelización que consideraríamos QUBO. Veamos como funcionan estas dos modelizaciones y sus incovenientes. Supongamos que tenemos el problema del viajante para el cuál N es el número de ciudades que hay que visitar más una ciudad extra correspondiente al punto inicial y final. Las siguientes variables y restricciones coinciden en ambos modelos. Sean las variables binarias $x_{i,j}$ donde $i, j \in \{0, ..., N+1\}$. Como ha aparecido de manera habitual en el trabajo, $x_{i,j} = 1$ significa que de la ciudad i viajamos a la ciudad j. Tenemos entonces que se debe garantizar las siguientes restricciones

1 $\{x_{i,j}\} \in \{0, 1\}$ para todo i, j.

2 Para cada $i \in \{0, ..., N\}$: $\sum_{j=1}^{N+1} x_{i,j} = 1$

3 Para cada $j \in \{1, ..., N+1\}$: $\sum_{i=0}^{N} x_{i,j} = 1$

La restricción 1 establece que las variables deben ser binarias, las restricciones 2 y 3 imponen las condiciones de salir y llegar a todas las ciudades respectivamente. Como explicamos en el capítulo 5, estas condiciones no son suficientes para garantizar que se forme un único ciclo que pase a través de todas las ciudades, por lo que veamos cuál es la restricción en cada modelo que consigue esta condición.

- Modelización DFJ: La restricción del modelo de Dantzig que trata de lograr que no se formaran ciclos corresponde con la siguiente

$$\sum_{i \in Q} \sum_{j \neq i j \in Q} x_{i,j} \leq |Q| - 1 \text{ para todo } Q \subset \{1, ..., N\},\ |Q| \geq 2 \tag{7.1}$$

  Esta ecuación trata de conseguir que para cualquier conjunto de ciudades exista un arista que salga de él, por lo que evita que se forme un ciclo en el cuál el viajero *quedaría atrapado*. Sin embargo, esta restricción tiene el inconveniente de que el número de subconjuntos de $\{1, ..., N\}$ corresponde con 2^N, por lo que el número de restricciones de esta modelización crece exponencialmente en función de N, lo cual convierte este modelo en completamente ineficiente.

- Modelización MTZ: La modelización MTZ trata de mejorar la anterior, de manera que evitemos un número de restricciones absurdamente grande. Para ello se plantea las siguientes restricciones.

$$u_i \in \mathbb{Z} \tag{7.2}$$

$$\text{Para cada } 2 \leq i \neq j \leq n\text{: } u_i - u_j + n x_{i,j} \leq n - 1 \tag{7.3}$$

$$\text{Para cada } 2 \leq i \leq n\text{: } 0 \leq u_i \leq n - 1 \tag{7.4}$$

En esta modelización la variable u_i representa la posición en el recorrido de la ciudad i. Notar que las variables u_i pueden tomarse continuas, por lo que para cada elección de las variables $x_{i,j}$ sería suficiente aplicar el método símplex para calcular los valores T_i. De esta manera podría manejarse un algoritmo heurístico que simplemente tomara valores en las variables $x_{i,j}$. Sin embargo, de cara a aplicarse para ser resuelta mediante *Quantum Annealing* nos encontramos con que para cada combinación de variables $x_{i,j}$ es necesario calcular los variables u_i. De esta manera llegamos a una situación que proporciona muy malos resultados (a parte de ser necesario el uso de $log(N)N^2$ variables auxiliares para introducir la restricción 7.3).

Problema del viajante con ventanas de tiempo

En la sección dedicada al problema del viajante con ventanas de tiempo, planteamos una modelización de este problema a partir de las variables $x_{i,j,r}$ que definimos en (5.13). Sin embargo, existe una modelización mejor debido a que no es necesario establecer un orden entre las variables $x_{i,j}$. Esto se debe a que al incluir las restricciones correspondientes a las ventanas de tiempo, se va a dotar de un orden en el recorrido que impide la formación de ciclos. Veamos el enunciado del problema y esta modelización.

