USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

FACULTADE DE MATEMÁTICAS

Traballo Fin de Grao

Modelos y algoritmos para el problema del viajante. Una aplicación en planificación socio sanitaria.

Dafne Lucía Arias Vilaboa

2020/2021

UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

GRAO DE MATEMÁTICAS

Traballo Fin de Grao

Modelos y algoritmos para el problema del viajante. Una aplicación en planificación socio sanitaria.

Dafne Lucía Arias Vilaboa

JULIO 2021

UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Trabajo propuesto

Área de Conocimiento: Estadística e Investigación Operativa
Título: Modelos y algoritmos para el problema del viajante. **Una aplicación en planificación socio sanitaria.**
Breve descripción del contenido:
En este trabajo se estudia el problema clásico del TSP desde dos enfoques. Primeramente desde su relación con la matemática discreta y en particular con los grafos ponderados, por lo cual se hará una revisión de conceptos y resultados en este contexto. El segundo enfoque enmarca el problema del viajante, TSP, dentro de la programación lineal y entera, de modo que se presentará el modelo de programación del TSP y algoritmos de resolución representativos. En la última parte del trabajo se presentan los denominados problemas de rutas de vehículos, VRP, como una extensión del TSP con restricciones adicionales. Se presentarán modelos del VRP con restricciones de capacidad, flota heterogénea y ventanas de tiempo. Esta teoría se aplicará a un problema práctico de la vida real: se considerará un centro de día de la ciudad de Lugo, orientado al cuidado de personas mayores y se va a estudiar la optimización de las salidas programadas de los usuarios con destino a sus domicilios teniendo en cuenta las restricciones relativas a características de movilidad de los usuarios y los distintos tipos de furgonetas disponibles en la empresa. Junto con la teoría anterior, para la resolución del problema se hará uso del lenguaje de modelado AMPL y del solucionador GUROBI.

Índice general

Resumen

El problema del viajante de comercio y su extensión, el problema de rutas de vehículos, son dos de los problemas de optimización combinatoria de clase NP-duros más estudiados a lo largo del tiempo. Su importancia se debe a que estos problemas cuentan con una gran cantidad de aplicaciones prácticas y el hecho de que sean problemas fáciles de entender pero con una resolución compleja ha motivado su gran investigación.

El objetivo de este trabajo es abordar el estudio de ambos problemas. En el primer capítulo se realizará una revisión bibliográfica de conceptos importantes sobre la teoría de grafos. A continuación, en el segundo capítulo se estudia el TSP así como sus múltiples aplicaciones y métodos de resolución, tanto exactos como heurísticos. Por otro lado, en el tercer capítulo se estudia el VRP de manera similar. En el capítulo final, se presenta una aplicación práctica de todo lo expuesto anteriormente. Nos centramos en el estudio de un problema que presenta un centro de día de la ciudad de Lugo y que se puede modelar siguiendo el esquema de una variante del TSP. Para su resolución se hace uso del modelador AMPL y del solucionador Gurobi a través del servidor de optimización NEOS.

Abstract

The travelling salesman problem (TSP) and its extent, the vehicle routing problem (VRP) are two of the most studied problems in the combinatorial optimization of NP-hard class throughout time. Its importance is due to their wide range of practical applications and the fact that they are easy to understand. However, their complex resolution has motivated research in this field.

The aim of this paper is analysing the study of both problems. In the first chapter, important graph theory concepts will be reviewed. In the second chapter, the TSP will be studied alongside with its multiple applications as well as exact and heuristic resolution methods. In the third chapter, the VRP will be studied likewise. In the final chapter, a

practical application of the previous will be represented. The focus will be the study of a problem in a day care centre in the city of Lugo. This problem will be solved following a scheme of a TSP variant. For its resolution we use the model AMPL and its solver Gurobi through the NEOS optimization server.

Capítulo 1

Introducción a la teoría de grafos

La programación lineal estudia problemas de optimización, es decir, la maximización o minimización, de una función lineal que satisface un conjunto de restricciones lineales de igualdad y/o desigualdad. En ocasiones, estos problemas se pueden representar con el apoyo de un grafo.

La **teoría de grafos** es una rama de las matemáticas que estudia las propiedades de los grafos. Nació en 1736 cuando Leonhard Euler resolvió el conocido *problema de los puentes de Könisgber*. Este problema consistía en encontrar un camino circular por la ciudad de Könisgber, llamada ahora Kaliningrado, de tal manera que solo se podía cruzar una única vez cada uno de los siete puentes que cruzan el río Pregel. Lo vemos representado en la Figura 1.1, donde se puede apreciar tanto el mapa como el grafo que lo representa.

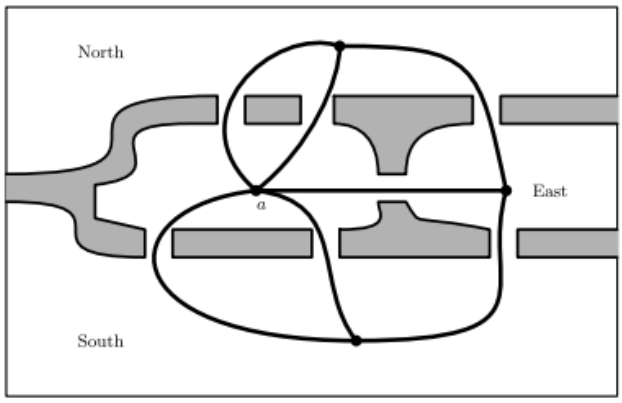


Figura 1.1: Problema de los puentes de Könisgber.

Introducimos a continuación unas nociones básicas, siguiendo la referencia [23], para facilitar la comprensión de este trabajo.

Definición 1.1. Un **grafo** G es un par $G = (V, E)$, donde V es un conjunto finito y E es un conjunto de subconjuntos de dos elementos de V. Los elementos de V se llaman **vértices** o **nodos** y los elementos de E se llaman **aristas** o **arcos** y son de la forma $e = (a, b)$, siendo a y b los vértices extremos de la arista e. Si existe la arista $e = (a, b)$ decimos que los nodos a y b son **adyacentes** o que son **incidentes** con e.

Definición 1.2. Un grafo **orientado** o **dirigido** es aquel en el que las aristas son pares ordenados, es decir, cada arista $e = (a, b)$ empieza en el nodo a y termina en el nodo b. En un grafo no orientado (a, b) y (b, a) representan la misma arista.

Definición 1.3. Un grafo es **completo** si todas las aristas posibles están presentes.

Definición 1.4. Un **subgrafo** de un grafo $G = (V, E)$ es un grafo $G' = (V', E')$ que tiene todos sus vértices y aristas en G, es decir, $V' \subseteq V$ y $E' \subseteq E$.

Definición 1.5. Un **subgrafo de expansión** de $G = (V, E)$ es un subgrafo $G' = (V', E')$ tal que $V' = V$.

Definición 1.6. Sea $G = (V, E)$ un grafo y $V' \subseteq V$. Denotamos por $E \mid V'$ el conjunto de todas las aristas $e \in E$ que tienen ambos vértices en V'. El grafo $(V', E \mid V')$ se denomina **grafo inducido en** V' y se denota por $G \mid V'$.

Definición 1.7. El **grado** de un nodo cualquiera de un grafo G es el número de aristas incidentes en él. Si todos los vértices de un grafo G tienen el mismo grado, G se llama **grafo regular**.

A menudo ilustraremos los grafos con dibujos en el plano. Tal y como se puede ver en las Figuras 1.2 y 1.3 los vértices están representados por puntos, en este caso numerados, y las aristas por líneas. Además en el caso del grafo orientado, las aristas indican un sentido de recorrido.

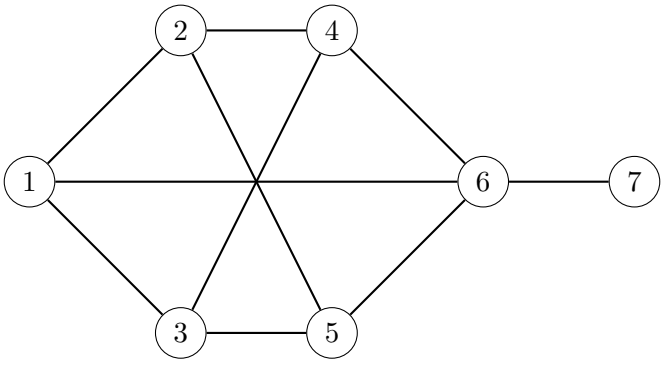


Figura 1.2: Ejemplo de grafo no orientado.

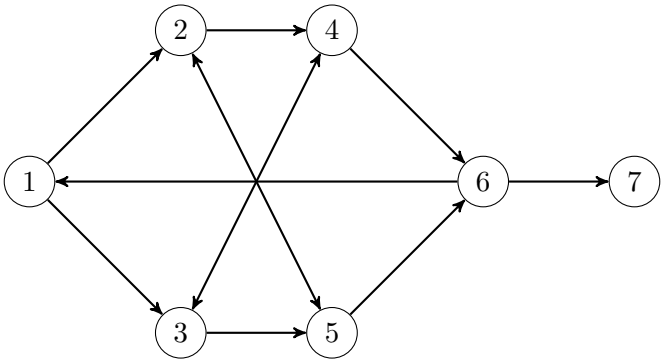


Figura 1.3: Ejemplo de grafo orientado.

A continuación se define qué es una cadena y qué tipos de cadenas hay. De aquí en adelante suponemos grafos no orientados, ya que la generalización a grafos orientados es sencilla.

Definición 1.8. Sea G un grafo no orientado y sea $(a_1, a_2, \ldots, a_n)$ una secuencia de aristas de G. Si existen vértices $v_0, \ldots, v_n$ tales que $a_i = (v_{i-1}, v_i)$ para $i = 1, \ldots, n$ entonces la secuencia es una **cadena**.

Para referirnos a una cadena podemos usar indistintamente la secuencia de las aristas o la secuencia de los nodos que la forman. Los vértices v_0 y v_n se denominan respectivamente **vértice inicial** y **vértice final**.

Necesitamos diferenciar varios tipos de cadena:

- Si $v_0 = v_n$ entonces se llama **cadena cerrada**.
- Una cadena en la que todos los vértices son distintos se llama **camino**.
- Una camino cerrado con $n \geq 3$, que tiene todos sus vértices, v_j, distintos (excepto, por supuesto, $v_0 = v_n$), se llama **circuito** o **ciclo**.

Denotamos como n a la **longitud** de la cadena que será el número de nodos que une la cadena.

En la Figura 1.4 podemos observar un grafo de 6 nodos unidos por un circuito.

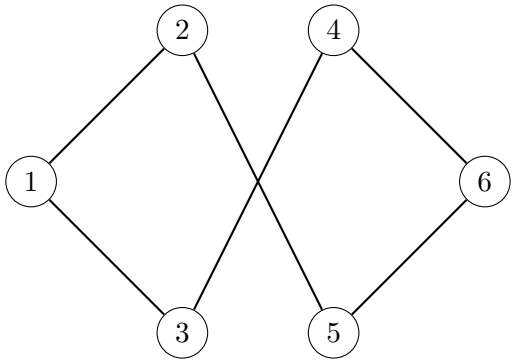


Figura 1.4: Ejemplo de grafo con circuito.

Definición 1.9. Decimos que dos vértices a y b, de un grafo G, están **conectados** si existe una cadena que empieza en el vértice a y finaliza en el vértice b. Si todos los pares de vértices del grafo G están conectados, se dice que el propio grafo G está **conectado**. Para cualquier vértice a, consideramos (a) como un camino trivial de longitud 0, de modo que cualquier vértice está conectado consigo mismo. Por lo tanto, la conectividad es una relación de equivalencia en el conjunto de vértices de G. Las clases de equivalencia de esta relación se denominan **componentes conexas** de G. De esta forma se dice que G es **conexo** si y solo si su conjunto de vértices V es su componente conexa única. Las componentes que contienen solo un vértice se denominan **vértices aislados**.

Algunos resultados que es interesante conocer son:

- Un grafo conexo con n vértices tiene al menos $n-1$ aristas.
- Un grafo acíclico, es decir que no contiene ningún ciclo, con n vértices tiene como máximo $n-1$ aristas.

Para comprender mejor la solución tanto del problema de los puentes de Königsberg como del problema del viajante de comercio, que veremos más adelante, vamos a introducir el teorema de Euler y algunas definiciones de interés.

Vamos a generalizar el concepto de grafo permitiendo que haya pares de vértices que están conectados por más de una arista. Intuitivamente, para un multigrafo sobre un conjunto de vértices V, queremos reemplazar el conjunto de aristas de un grafo ordinario por una familia E de subconjuntos de dos elementos de V.

Definición 1.10. Un **grafo ponderado** es aquel en el que cada arista tiene un peso o un valor asociado. Formalmente es un grafo $G = (V, E, W)$ donde $W = \{w_1, \ldots, w_m\}$ es el conjunto de pesos asociados a cada arista. El **peso** de una cadena o camino es la suma de los pesos de las aristas que la componen.

Definición 1.11. Para poder distinguir diferentes aristas que conectan el mismo par de vértices definimos un **multigrafo** como una terna (V, E, J) donde V y E son conjuntos disjuntos y J es una aplicación de E al conjunto de subconjuntos de dos elementos de V, la aplicación de incidencia. La imagen $J(e)$ de una arista e es el conjunto $\{a, b\}$ de vértices extremos de e. Las aristas e y e' con $J(e) = J(e')$ se llaman **paralelas**.

Para comprender mejor la idea de multigrafo, en la Figura 1.5, se representa un problema de la vida real. Se muestra un grafo que conecta las 5 principales ciudades gallegas por carretera/autovía o autopista (en caso de que exista). Es un claro ejemplo de nodos conectados por varias aristas a las que podemos asignar pesos diferentes, ya que no costará ni llevará el mismo tiempo desplazarse por carretera (C) o desplazarse por autopista (AP). Si necesitamos ir, por ejemplo, de Coruña a Ourense tendríamos que decidir si ir por (C) pasando por Lugo o por Santiago, o bien por (AP) o una combinación de ambas.

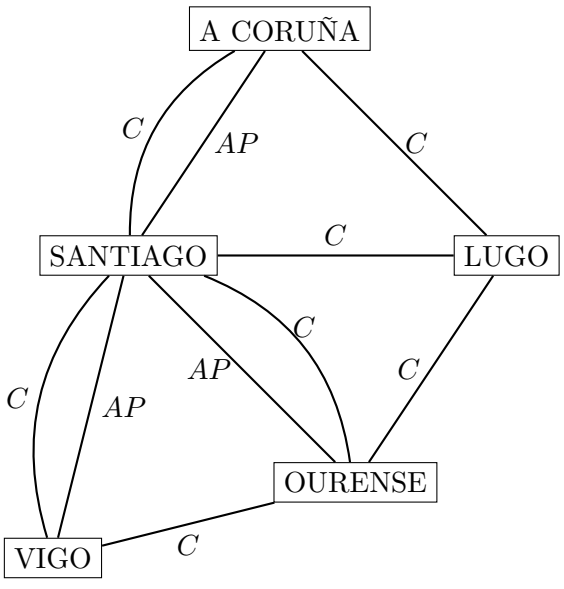


Figura 1.5: Un multigrafo que representa la conexión entre las principales ciudades gallegas por carretera/autovía o autopista.

Este ejemplo de multigrafo es una motivación de los problemas de minimización de rutas que veremos más adelante.

Definición 1.12. Una **cadena euleriana** de un multigrafo G es una cadena que contiene cada arista de G exactamente una vez. Un grafo se llama **euleriano** si contiene una cadena euleriana cerrada.

Teorema 1.13 (Teorema de Euler)**.** *Un multigrafo $G = (V, E, J)$ es Euleriano si y sólo si las siguientes afirmaciones son equivalentes:*

a) G es conexo.

b) Cada vértice de G tiene grado par.

Demostración. Siguiendo la referencia [19] veremos que estas condiciones son necesarias. Por una parte si el grafo no es conexo no hay ninguna cadena cerrada que recorra todos los nodos y, además, en cualquier cadena cerrada tendremos que el número de aristas incidentes en cada nodo es par. La suficiencia se prueba por inducción en el número de aristas de G:
- Para $n = 0$, un multigrafo con 0 aristas que verifica a) y b) tendrá un único nodo y será euleriano.
- Para $n \neq 0$. Supongamos ahora que tenemos un multigrafo G verificando a) y b) y tal que todos los multigrafos con menos aristas que G que verifiquen a) y b) son eulerianos. Elijamos ahora un vértice i de G y empecemos en él una cadena cerrada de aristas de G, que nunca repita dos veces la misma arista; claramente, b) nos asegura que esto será posible. Ahora, eliminemos de G las aristas de esta cadena. Si ya no queda ninguna arista, entonces tendremos una cadena euleriana. En otro caso, tendremos una o más componentes conexas (subgrafos conexos de G). Para cada nodo hemos eliminado una cantidad par de aristas incidentes en él, con lo que cada componente conexa verificará a) y b) y, por inducción, será euleriana. Ahora podemos crear una cadena euleriana en G concatenando las distintas cadenas eulerianas con la cadena de partida, apoyándonos en a).

Proposición 1.14. *Las anteriores afirmaciones también son equivalentes a la siguiente: El conjunto de aristas de G se puede dividir en ciclos.*

Definición 1.15. Un **circuito hamiltoniano** en un grafo G es un circuito que contiene a todos los vértices de G (por ser circuito, solo podrá repetir el primer nodo, que coincidirá con el último). Un grafo que contiene un circuito hamiltoniano se llama **grafo hamiltoniano**.

Aunque las cadenas de Euler y los ciclos hamiltonianos tienen definiciones similares son conceptos bastante diferentes.

Supongamos que tenemos un grafo G en el que cada arista tiene asociado un coste, por ejemplo, el tiempo que se tarda en atravesar dicha arista, y tenemos que encontrar un circuito hamiltoniano que minimice el tiempo que se tarda en recorrer todos los nodos del grafo. Nos encontramos ante uno de los problemas más fundamentales en la teoría de grafos, el **problema del viajante de comercio (TSP)** del que hablaremos extensamente en el siguiente capítulo.

Capítulo 2

Problema del viajante de comercio

En este capítulo se realiza una revisión bibliográfica sobre el problema del viajante de comercio. Comenzamos con una introducción histórica del problema y a continuación se describirá junto con algunas aplicaciones y métodos de resolución.

2.1. Un poco de historia

El problema del viajante del comercio, en inglés *Traveling Salesman Problem (TSP)*, es uno de los problemas de optimización combinatoria NP-duros más importantes, y por tanto, más ampliamente estudiado. El hecho de que sea un problema fácil de entender y a la vez no se haya encontrado una solución general ha hecho que sea uno de los pocos problemas contemporáneos en matemáticas que ya forma parte de la cultura popular.

