VNIVERSITAT DE VALÈNCIA

FACULTAD DE MATEMÁTICAS - Curso 2018/ 2019
Trabajo fin de grado:

Algunos algoritmos heurísticos para el Problema de Rutas de Vehículos con Capacidades

Realizado por:
Juan Francisco Gómez González

Dirigido por:
Ángel Corberán Salvador
Departamento de estadística e investigación operativa

Facultat de Ciències Matemàtiques

Dedicado a Manuel, gran amigo y compañero

Índice general

1. Introducción **1**

2. Problemas de Rutas de vehículos **3**
2.1. Problema de rutas por vértices . . . 3
2.2. Problemas de rutas por arcos . . . 5

3. Preliminares matemáticos **6**
3.1. Teoría de grafos . . . 6
3.2. Programación lineal y entera . . . 8
3.2.1. Programación lineal . . . 8
3.2.2. Programación Lineal Entera . . . 10
3.3. Algoritmos heurísticos y metaheurísticos . . . 12
3.3.1. Algoritmos heurísticos . . . 13
3.3.2. Algoritmos metaheurísticos . . . 13

4. El problema del viajante **14**
4.1. Formulación . . . 14
4.2. Un algoritmo heurístico para el TSP . . . 15
4.2.1. Algoritmo de los ahorros (savings) . . . 15
4.2.2. Un procedimiento de búsqueda local . . . 16
4.2.3. Aleatoriedad en la elección del s_{ij} . . . 16

5. El problema de rutas de vehículos con capacidades **17**
5.1. Definición del problema . . . 17
5.2. Formulación del problema . . . 18

6. Dos algoritmos heurísticos para la resolución del CVRP **20**
6.1. Algoritmo de los ahorros para el CVRP . . . 20
6.1.1. Ejemplo del algoritmo . . . 20
6.1.2. Un procedimiento de búsqueda local . . . 23
6.1.3. Aleatorización de los savings . . . 24
6.2. Algoritmo heurístico extendido de Ulusoy (Extended Ulusoy's heuristic, EUH) . . . 24
6.2.1. Camino más corto (CMC) . . . 24
6.2.2. Ejemplo del algoritmo de Ulusoy . . . 26

7. Resultados computacionales **30**
7.1. Instancias utilizadas 30
7.2. Ajuste del parámetro α 31
7.3. Comparación de los algoritmos propuestos 37
7.4. Comparación del óptimo de las instancias con la solución computada 40

8. Conclusiones **43**

Índice de cuadros

7.1. Medias y varianzas asociadas a cada valor de α al utilizar el algoritmo de los ahorros . 31
7.2. Medias y varianzas con los distintos α en las instancias con 35 ciudades 32
7.3. Medias y varianzas con los distintos α en las instancias con 50 ciudades 32
7.4. Medias y varianzas con los distintos α en las instancias con 65 ciudades 33
7.5. Medias y varianzas asociadas a cada valor de α para las instancias con 3 camiones . . 33
7.6. Medias y varianzas asociadas a cada valor de α para las instancias con 4 camiones . . 33
7.7. Medias y varianzas asociadas a cada valor de α para las instancias con 5 camiones . . 33
7.8. Medias y varianzas asociadas a cada valor de α para las instancias con 6 camiones . . 33
7.9. Medias y varianzas asociadas a cada valor de α para las instancias con 7 camiones . . 33
7.10. Medias y varianzas asociadas a cada valor de α para las instancias con 8 camiones . . 34
7.11. Medias y varianzas asociadas a cada valor de α al utilizar el algoritmo de Ulusoy . . . 34
7.12. Medias y varianzas con los distintos α en las instancias con 35 ciudades 35
7.13. Medias y varianzas con los distintos α en las instancias con 50 ciudades 35
7.14. Medias y varianzas con los distintos α en las instancias con 65 ciudades 35
7.15. Medias y varianzas asociadas a cada valor de α para las instancias con 3 camiones . . 36
7.16. Medias y varianzas asociadas a cada valor de α para las instancias con 4 camiones . . 36
7.17. Medias y varianzas asociadas a cada valor de α para las instancias con 5 camiones . . 36
7.18. Medias y varianzas asociadas a cada valor de α para las instancias con 6 camiones . . 36
7.19. Medias y varianzas asociadas a cada valor de α para las instancias con 7 camiones . . 36
7.20. Medias y varianzas asociadas a cada valor de α para las instancias con 8 camiones . . 36
7.21. Resumen de cada algoritmos 37
7.22. Resultados de los algoritmos en las instancias con 35 ciudades 38
7.23. Resultados de los algoritmos en las instancias con 50 ciudades 38
7.24. Resultados de los algoritmos en las instancias con 65 ciudades 39
7.25. Resultados de los algoritmos en las instancias con 3 camiones disponibles 39
7.26. Resultados de los algoritmos en las instancias con 4 camiones disponibles 39
7.27. Resultados de los algoritmos en las instancias con 5 camiones disponibles 39
7.28. Resultados de los algoritmos en las instancias con 6 camiones disponibles 39
7.29. Resultados de los algoritmos en las instancias con 7 camiones disponibles 40
7.30. Resultados de los algoritmos en las instancias con 8 camiones disponibles 40
7.31. Comparación de resultados 41
7.32. Desviaciones respecto del valor óptimo (gap) 41