Sea N el número de ciudades que hay que recorrer, más la ciudad 0 que corresponde con el origen y la ciudad $N+1$ que corresponde con el final de la ruta y sea $d_{i,j}$ el tiempo necesario para viajar de la ciudad i a la j. Supongamos que los intervalos de tiempo para cada ciudad se corresponden con los intervalos $[a_n, l_n]$ para $n \in \{1, ..., N\}$, y M se corresponde con el valor máximo que podemos tardar en completar el recorrido. Si $l_n < \infty$ para todo n, podemos estimar M de manera sencilla tomando $M = \text{máx}_{n\in\{1,...,N\}} l_n + d_{n,0}$. Para tratar de resolver este problema vamos a utilizar las siguientes variables:

- Sean las variables $x_{i,j}$, con $i, j \in \{0, ..., N+1\}$, tales que $x_{i,j} = 1$ si de la ciudad i vamos a la ciudad j, y $x_{i,j} = 0$ si no. En el modelo están eliminadas las variables $x_{i,i}$.
- Por otro lado, sea T_i el instante en el que pasamos por la ciudad i. Tenemos entonces que para cada i:
$$T_i = \sum_{h=0}^{log_2(M)} 2^h t_{i,h}.$$
Para facilitar la notación, tomaremos en las ecuaciones T_i en lugar del sumatorio (su expresión en función de las variables $t_{i,h}$).

Una vez que tenemos las variables que van a formar el modelo, veamos cuáles son las restricciones que se deben verificar. Recordemos que al tratar los sumatorios estamos obviando las variables $x_{i,i}$.

- *Restricción 1*: Debemos llegar a cada ciudad una y solo una vez, luego
$$\text{Para cada } j \in \{1, ..., N+1\}\text{: } \sum_{i=0}^{N} x_{i,j} = 1. \tag{7.5}$$
- *Restricción 2*: Debemos salir de cada ciudad una y solo una vez, luego
$$\text{Para cada } i \in \{0, ..., N\}\text{: } \sum_{j=1}^{N+1} x_{i,j} = 1. \tag{7.6}$$

Veamos las restricciones correspondientes a que se cumplan los intervalos de tiempo.

- *Restricción 3*: Cada valor T_i debe pertenecer al intervalo $[a_i, l_i]$, por lo que

$$\text{Para cada } i \in \{1, ..., N\}\text{: } a_i \leq T_i \leq l_i. \tag{7.7}$$

- *Restricción 4*: Se debe cumplir que si de la ciudad i viajamos a la ciudad j, entonces $T_i + d_{i,j} \leq T_j$. Podemos expresar esta restricción con la siguiente ecuación

$$\text{Para cada } i, j \in \{0, ..., N+1\}\text{: } T_i + d_{i,j} \leq T_j + M(1 - x_{i,j}). \tag{7.8}$$

Una vez que se cumplen estas condiciones, la función a minimizar coincide con T_{N+1}.

Problema N-reinas

A lo largo del trabajo aparecen distintas referencias a este problema.

Máximo penalización N-Reinas

En la sección 3.5.1 se presentaban los valores de restricción asociados a una penalización y aparecía como ejemplo el problema de las N-Reinas. Veamos como calcular el máximo de la función de penalización asociada a la restricción que evita que existan dos reinas atacándose entre ellas. Recordemos que la función de penalización que vamos a manejar es la siguiente

$$\sum_{i_1=1}^{n}\sum_{i_2=1}^{n}\sum_{i_3=1}^{n}\sum_{i_4=1}^{n} J_{i_1,i_2,i_3,i_4} x_{i_1,i_2} x_{i_3,i_4} \tag{7.9}$$