La simplicidad del TSP, junto con su profunda dificultad, lo convierte en una plataforma ideal para desarrollar ideas y técnicas para poder atacar problemas computacionales en general.

El origen del problema del transporte no está del todo claro, ya que no existe ningún documento oficial donde aparezca el nombre del creador. El problema aparece por primera vez en un trabajo publicado por Karl Menger en 1932 [30], bajo el nombre de *El problema del mensajero*, el cual consistía en encontrar el camino de mínima longitud de manera que uniésemos todos los puntos de un conjunto cuya distancia era conocida. Más adelante, en 1949, en Estados Unidos, se publicó el primer informe usando el nombre "traveling salesman problem" como un problema de optimización numérica, [42].

De forma paralela, a pesar de no conocer quien introdujo este problema en el mundo matemático, su principal difusor fue Merrill Flood. Empezó a investigar sobre el mismo en la Universidad de Princenton. A raíz de sus investigaciones, aparecieron nuevos trabajos e investigadores, como por ejemplo Koopmans, que versionó el TSP al "Problema de los 48

estados" de Hassler Whitney, cuando trataba de encontrar una ruta del autobús escolar en Virginia.

En 1948, Jonh Williams convenció a Flood para que difundiese y popularizase el TSP en la RAND Corporation con el objetivo de crear retos intelectuales para motivar el estudio de modelos fuera de la teoría de grafos. Durante esos años la popularidad del problema fue creciendo entre el círculo de científicos de Europa y Estados Unidos, pero no fue hasta 1954 cuando Dantzig, Fulkerson y Johnson lo publicaron en su obra, [13]. En esta, lo expresaron como un problema de Programación Lineal Entera y desarrollaron el *"método de los Planos de Corte"* para su resolución, con el que resolvieron el problema para 49 ciudades, una por cada estado de Estados Unidos y Washington, creando un recorrido óptimo y probando que no era posible construir otro recorrido mejor.

Otro acontecimiento interesante tuvo lugar en 1972, cuando Richard M. Karp demostró que el *Problema del ciclo de Hamilton* era un problema NP-Completo, y como consecuencia el TSP sería un problema NP-Duro.

Más adelante, en 1987, Grötschel, Padberg, Rinaldi y otros matemáticos consiguieron resolver el problema para 2392 ciudades usando el método de Planos de Corte y Ramificación y Acotación.

En los 90, Applegate, Bixby, Chvátal y Cook desarrollaron un programa, llamado *Concorde TSP Solver* [1], con el cual se resolvió el problema de 33810 ciudades en 2005 y el actual récord en 2006 con 85900 ciudades.

En la Tabla 2.1 observamos la evolución de las investigaciones aumentando el número de ciudades. Es importante destacar la última solución, debido a que ha sido el TSP más grande que se ha resuelto de forma óptima. Como se ha dicho, en 2006 se encontró una solución para un problema de 85900 ciudades. En este caso era una aplicación del TSP a un problema de minimización del tiempo total que empleaba un láser en la elaboración de chips. Las ciudades eran, en este caso, las ubicaciones de interconexiones y el peso de las aristas señalaban el coste de pasar de una interconexión a otra [1].

En 2016, Hans Mittelmann creó un Servidor NEOS para Concorde, que es capaz de resolver problemas tipo TSP simétricos de forma online, [37].

Actualmente existen varios retos abiertos relativos al TSP, uno de los cuales es el "Monalisa TSP Challenge" que podemos encontrar en [35]. Consiste en encontrar la solución óptima para un TSP de 100000 ciudades, de manera que al unirlas mediante una línea continua representan el dibujo de la Mona Lisa de Leonardo da Vinci, como se puede ver en la Figura 2.1. Encontrar una solución óptima para este TSP establecería un nuevo récord mundial. La mejor solución hasta el momento se encontró en 2012 pero no es la óptima. Para motivar su búsqueda se ofrece una recompensa de $1000 a la persona que

consiga mejorarla.

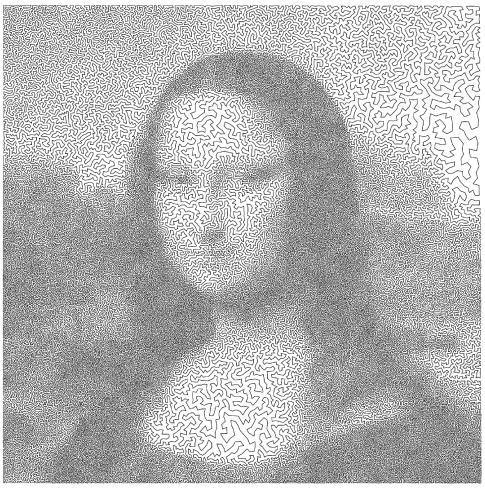


Figura 2.1: "Monalisa TSP Challenge"

En la página web [34] se pueden encontrar otros ejemplos muy interesantes de arte y figuras construidas mediante la resolución de TSP.

AÑO	AUTORES	NÚMERO CIUDADES
1954	G. Dantzig, R. Fulkerson, S. Johnson	49
1971	M. Held , R.M. Karp	64
1975	P.M. Camerini, L. Fratta, F. Maffioli	67
1977	M. Grötschel	120
1980	H. Crowder, M.W. Padberg	318
1987	M. Padberg , G. Rinaldi	532
1987	M. Grötschel , O. Holland	666
1991	M. Padberg , G. Rinaldi	2392
1994	D. Applegate, R. Bixby, V. Chvátal, W. Cook	7397
1998	D. Applegate, R. Bixby, V. Chvátal, W. Cook	13509
2001	D. Applegate, R. Bixby, V. Chvátal, W. Cook	15112
2004	D. Applegate, R. Bixby, V. Chvátal, W. Cook	18512
2004	D. Applegate, R. Bixby, V. Chvátal, W. Cook, K. Helsgaun	24978
2005	W. Cook, D. Espinoza, M. Goycoolea	33810
2006	D. Applegate, R. Bixby, V. Chvátal, W. Cook	85900

Tabla 2.1: Evolución histórica del tamaño del TSP resuelto.

2.2. Descripción del problema

El problema del viajante consiste en encontrar la ruta más corta que debe realizar un comercial de forma que recorra un conjunto de n ciudades, empezando y finalizando en la misma, de modo que la única ciudad que visita más de una vez es la de partida.

Este problema lo podemos expresar de forma matemática usando la teoría de grafos. El problema se traduciría en encontrar el circuito hamiltoniano de menor peso en un grafo ponderado de n nodos. Los nodos representan las ciudades, las aristas son las conexiones entre las ciudades y los pesos de las aristas representan, por ejemplo, la distancia que las separa o el tiempo que se tarda en llegar de una ciudad a otra.

Habitualmente trabajaremos con grafos completos, es decir, cada par de vértices siempre está conectado por una arista, de esta manera, podemos construir la matriz de costes asociada a nuestro problema. En caso de que no existiese el camino entre un par de ciudades, se añade una arista larga para completar el grafo, sin que esta afecte al recorrido óptimo. La matriz de costes es de la forma

$$C = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix},$$

donde cada coeficiente c_{ij} representa el coste de viajar desde la ciudad i a la ciudad j.

Observación 2.1. Es interesante diferenciar si el TSP es simétrico o asimétrico. El problema será simétrico si el peso de la arista (i, j) es el mismo que el peso de la arista (j, i), es decir, si nos encontramos en un grafo no dirigido. Si el grafo es dirigido, la matriz de costes no tendrá por qué ser necesariamente simétrica.

Observación 2.2. Para un conjunto de n nodos tenemos $n!$ rutas posibles. Aunque lo podemos simplificar, ya que, como es un circuito circular el nodo de partida podría ser cualquiera, entonces tendríamos $(n-1)!$ rutas. Además, si estamos ante un grafo no dirigido, cada arista se podría recorrer en ambos sentidos, reduciendo entonces las rutas a la mitad. Tendríamos finalmente $\frac{(n-1)!}{2}$ rutas posibles. Si el TSP es asimétrico, puede no existir caminos en ambas direcciones, por ejemplo, no todas las calles son de doble sentido.

Aún así, el número de rutas crece exponencialmente, ya que, para un TSP de 10 ciudades, existen $10! = 3628800$ rutas, pero simplificando tendríamos $\frac{(10-1)!}{2} = 181440$ rutas diferentes.

En la Figura 2.2 podemos ver un ejemplo de un TSP, donde se ha buscado el circuito hamiltoniano de menor coste posible. Es decir, el recorrido de menor peso de manera que se unan todos los nodos. En este caso, el coste asociado a ese ciclo hamiltoniano es de 6 unidades.

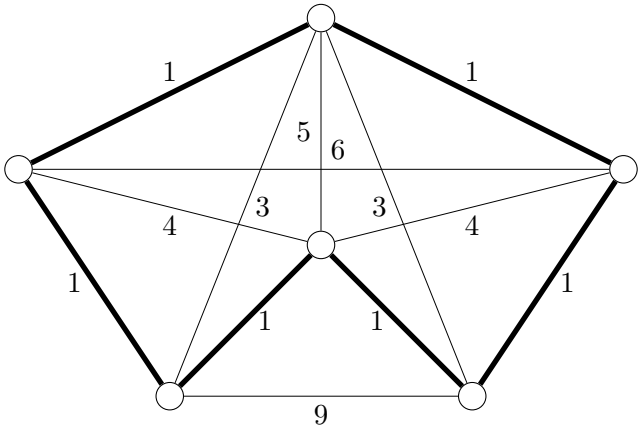
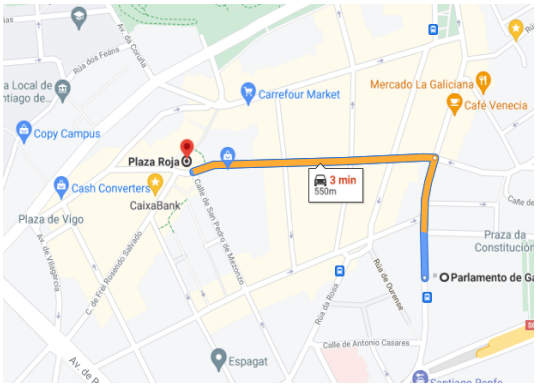
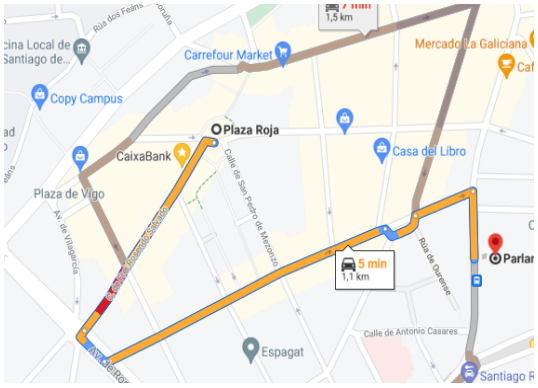


Figura 2.2: Solución de un TSP.

En la Figura 2.3 se puede ver, de forma sencilla, la existencia de un camino asimétrico. El recorrido en coche desde el Parlamento de Galicia a Plaza Roja no es el mismo que el recorrido desde Plaza Roja al Parlamento de Galicia. Esto es debido al sentido de las calles que recorremos.



(a) Recorrido del Parlamento a Plaza Roja.



(b) Recorrido de Plaza Roja al Parlamento.

Figura 2.3: Un camino asimétrico entre dos lugares de Santiago.

Matemáticamente, definimos el problema del viajante como un sistema de optimización lineal entero.

La primera formulación matemática fue realizada en el siglo XIX por W.R. Hamilton y T. Kirkman. Actualmente, las tres formulaciones más comunes son la de Dantzig-Fulkerson-Jonhson (DFJ), la de Miller-Tucker-Zemlin (MTZ) y la formulación basada en el flujo de redes. La principal diferencia entre ellas es la definición de la restricción encargada de que no se formen subciclos. A pesar de ser matemáticamente equivalentes, su rendimiento es diferente ya que estamos ante un problema computacionalmente complejo.

De aquí en adelante utilizaremos la formulación MTZ [31] por ser la que tiene mejor rendimiento y obtiene resultados de forma más eficiente a medida que aumentan los nodos del problema.

Definición 2.3 (TSP-Formulación MTZ). Sea $C = (c_{ij})$ una matriz de costes, donde c_{ij} representa el coste de ir del nodo i al nodo j para $i, j = 1, ..., n$, es decir, el peso de la arista (i, j). Se trata de resolver el siguiente problema de optimización:

$$\text{mín} \qquad \sum_{i=1}^{n} \sum_{j=1, i\neq j}^{n} x_{ij} \cdot c_{ij} \tag{2.1}$$

sujeto a:

$$\sum_{i=1, i\neq j}^{n} x_{ij} = 1 \qquad j = 1, \ldots, n \tag{2.2}$$

$$\sum_{j=1, i\neq j}^{n} x_{ij} = 1 \qquad i = 1, \ldots, n \tag{2.3}$$

$$u_i - u_j + n \cdot x_{ij} \leq n - 1 \qquad 2 \leq i \neq j \leq n \tag{2.4}$$

$$\text{donde } x_{ij} = \begin{cases} 1, & \text{si la ruta incluye la arista del nodo } i \text{ al } j \\ 0, & \text{en otro caso} \end{cases}$$

y u_i es una variable entera que representa, para cada nodo, el número de nodos que le precede en el ciclo creado.

La ecuación (2.1) es la función objetivo del problema. La ecuación (2.2) garantiza que solo se llegue a cada ciudad una vez. La ecuación (2.3) garantiza que solo se salga de cada ciudad una vez. La ecuación (2.4) obliga a que todas las ciudades se conecten por un solo camino y no varios, además anula la formación de ciclos.

Observación 2.4. En el caso de que no se pudiese ir de un nodo a otro, esa arista tendría coste infinito. Es decir, si no existe $(\hat{i}, \hat{j})$ entonces $c_{\hat{i}\hat{j}} = \infty$.

Observación 2.5. La diferencia entre la formulación MTZ y DFJ consistiría en cambiar la restricción (2.4) por la (2.5) que definimos a continuación:

$$\sum_{i \in S} \sum_{i \neq j, j \in S} x_{ij} \leq |S| - 1 \qquad \begin{array}{l} \forall S \subsetneq \{1, \cdots, n\}, \\ card(S) \geq 2. \end{array} \tag{2.5}$$

2.2.1. Algunas aplicaciones prácticas del TSP

El problema del viajante ha sido ampliamente estudiado debido a que la mayoría de las investigaciones están motivados por problemas de la vida real. Las aplicaciones del TSP son muy extensas, desde las más simples, como puede ser la optimización de las rutas de los buses escolares que estudió Flood [14] en los años cuarenta o el reparto de mercancías o correo, hasta algunas más complejas como son los problemas de *scheduling* o la secuenciación del genoma.

A continuación vemos algunas aplicaciones del TSP en la vida real:

- Rutas turísticas y viajes de negocios. Es la aplicación más obvia, puesto que se corresponde directamente con la definición del TSP. Al planear un viaje se busca la manera más eficiente para conocer todos los puntos de interés del lugar que visitamos. Aunque mucha gente no sea consciente, se está haciendo uso del TSP.

- Problemas de *scheduling.* Quizás el área de aplicación del TSP más estudiada sea la secuenciación y programación de máquinas. Los problemas de *scheduling* consisten en organizar un conjunto de operaciones, con una cantidad limitada de recursos. Deben satisfacer una serie de restricciones y el objetivo suele ser la minimización de tiempo. Son problemas que aparecen a menudo en los procesos de producción y organización de las empresas.

- Redes y telecomunicaciones. El TSP se ha usado para colocar cables de forma estratégica y así suministrar energía para garantizar las conexiones de fibra óptica en los hogares.

- Red de recolección de residuos. Ha sido una de las primeras aplicaciones prácticas del problema.

- Placa de circuitos impresos. La aplicación del TSP para la realización de placas de circuitos ha sido un gran avance en la industria, ya que se enfoca en dos subproblemas, por un lado, el problema de perforado de las placas y por otro lado, el problema de conexión de chips, cuyo objetivo es encontrar la cantidad mínima de cable que se necesita para unir todos los puntos de la placa sin que haya interferencias.

- Secuenciación del genoma. El uso del TSP ayudó en gran medida a descifrar el mapa genético del gato doméstico, el séptimo genoma descifrado de un mamífero. Al descifrar el mapa genético de este mamífero, que consta de 20285 genes, se produjeron grandes avances en la medicina, ya que muchas de las enfermedades de los gatos aparecen en los humanos de forma similar.

- Entrega de correo. Las empresas de transporte usan programas y algoritmos a diario para el reparto de paquetería.

Observación 2.6. Quizás el mayor TSP planteado actualmente sea el estudio realizado por la Agencia Espacial Europea (ESA) con el proyecto de su misión Gaia [39], cuyo objetivo es hacer el mapa tridimensional más grande y preciso de nuestra galaxia. Para ello, están resolviendo un TSP en 3D con 1.33 billones de estrellas [4].

En 1972 se demostró que el TSP es un problema de tipo NP-duro, por lo que los métodos de resolución no son sencillos. Debemos diferenciar dos enfoques para abordar el problema.

1. La primera será mediante la aplicación de métodos exactos, pero la búsqueda de soluciones se limita a problemas de pequeño tamaño.

2. La segunda será utilizando métodos heurísticos o de aproximación, los cuales, aunque no son exactos, nos proporcionan soluciones muy buenas para problemas de mayor tamaño.

2.3. Métodos exactos

La forma más evidente de atacar el TSP sería calcular las $n!$ posibles rutas y sus costes asociados y ver cual de ellas tiene menor peso. Este método se llama *Búsqueda por Fuerza Bruta*. Es fácil darse cuenta de que aunque es el método más obvio, sería inviable utilizarlo para resolver cualquier TSP un poco grande, puesto que para un problema de 10 ciudades, habría que calcular las $10! = 3628800$ rutas posibles, es decir, más de 3 millones

de iteraciones. El tiempo de ejecución de este método es un factor polinómico de orden $\mathcal{O}(n!)$.

Visto que este método es inviable para resolver un TSP de gran magnitud, aparecen otros métodos exactos como son:

1. Algoritmos de ramificación y acotación. Estos pueden resolver problemas de entre 40 y 60 ciudades en un tiempo razonable. En el siguiente apartado realizaremos un breve resumen de este algoritmo por ser el primero utilizado para la resolución del TSP.

2. Algoritmos de mejoras progresivas (iterativas) usando técnicas de programación lineal, trabajan bien en TSP de hasta 200 ciudades.