Índice de figuras

1.1. Solución gráfica 2

3.1. Nodos 1 y 2 11
3.2. Árbol del problema 12

6.1. Ejemplo algoritmo de los ahorros 23
6.2. Instancia con 13 clientes 26
6.3. Solución del TSP (1) y reordenación de los vértices (2) 27
6.4. Construcción el grafo (3) 29
6.5. Solución proporcionada por el algoritmo de Ulusoy (5) 29

7.1. Diagrama de cajas para α_3 y α_4 en el algoritmo de los ahorros 32
7.2. Diagrama de cajas para α_2 y α_3 en el algoritmo de Ulusoy 35
7.3. Diagrama de cajas para el algoritmo de los ahorros y de Ulusoy 38
7.4. Solución computada 42
7.5. Solución óptima 42

Capítulo 1

Introducción

Los problemas de rutas de vehículos son problemas que se resuelven a diario en empresas de reparto de mercancías como Amazon, Correos, Seur, etc. Algunas de estas empresas tienen la opción de ***entregas en un día***, lo cual hace que tengan menos tiempo para hallar la solución óptima. Por este motivo hacen falta buenos algoritmos que calculen soluciones buenas en poco tiempo, dejando de lado llegar al óptimo. Los algoritmos heurísticos no solo hallan una solución buena del problema, sino que también nos sirven de ayuda para encontrar la solución óptima de una manera más rápida usando ***Branch and Bound*** o ***Branch and cut*** (3.2.2).
En este trabajo presentamos algunos algoritmos para el problema de rutas de vehículos con capacidades (capacitated vehicle routing problem, CVRP), los cuales dan soluciones cercanas a las óptimas en un tiempo de computación razonable para instancias con un tamaño considerablemente grande.

La siguiente representación gráfica es una solución a un problema de rutas de vehículos con capacidades. (6.1).

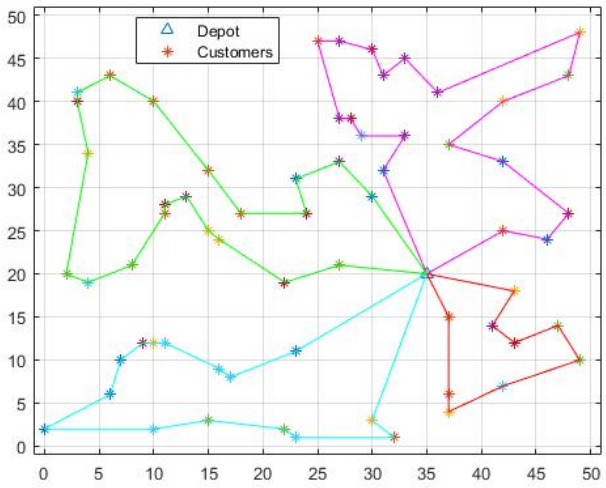

Figura 1.1: Solución gráfica

El depósito es el lugar donde todos los camiones han de salir y donde han de regresar, lo representamos con un triangulo. Los clientes son los lugares donde hemos de ir, se representan con un asterisco.