Como comentamos en la sección a la que hacemos referencia, el máximo de esta función se alcanza al tomar $x_{i,j} = 1$ para todo i, j; es decir, colocar una reina en cada casilla del tablero. Con esta disposición cada reina ataca a todas las reinas de su misma fila, columna y las de las diagonales. Tenemos entonces que el máximo se corresponde con

$$2N^2(N-1) + \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} (\text{mín}(i,j) + \text{mín}(i, N-1-j) + \text{mín}(N-1-i, j) + \text{mín}(N-1-i, N-1-j)), \tag{7.10}$$

donde el primer término corresponde a que cada reina colocada en el tablero ataca a las $2(N-1)$ reinas con las que comparte fila o columna, y el sumatorio corresponde al número de reinas con las que cada una comparte diagonal en función de la casilla que ocupa. Es fácil ver, mediante cambios de variable de la forma $k = N - 1 - p$, que la expresión anterior coincide con

$$2N^2(N-1) + 4\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} \text{mín}(i,j) \tag{7.11}$$

Manejando los sumatorios en función de los valores de i y j, tenemos que la función anterior es equivalente a

$$2N^2(N-1) + 8\sum_{i=1}^{N-1}\sum_{j=0}^{i-1} j + 4\sum_{i=0}^{N-1} i \tag{7.12}$$

Ya que por un lado hemos tomado los puntos donde $i = j$ y los casos $i < j$ y $j < i$ se pueden agrupar por simetría en $2\sum_{i=1}^{N-1}\sum_{j=0}^{i-1} j$. Llegado a este punto, desarrollar la función anterior es sencillo:

$$= 2N^2(N-1) + 4\sum_{i=1}^{N-1}(i^2 - i) + 2N(N-1) =$$

$$= 2N^2(N-1) + \frac{2}{3}(N-1)N(2N-1) - 2N(N-1) + 2N(N-1) =$$

$$= \frac{1}{3}(N-1)(10N-2)N$$

Podemos comprobar que este valor es correcto de manera sencilla mediante el algoritmo de Quantum Annealing, ya que corresponde con encontrar el máximo de la función

$$\sum_{i_1=1}^{n}\sum_{i_2=1}^{n}\sum_{i_3=1}^{n}\sum_{i_4=1}^{n} J_{i_1,i_2,i_3,i_4} x_{i_1,i_2} x_{i_3,i_4}$$

Podemos aplicar el siguiente código escrito en el lenguaje Python para verificar que en efecto se cumple la igualdad:

```
N = 8
sol = 0
for i1 in range(N):
    for i2 in range(N):
        for i3 in range(N):
            for i4 in range(N):
                cond1 = i1 == i3
                cond2 = i2 == i4
                cond3 = i1 - i3 == i2 - i4
                cond4 = i1 - i3 == i4 - i2
                if cond1 or cond2 or cond3 or cond4:
                    if not (i1 == i3 and i2 == i4):
                        sol += 1
print("La␣suma␣es",sol)
print("1/3(N-1)(N)(10N-2)␣=",((N-1)*(N)*(10*N-2))/3)
```