3. Implementación de ramificación y acotación y un problema específico de generación de cortes: ramificación y poda. Este algoritmo funciona bien con grandes problemas y se ha utilizado para resolver el récord vigente, considerando 85900 ciudades, conseguido por Applegate et al. en 2006 [1].

4. Planos de corte de Dantzig, Fulkerson y Jonhson (1954) [13] basados en la programación lineal.

2.3.1. Ramificación y acotación

El *método de ramificación y acotación*, o también conocido por su nombre en inglés *branch and bound*, debe su origen al TSP. Es un método que se aplica en problemas de programación lineal entera con restricciones. Este método debe su nombre a Little, Murty, Sweeney y Karel (1963), que en su obra [26] lo describen como una extensión del método de planos de corte.

Este método consiste en realizar una búsqueda exhaustiva, de forma organizada, de la mejor solución en un conjunto específico. Cada paso de ramificación divide el espacio de búsqueda en dos o más subconjuntos, de modo que el problema se va dividiendo en problemas cada vez más pequeños cuya solución será más sencilla de encontrar que en el problema original.

Veamos un ejemplo sencillo. Supongamos que tenemos un TSP de un conjunto de ciudades de España. No sabemos si debemos viajar o no entre dos ciudades, sean por ejemplo, Barcelona y Madrid. Podemos dividir el conjunto de todas las rutas en aquellas que conecten directamente Barcelona y Madrid y las que no. Al hacer repetidamente tales pasos de ramificación creamos una colección de subproblemas que necesitan ser resueltos, cada uno definido por un subconjunto de rutas que incluyen ciertos caminos y excluyen

otros. Antes de buscar un subproblema y posiblemente dividirlo más, se calcula un límite sobre el coste de las rutas en su subconjunto. En este ejemplo, un límite sencillo puede ser la suma de los gastos de viaje en todos los tramos de los recorridos que hemos decidido hacer. El propósito del paso delimitador es tratar de evitar una búsqueda sin mejora de un subproblema que no contiene mejor solución que las que ya hemos descubierto. La idea es que si el límite es mayor o igual al costo de un tour que ya hemos encontrado, entonces podemos descartar el subproblema sin peligro de perdernos un camino mejor.

2.4. Algoritmos heurísticos

Debido a que el TSP se ha calificado como un problema NP-duro, muchas veces no somos capaces de crear algoritmos para encontrar la solución exacta en tiempo polinómico. Entonces se recurre a métodos heurísticos, es decir, métodos que pueden encontrar soluciones muy buenas, pero que no son necesariamente la óptima, con un coste computacional razonable.

Los algoritmos heurísticos suelen ser *ad hoc*, lo que quiere decir que en general cada método se diseña especialmente para abordar un tipo de problema específico. Suelen ser algoritmos iterativos donde cada iteración implica la búsqueda de una solución mejor que las calculadas anteriormente.

Aunque el TSP es computacionalmente complejo, existe una gran cantidad de métodos heurísticos para su resolución, como por ejemplo, el algoritmo de Christofides (es el algoritmo aproximado que mejor funciona actualmente para la resolución del TSP), el método de búsqueda tabú, el algoritmo genético, el método de aproximación de Vogel, el algoritmo de Clarke and Wright, ...

A continuación, siguiendo la referencia [21], presentaremos a modo ilustrativo uno de estos algoritmos: el método de búsqueda tabú.

2.4.1. Búsqueda tabú

La búsqueda tabú es un método heurístico utilizado para resolver problemas de optimización combinatoria, como el TSP. Usa un procedimiento de búsqueda local o por vecindades para moverse desde una solución inicial factible x hacia una solución $x*$ mejorada hasta que se cumpla algún criterio de detención. La búsqueda local funciona como un procedimiento de mejora local (excepto que no requiere que cada nueva solución de prueba sea mejor que la solución de prueba anterior) de manera usual, es decir, en cada iteración se busca una solución mejor que la anterior para encontrar un óptimo local.

La estrategia más importante de la búsqueda tabú es que permite “movimientos sin

mejora", de esta manera puede escapar de óptimos locales y continuar estratégicamente hacia soluciones mejores. Para evitar que se forme un ciclo que nos aleje del óptimo local, este algoritmo prohíbe los movimientos que puedan regresar a una solución ya visitada. De esta manera se crea una *lista tabú* que registra esos movimientos prohibidos, llamados *movimientos tabú*. Se produce una excepción cuando un movimiento tabú es mejor que la mejor solución factible que se haya encontrado hasta ese momento.

ESQUEMA DEL ALGORITMO DE BÚSQUEDA TABÚ BÁSICO:

PASO INICIAL	Se comienza dando una solución de prueba inicial factible.
ITERACIÓN	Se utiliza un método de búsqueda local diseñado para realizar movimientos adecuados en el vecindario local de la solución de la prueba actual. Está prohibido considerar un movimiento de la lista tabú, excepto que ese movimiento proporcione una mejor solución que la mejor encontrada hasta el momento. Vemos cual de los movimientos restantes mejora esa solución y la fijamos como la próxima solución de prueba, sin importar si esta es mejor o peor que la solución de prueba actual. Se actualiza la lista tabú para evitar el regreso a la última solución de prueba actual. En el caso de que la lista tabú esté completa, se borrará el elemento más antiguo con el objetivo de dar más flexibilidad a los movimientos futuros.
REGLA DE DETENCIÓN	La búsqueda se detiene cuando se cumple el criterio de detención, que puede ser un número fijo de iteraciones, un tiempo limitado o un número de iteraciones consecutivas que no mejoren la mejor solución dada, es decir que no puedan mejorar la solución de la función objetivo. Cuando no existan movimientos factibles en la vecindad local de la solución de prueba actual, el algoritmo se detendrá y la solución final será la mejor solución que se haya encontrado durante todo el proceso.

Este algoritmo hace uso de 3 conceptos importantes:

1. El uso de memoria, que es una característica distintiva de la búsqueda tabú, ya que es imprescindible para dirigir la búsqueda usando las listas tabú registradas.

2. La intensificación, que consiste en volver a la parte más prometedora de la región factible (ya analizada), para explorarla de nuevo de forma más exhaustiva.

3. La diversificación, que implica forzar la búsqueda a entrar en áreas de la región factible que no se hayan explorado anteriormente.

Este método nos proporciona una estructura general y directrices estratégicas para desarrollar un método heurístico para un problema particular. Sin embargo, no responde a algunas preguntas importantes que debemos de tener en cuenta para atacar el problema específico. Estas cuestiones serán: ¿qué procedimiento de búsqueda local debemos utilizar?, ¿cómo se define la vecindad y qué soluciones son vecinas inmediatas?, ¿cómo se colocan los movimientos tabú en la lista?, ¿cuál movimiento tabú se coloca en la lista tabú en cada iteración?, ¿cuanto puede moverse un movimiento en la lista tabú? y ¿qué regla de detención debemos usar? Todas estas preguntas debemos responderlas antes de iniciar el algoritmo.

A continuación aplicamos el método de búsqueda tabú a un sencillo ejemplo para facilitar su comprensión.

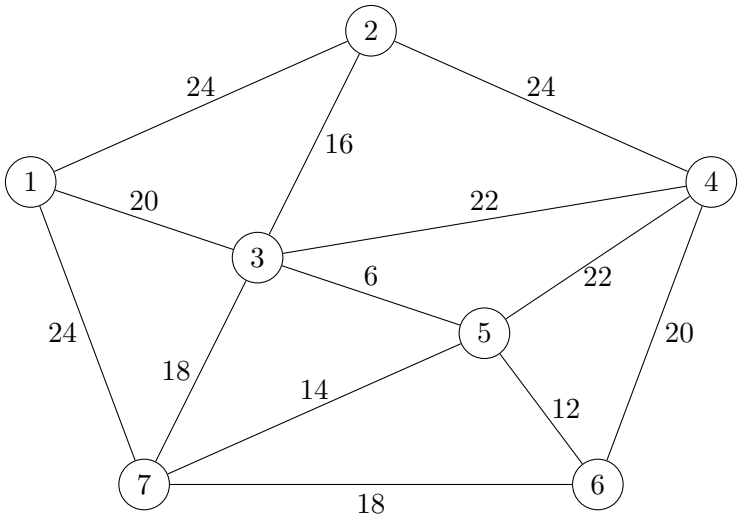


Figura 2.4: Ejemplo de TSP.

Para el TSP representado en la Figura 2.4 realizamos un procedimiento de búsqueda local intuitivo para añadir o eliminar aristas en la solución de prueba actual para obtener una nueva solución de prueba.

Antes de empezar con el procedimiento debemos responder a las seis preguntas formuladas anteriormente:

1. Procedimiento de búsqueda local: en cada iteración se optará por el mejor vecino inmediato de la solución de prueba actual (si no está en la lista tabú).

2. Estructura de vecindad: un vecino inmediato será el que se obtiene invirtiendo la forma de recorrer una sucesión de aristas consecutivas. Al realizar esta inversión se crearán dos ligaduras y se romperán otras dos.

3. Forma de los movimientos tabú: se enumerarán los movimientos tabú para evitar que se regrese a la solución de prueba anterior.

4. Adicción de un movimiento tabú en la lista: se añadirán a la lista las dos ligaduras que se incorporan en la actual iteración.

5. Tamaño máximo de la lista tabú (L.T.): en este caso será cuatro. Cada vez que se añadan ligaduras a una lista tabú llena se eliminarán las dos más antiguas.

6. Regla de detención: se detendrá el proceso después de tres iteraciones sin mejora.

Procedemos ahora a la aplicación del algoritmo.
En el primer paso damos una solución inicial, la más obvia y calculamos el coste asociado a esa solución.

Solución de prueba inicial: 1-2-3-4-5-6-7-1	
Lista tabú	ø
Distancia	138

Es obvio que existen rutas con un coste menor. A simple vista podemos observar que la arista con menor peso es 3-5 entonces, debemos de hacer un cambio para añadir esa arista a nuestra siguiente solución de prueba.

Iteración $K = 1$	
Se elige invertir la ligadura 3-4	
Ligaduras rotas	2-3 y 4-5
Ligaduras agregadas	2-4 y 3-5
Lista tabú	2-4 y 3-5
Nueva solución de prueba	1-2-4-3-5-6-7-1
Distancia	130

Hemos reducido un coste de 8 unidades. Buscamos otra solución vecina que pueda mejorar la solución de prueba actual.

Iteración $K = 2$	
Se elige invertir la secuencia 3-5-6	
Ligaduras rotas	4-3 y 6-7 (no están en la L.T)
Ligaduras agregadas	4-6 y 3-7
Lista tabú	2-4, 3-5, 4-6 y 3-7
Nueva solución de prueba	1-2-4-6-5-3-7-1
Distancia	128

El algoritmo intenta escapar de ese óptimo local buscando nuevas soluciones, aunque empeoren la distancia. Nos movemos hacia el mejor vecino inmediato aunque la distancia sea mayor.
En este caso, debido a la limitada cantidad de ligaduras que posee el problema, solamente tenemos dos vecinos inmediatos. En este caso serán:

- Se elige invertir 6-5-3. Daría como resultado la solución de prueba de la iteración K=2 y el valor de la distancia sería 130. Está prohibido volver a esta solución porque las ligaduras 4-6 y 3-7 están en la lista tabú.
- Se elige invertir 3-7. Daría como resultado una solución peor, aumentando la distancia a 132. Es la única opción que se puede elegir al descartar la anterior.

Iteración $K = 3$	
Se elige invertir la ligadura 3-7	
Ligaduras rotas	5-3 y 7-1
Ligaduras agregadas	5-7 y 3-1
Lista tabú	4-6, 3-7, 5-7 y 3-1
Nueva solución de prueba	1-2-4-6-5-7-3-1
Distancia	132

En esta iteración, como el tamaño de la lista tabú es cuatro, se eliminaron las dos ligaduras más antiguas, en este caso 2-4 y 3-5.
Esta nueva solución de prueba tiene cuatro vecinos inmediatos. En este caso serán:

- Se elige invertir 2-4-6-5-7. Daría como resultado la solución de prueba 1-7-5-6-4-2-3-1 y el valor de la distancia sería 130, un resultado peor que en la solución de prueba de la iteración K=2.

- Se elige invertir 6-5. Daría como resultado la solución de prueba 1-2-4-5-6-7-3-1 y el valor de la distancia sería 138. Se descarta esta solución porque las ligaduras utilizadas están en la última lista tabú.

- Se elige invertir 5-7. Daría como resultado la solución de prueba 1-2-4-6-7-5-3-1 y el valor de la distancia sería 126. De momento es la mejor solución de prueba.

- Se elige invertir 7-3. Daría como resultado la solución de prueba 1-2-4-6-5-3-7-1 y el valor de la distancia sería 128. Se descarta esta solución porque las ligaduras utilizadas están en la última lista tabú.

En este momento, solo podríamos decidir entre la primera y la tercera opción. Se elegirá la tercera por ser la mejor solución de las dos.

Iteración $K = 4$	
Se elige invertir la ligadura 5-7	
Ligaduras rotas	6-5 y 7-3
Ligaduras agregadas	6-7 y 5-3
Lista tabú	5-7, 3-1, 6-7 y 5-3
Nueva solución de prueba	1-2-4-6-7-5-3-1
Distancia	126

En esta última iteración se han eliminado las dos ligaduras más antiguas de la lista tabú, que son 4-6 y 3-7.
La nueva solución de prueba 1-2-4-6-7-5-3-1 tiene la mejor distancia entre todas las que hemos calculado. Esta solución es, en realidad, la solución óptima. Las soluciones vecinas inmediatas están descartadas ya por estar las ligaduras en la lista tabú o por ser soluciones ya visitadas. Como no existe otro vecino inmediato, la regla de detención da por finalizado el algoritmo, siendo la solución calculada en la iteración K=4 la óptima. En la figura 2.5 se representa la solución final.

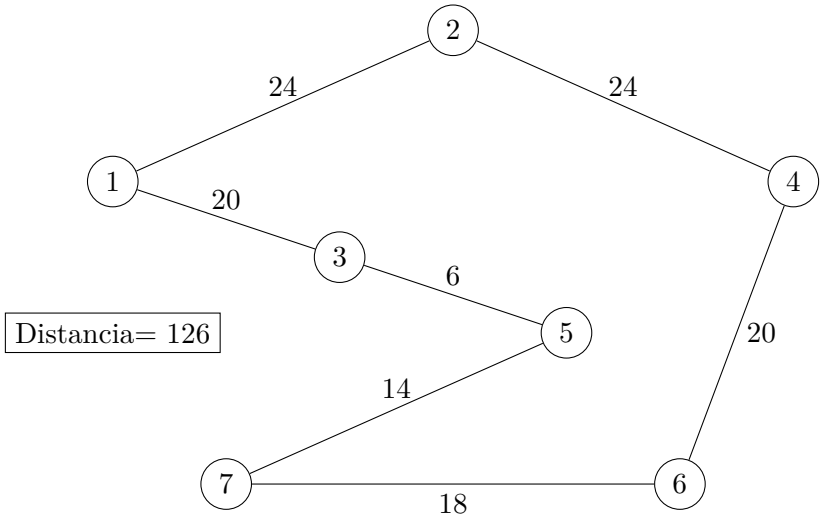


Figura 2.5: Solución del problema.

Capítulo 3

Problemas de rutas de vehículos

Tanto los problemas del viajante como los de rutas de vehículos han sido los problemas más estudiados a lo largo del tiempo en Investigación Operativa.

Por un lado, en el capítulo anterior, hemos visto que el *problema del viajante*, TSP, se basa en encontrar la ruta de distancia mínima, partiendo de un lugar y regresando al mismo de modo que un comercial visite a cada uno de sus clientes y regrese a su punto de partida. Por otro lado, el *problema de rutas de vehículos*, en inglés *Vehicle Routing Problem (VRP)*, tiene como objetivo encontrar el conjunto de rutas, de menor coste, para que la flota de vehículos de una empresa reparta entre sus clientes una mercancía de manera que los repartidores salgan desde el almacén de la empresa, satisfagan todas las necesidades de sus clientes y vuelvan al almacén.

Es fácil darse cuenta que ambos problemas están muy relacionados entre sí. El VRP surge como una extensión del TSP para un determinado caso en el que existe una flota de vehículos y la capacidad de los vehículos es limitada, por lo que, sería necesario realizar varias rutas.

De la misma manera que con el TPS, encontrar una solución óptima para el VRP se considera un problema NP-duro, ya que, no es posible resolverlo en un tiempo polinómico. Casi siempre se recurre a usar métodos heurísticos o meta-heurísticos, debido a que en problemas de gran tamaño, cómo los de la vida real, suelen dar muy buenos resultados.

Entre las diversas referencias sobre el VRP debemos destacar los libros [45], [18] y [9], ya que han sido las principales fuentes de este capítulo.

3.1. Un poco de historia

Las primeras referencias y aplicaciones prácticas en el campo del VRP aparecen por primera vez en un artículo escrito por Dantzig y Ramser en 1959 [11], donde tratan de

encontrar, planteando una aproximación algorítmica, una solución para un problema de reparto de gasolina. Este problema tenía como objetivo encontrar el conjunto de rutas óptimas para que una flota de camiones abasteciese de gasolina, desde un único depósito, a una gran cantidad de estaciones de servicio. En concreto se encontró la solución para 20 estaciones. Desde que Dantzig y Ramser introdujeron el VRP, muchos matemáticos empezaron a estudiarlo de forma masiva. En 1960, Miller, Tucker y Zemlin [31], dan una formulación formal al TSP con múltiples vehículos, casi similar a la definición de VRP.

Más adelante, en 1964, Clarke y Wright [8] mejoraron la aproximación de Dantzig y Ramser utilizando una aproximación "greedy" conocida como algoritmo de ahorros.

3.2. El problema

El problema de rutas de vehículos es un problema de optimización combinatoria cuyo objetivo es encontrar el conjunto de rutas de mínimo coste para que una empresa satisfaga las demandas de sus clientes, saliendo y regresando del mismo depósito.

Para poder definir un problema de rutas de vehículos necesitamos los siguientes elementos:

1. La existencia de un depósito o conjunto de depósitos.
2. Un conjunto de demandas, es decir, clientes.
3. Una flota de vehículos para su transporte.

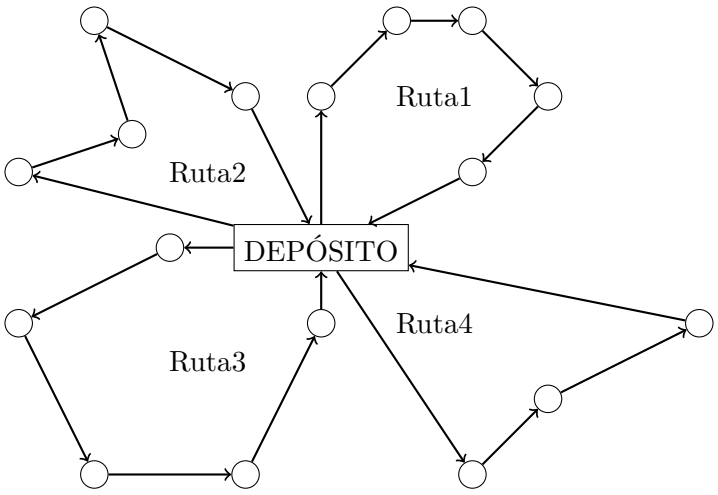


Figura 3.1: Representación de un VRP.

En la Figura 3.1 se representa un esquema básico para un VRP. De un depósito central, deben partir una flota de vehículos que tienen que satisfacer las demandas de sus clientes y volver al depósito. El objetivo del VRP es encontrar el conjunto de rutas que minimicen los costes de transporte.

3.3. Formulación matemática del problema

A continuación, definimos los conjuntos, parámetros y variables utilizados para poder modelar el problema de forma matemática.

- $V = \{0, 1, \ldots, n\}$ es el conjunto de vértices que representan las ubicaciones del problema.
- $N = \{1, \ldots, n\}$ es el conjunto de clientes.
- $i = 0$ representa el almacén o depósito.
- $A = \{(i, j)/i, j \in V, i \neq j\}$ es el conjunto de aristas que unen los vértices.
- $C = (c_{ij})$ es la matriz que representa los costes de desplazamiento entre el nodo i y el nodo j con $(i, j) \in A$.
- $M = (1, \ldots, m)$ es el conjunto de vehículos de los que disponemos (suponemos que todos generan los mismos costes).
- Q representa la capacidad máxima de los vehículos (supongamos flota homogénea).
- $q_i, i \in N$ representa la demanda de cada cliente, es decir, la cantidad de mercancía que debemos suministrarle.
- $\delta^+(i) = \{j \in V, (i, j) \in A\}$ y $\delta^-(j) = \{i \in V, (i, j) \in A\}$ representan los conjuntos de aristas elegidas, de salida y de entrada en los vértices respectivamente.

A partir de la formulación del TSP se construye la siguiente formulación para el VRP.

$$\text{mín} \quad \sum_{i=0}^{n}\sum_{j=0}^{n}\sum_{k=1}^{m} c_{ij} \cdot x_{ijk} \tag{3.1}$$

sujeto a

$$\sum_{k\in M}\sum_{j\in\delta^+(i)} x_{ijk} = 1 \qquad i \in N \tag{3.2}$$

$$\sum_{i\in\delta^-(j)} x_{ijk} - \sum_{i\in\delta^+(j)} x_{jik} = 0 \qquad j \in N, k \in M \tag{3.3}$$

$$\sum_{j\in\delta^+(0)} x_{0jk} = 1 \qquad k \in M \tag{3.4}$$

$$\sum_{i\in\delta^-(0)} x_{i0k} = 1 \qquad k \in M \tag{3.5}$$

$$\sum_{i\in N} q_i \sum_{j\in\delta^+(i)} x_{ijk} \leq Q \qquad k \in M \tag{3.6}$$

$$u_{ik} - u_{jk} + n \cdot x_{ijk} \leq n + 1 \qquad i, j \in N, K \in M \tag{3.7}$$

donde $x_{i,j,k} = \begin{cases} 1, & \text{si se va desde el nodo } i \text{ al nodo } j \text{ con el vehículo } k \in M \\ 0, & \text{en otro caso} \end{cases}$

y $u_{ik} \geq 0$ y enteras

La ecuación (3.1) representa la función objetivo del VRP. La ecuación (3.2) garantiza que cada cliente solo puede ser visitado una única vez por un único vehículo. La ecuación (3.3) representa la conservación del flujo, es decir, si un vehículo visita a un cliente este vehículo también debe abandonarlo. La ecuación (3.4) garantiza que todos los vehículos tienen que partir desde el almacén. La ecuación (3.5) garantiza que todos los vehículos deben de acabar sus rutas en el almacén. La ecuación (3.6) indica que se tiene que respetar la capacidad máxima de los vehículos. La ecuación (3.7) garantiza que no se formen ciclos indeseados.

3.4. Variantes del VRP

Como hemos comentado antes, el gran interés de los investigadores matemáticos sobre las diferentes variantes del VRP no solo está motivado por su gran dificultad en cuanto a la optimización combinatoria, sino también, por su gran importancia en las aplicaciones prácticas.

En esta sección se hace un resumen de las variantes del VRP más importantes, siguiendo las referencias [18] y [22].

Para su clasificación debemos tener en cuenta las características de los depósitos, clientes y la flota de vehículos ya que darán lugar a las diferentes restricciones del VRP. Estas restricciones transformarán el problema original en variantes específicas con un método de resolución propio. Algunas de estas características son:

- La capacidad limitada de los vehículos.
- Las características de nuestra flota de vehículos (homogénea o heterogénea, ver Obs. 3.1).
- Los costes que lleva asociado cada vehículo (por ejemplo, no consumirá lo mismo una moto, un avión o un trailer).
- La capacidad limitada de los depósitos.
- La ubicación de los depósitos.
- Las franjas horarias en las que debe de ser visitado cada cliente.
- La cantidad de puntos de suministro con los que contamos.
- La existencia de variables aleatoria (variación diaria del número de clientes, las demandas, ...).
- Horarios o entregas periódicas de los clientes.

Observación 3.1. Cuando tenemos una flota de vehículos donde todos poseen las mismas características, es decir, la misma capacidad y los mismos costes asociados se dice que tenemos una *flota homogénea*. Si hay diferencias en las características decimos que tenemos una *flota heterogénea*.

Para resumir, siguiendo el trabajo de Toth y Vigo [45] se representan en la Tabla 3.1, las variantes del VRP más comunes.

NOMBRE DE LA VARIANTE	PARTICULARIDAD
AVRP *(Asymmetric VRP)*	El coste de desplazamiento entre dos puntos depende del sentido de recorrido.
CVRP *(Capacitated VRP)*	El vehículo tiene una capacidad limitada.
FRP *(Fixed Routes Problem)*	Una vez se han fijado las rutas no se pueden cambiar durante un tiempo.
FSMVRP *(Fleet Size and Mix VRP)*	La flota de vehículos es ilimitada y diversa. Cada tipo de vehículo tiene un coste fijo o costes variables homogéneos.
DCVRP *(Distance Constrained VRP)*	La distancia total recorrida o el número de clientes visitados está fijada de antemano.
DVRP *(Dynamic VRP)*	Algunos parámetros varían en función del tiempo.
MCVRP *(Multi Compartment VRP)*	Los vehículos están organizados mediante compartimentos. Transportan varios productos pero estos deben de ir en compartimentos separados.
MDVRP *(Multiple Depot VRP)*	Existe un conjunto de depósitos y los vehículos tienen un depósito origen y depósito destino fijados.
OVRP *(Open VRP)*	Algunos vehículos de la flota no necesitan terminar en el depósito.
PVRP *(Periodic VRP)*	Hay un tiempo establecido para atender las necesidades de cada cliente.
VRPLC *(VRP with Length Constraint)*	Está establecida una distancia máxima para cada ruta.
VRPMT *(VRP with Multiple Travel)*	Cada vehículo puede realizar más de una ruta en un cierto período de tiempo.
VRPPC *(VRP with Precedent Constraints)*	Existen relaciones de precedencia. Antes de visitar a un cliente, se debe visitar un grupo de ellos.
VRPPD *(VRP with Pickups and Deliveries)*	El vehículo debe de recoger mercancía en un punto y llevarla a otro.
VRPSD *(VRP with Split Delivery)*	Si se necesitan varios vehículos para satisfacer las demandas de un cliente. La mercancía puede estar dividida en varios vehículos que van por diferentes rutas.
VRPSF *(VRP with Satellite Facilities)*	Si existen depósitos intermedios sin necesidad de volver al depósito inicial para recargar mercancía.

Tabla 3.1: Algunas variantes importantes del VRP.

3.5. Métodos de resolución

Como se ha comentado al inicio del capítulo, para resolver el VRP casi siempre se recurre al uso de métodos heurísticos o meta-heurísticos ya que nos permiten encontrar soluciones casi óptimas en tiempos computacionales razonables. Los problemas de rutas de vehículos, en general, tienen un gran historial de implementaciones meta-heurísticas exitosas.

A continuación, siguiendo la referencia [18], citaremos y explicaremos brevemente la idea principal de las meta-heurísticas más populares para la resolución de este tipo de problemas.

- **Optimización por colonias de hormigas.** Este método meta-heurístico se inspiró en una metáfora natural: "los mecanismos de comunicación y cooperación entre hormigas para encontrar la ruta más corta desde el hormiguero hasta las fuentes de alimento". En este caso en particular, el medio de comunicación de las hormigas es el rastro de feromonas que desprenden al desplazarse. De esta manera, cada vez que una hormiga encuentra un rastro de feromonas continua por él, intensificando así la cantidad de feromonas del camino. Esto llama la atención de otras hormigas, atraídas por la sustancia, aumentando el número de hormigas que circulan por ese camino. De esta manera se encontraría el camino más corto puesto que el rastro de feromonas se intensificaría en dicho camino.

  Este algoritmo utiliza el mismo procedimiento. Se construye un conjunto de soluciones iniciales aleatorio y mediante un proceso iterativo se van incorporando elementos en la solución parcial. En cada paso se calcula la cantidad de feromonas, cuantas más feromonas hay, más posibilidades existen de que sea la mejor solución. Las feromonas de las hormigas representan la memoria del algoritmo.

- **Algoritmo de ahorro de Clarke and Wright**. Como se comenta anteriormente, este algoritmo fue uno de los primeros métodos heurísticos creados para resolver el VRP y a día de hoy es, sin duda, el algoritmo más utilizado para su resolución. La idea principal del algoritmo es calcular el ahorro que supone unir dos puntos que no están en la misma ruta en la solución factible inicial. Mediante una comparación de ahorros se construye la ruta de menor coste.

- **Algoritmos genéticos**. Estos algoritmos nace inspirándose en la teoría de la evolución darwiniana. La idea principal consiste en imitar la manera en la que las especies evolucionan y se adaptan a su entorno, siguiendo la teoría darwiniana de selección natural. Se parte de una población inicial de individuos que representan un conjunto

de soluciones iniciales factibles. Cada solución factible sería un individuo. Las estrategias de evolución del algoritmo actúan sobre los individuos (es decir el conjunto inicial de soluciones) aplicando una serie de operadores evolutivos que modifican y combinan a los individuos, de forma que se crea una nueva. Estos operadores, como en el caso natural, son la selección del más apto, el cruce genético y la mutación. A través del proceso de selección solo se mantienen los mejores, dando lugar a nuevos descendientes mejorados. El proceso de cruce genético combina las características más importantes de dos buenas soluciones creando una solución mejorada. Finalmente cada descendiente es modificado aleatoriamente por un factor de mutación. Tras un proceso iterativo la mejor solución obtenida se considera la solución final.

- **Greedy Randomized Adaptive Search Procedure GRA-SP**. La idea principal de este algoritmo es utilizar un heurístico de construcción aleatorizado para generar un conjunto de soluciones iniciales. En cada paso, los elementos aún no incorporados en la solución parcial actual se evalúan y los mejores se añaden a una lista restringida de elementos candidatos. Se elige un elemento al azar de esta lista y se incorpora a la solución parcial. Tras un proceso iterativo la mejor solución obtenida se considera la solución final.

- **Búsqueda tabú**. Es un algoritmo basado en la búsqueda local donde, en cada iteración, se busca una solución mejorada en la vecindad de la solución de prueba actual, incluso si aumenta el coste de la solución. De esta manera, el algoritmo puede escapar de óptimos locales lo que le permite acercarse a la solución óptima. La ventaja de este método es el uso de memoria, llamado lista tabú, donde se van almacenando las soluciones visitadas recientemente para evitar que entre en un bucle. El algoritmo para al cumplirse un criterio de detención, que normalmente es al llegar a un número de iteraciones dadas o al realizar un número de iteraciones sin mejora de la solución de prueba. La última solución calculada se considera la solución final. En el capítulo 2 se muestra un ejemplo para la resolución de un TSP utilizando el método de búsqueda tabú.

Capítulo 4

Una aplicación socio sanitaria

En los capítulos anteriores se hace una revisión sobre dos problemas de programación lineal, el TSP y el VRP, así como su formulación matemática y métodos de resolución. En este capítulo se pone en práctica todo lo expuesto anteriormente, modelizando un problema de la vida real como un VRP. Se hará una comparación entre las diversas soluciones obtenidas modelando el problema con AMPL.

4.1. Introducción

Arroupar es un centro de día orientado a personas mayores y/o dependientes en la ciudad de Lugo. Este centro ofrece una atención integral, personalizada y adaptada a las necesidades de cada uno de sus usuarios, permitiendo así que puedan seguir viviendo en sus casas pero estando atendidos y activos durante el día.

Es un centro pequeño que dispone de 50 plazas lo cual garantiza que los usuarios estén en un ambiente familiar, acogedor y con una atención directa. Entre sus trabajadores se encuentran varios profesionales de la salud que cuentan con una formación continua para poder atender las diversas enfermedades de los usuarios, como son la demencia, alzheimer, artrosis, etc.

Esta empresa además de realizar muchas actividades a lo largo del día presta un servicio de recogida y de regreso de cada usuario a su domicilio.

4.2. El problema

El problema que presenta esta empresa consiste en organizar las rutas de manera más eficiente y económica para llevar de vuelta a cada usuario a su domicilio tras la realización de las actividades. El objetivo es minimizar tanto el tiempo que se tarda en llevar a todos los usuarios como las distancias, lo cual supondría un ahorro en tiempo y en costes de transporte.

La optimización de las recogidas de los usuarios de forma programada es una cuestión compleja pero a la vez sencilla ya que las horas y el orden de recogida dependen de las necesidades de la familia de cada usuario día a día. Debido a la familiaridad de la empresa, esta intenta adaptarse en la mayor medida posible a sus rutinas, por lo tanto, no se puede optimizar de forma rígida ya que está en constante variación. Debido al pequeño número de usuarios que tiene, el centro permite una gran flexibilidad de horarios de entrada. Por ejemplo, a varios usuarios los acercan sus familiares de camino al trabajo, otros prefieren ir paseando y otros más perezosos prefieren disfrutar del descanso y que luego los recojan. Por lo tanto, el número de recogidas es mucho más pequeño y las horas son dispares.

Al mediodía nos encontramos con una situación similar. El centro dispone de servicio de comedor, por lo que la mayoría de los usuarios se quedan a comer. También existe la opción de media jornada, pero solo la tienen un número muy reducido de usuarios, por lo tanto, la organización del transporte es sencilla.

Sin embargo, a la tarde la situación es diferente. Los usuarios deben salir de forma programada ya que el centro cierra a las 20:30h y a esa hora deben de estar todos fuera, ya sea en sus respectivos domicilios o camino a ellos.

Para ello la empresa cuenta con dos conductores y dos furgonetas adaptadas. La furgoneta blanca tiene capacidad para 7 usuarios, todos ellos sentados, y la furgoneta amarilla tiene capacidad para 4 usuarios sentados y 2 sillas de ruedas, siendo así un total de 6 plazas. De modo que clasificaremos los usuarios en dos grupos:

- Grupo 1: Usuarios válidos, que van sentados en los asientos de la furgoneta.
- Grupo 2: Usuarios inválidos, que necesitan silla de ruedas.

Aunque para la empresa es más cómodo trasladar a los usuarios en silla de ruedas, intentan minimizar el uso de estas ya que las sillas son rígidas y no poseen amortiguación y debido a los múltiples badenes, baches o desniveles que hay en la ciudad de Lugo, las frágiles caderas de las personas mayores se verían resentidas a largo plazo.

4.3. Datos del problema

A continuación detallamos los datos que hemos recopilado para la correcta resolución del problema:

1. Número de usuarios que cuentan con el servicio de transporte en agosto de 2019: 26 usuarios.
2. Número de usuarios que van en silla de ruedas: 3 usuarios.
3. Número de vehículos de los que disponemos: 2 vehículos.
4. Número de plazas en cada vehículo. En la furgoneta blanca disponemos de 7 plazas para válidos y en la furgoneta amarilla disponemos de 4 plazas para válidos y 2 para inválidos.
5. Número de rutas actuales realizadas por la empresa: 6. Como estamos ante un problema de minimización de costes, se intentará reducirlas. Entonces el número de rutas, a efectos prácticos, será una variable aunque se trate como un dato.
6. Tabla de direcciones de los usuarios. Consultar apéndice A.1.
7. Tabla de distancias entre cada nodo. Consultar apéndice A.2.
8. Tabla de tiempos que se tarda en ir de un nodo a otro. Consultar apéndice A.3.

Observación 4.1. Es interesante destacar que no es un problema simétrico ya que, por lo general, la distancia (i, j) no es la misma que (j, i). Esto es debido a que no todas las calles son de doble sentido, hay calles cortadas, etc.

En la Figura 4.1 se representa gráficamente donde están situados los domicilios de los usuarios que tienen servicio de transporte. En ella se puede apreciar la considerable distancia que separa al centro de día de los dos domicilios de las usuarias que viven en Coeses y el usuario que vive en Mazoi.

Figura 4.1: Situación de los 26 domicilios en la ciudad de Lugo.

En la Figura 4.2 se amplía el centro de Lugo, dejando fuera los 3 domicilios que están más alejados del centro de día para poder ver con claridad la situación de los nodos.

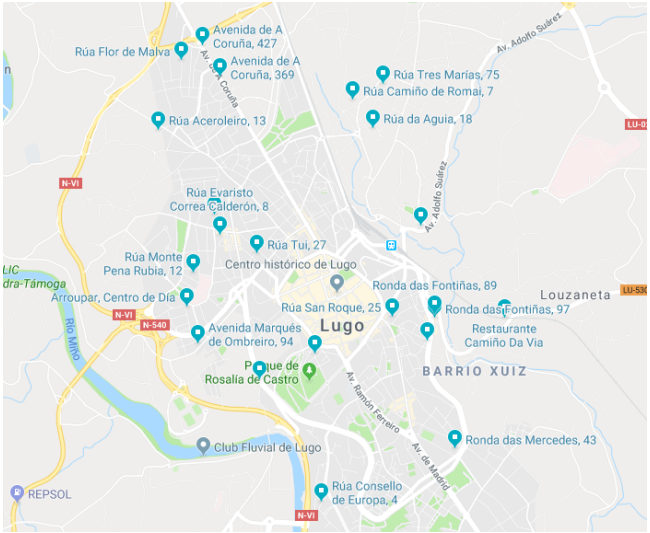


Figura 4.2: Centro de Lugo aumentado.