Imagen 50-reinas

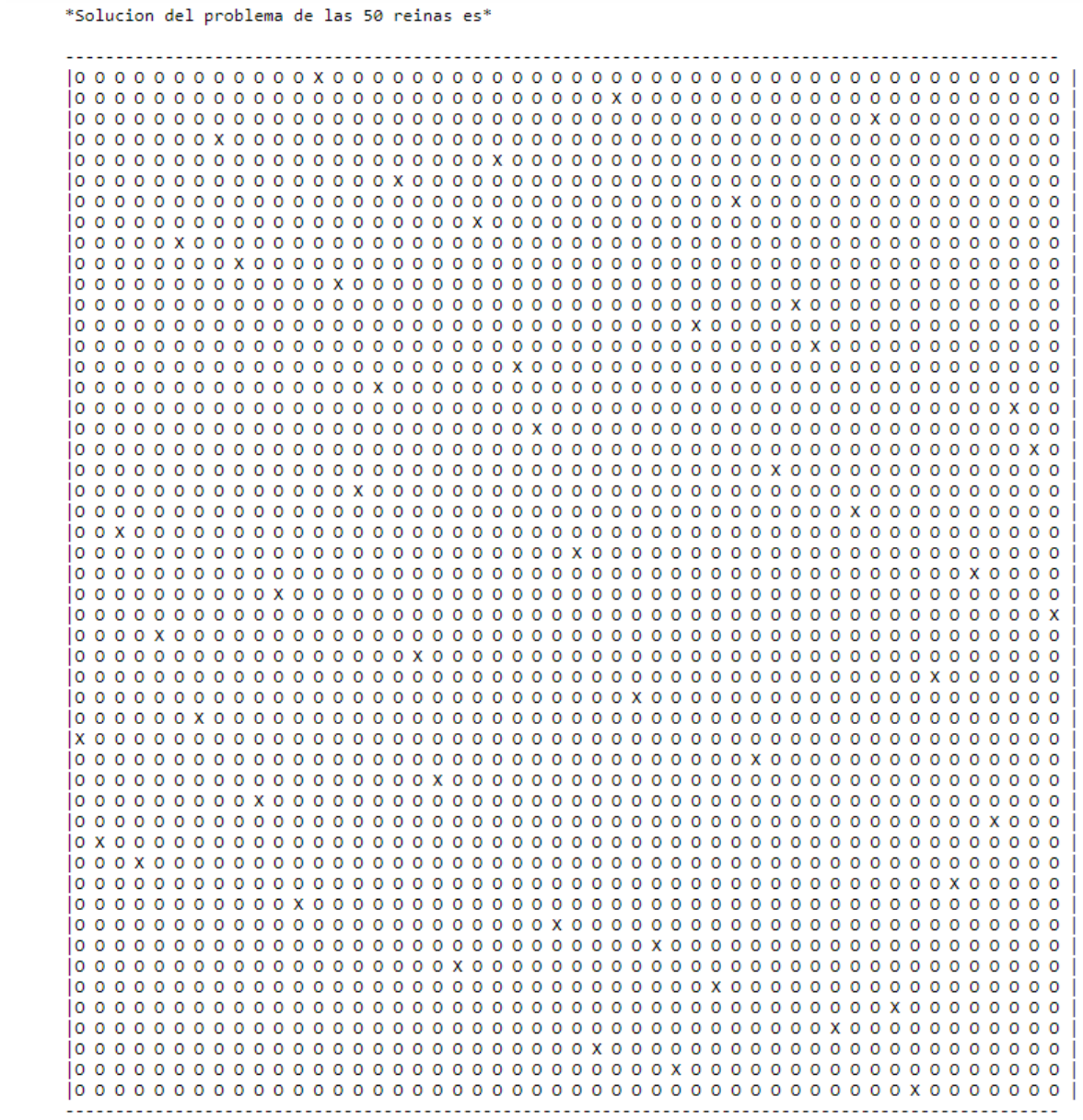
En distintas partes del trabajo se ha mencionado como la herramienta *Quantum Annealing* proporciona muy buenos resultados para ciertos problemas. Podemos apreciar ese hecho en la siguiente imagen, que corresponde con una solución para el problema de las 50-reinas. La imagen consiste en un tablero de dimensiones 50×50 donde los lugares ocupados por X corresponden a la posición de una dama y los lugares ocupados por 0 corresponden a casillas vacías.
El código utilizado para generar la imagen anterior lo podemos encontrar en [1].