4.4. Descripción de las variables y parámetros

A continuación definiremos los conjuntos, parámetros y las variables necesarias para la comprensión y el correcto funcionamiento del problema.

$N = \{1, \ldots, 26\}$	Conjunto de usuarios que necesitan transporte a sus domicilios.
$\{0\}$	El nodo 0 representa el centro de día Arroupar.
$n = card(N)$	Número de usuarios.
$s = 3$	Número de usuarios que usan silla de ruedas para el traslado.
$R = \{1, 2, 3, ...\}$	Conjunto de rutas. Este conjunto lo vamos modificando hasta encontrar la solución óptima.
$V = \{1, 2\}$	Conjunto de vehículos de los que disponemos.
c_{ij}	Coste del desplazamiento del nodo i al j. Lo expresamos en kilómetros o minutos.
$x_{i,j,v,r}$	Variable binaria que toma valor 1 cuando vamos del nodo i al nodo j con el vehículo v por la ruta r. Toma el valor 0 cuando no se cumple alguna de esas condiciones.
$u_{i,j,r}$	Variable entera auxiliar necesaria para que no se formen ciclos indeseados.

4.5. Resolución del problema

Para la resolución de este problema hacemos uso del *problema del viajante* y su variación, el *problema de rutas de vehículos*.

Modelaremos el problema como un problema de programación lineal entera y para su resolución utilizamos AMPL (*A Mathematical Programming Language*), que es un lenguaje de programación algebraica que se usa sobre todo para describir y solucionar problemas de complejidad matemática media o alta, como ocurre en este caso. Un problema de 27 nodos es considerado bastante grande puesto que son necesarias más de un millón de iteraciones para poder llegar a algún resultado óptimo. Para resolverlo por fuerza bruta habría que realizar $26! \approx 4{,}10^{26}$ operaciones. Además como ya hemos visto tanto el TSP como el VRP

son problemas NP-duros, por lo tanto necesitamos un solucionador potente.

AMPL es capaz de expresar en notación algebraica problemas de optimización como los de programación lineal y resolverlos gracias a los diferentes solucionadores de los que dispone. Este programa nos permite representar de forma sencilla problemas de minimización o maximización, dando la función objetivo y las diferentes restricciones. Con ayuda de los solucionadores con los que cuenta resuelve el problema dado. En este caso usaremos el solucionador Gurobi.

Para programar en AMPL se puede optar por dos maneras diferentes, mediante la terminal propia, con la licencia académica, o mediante el servidor NEOS, que es un servicio gratuito en Internet, con base en la Universidad de Wisconsin en Madison, que se utiliza para resolver problemas de optimización numérica. El servidor NEOS proporciona acceso a numerosos solucionadores de última generación que se ejecutan en computadoras de alto rendimiento distribuidas por todo el mundo.

Uno de los solucionadores más potentes para la resolución de problemas a gran escala es Gurobi. Está especializado en problemas de programación lineal (LP), problemas de programación lineal entera mixta (MILP) y problemas de programación cónica de segundo orden (SOCP). Para su resolución utiliza algoritmos de ramificación y acotación. Gurobi es un solucionador de última generación para programación matemática.

4.5.1. Formulación matemática

A continuación vamos a escribir la formulación matemática como un problema de programación lineal:

$$\text{mín} \quad \sum_{i,j \in N \cup \{0\}, v \in V, r \in R} x_{i,j,v,r} \cdot c_{ij} \tag{4.1}$$

sujeto a

$$\sum_{j \in N, v \in V} x_{0,j,v,r} = 1 \qquad \forall r \in R \tag{4.2}$$

$$\sum_{i \in N, v \in V} x_{i,0,v,r} = 1 \qquad \forall r \in R \tag{4.3}$$

$$\sum_{j \in N \cup \{0\}, v \in V, r \in R} x_{i,j,v,r} = 1 \qquad \forall i \in N \tag{4.4}$$

$$x_{i,i,v,r} = 0 \qquad \begin{array}{l} \forall i \in N \cup \{0\}, \\ \forall v \in V, \forall r \in R \end{array} \tag{4.5}$$

$$\sum_{j \in N \cup \{0\}} x_{i,j,v,r} - \sum_{j \in N \cup \{0\}} x_{j,i,v,r} = 0 \qquad \begin{array}{l} \forall i \in N, \forall v \in V, \\ \forall r \in R \end{array} \tag{4.6}$$

$$u_{i,v,r} - u_{j,v,r} + n \cdot x_{i,j,v,r} \leq n - 1 \qquad \begin{array}{l} \forall i, j \in N, \\ \forall v \in V, \forall r \in R \end{array} \tag{4.7}$$

$$x_{i,j,1,r} = 0 \qquad \begin{array}{l} \forall i \in N \cup \{0\}, \\ \forall j \in \{1, .., s\}, \forall r \in R \end{array} \tag{4.8}$$

$$\sum_{i \in N, j \in N \cup \{0\}} x_{i,j,1,r} \leq 7 \qquad \forall r \in R \tag{4.9}$$

$$\sum_{i \in \{1,..,s\}, j \in N \cup \{0\}} x_{i,j,2,r} \leq 2 \qquad \forall r \in R \tag{4.10}$$

$$\sum_{i \in \{1+s,..,n\}, j \in N \cup \{0\}} x_{i,j,2,r} \leq 4 \qquad \forall r \in R \tag{4.11}$$

$$\text{donde } x_{i,j,v,r} = \begin{cases} 1, & \text{si el vehículo } v \text{ va por la ruta } r \text{ desde el nodo } i \text{ al nodo } j \\ 0, & \text{en otro caso.} \end{cases}$$

La ecuación (4.1) es nuestra función objetivo. La ecuación (4.2) indica que cada ruta debe de empezar siempre en el nodo 0, es decir, el Centro de día. La ecuación (4.3) indica que cada ruta debe de acabar siempre en el nodo 0. La ecuación (4.4) indica que se deben de recoger a todos los usuarios. La ecuación (4.5) indica que no se pueden formar lazos. La ecuación (4.6) refleja la conservación de flujo (sólo se puede llegar y salir de la dirección de un usuario cada vez). La ecuación (4.7) indica que no se pueden formar ciclos improcedentes. La ecuación (4.8) indica que en el vehículo 1 no pueden ir sillas. La ecuación (4.9) indica que la capacidad del vehículo 1 son 7 usuarios. La ecuación (4.10) indica que en el vehículo 2 sólo caben 2 sillas. La ecuación (4.11) indica que en el vehículo 2 caben a lo sumo 4 usuarios sentados.

4.5.2. Método de resolución

Al tratarse de un problema de rutas de vehículos, que es una extensión del TSP, usaremos el método exacto de ramificación y acotación implementado por el solucionador Gurobi.

Como ya hemos explicado, para modelar este problema de optimización lineal entera usaremos el programa AMPL, ya que es compatible con los mejores algoritmos de programación lineal conocidos a día de hoy y su gran efectividad viene impulsada por la separación del modelo por un lado, los datos por otro y los comandos a ejecutar por otro.

Para trabajar con NEOS necesitamos crear 3 archivos:

1. El primer archivo será el modelo, donde se define el problema, la función objetivo y las restricciones. Consultar apéndice B.1

2. En el segundo archivo se proporcionan los datos del problema. En nuestro caso, iremos modificando el número de rutas y a veces quitando nodos para encontrar la solución óptima. Hemos escritos dos ficheros, en el apéndice B.2 se representan los datos de distancias en kilómetros y en el apéndice B.3 se representan los datos de tiempo en minutos.

3. En el tercer archivo se proporcionan los comandos a ejecutar, es decir, una vez dado el problema y los datos, se escriben las órdenes de resolución. En el apéndice B.4 se presentan 3 archivos que utilizaremos en función de lo que necesitemos en cada momento.

Los resultados obtenidos son enviados a la dirección de correo electrónico aportada por el usuario, facilitando así su recolección.

Antes de continuar, introducimos el concepto de *"gap"* ya que es un dato muy relevante a la hora de comparar las soluciones obtenidas.

El método de ramificación y acotación termina cada iteración con una cota superior, S, y una cota inferior, I, para el valor del objetivo en el óptimo. El textitgap absoluto (*absmipgap*) representa la diferencia, en valor absoluto, de esas dos cotas. El textitgap relativo (*relmipgap*) representa el textitgap absoluto dividido por el valor absoluto de la cota superior, es decir, expresado,si lo multiplicamos por cien, en tanto por cien.

4.6. Optimización de la distancia

En este apartado modelamos el problema usando los datos relativos a las distancias, medidos en kilómetros.

Primero se prueba para el caso de 6 rutas y se van realizando modificaciones. Para comprobar la mejora de los resultados se va aumentando el tiempo de trabajo del solucionador Gurobi.

Para la comparación de las soluciones se ha utilizado el tercer fichero del apéndice B.3. ya que esta opción es una combinación de las dos anteriores y nos permite comparar, dando los resultados del *gap*, como de buenas son las soluciones obtenidas.

En las salidas podemos ver el número de variables y restricciones del problema, los nodos explorados en el proceso de ramificación y acotación o el número de veces que se aplica como subrutina el algoritmo del *simplex*. Por medio de una tabla mostraremos, de forma más visual, la solución obtenida.

4.6.1. Caso de 6 rutas

La empresa habitualmente realizaba 6 viajes, por ese motivo empezamos modelando el problema para el caso de 6 rutas.

A continuación se explican las salidas recibidas más importantes. Hemos probado con varios tiempos de ejecución hasta llegar a soluciones que merecen la pena.

SALIDA 1 DE AMPL, 6 rutas y tiempo 21600s.

```
Presolve eliminates 1140 constraints and 1224 variables.
Adjusted problem:
7836 variables:
```

```
7524 binary variables
312 linear variables
8150 constraints, all linear; 52044 nonzeros
332 equality constraints
7818 inequality constraints
1 linear objective; 7512 nonzeros.

Gurobi 9.1.1: timelim 21600
bestbound 1
threads=4
Gurobi 9.1.1: time limit with a feasible solution; objective 80.6
208519878 simplex iterations
6915385 branch-and-cut nodes
absmipgap = 3.42, relmipgap = 0.0424
No basis.
No dual variables returned.
```

Interpretación:

Nº RUTA	VEHÍCULO	USUARIOS	Nº USUARIOS
1	2	$0 \to 19 \to 2 \to 25 \to 26 \to 1 \to 9 \to 0$	6
2	1	$0 \to 18 \to 4 \to 12 \to 11 \to 23 \to 13 \to 0$	6
3	1	$0 \to 21 \to 10 \to 24 \to 15 \to 17 \to 16 \to 8 \to 0$	7
4	2	$0 \to 22 \to 3 \to 6 \to 5 \to 7 \to 0$	5
5	2	$0 \to 14 \to 0$	1
6	1	$0 \to 20 \to 0$	1
GAP = 0.0424			
TOTAL DE KMS RECORRIDOS: 80.6			

Tabla 4.1: Rutas de la salida 1 con 6 rutas y tiempo límite 21600s.

SALIDA 2 DE AMPL, 6 rutas y tiempo 28800s.

```
Presolve eliminates 1140 constraints and 1224 variables.
Adjusted problem:
7836 variables:
7524 binary variables
```

```
312 linear variables
8150 constraints, all linear; 52044 nonzeros
332 equality constraints
7818 inequality constraints
1 linear objective; 7512 nonzeros.

Gurobi 9.1.1: timelim 28800
bestbound 1
threads=4
Gurobi 9.1.1: time limit with a feasible solution; objective 80.6
102788813 simplex iterations
3320925 branch-and-cut nodes
absmipgap = 3.95, relmipgap = 0.049
No basis.
No dual variables returned.
```

Interpretación:

Nº RUTA	VEHÍCULO	USUARIOS	Nº USUARIOS
1	2	$0 \to 19 \to 2 \to 25 \to 26 \to 1 \to 9 \to 0$	6
2	1	$0 \to 18 \to 4 \to 12 \to 11 \to 23 \to 13 \to 0$	6
3	1	$0 \to 21 \to 10 \to 24 \to 15 \to 17 \to 16 \to 8 \to 0$	7
4	2	$0 \to 22 \to 3 \to 6 \to 5 \to 7 \to 0$	5
5	2	$0 \to 14 \to 0$	1
6	1	$0 \to 20 \to 0$	1
GAP = 0.049			
TOTAL DE KMS RECORRIDOS: 80,6			

Tabla 4.2: Rutas de la salida 2 con 6 rutas y tiempo límite 28800s.

Conclusiones:

Después de realizar varias pruebas, vemos que nos da las mismas rutas y con un valor del *gap* bastante pequeño. Se ha intentado conseguir un textitgap de 0.01 pero no ha sido posible ya que se excede el tiempo máximo permitido para la resolución de este problema. Entonces podemos concluir, que si queremos realizar 6 rutas, la mejor solución posible será la representada en las tablas 4.1 y 4.2. El valor mínimo de la función objetivo será 80,6 km.

4.6.2. Caso de 5 rutas

En el apartado anterior podemos ver que dos de las rutas dadas sólo llevarían a un usuario. Vamos a ver que ocurre si decidimos reducir una ruta.

SALIDA 1 DE AMPL, 5 rutas y tiempo 21600s.

```
Presolve eliminates 950 constraints and 1020 variables.
Adjusted problem:
6530 variables:
6270 binary variables
260 linear variables
6796 constraints, all linear; 43370 nonzeros
281 equality constraints
6515 inequality constraints
1 linear objective; 6260 nonzeros.

Gurobi 9.1.1: timelim 21600
bestbound 1
threads=4
Gurobi 9.1.1: time limit with a feasible solution; objective 79.2
140310794 simplex iterations
3705334 branch-and-cut nodes
absmipgap = 5.2, relmipgap = 0.0788
No basis.
No dual variables returned.
```

Interpretación:

NºRUTA	VEHÍCULO	USUARIOS	NºUSUARIOS
1	2	$0 \to 13 \to 21 \to 1 \to 9 \to 23 \to 0$	5
2	2	$0 \to 22 \to 3 \to 25 \to 26 \to 2 \to 24 \to 0$	6
3	1	$0 \to 14 \to 0$	1
4	1	$0 \to 20 \to 11 \to 19 \to 6 \to 5 \to 7 \to 10 \to 0$	7
5	1	$0 \to 18 \to 4 \to 12 \to 15 \to 17 \to 16 \to 8 \to 0$	7
GAP = 0.0788			
TOTAL DE KMS RECORRIDOS: 79.2			

Tabla 4.3: Rutas de la salida 1 con 5 rutas y tiempo límite 21600s.

SALIDA 2 DE AMPL, 5 rutas y tiempo 28800s.

```
Presolve eliminates 950 constraints and 1020 variables.
Adjusted problem:
6530 variables:
6270 binary variables
260 linear variables
6796 constraints, all linear; 43370 nonzeros
281 equality constraints
6515 inequality constraints
1 linear objective; 6260 nonzeros.

Gurobi 9.1.1: timelim 28800
bestbound 1
threads=4
Gurobi 9.1.1: time limit with a feasible solution; objective 79.2
150832850 simplex iterations
3992759 branch-and-cut nodes
absmipgap = 5.13, relmipgap = 0.0648
No basis.
No dual variables returned.
```

Interpretación:

NºRUTA	VEHÍCULO	USUARIOS	NºUSUARIOS
1	2	$0 \to 13 \to 21 \to 1 \to 9 \to 23 \to 0$	5
2	2	$0 \to 22 \to 3 \to 25 \to 26 \to 2 \to 24 \to 0$	6
3	1	$0 \to 14 \to 0$	1
4	1	$0 \to 20 \to 11 \to 19 \to 6 \to 5 \to 7 \to 10 \to 0$	7
5	1	$0 \to 18 \to 4 \to 12 \to 15 \to 17 \to 16 \to 8 \to 0$	7
GAP = 0.0648			
TOTAL DE KMS RECORRIDOS: 79.2			

Tabla 4.4: Rutas de la salida 2 con 5 rutas y tiempo límite 28800s.

Conclusiones:

Después de realizar varias pruebas, se pude apreciar que el valor del *gap* en la salida 2 es menor que en la salida 1. Se ha intentado aumentar el tiempo de ejecución para obtener un valor del *gap* más pequeño pero no ha sido posible ya que se excede el tiempo máximo permitido. Entonces podemos concluir, que realizando 5 viajes se reduce el valor de la función objetivo a 79,2 km. Aunque es una disminución pequeña, se trata de la disminución diaria por lo que finalmente, en el cómputo mensual supone un ahorro interesante. Las mejor solución es la dada en las tablas 4.3 y 4.4.

4.6.3. Caso de 4 rutas

Al programar el problema con 4 rutas nos da error, lo cual es obvio, debido a que los usuarios que tienen contratado el servicio de transporte son 23 válidos y 3 inválidos. Tendríamos que hacer mínimo dos viajes con la furgoneta mixta, de manera que fueran 2 inválidos + 4 válidos en un viaje y 1 inválido + 4 válidos en el segundo viaje. Un total de 11 usuarios en la furgoneta mixta. Por otro lado, en la otra furgoneta quedarían por realizar dos rutas, con 7 usuarios en cada una, siendo un total de 14 usuarios.

Entonces habríamos llevado a sus domicilios a 11 + 14 = 25 usuarios. En el caso de que algún día uno de los usuarios no acudiese al centro se podrían realizar 4 rutas en vez de 5. Como se puede observar en los apartados anteriores, hay una ruta que lleva a un sólo usuario, el 14. Prescindir de esta ruta sería la forma más sencilla para realizar solamente 4 rutas.

4.6.4. Conclusión final

Como acabamos de explicar, el diseño de rutas óptimo sería el de la tabla 4.3, con un valor mínimo de la función objetivo de 79,2 km, siendo el conjunto de rutas de la tabla 4.5 la mejor solución encontrada.

NºRUTA	VEHÍCULO	USUARIOS	NºUSUARIOS
1	2	$0 \to 13 \to 21 \to 1 \to 9 \to 23 \to 0$	5
2	2	$0 \to 22 \to 3 \to 25 \to 26 \to 2 \to 24 \to 0$	6
3	1	$0 \to 14 \to 0$	1
4	1	$0 \to 20 \to 11 \to 19 \to 6 \to 5 \to 7 \to 10 \to 0$	7
5	1	$0 \to 18 \to 4 \to 12 \to 15 \to 17 \to 16 \to 8 \to 0$	7
TOTAL DE KMS RECORRIDOS: 79.2			

Tabla 4.5: Solución final para la optimización de distancia.

4.7. Optimización del tiempo

En este apartado modelamos el problema usando los datos relativos al tiempo, medidos en minutos.

Utilizamos un procedimiento similar al caso de optimización de distancias. Empezamos probando para el caso de 6 rutas y vamos realizando modificaciones y comparaciones de soluciones. Para comprobar la mejora de los resultados se va aumentando el tiempo de trabajo del solucionador Gurobi.

Para la comparación de las soluciones se ha utilizado el tercer fichero del apéndice B.3. ya que esa opción es una combinación de las dos anteriores y nos permite comparar, dando los resultados del *gap*, como de buenas son las soluciones obtenidas.

4.7.1. Caso de 6 rutas

La empresa habitualmente realizaba 6 viajes, por ese motivo empezamos modelando el problema para el caso de 6 rutas.

A continuación se explican las salidas más importantes recibidas. Hemos probado con varios tiempos de ejecución hasta llegar a soluciones que merecen la pena.

SALIDA 1 DE AMPL, 6 rutas y tiempo 21600s:

```
Presolve eliminates 1140 constraints and 1224 variables.
Adjusted problem:
7836 variables:
7524 binary variables
312 linear variables
8150 constraints, all linear; 52044 nonzeros
332 equality constraints
7818 inequality constraints
1 linear objective; 7488 nonzeros.

Gurobi 9.1.1: timelim 21600
bestbound 1
threads=4
Gurobi 9.1.1: time limit with a feasible solution; objective 154
173640916 simplex iterations
3662456 branch-and-cut nodes
absmipgap = 6, relmipgap = 0.039
```

```
No basis.
No dual variables returned.
```

Interpretación:

Nº RUTA	VEHÍCULO	USUARIOS	Nº USUARIOS
1	2	$0 \to 19 \to 9 \to 1 \to 21 \to 23 \to 0$	5
2	1	$0 \to 13 \to 0$	1
3	1	$0 \to 14 \to 0$	1
4	1	$0 \to 22 \to 20 \to 4 \to 12 \to 11 \to 18 \to 0$	6
5	2	$0 \to 25 \to 26 \to 2 \to 5 \to 3 \to 24 \to 0$	6
6	1	$0 \to 15 \to 17 \to 16 \to 8 \to 6 \to 7 \to 10 \to 0$	7
GAP = 0.039			
TOTAL DE MINS: 154			

Tabla 4.6: Rutas de la salida 1 con 6 rutas y tiempo límite 21600s.

SALIDA 2 DE AMPL, 6 rutas y tiempo 28800s.

```
Presolve eliminates 1140 constraints and 1224 variables.
Adjusted problem:
7836 variables:
7524 binary variables
312 linear variables
8150 constraints, all linear; 52044 nonzeros
332 equality constraints
7818 inequality constraints
1 linear objective; 7488 nonzeros.

Gurobi 9.1.1: timelim 28800
bestbound 1
threads=4
Gurobi 9.1.1: time limit with a feasible solution; objective 154
421683594 simplex iterations
10111046 branch-and-cut nodes
absmipgap = 3, relmipgap = 0.0195
No basis.
```

```
No dual variables returned.
```

Interpretación:

NºRUTA	VEHÍCULO	USUARIOS	NºUSUARIOS
1	2	$0 \to 19 \to 9 \to 1 \to 21 \to 23 \to 0$	5
2	2	$0 \to 13 \to 0$	1
3	1	$0 \to 14 \to 0$	1
4	1	$0 \to 22 \to 20 \to 4 \to 12 \to 11 \to 18 \to 0$	6
5	2	$0 \to 25 \to 26 \to 2 \to 5 \to 3 \to 24 \to 0$	6
6	1	$0 \to 15 \to 17 \to 16 \to 8 \to 6 \to 7 \to 10 \to 0$	7
GAP = 0.0195			
TOTAL DE MINS: 154			

Tabla 4.7: Rutas de la salida 2 con 6 rutas y tiempo límite 28800s.

Conclusiones:

Después de realizar varias pruebas, vemos que ambas salidas nos dan el mismo conjunto de rutas pero la salida 2 tiene un textitgap mucho más pequeño. Se ha intentado conseguir un textitgap de 0.01 pero no ha sido posible ya que se excede el tiempo máximo permitido para la resolución de este problema. Entonces podemos concluir, que si queremos realizar 6 rutas, la mejor solución posible será la representada en la tabla 4.7 . El valor mínimo de la función objetivo para la minimización del tiempo es 154 min $\approx$ 2h y media.

4.7.2. Caso de 5 rutas

En el apartado anterior podemos observar que dos de las rutas creadas tan solo llevarán un usuario cada una, lo cual incrementa notablemente el valor de la función objetivo. Vamos a ver que ocurre si decidimos reducir una ruta.

SALIDA 1 DE AMPL, 5 rutas y tiempo 21600s.

```
Presolve eliminates 950 constraints and 1020 variables.
Adjusted problem:
6530 variables:
6270 binary variables
260 linear variables
6796 constraints, all linear; 43370 nonzeros
281 equality constraints
```

```
6515 inequality constraints
1 linear objective; 6240 nonzeros.

Gurobi 9.1.1: timelim 21600
bestbound 1
threads=4
Gurobi 9.1.1: time limit with a feasible solution; objective 149
124002096 simplex iterations
3755093 branch-and-cut nodes
absmipgap = 9, relmipgap = 0.0608
No basis.
No dual variables returned.
```

Interpretación:

Nº RUTA	VEHÍCULO	USUARIOS	Nº USUARIOS
1	2	$0 \to 19 \to 1 \to 9 \to 21 \to 24 \to 0$	5
2	1	$0 \to 15 \to 17 \to 8 \to 16 \to 5 \to 7 \to 10 \to 0$	7
3	1	$0 \to 14 \to 0$	1
4	2	$0 \to 25 \to 26 \to 3 \to 6 \to 2 \to 13 \to 0$	6
5	1	$0 \to 22 \to 20 \to 4 \to 12 \to 11 \to 18 \to 23 \to 0$	7
GAP = 0.0608			
TOTAL DE MINS: 149			

Tabla 4.8: Rutas de la salida 1 con 5 rutas y tiempo límite 21600s.

SALIDA 2 DE AMPL, 5 rutas y tiempo 28800s.

```
Presolve eliminates 950 constraints and 1020 variables.
Adjusted problem:
6530 variables:
6270 binary variables
260 linear variables
6796 constraints, all linear; 43370 nonzeros
281 equality constraints
6515 inequality constraints
1 linear objective; 6240 nonzeros.
```

```
Gurobi 9.1.1: timelim 28800
bestbound 1
threads=4
Gurobi 9.1.1: time limit with a feasible solution; objective 148
272548832 simplex iterations
8472129 branch-and-cut nodes
absmipgap = 7, relmipgap = 0.0473
No basis.
No dual variables returned.
```

Interpretación:

NºRUTA	VEHÍCULO	USUARIOS	NºUSUARIOS
1	2	$0 \to 19 \to 1 \to 9 \to 21 \to 13 \to 0$	5
2	1	$0 \to 15 \to 17 \to 16 \to 8 \to 6 \to 7 \to 10 \to 0$	7
3	1	$0 \to 14 \to 0$	1
4	2	$0 \to 25 \to 26 \to 2 \to 5 \to 3 \to 24 \to 0$	6
5	1	$0 \to 22 \to 20 \to 4 \to 12 \to 11 \to 18 \to 23 \to 0$	7
GAP = 0.0473			
TOTAL DE MINS: 148			

Tabla 4.9: Rutas de la salida 2 con 5 rutas y tiempo límite 28800s.

Conclusiones:

Después de realizar varias pruebas, aumentando el tiempo de ejecución, vemos que tanto el textitgap como la solución de la función objetivo ha mejorado en la segunda salida.

Entonces podemos concluir, que realizando 5 viajes en vez de 6 se reduce el valor de la función objetivo. Además entre las dos salidas, la mejor es la dada en la Tabla 4.9, cuyo valor de la función objetivo es 148 min = 2h y 28 min.

4.7.3. Caso de 4 rutas

Como ocurre en el caso de optimización en distancia, para 4 rutas el solucionador responde con un error, es evidente ya que no podemos distribuir los 26 usuarios en 4 rutas.

En el caso de que algún día uno de los usuarios no acudiese al centro se podrían realizar 4 rutas en vez de 5.

4.7.4. Conclusión final

Como acabamos de explicar el diseño de rutas óptimo sería el de la tabla 4.9, con un valor mínimo de la función objetivo de 148 mins. En cuanto a la comparación entre realizar 5 o 6 rutas, observamos que el tiempo es similar (148 mins frente a 154 mins) pero la diferencia entre realizar 1 ruta más o menos supone un gran ahorro en cuanto a costes de transporte y de organización.

El conjunto de rutas de la tabla 4.10 es la mejor solución encontrada.

Nº RUTA	VEHÍCULO	USUARIOS	Nº USUARIOS
1	2	$0 \to 19 \to 1 \to 9 \to 21 \to 13 \to 0$	5
2	1	$0 \to 15 \to 17 \to 16 \to 8 \to 6 \to 7 \to 10 \to 0$	7
3	1	$0 \to 14 \to 0$	1
4	2	$0 \to 25 \to 26 \to 2 \to 5 \to 3 \to 24 \to 0$	6
5	1	$0 \to 22 \to 20 \to 4 \to 12 \to 11 \to 18 \to 23 \to 0$	7
TOTAL DE MINS: 148			

Tabla 4.10: Solución final para la optimización de tiempo.

4.8. Resultado final

En la Figura 4.3 se representan las rutas de la mejor solución obtenida para la optimización de distancias. Son las correspondientes a la Tabla 4.5.

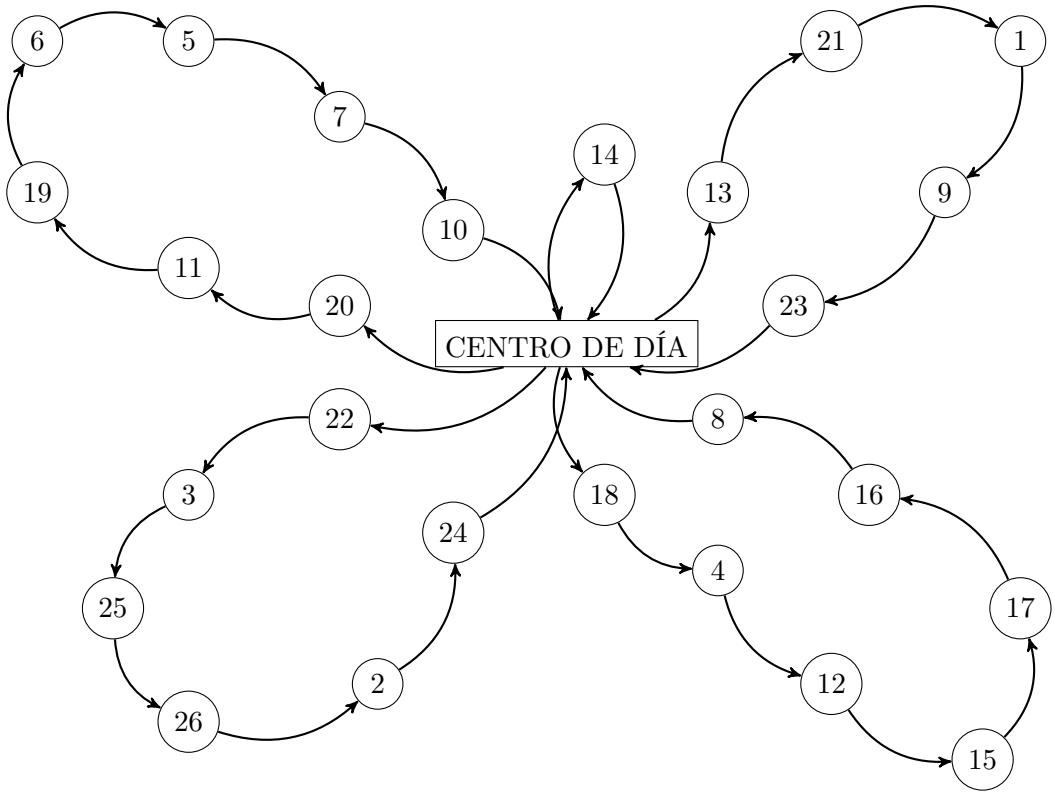


Figura 4.3: Representación de la mejor solución para la optimización de las distancias.

En la Figura 4.4 se representan las rutas de la mejor solución obtenida para la optimización de tiempo. Son las correspondientes a la Tabla 4.10.

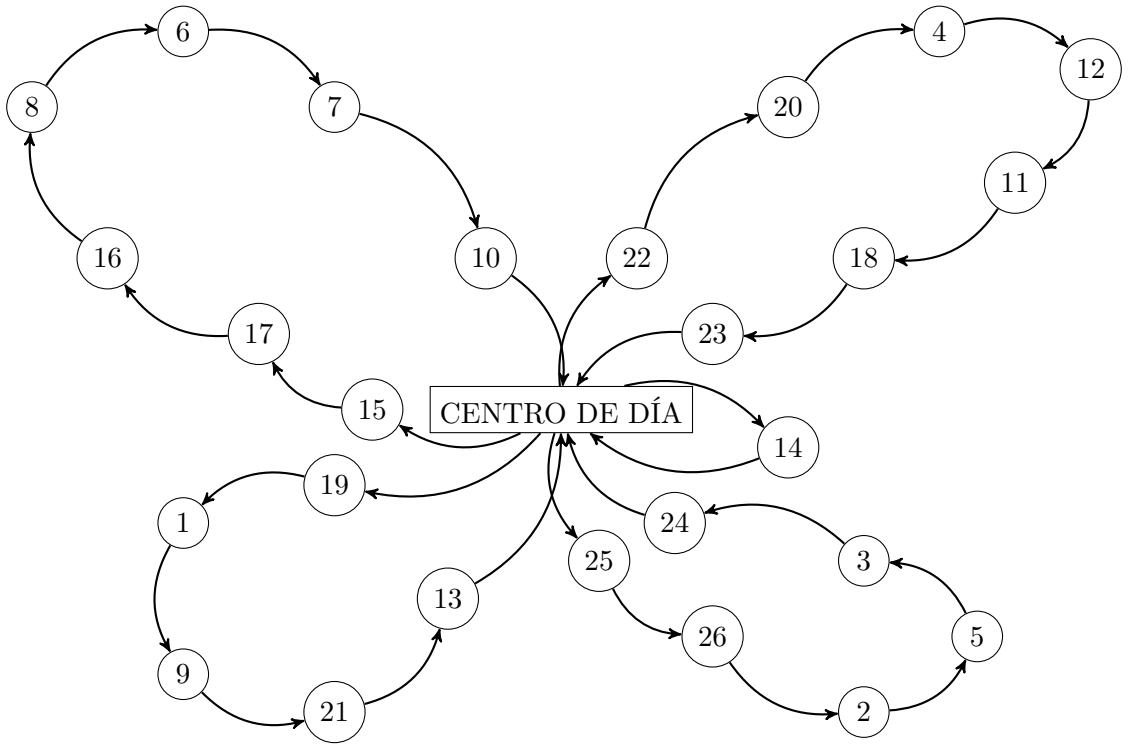


Figura 4.4: Representación de la mejor solución para la optimización del tiempo.

Como se puede observar, las rutas no son las mismas. Para finalizar optaríamos por la optimización de distancias, ya que estas no suelen variar, sin embargo, la optimización del tiempo está sujeta a muchas otras variables, por ejemplo atascos, semáforos, por lo que sería menos precisa.

Apéndices

Apéndice A

Tablas de datos

A.1. Direcciones de los usuarios

En la Tabla A.1 se representan las direcciones de los usuarios del centro utilizadas para la resolución del problema. Estos datos fueron cedidos por la empresa.

Las direcciones son reales ya que el proyecto fue aplicado en la empresa pero los nombres han sido modificados para preservar la identidad de los usuarios.

Observación A.1. Los usuarios que usan silla de ruedas se corresponden con los nodos 1, 2 y 3 para facilitar el manejo de datos en el código de programación.

	NOMBRE	DIRECCIÓN	SILLA DE RUEDAS
Salida	Centro de día	Rúa da Pomba, 7	
Usuario 1	Maricarmen V.	Rúa Consello de Europa, 4	Sí
Usuario 2	Ángel A.	Ronda das Mercedes, 43	Sí
Usuario 3	Aldara A.	Rúa Camiño da Vila, 15	Sí
Usuario 4	Pepa L.	Rúa Flor de Malva, 42	
Usuario 5	Andrés G.	Ronda das Fontiñas, 97	
Usuario 6	Lucía R.	Ronda das Fontiñas, 142	
Usuario 7	Pablo R.	Ronda das Fontiñas, 89	
Usuario 8	Alba C.	Camiño de Romay, 7	
Usuario 9	Noelia C.	Rúa Consello de Europa, 4	
Usuario 10	Laura A.	San Roque, 25	
Usuario 11	Paula M.	Avenida da Coruña, 369	
Usuario 12	Yolanda A.	Avenida da Coruña, 427	
Usuario 13	Elena A.	Marqués de Hombreiro, 94	
Usuario 14	Mónica A.	Monte Pena Rubia, 12	
Usuario 15	María A.	Mazoy, 13	
Usuario 16	Susa G.	Rúa da Aguia, 18	
Usuario 17	Marta F.	Rúa Tres Marías, 75	
Usuario 18	Lourdes C.	Rúa Aceroleiro, 13	
Usuario 19	Xosé María	Rúa General Tella, 4	
Usuario 20	Natalia G.	Rúa Doña Urraca, 19	
Usuario 21	Sara	Ronda do Carmen, 19	
Usuario 22	María G.	Rúa Evaristo Correa Calderón, 8	
Usuario 23	Paula A.	Rúa Tui, 27	
Usuario 24	Marta B.	Rúa Adolfo Suárez, 15	
Usuario 25	Uxía	Coeses Fontemaior, 2	
Usuario 26	Pepe G.	Coeses Fontemaior, 1	

Tabla A.1: Direcciones de los usuarios.

A.2. Distancia entre nodos

En la Tabla A.2 se representan las distancias, medidas en kilómetros, entre las direcciones de los usuarios.

Estos datos han sido obtenidos con la aplicación *Google Maps* a partir de las direcciones dadas en la Tabla A.1. Además se ha empleado una aplicación llamada *Map Marker* para situar gráficamente los puntos en el plano.