Para ayudarnos a entender la potencia del algoritmo del Quantum Annealing vamos a realizar un estudio del espacio de soluciones de nuestro problema. En nuestra modelización estamos utilizando 50^2 qubits, lo que supone que el cardinal del espacio donde se buscan las soluciones es $2^{(50^2)} = 2^{2500}$. Si tratamos de estimar la proporción del número de posibles soluciones P con el número de soluciones factibles F, nos encontramos lo siguiente. El número de soluciones factibles es complicado de estimar para este problema, sin embargo es claro que dicho valor es mucho menor que 50!, ya que solo puede haber una reina por fila y columna. Tenemos entonces que

$$\frac{F}{P} \leq \frac{50!}{2^{2500}} \approx \frac{50^{50}2^{50}\sqrt{100\pi}}{2^{2500}e^{50}} \leq \frac{5^{100}\sqrt{100\pi}}{2^{2500}} \leq \frac{2^{300}2^5}{2^{2500}} \leq \frac{1}{2^{2000}} \leq \frac{1}{10^{500}} \tag{7.13}$$

Donde en la equivalencia hemos aplicado la fórmula de Stirling para el factorial. De esta cota deducimos la capacidad que tiene el Quantum Annealing de encontrar soluciones a problemas es espacios inmensamente grandes.

Figura 7.1: Tablero en el que se muestra una solución del problema de las N-reinas para N=50.



Código en Python

Veamos unas imágenes del código en Python utilizado para conseguir los resultados de las simulaciones.

Figura 7.2: Código de definición del problema Q-Robots.

In [1]:
```
## Librerias necesarias
import qubovert
import math
from neal import SimulatedAnnealingSampler

import numpy as np
import matplotlib.pyplot as plt

N = 6
Q = 3
def fnorm(v):  ## v must be a np.array
    return np.sqrt(np.sum(v**2))
puntos = np.random.rand(N+1,2)
#plt.plot(puntos[:,0],puntos[:,1],'o')
dist = np.zeros((N+2,N+2))
for i in range(N):
    for j in range(i+1,N+1):
        aux  =  fnorm(puntos[i,:]-puntos[j,:])
        dist[i,j],dist[j,i] = aux,aux
for j in range(0,N+1):
    i = N+1
    aux  =  fnorm(puntos[0,:]-puntos[j,:])
    dist[i,j],dist[j,i] = aux,aux
dist = np.floor(dist*1000)  ## Tomamos los puntos como enteros

## Indices de los sumatorios
R = 5
lis_n = range(0,N+2)
lis_q = range(1,Q+1)
lis_r = range(R)
lis_h = range(bmax+1)


# Creamos las variables de nuestro modelo
## Variables x_{i,j,r,q}
coef = qubovert.QUBO()
for i in lis_n:
    for j in lis_n:
        for r in lis_r:
            for q in lis_q:
                coef.create_var(f"x_{i}_{j}_{r}_{q}")
## Variables a_{i,j}
for i in lis_n:
    for j in lis_n:
        coef.create_var(f"a_{i}_{j}")
## Variables b_{h,q}
for h in lis_h:
    for q in range(2,Q+1):
        coef.create_var(f"b_{h}_{q}")
## Introducimos las variables auxiliares
for j in range(1,N+2):
    for q in lis_q:
        coef.create_var(f"aux1_{j}_{q}")
for i in lis_n:
    for j in lis_n:
        coef.create_var(f"aux2_{i}_{j}")
```

Figura 7.3: Fragmento de código de introducción de las restricciones del problema Q-Robots.