KM	C.D	Us.1	Us.2	Us.3	Us.4	Us.5	Us.6	Us.7	Us.8	Us.9	Us.10	Us.11	Us.12	Us.13	Us.14	Us.15	Us.16	Us.17	Us.18	Us.19	Us.20	Us.21	Us.22	Us.23	Us.24	Us.25	Us.26
C.D	0	4.6	5.9	7.1	3.6	6	5.4	6.1	5.5	4.6	4.7	3.8	3.6	2.1	0.65	9.1	5.5	5	1.7	3.8	0.9	3	0.95	2.5	5.4	18	18.5
Us.1	3.2	0	2.1	4.6	8.1	2.9	3	3	4.2	0	2	41	5.6	2.9	3.4	8.5	4.3	4.1	3.9	2.1	2.9	1.8	2.7	2.6	2.7	15	15.5
Us.2	6.1	2.1	0	2.8	8.7	1.2	1.3	1.2	3.5	2.1	1.5	3.7	4.9	5.5	6.6	8.4	3.5	3.7	4.3	2	3.3	2.2	3.2	3.1	2.3	15	15.5
Us.3	4.1	4.9	2.8	0	4.8	1.8	1.8	1.8	3.1	4.9	1.8	4.8	4.6	3.8	4.4	8.7	3.1	3.3	5.1	3.5	3.7	4	3.6	3.9	2.7	16	16.5
Us.4	4.2	6,6	5.1	4,8	0	4,1	4,2	4,1	2,7	6,6	4,1	0,65	0,4	4,6	2,4	5,9	2,7	2,9	0,65	6,1	1,8	3,4	1,8	1,9	3,4	21	21,5
Us.5	6	3,5	1,4	1,6	4	0	0,4	0,03	2,3	3,5	0,5	4	3,8	3	3,5	7,2	2,3	2,5	4,2	2,7	2,9	3,1	2,8	2,6	1,9	18	18,5
Us.6	3,5	3,1	0,95	1,9	4,2	0,27	0	0,3	2,6	3,1	0,65	4,3	4	3,3	3,8	7,5	2,6	2,8	5	3	3,2	3,5	3,1	29	2,2	16	16,5
Us.7	3,2	3,4	1,3	1,6	3,9	0,35	0,23	0	2,3	3,4	0,45	4	3,7	3	3,5	7,2	2,3	2,5	4,2	2,7	2,9	3,1	2,8	2,6	1,9	18	18,5
Us.8	3,4	4,5	3,5	3,2	2,7	2,5	2,5	2,5	0	4,5	2,5	2,7	2,5	3,1	3,9	5,2	0,3	0,55	2,9	2,8	3,4	3,3	3	2,7	1,8	19	19,5
Us.9	3,2	0	2,1	4,6	8,1	2,9	3	3	4,2	0	2	4,1	5,6	2,9	3,4	8,5	4,3	4,1	3,9	2,1	2,9	1,8	2,7	2,6	2,7	15	15,5
Us.10	2,6	3,6	2,2	2,2	4,1	1,2	1,2	1,2	2,4	3,6	0	2,3	2,6	2,3	2,8	6,7	2,2	2,5	2,7	2	2,2	2,8	2,1	1,9	0,9	19	19,5
Us.11	4,5	6,8	5,1	4,8	0,7	4,2	4,2	4,2	2,8	6,8	4,2	0	0,2	4,8	2,4	6	2,8	2,5	1,1	2,7	1,8	3,4	1,7	1,9	3,5	22	22,5
Us.12	4,2	6,6	4,9	4,6	0,45	3,9	3,9	3,9	2,5	6,6	3,9	0,2	0	4,6	2,6	5,7	2,5	2,7	0,95	3,1	2	5,2	1,9	2,1	3,2	22	22,5
Us.13	0,65	2,6	3,8	4,7	4,2	4,7	3,4	4,7	2,6	2,6	2,9	4,5	4,2	0	0,95	9,8	2,6	2,8	2	1,8	1,2	1,1	1,2	1,1	3,3	17	17,5
Us.14	1,6	5,1	8,1	8,8	3,9	7,7	6,3	8,9	6,4	5,1	4	4,2	3,9	2,9	0	9,5	6,3	6,6	1,4	4,1	0,7	3,8	0,8	2	7	19	19,5
Us.15	9,7	9,3	8,4	8,6	5,9	7,4	7,4	7,4	5,2	9,3	7,4	6	5,7	10	11	0	5,3	4,8	6,7	7,7	7,2	11	7,1	7,3	6,1	23	23,5
Us.16	3,4	4,5	3,5	3,2	2,7	2,5	2,6	2,6	0,3	4,5	2,5	2,6	2,5	3,2	4	5,3	0	0,4	3	2,9	3,1	3,3	3	3	1,8	19	19,5
Us.17	3,6	4,6	3,7	3,4	2,8	2,7	2,7	2,7	0,55	4,6	2,7	2,8	2,6	3,3	4,1	4,8	0,4	0	3,1	3	3,5	3,5	3,1	2,9	1,6	18	18,5
Us.18	2	5,5	6,1	5,8	0,65	5,1	4,5	5,1	3	5,5	5,1	0,9	1,4	3,3	1,6	6,9	3,7	3,9	0	2,5	1,3	3	1,4	1,5	3,7	20	20,5
Us.19	2,9	1,4	2,1	3	3,8	1,9	2	2	3,2	1,4	1,6	3,1	4,7	2,7	3,2	7,5	3,2	3,2	3,6	0	2,6	1,5	2,5	2,4	1,7	16	16,5
Us.20	1,6	4	4,4	5,1	1,5	4,4	4	4	3	4	4,4	1,3	2,3	3	0,9	7,8	3,1	3,2	1,1	2,4	0	2,8	0,6	1,3	3,3	19	19,5
Us.21	3	1,4	2,7	4,5	4,6	3,5	2,8	3,6	4,1	1,4	1,8	3,9	6,2	2,8	3,3	8,4	4	4,1	3,7	2	2,7	0	2,6	2,5	2,6	16	16,5
Us.22	1,5	5,4	4,2	2,7	2,5	4,6	4,1	4,7	3,2	5,4	4,6	1,7	1,9	2,8	0,8	7,2	3,3	3,4	1,3	2,4	0,2	3,9	0	1,4	4	19	19,5
Us.23	1,2	2,5	2,8	3,9	2,6	3,2	3,3	3,3	1,9	2,5	2,9	1,8	2,1	0,9	1,1	6,9	2,1	2	1,6	1,1	0,6	1,1	0,5	0	2,6	18	18,5
Us.24	3	4,1	3,1	2,8	3,4	2,1	2,1	2,1	1,8	4,1	2,1	3,5	3,2	2,7	3,3	5,8	1,8	1,5	3	2,4	2,7	2,9	2,6	2,3	0	21	21.5
Us.25	18	15	15	17	20	16	19	18	20,5	15	19	21	20	17	18	24	20	22	20	17	19	18	19	18	21,5	0	0,5
Us.26	18,5	15,5	15,5	17,5	21,5	16,5	19,5	18,5	21	15,5	19,5	21,5	20,5	17,5	18,5	24,5	20,5	22,5	20,5	17,5	19,5	18,5	19,5	18,5	22	0,5	0

Tabla A.2: Tabla de distancias (en kilómetros).

A.3. Tiempo entre nodos

En la Tabla A.3 se representan los tiempos, medidos en minutos, entre las direcciones de los usuarios.

Estos datos han sido obtenidos con la aplicación *Google Maps* a partir de las direcciones dadas en la Tabla A.1. Se han recogido entre las 19h y las 20:30h, para que fuesen lo más realistas posibles, ya que el tráfico varía a lo largo del día.

MIN	C.D	Us.1	Us.2	Us.3	Us.4	Us.5	Us.6	Us.7	Us.8	Us.9	Us.10	Us.11	Us.12	Us.13	Us.14	Us.15	Us.16	Us.17	Us.18	Us.19	Us.20	Us.21	Us.22	Us.23	Us.24	Us.25	Us.26
C.D	0	11	15	15	5	13	13	13	10	11	15	7	5	5	3	13	10	15	5	8	3	8	3	8	15	18	19
Us.1	9	0	8	11	11	9	9	9	11	0	6	13	13	7	10	18	11	11	12	6	8	4	7	8	8	17	18
Us.2	12	6	0	5	9	3	4	3	8	6	5	11	10	6	8	14	7	8	12	6	8	5	8	9	6	15	16
Us.3	11	13	6	0	5	4	4	4	5	13	5	8	7	9	11	11	5	6	9	8	9	9	9	8	4	16	17
Us.4	6	9	10	8	0	8	8	8	6	9	8	3	1	6	7	9	6	6	2	9	5	10	5	6	6	19	20
Us.5	13	10	4	2	7	0	1	0	5	10	2	8	7	8	11	13	5	6	10	8	9	8	9	8	4	15	16
Us.6	11	8	2	4	8	1	0	1	5	8	4	9	8	9	11	13	6	6	11	8	11	9	10	9	5	17	18
Us.7	10	9	3	3	7	1	1	0	5	9	2	8	7	9	11	12	5	5	10	7	9	8	9	8	4	16	17
Us.8	10	12	8	5	5	5	5	6	0	12	6	6	4	9	11	9	1	2	7	7	9	9	9	8	4	20	21
Us.9	9	0	8	11	11	9	9	9	11	0	6	13	13	7	10	18	11	11	12	6	8	4	7	8	8	17	18
Us.10	9	9	5	5	8	4	4	4	6	9	0	7	8	7	10	12	6	5	9	6	7	8	7	6	3	19	20
Us.11	8	9	10	8	2	8	8	8	5	9	9	0	0	6	8	9	6	6	2	9	5	9	5	6	6	20	21
Us.12	7	10	9	7	1	7	7	8	5	10	8	1	0	5	8	8	5	5	3	9	6	7	6	6	6	19	20
Us.13	2	6	10	10	5	11	11	9	9	6	8	5	5	0	3	13	8	8	6	6	4	2	4	4	9	17	18
Us.14	5	9	12	13	7	13	15	14	13	9	9	8	7	6	0	15	11	11	4	10	2	9	2	6	12	20	21
Us.15	15	20	14	11	9	13	13	13	9	20	14	10	9	13	15	0	9	8	11	14	14	15	14	14	10	22	22
Us.16	10	12	7	6	5	5	6	6	1	12	6	6	5	9	11	9	0	2	7	7	9	8	8	8	4	19	20
Us.17	11	12	7	6	6	6	6	6	2	12	7	7	5	9	11	8	2	0	8	8	9	8	9	8	4	18	19
Us.18	6	11	10	9	2	9	10	10	8	11	10	3	3	7	5	10	7	7	0	7	4	7	4	4	7	19	20
Us.19	8	4	8	8	11	7	7	7	9	4	6	10	10	7	9	14	9	9	10	0	7	4	7	7	6	18	19
Us.20	6	10	12	10	4	11	11	12	8	10	11	5	5	7	3	12	8	8	4	6	0	7	2	4	8	19	20
Us.21	9	4	7	9	12	9	7	8	10	4	6	10	11	7	9	15	10	10	10	5	7	0	6	6	6	16	16
Us.22	5	10	13	11	8	11	13	11	9	10	12	6	6	6	3	14	8	8	4	7	1	8	0	5	9	19	20
Us.23	4	7	9	9	8	9	9	9	7	7	9	6	7	3	4	14	7	6	6	4	1	3	1	0	7	19	20
Us.24	9	11	6	5	6	5	4	5	3	11	5	6	6	8	6	10	3	4	8	6	7	7	7	7	0	17	18
Us.25	19	18	17	16	19	18	19	16	20	18	19	20	20	17	19	25	19	20	19	18	20	19	20	20	20	0	0,5
Us.26	19	19	17	17	20	19	19	17	21	19	20	21	21	18	20	25	20	21	20	19	21	20	21	21	21	0,5	0

Tabla A.3: Tabla de tiempos (en minutos).

Apéndice B

Código AMPL

B.1. Problema

En el fichero aquí mostrado escribiremos la formulación del problema que queremos resolver. Indicamos las variables, la función objetivo y las restricciones de nuestro problema.

```
set N ordered;  # Conjunto de usuarios (el nodo 0 representa el centro
    de día).

param n:= card(N);  # Número de usuarios.

param s >=0;  # Número de usuarios que usan silla de ruedas (siempre los
     colocamos como los s primeros en el fichero de datos).

set R ordered;  # Conjunto de rutas.

set V ordered;  # Conjunto de vehículos (disponemos de 2 vehículos; el
    primero para 7 válidos y el segundo para 2 sillas y 4 válidos).

param C {i in N union {0}, j in N union {0}} >= 0;  # Tiempos.

var X{i in N union {0}, j in N union {0}, v in V, r in R} binary;  #
    variable binaria que vale 1 si el vehículo v en la ruta r va de un
    centro/usuario i al centro/usuario j.

var U{i in N, v in V, r in R} >= 0;  # Variable auxiliar que se usa al
    poner las restricción para que no se formen ciclos improcedentes.
```

```
# OBJETIVO:

minimize Time: sum{i in N union {0}, j in N union {0}, v in V, r in R} C
    [i,j]*X[i,j,v,r];  # Función objetivo del problema, en este caso,
    minimizar el tiempo total.

# RESTRICCIONES:

subject to Start {r in R}: sum{j in N, v in V} X[0,j,v,r] = 1;  #
    Restricción 1: indica que cada ruta debe empezar en el nodo 0, es
    decir,  el centro de día.

subject to End {r in R}: sum{j in N, v in V} X[j,0,v,r] = 1;  #
    Restricción 2: indica que cada ruta acaba en el centro de día.

subject to Allpickup {i in N}: sum{j in N union {0}, v in V, r in R} X[i
    ,j,v,r] =1;      # Restricción 3: recogemos a todos los usuarios.

subject to Nolink {i in N union {0}, v in V, r in R}: X[i,i,v,r]=0;  #
    Restricción 4: no se pueden forman lazos.

subject to Balance {i in N , v in V, r in R}: sum{j in N union {0}} X[i,
    j,v,r] - sum{j in N union {0}} X[j,i,v,r] = 0;  # Restricción 5:
    conservación del flujo, a cada dirección solo se llega y se sale una
     vez.

subject to SubtourElimination {i in N, j in N, v in V, r in R}: U[i,v,r
    ]-U[j,v,r]+n*X[i,j,v,r]<=n-1;  # Restricción 6: no se pueden formar
    ciclos improcedentes.

subject to NoWheelchairs1 {i in N union {0}, j in first(N)..first(N)+s
    -1, r in R}: X[i,j,1,r] =0;  # Restricción 7: en el vehículo 1 no
    pueden ir sillas.

subject to cap1 {r in R}: sum{i in N, j in N union {0}} X[i,j,1,r] <= 7;
      # Restricción 8: La capacidad de vehículo 1 son 7 usuarios.

subject to cap2 {r in R}: sum{i in first(N)..first(N)+s-1, j in N union
    {0}} X[i,j,2,r] <= 2; # Restricción 9: En el vehículo 2 van a lo
    sumo 2 sillas.

subject to cap2b {r in R}: sum{i in first(N)+s..first(N)+n-1, j in N
    union {0}} X[i,j,2,r] <= 4; # Restricción 10: En el vehículo 2 van a
     lo sumo 4 usuarios válidos.
```