[37]:
```
## Introducimos las restricciones
## Restricción 1
for i in lis_n:
    for j in lis_n:
        for q in lis_q:
            for r in lis_r:
                coef[(f"x_{i}_{j}_{r}_{q}",)] += -2*(lambda_1)
            for r1 in lis_r:
                for r2 in lis_r:
                    coef[(f"x_{i}_{j}_{r1}_{q}",f"x_{i}_{j}_{r2}_{q}")] +=  (lambda_1)

## Restriccion 2
for q in lis_q:
    for j in range(1,N+2):
        coef[(f"x_{0}_{j}_{1}_{q}",)] = coef[(f"x_{0}_{j}_{1}_{q}",)] -2*lambda_2
    for j1 in range(1,N+2):
        for j2 in range(1,N+2):
            coef[(f"x_{0}_{j1}_{1}_{q}",f"x_{0}_{j2}_{1}_{q}")] = coef[(f"x_{0}_{j1}_{1}_{q}",f"x_{0}_{j2}_{1}_{q}")]+lambda_2

    ## Restriccion 2.b
    for i in range(0,N+2):
        coef[(f"x_{i}_{0}_{1}_{q}",)] = coef[(f"x_{i}_{0}_{1}_{q}",)] + lambda_2_ext


## Restriccion 3
for q in lis_q:
    for i in range(0,N+1):
        coef[(f"x_{i}_{N+1}_{1}_{q}",)] = coef[(f"x_{i}_{N+1}_{1}_{q}",)] -2*lambda_3
    for i1 in range(0,N+1):
        for i2 in range(0,N+1):
            coef[(f"x_{i1}_{N+1}_{1}_{q}",f"x_{i2}_{N+1}_{1}_{q}")] = coef[(f"x_{i1}_{N+1}_{1}_{q}",f"x_{i2}_{N+1}_{1}_{q}")]+lar
    ## Restriccion 3 extra
    for j in range(N+2):
        coef[(f"x_{N+1}_{j}_{1}_{q}",)] = coef[(f"x_{N+1}_{j}_{1}_{q}",)] + lambda_3_ext


## Restriccion 4
for i in range(1,N+1):
    for q in lis_q:
        for j in range(1,N+2):
            coef[(f"x_{i}_{j}_{1}_{q}",)] +=  -2*(lambda_4 )
    for q1 in lis_q:
        for j1 in range(1,N+2):
            for q2 in lis_q:
                for j2 in range(1,N+2):
                    coef[(f"x_{i}_{j1}_{1}_{q1}",f"x_{i}_{j2}_{1}_{q2}")] += (lambda_4)

## Restriccion 5
for j in range(1,N+1):
    for q in lis_q:
        for i in range(0,N+1):
            coef[(f"x_{i}_{j}_{1}_{q}",)] +=  -2*lambda_5
```

Figura 7.4: Código para la realización de las simulaciones y ploteo del problema Q-Robots.

In [39]:
```
## Cargamos el diccionario con las variables
dwave_dic = {}
for i in coef:
    if len(i) == 1:
        dwave_dic[(i[0],i[0])] = coef[i]
    else:
        dwave_dic[i] = coef[i]

from neal import SimulatedAnnealingSampler

n_samples = 5000 # número de veces que ejecutamos el sistema


sampler = SimulatedAnnealingSampler()  ## LLamada simulacion
#sampler = EmbeddingComposite(DWaveSampler())  ## LLamada ordenador real

sampleset = sampler.sample_qubo(dwave_dic, num_reads = n_samples,auto_scale=True) ## Simulacion

solution = sampleset.first.sample

## Mejor energia
## Matriz solucion
for q in lis_q:
    mat_sol = np.zeros((N+2,N+2))
    for i in  range(N+2):
        for j in range(N+2):
            if solution[f"x_{i}_{j}_{1}_{q}"] == 1:
                mat_sol[i,j] = 1
    print(mat_sol)

    ## Pintamos el camino propuesto
    plt.plot(puntos[:,0],puntos[:,1],'o')
    vaux = np.array(list(range(N+2)))
    suma_ruta = 0
    for i in range(N+1):
        sig_aux = mat_sol[i,:]==1
        if np.sum(sig_aux) > 0:
            sig = (int(vaux[sig_aux][0]))%(N+1)
            plt.plot(puntos[(i,sig),0],puntos[(i,sig),1])
            suma_ruta += np.floor(1000*fnorm(puntos[i,:]-puntos[sig,:]))
    plt.show()
    print("El robot ",q," recorre ",suma_ruta)
```