B.2. Fichero de los datos de distancias

En el siguiente fichero se muestran los datos de distancias proporcionados al programa:

```
# Datos de ejemplo del problema del centro de día de Lugo. Usuarios:
    todos, del 1 al 26.

set N := 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
    25 26  ;  # Conjunto de los usuarios (centro de día: 0).

set R := 1 2 3 4 5 6;        # Conjunto de rutas. Fijamos el número de
    las rutas dependiendo del caso en el que estemos. Hemos probado con
    6, 5 y 4 rutas.

set V := 1 2 ; # Conjunto de vehículos: 2. El 1 es de válidos y el 2 es
    mixto (válidos y sillas).

# Para poder visualizar correctamente los datos en Latex, he dividido la
     tabla en dos, de manera que quedan ordenados y legibles.

param C:

     0    1    2    3    4    5    6    7    8    9    10   11   12

 0   0    4.6  5.9  7.1  3.6  6    5.4  6.1  5.5  4.6  4.7  3.8  3.6
 1   3.2  0    2.1  4.6  8.1  2.9  3    3    4.2  0    2    4.1  5.6
 2   6.1  2.1  0    2.8  8.7  1.2  1.3  1.2  3.5  2.1  1.5  3.7  4.9
 3   4.1  4.9  2.8  0    4.8  1.8  1.8  1.8  3.1  4.9  1.8  4.8  4.6
 4   4.2  6.6  5.1  4.8  0    4.1  4.2  4.1  2.7  6.6  4.1  0.65 0.4
 5   6    3.5  14   1.6  4    0    0.4  0.03 2.3  3.5  0.5  4    3.8
 6   3.5  3.1  0.95 1.9  4.2  0.27 0    0.3  2.6  3.1  0.65 4.3  4
 7   3.2  3.4  1.3  1.6  3.9  0.35 0.23 0    2.3  3.4  0.45 4    3.7
 8   3.4  4.5  3.5  3.2  2.7  2.5  2.5  2.5  0    4.5  2.5  2.7  2.5
 9   3.2  0    2.1  4.6  8.1  2.9  3    3    4.2  0    2    4.1  5.6
10   2.6  3.6  2.2  2.2  4.1  1.2  1.2  1.2  2.4  3.6  0    2.3  2.6
11   4.5  6.8  5.1  4.8  0.7  4.2  4.2  4.2  2.8  6.8  4.2  0    0.2
12   4.2  6.6  4.9  4.6  0.45 3.9  3.9  3.9  2.5  6.6  3.9  0.2  0
13   0.65 2.6  3.8  4.7  4.2  4.7  3.4  4.7  2.6  2.6  2.9  4.5  4.2
14   1.6  5.1  8.1  8.8  3.9  7.7  6.3  8.9  6.4  5.1  4    4.2  3.9
15   9.7  9.3  8.4  8.6  5.9  7.4  7.4  7.4  5.2  9.3  7.4  6    5.7
16   3.4  4.5  3.5  3.2  2.7  2.5  2.6  2.6  0.3  4.5  2.5  2.6  2.5
17   3.6  4.6  3.7  3.4  2.8  2.7  2.7  2.7  0.55 4.6  2.7  2.8  2.6
18   2    5.5  6.1  5.8  0.65 5.1  4.5  5.1  3    5.5  5.1  0.9  1.4
```

```
19  2.9  1.4  2.1   3   3.8  1.9   2    2   3.2  1.4  1.6  3.1   4.7
20  1.6   4   4.4  5.1  1.5  4.4   4    4    3    4   4.4  1.3   2.3
21   3   1.4  2.7  4.5  4.6  3.5  2.8  3.6  4.1  1.4  1.8  3.9   6.2
22  1.5  5.4  4.2  2.7  2.5  4.6  4.1  4.7  3.2  5.4  4.6  1.7   1.9
23  1.2  2.5  2.8  3.9  2.6  3.2  3.3  3.3  1.9  2.5  2.9  1.8   2.1
24   3   4.1  3.1  2.8  3.4  2.1   21  2.1  1.8  4.1  2.1  3.5   3.2
25   18   15   15   17   20   16   19   18  20.5  15   19   21    20
26  18.5 15.5 15.5 17.5 21.5 16.5 19.5 18.5  21  15.5 19.5 21.5  20.5


 13   14    15   16   17   18   19   20   21   22    23  24   25   26 :=

2.1  0.65  9.1  5.5   5   1.7  3.8  0.9   3   0.95  2.5  5.4  18  18.5
2.9  3.4   8.5  4.3  4.1  3.9  2.1  2.9  1.8  2.7   2.6  2.7  15  15.5
5.5  6.6   8.4  3.5  3.7  4.3   2   3.3  2.2  3.2   3.1  2.3  15  15.5
3.8  4.4   8.7  3.1  3.3  5.1  3.5  3.7   4   3.6   3.9  2.7  16  16.5
4.6  2.4   5.9  2.7  2.9  0.65 6.1  1.8  3.4  1.8   1.9  3.4  21  21.5
 3   3.5   7.2  2.3  2.5  4.2  2.7  2.9  3.1  2.8   2.6  1.9  18  18.5
3.3  3.8   7.5  2.6  2.8   5    3   3.2  3.5  3.1    29  2.2  16  16.5
 3   3.5   7.2  2.3  2.5  4.2  2.7  2.9  3.1  2.8   2.6  1.9  18  18.5
3.1  3.9   5.2  0.3  0.55 2.9  2.8  3.4  3.3   3    2.7  1.8  19  19.5
2.9  3.4   8.5  4.3  4.1  3.9  2.1  2.9  1.8  2.7   2.6  2.7  15  15.5
2.3  2.8   6.7  2.2  2.5  2.7   2   2.2  2.8  2.1   1.9  0.9  19  19.5
4.8  2.4    6   2.8  2.5  1.1  2.7  1.8  3.4  1.7   1.9  3.5  22  22.5
4.6  2.6   5.7  2.5  2.7  0.95 3.1   2   5.2  1.9   2.1  3.2  22  22.5
0    0.95  9.8  2.6  2.8   2   1.8  1.2  1.1  1.2   1.1  3.3  17  17.5
2.9   0    9.5  6.3  6.6  1.4  4.1  0.7  3.8  0.8    2    7   19  19.5
10    11    0   5.3  4.8  6.7  7.7  7.2   11  7.1   7.3  6.1  23  23.5
3.2   4    5.3   0   0.4   3   2.9  3.1  3.3   3     3   1.8  19  19.5
3.3  4.1   4.8  0.4   0    3.1   3   3.5  3.5  3.1   2.9  1.6  18  18.5
3.3  1.6   6.9  3.7  3.9   0   2.5  1.3   3   1.4   1.5  3.7  20  20.5
2.7  3.2   7.5  3.2  3.2  3.6   0   2.6  1.5  2.5   2.4  1.7  16  16.5
 3   0.9   7.8  3.1  3.2  1.1  2.4   0   2.8  0.6   1.3  3.3  19  19.5
2.8  3.3   8.4   4   4.1  3.7   2   2.7   0   2.6   2.5  2.6  16  16.5
2.8  0.8   7.2  3.3  3.4  1.3  2.4  0.2  3.9   0    1.4   4   19  19.5
0.9  1.1   6.9  2.1   2   1.6  1.1  0.6  1.1  0.5    0   2.6  18  18.5
2.7  3.3   5.8  1.8  1.5   3   2.4  2.7  2.9  2.6   2.3   0   21  21.5
 17   18    24   20   22   20   17   19   18   19    18  21.5  0   0.5
17.5 18.5  24.5 20.5 22.5 20.5 17.5 19.5 18.5 19.5  18.5  22  0.5   0  ;

# km

param s:=3; # Indica que son 3 los usuarios que necesitan silla de
    ruedas.
```

B.3. Fichero de los datos de tiempos

En el siguiente fichero se muestran los datos de tiempos proporcionados al programa:

```
# Datos de ejemplo del problema del centro de día de Lugo. Usuarios:
    todos, del 1 al 26.

set N := 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
    25 26  ;  # Conjunto de los usuarios (centro de día: 0).

set R := 1 2 3 4 5 6;      # Conjunto de rutas. Fijamos el número de las
    rutas dependiendo del caso en el que estemos. Hemos probado con 6, 5
     y 4 rutas.

set V := 1 2 ; # Conjunto de vehículos: 2. El 1 es de válidos y el 2 es
    mixto (válidos y sillas).

# Para poder visualizar correctamente los datos en Latex, he dividido la
     tabla en dos, de manera que quedan ordenados y legibles.

param C:
     0     1     2     3     4     5     6     7     8     9     10    11    12

0    0    11    15    15     5    13    13    13    10    11    15     7     5
1    9     0     8    11    11     9     9     9    11     0     6    13    13
2   12     6     0     5     9     3     4     3     8     6     5    11    10
3   11    13     6     0     5     4     4     4     5    13     5     8     7
4    6     9    10     8     0     8     8     8     6     9     8     3     1
5   13    10     4     2     7     0     1     0     5    10     2     8     7
6   11     8     2     4     8     1     0     1     5     8     4     9     8
7   10     9     3     3     7     1     1     0     5     9     2     8     7
8   10    12     8     5     5     5     5     6     0    12     6     6     4
9    9     0     8    11    11     9     9     9    11     0     6    13    13
10   9     9     5     5     8     4     4     4     6     9     0     7     8
11   8     9    10     8     2     8     8     8     5     9     9     0     0
12   7    10     9     7     1     7     7     8     5    10     8     1     0
13   2     6    10    10     5    11    11     9     9     6     8     5     5
14   5     9    12    13     7    13    15    14    13     9     9     8     7
15  15    20    14    11     9    13    13    13     9    20    14    10     9
16  10    12     7     6     5     5     6     6     1    12     6     6     5
17  11    12     7     6     6     6     6     6     2    12     7     7     5
18   6    11    10     9     2     9    10    10     8    11    10     3     3
19   8     4     8     8    11     7     7     7     9     4     6    10    10
```

```
 20   6   10   12   10    4   11   11   12    8   10   11    5    5
 21   9    4    7    9   12    9    7    8   10    4    6   10   11
 22   5   10   13   11    8   11   13   11    9   10   12    6    6
 23   4    7    9    9    8    9    9    9    7    7    9    6    7
 24   9   11    6    5    6    5    4    5    3   11    5    6    6
 25  19   18   17   16   19   18   19   16   20   18   19   20   20
 26  19   19   17   17   20   19   19   17   21   19   20   21   21



 13   14   15   16   17   18   19   20   21   22   23   24   25   26   :=

  5    3   13   10   15    5    8    3    8    3    8   15   18   19
  7   10   18   11   11   12    6    8    4    7    8    8   17   18
  6    8   14    7    8   12    6    8    5    8    9    6   15   16
  9   11   11    5    6    9    8    9    9    9    8    4   16   17
  6    7    9    6    6    2    9    5   10    5    6    6   19   20
  8   11   13    5    6   10    8    9    8    9    8    4   15   16
  9   11   13    6    6   11    8   11    9   10    9    5   17   18
  9   11   12    5    5   10    7    9    8    9    8    4   16   17
  9   11    9    1    2    7    7    9    9    9    8    4   20   21
  7   10   18   11   11   12    6    8    4    7    8    8   17   18
  7   10   12    6    5    9    6    7    8    7    6    3   19   20
  6    8    9    6    6    2    9    5    9    5    6    6   20   21
  5    8    8    5    5    3    9    6    7    6    6    6   19   20
  0    3   13    8    8    6    6    4    2    4    4    9   17   18
  6    0   15   11   11    4   10    2    9    2    6   12   20   21
 13   15    0    9    8   11   14   14   15   14   14   10   22   22
  9   11    9    0    2    7    7    9    8    8    8    4   19   20
  9   11    8    2    0    8    8    9    8    9    8    4   18   19
  7    5   10    7    7    0    7    4    7    4    4    7   19   20
  7    9   14    9    9   10    0    7    4    7    7    6   18   19
  7    3   12    8    8    4    6    0    7    2    4    8   19   20
  7    9   15   10   10   10    5    7    0    6    6    6   16   16
  6    3   14    8    8    4    7    1    8    0    5    9   19   20
  3    4   14    7    6    6    4    1    3    1    0    7   19   20
  8    6   10    3    4    8    6    7    7    7    7    0   17   18
 17   19   25   19   20   19   18   20   19   20   20   20    0    1
 18   20   25   20   21   20   19   21   20   21   21   21    1    0    ;

#min

param s:=3; # Indica que son 3 los usuarios que necesitan silla de
    ruedas.
```

B.4. Fichero de ejecución

En los siguientes ficheros se dan las órdenes al solucionador, es decir, le decimos que queremos que haga con el problema y los datos que le hemos enviado. Son tres ficheros diferentes porque en cada uno damos una orden diferente en función de las necesidades.

```
option gurobi_options $gurobi_options 'timelim 14400';  # El dato del
    timelim lo vamos variando. Es el tiempo de ejecución de nuestro
    programa, por ejemplo, ''timelim 14400'', da la orden de que al
    cumplirse 14400 s = 4h de tiempo de ejecución, el programa debe
    parar, se fuerza el cierre.

solve ; # Comando que manda resolver el problema.
display _varname, _var, Time;  #Es el comando que indica que queremos
    que nos devuelva las variables y el tiempo de ejecución.
```

```
option gurobi_options $gurobi_options 'mipgap 0.01';  # La opción de ''
    mipgap 0.01'' implica que se de la solución a la que ha llegado
    cuando el gap sea 0.01.

solve ; # Comando que manda resolver el problema.
display _varname, _var, Time;  #Es el comando que indica que queremos
    que nos devuelva las variables y el tiempo de ejecución.
```

Esta opción es una combinación de las dos anteriores y será la que finalmente utilicemos en todos las salidas expuestas tras comparar los resultados obtenidos utilizando los dos ficheros anteriores. Esta opción nos permite comparar como de buenas son las soluciones obtenidas.

```
option gurobi_options $gurobi_options 'timelim 7200 bestbound 1'; # Esta
     opción es una combinación de las dos anteriores. ''timelim 7200
    bestbound 1'' indica que al llegar al tiempo establecido, en este
    caso 7200s, además de devolver la solución, calcule el gap obtenido.

solve ; # Comando que manda resolver el problema.
display _varname, _var, Time;  #Es el comando que indica que queremos
    que nos devuelva las variables y el tiempo de ejecución.
```

Bibliografía

[1] Applegate, D.L., Bixby, R.E., Chvátal, V. and Cook, W.J. (2006). *The Traveling Salesman Problem: A Computational Study.* Princeton University Press, cop.

[2] Applegate, D.L., Bixby, R.E., Chvátal, V. and Cook, W.J. (2011). *The Traveling Salesman Problem.* Princeton University Press, cop.

[3] Aynos, A. y de Pinto, L. *El problema del viajante de comercio: análisis teórico y estrategias de resolución.* Universidad Carlos III, Madrid.

[4] Bailer-Jones, C.A.L., Rybizki, J., Fouesneau, M., Mantelet, G. and Andrae, R. (2018) *Estimating distances from parallaxes IV: Distances to 1.33 billion stars in Gaia. Data relase 2.* Astronomical Journal, Vol 156, 58. Max Planck Institute for Astronomy, Heidelberg, Germany.

[5] Bazaraa, M., Jarvis, J.J. and Sherali, H. (2011). *Programación lineal y flujo en redes.* México, Limusa.

[6] Calviño, A. (2011). *Cooperación en los problemas del viajante (TSP) y de rutas de vehículos (VRP): una panorámica.* Tesis del Máster Interuniversitario en Técnicas Estadísticas. Universidad de la Coruña, Universidad de Santiago de Compostela y Universidad de Vigo.

[7] Campos Laclaustra, J. (1995). *Estructuras de datos y algoritmos.* Zaragoza, Prensas Universitarias de Zaragoza.

[8] Clarke, G. and Wright, J.R. (1964). *Scheduling of Vehicle Routing Problem from a Central Depot to a Number of Delivery Points.* Operations Research, Vol 12, 568-581.

[9] Cordeau, J.F., Laporte, G., Savelsbergh, M.W.P and Vigo, D. (2007). *Transportation. HandBooks in Operations Research and Management Science.* Vol 14, 367-428. Amsterdam, Elsevier.

[10] Dantzig, G.B. (1963). *Linear Programming and Extensions.* Pricenton, Pricenton University Press.

[11] Dantzig, G.B. and Ramser, J.H. (1959). *The Truck Dispatching Problem.* Management Science, Vol 6, n.1, 80-91.

[12] Dantzig, G.B., Fulkerson, D.R. and Johnson, S.M. (1954). *On a linear programming combinatorial approach to the traveling-salesman problem.* Operations Research, Vol 7, 58-66.

[13] Dantzig, G.B., Fulkerson, D.R. and Johnson, S.M. (1954). *Solution of a large scale traveling salesman problem.* Operations Research, Vol 2, 393-410.

[14] Flood, M.M. (1956). *The traveling salesman problem.* Operations Research, Vol 4, 61-75.

[15] Flores Cabezas, X.A. (2014). *Problemas del Milenio: P vs NP.* Revista de divulgación. Asociación Amarum, Vol 1, 1-15.

[16] Fourer, R., Gay, D. and Kernighan, B. (2003). *AMPL. A modeling language por mathematical programming.* Canada, Thomson Learning.

[17] García, J.J., Hontoria, E. and Aleksovski, D. (2015). *El problema del viajante de comercio: Búsqueda de soluciones y herramientas asequibles.* Revista electrónica de comunicaciones y trabajos de ASEPUMA. Vol 16, 117-133.

[18] Golden, B., Raghavan, S. and Wasil, E. (2008). *The vehicle routing problem: latest advances and new challenges.* New York, Springer.

[19] González Diaz, J. (2020). *Apuntes de la materia Programación Lineal e Enteira. Curso 2020-2021.* Universidad de Santiago de Compostela.

[20] Gutin, G. and Punnen, A. (2007). *The traveling salesman problem and its variations.* New York, Springer.

[21] Hillier, F.S. and Lieberman, G.J. (2010). *Introducción a la investigación de operaciones.* México, Mc Graw Hill. Novena edición.

[22] Irnich, S., Toth, P. and Vigo, D. (2014). *Chapter 1: The Family of Vehicle Routing Problems.* Researchgate.net. Artículo disponible en: https://www.researchgate.net/publication/284529903_Chapter_1_The_Family_of_Vehicle_Routing_Problems (Visitada 30 de junio de 2021)

[23] Jungnickel, D. (2012). *Graphs, Networks and Algorithms.* New York, Springer.

[24] Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G. and Shmoys, D.B. (1985). *The Traveling Salesman Problem.* New York, John Wiley and Sons.

[25] Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G. and Shmoys, D.B. (1991). *The Traveling Salesman Problem: A guided tour of combinatorial optimization.* New York, John Wiley and Sons.

[26] Little, J., Murky, K., Sweedney, D. and Karel, C. (1963). *An algorithm for traveling salesman problem.* Operations Research, Vol 11, 972-989.

[27] Manber, U. (1989). *Introduction to Algorithms. A Creative Approach.* Addison-Wesley.

[28] Mencía Castellana, R. (2017). *Tesis doctoral: Metaheurísticas para problemas de scheduling con múltiples recursos.* Universidad de Oviedo, Dep. Informática.

[29] Méndez, I. (2018). *Modelización y heurísticas para un problema de planificación de tareas en una empresa de atención a personas dependientes.* Tesis del Máster Interuniversitario en Técnicas Estadísticas. Universidad de la Coruña, Universidad de Santiago de Compostela y Universidad de Vigo.

[30] Menger, K. (1932). *Das botenproblem.* Ergebnisse Eines Mathematischen Kolloquiums, Vol 2, 11-12.

[31] Miller, C.E., Tucker, A.W. and Zemlin, R.A. (1960). *Integer Programming Formulations and Traveling Salesman Problems.* Journal of Association for Computing Machinery, Vol 7, 326-329.

[32] Moreno Soto, F. (2016). *Cómo la naturaleza nos muestra una solución a algunos problemas difíciles: el recocido simulado.* Números: Revista de Didáctica de las Matemáticas. http://www.sinewton.org/numeros. ISSN: 1887-1984. Vol 92, julio 2016, 35-48.

[33] Página web oficial de AMPL: https://ampl.com/ (Visitada 23 de junio de 2021)

[34] Página web de Arte mediante la resolución de TSP: https://www2.oberlin.edu/math/faculty/bosch/tspart-page.html (Visitada 27 de junio de 2021)

[35] Página web del Monalisa TSP Challenge: http://www.math.uwaterloo.ca/tsp/data/ml/monalisa.html (Visitada 21 de junio de 2021)

[36] Página web oficial de NEOS: https://neos-server.org/neos/ (Visitada 23 de junio de 2021)

[37] Página web oficial del Solver CONCORDE: http://www.math.uwaterloo.ca/tsp/concorde/index.html (Visitada 23 de junio de 2021)

[38] Página web oficial del solver GUROBI: https://www.gurobi.com/ (Visitada 23 de junio de 2021)

[39] Página web del TSP Star Tours: http://www.math.uwaterloo.ca/tsp/star/index.html (Visitada 27 de junio de 2021)

[40] Página web de la Universidad de Waterloo: http://www.math.uwaterloo.ca/tsp/index.html (Visitada 28 de junio de 2021)

[41] Reinelt, G. (1994). *The Traveling salesman: computational solutions for TSP applications.* Berlin, Springer-Verlag.

[42] Robinson, J.B. (1949). *On the Hamiltonian game (a traveling-salesman problem).* Rand Corporation.

[43] Rocha, L.B., González, C. and Orjuela, J. (2011). *Una revisión al estado del arte del problema de ruteo de vehículos. Evolución histórica y métodos de solución.* Artículo de revisión. Universidad Districtal José de Caldas. Ingeniería, Vol 2, n 2.

[44] Rodríguez Pérez, J. (2012). *Caracterización, modelado y determinación de las rutas de la flota en una empresa de rendering.* Disponible en: http://bibing.us.es/proyectos/abreproy/70319/fichero/Jorge+Rodriguez+Perez _TRABAJO+FIN+DE+MASTER.pdf

[45] Vigo, D. and Toth, P. (2002). *The vehicle routing problem.* Philadelphia, Society for Industrial and Applied Mathematics.