En este trabajo haremos una introducción al problema del viajante (4), del cual hablaremos de su solución exacta y para el que presentaremos un algoritmo heurístico que se usará, haciendo una pequeña modificación, para la resolución del problema de rutas de vehículos con capacidades.

El capítulo 5 empieza definiendo el problema de rutas de vehículos con capacidades junto con dos de las formulaciones de este problema.

En el capítulo 6 se exponen los dos algoritmos heurísticos que hemos implementado y utilizado en este trabajo. Se les han añadido una componente de aleatoriedad a través de un parámetro, α. Esta componente es calculada en el capítulo 7 mediante unos experimentos computacionales previos. También se ha determinado qué algoritmo funciona mejor y en qué situación es interesante y más ventajoso utilizar dichos algoritmos. Para ello hemos usado 360 instancias de distintas características.
Para hallar el valor del parámetro α, y decidir qué algoritmo es mejor en cada caso, se ha realizado un análisis estadístico con los costes que hemos obtenido al ejecutar las instancias con los algoritmos que hemos programado.

También hemos realizado una comparación entre las soluciones óptimas, conocidas para un conjunto de instancias de la base de datos de la universidad de Málaga [17], y las obtenidas con nuestros dos algoritmos heurísticos (7.4). Esto nos servirá para ver qué desviación (gap) hay entre las soluciones óptimas y las obtenidas por los algoritmos.

Capítulo 2

Problemas de Rutas de vehículos

En este capítulo presentaremos un resumen de los problemas de rutas de vehículos más conocidos. Los problemas de rutas son problemas de optimización combinatoria, que consisten en encontrar la mejor solución (óptima) entre un número finito (pero enorme) o infinito numerable de soluciones posibles. El número de soluciones posibles generalmente depende de $n!$ o de k^n, donde n representa el tamaño del problema y k un entero mayor o igual que 2.

Los problemas de rutas se modelizan en un grafo G, donde existe un coste asociado a los arcos o/y aristas. Estos problemas se dividen en dos clases, los problemas de rutas por vértices y problemas de rutas por arcos. Los primeros se caracterizan porque el servicio a realizar se hace en los vértices del grafo mientras que en los segundos, el servicio se realiza en los arcos o aristas del grafo.

La importancia de esta clase de problemas se debe al gran número de situaciones reales en las que se pueden aplicar. Algunos ejemplos típicos son el reparto de correo, la recogida de basura o el reparto o recogida de las empresas de transporte. Estos problemas no solo pueden ser aplicados en logística y distribución, también sirven para modelizar y resolver otras situaciones como la producción de circuitos electrónicos integrados o la secuenciación de tareas. Esta es la razón por la que en los últimos 40 años se haya hecho un estudio importante de este campo, consiguiendose un avance significativo tanto en la formulación de los problemas como en el diseño, análisis e implementación de algoritmos para su resolución.

2.1. Problema de rutas por vértices

Como hemos comentado anteriormente, la característica principal de estos problemas es que los clientes que requieren servicio están representados como vértices del grafo. Básicamente, consisten en la obtención de uno o varios ciclos que pasen por todos (o algunos de) los vértices del grafo y de forma que la distancia total recorrida sea mínima. Dentro de este tipo de problemas podemos destacar:

- El problema del viajante, **TSP**, donde un único vehiculo ha de visitar todos los vértices del grafo una única vez de manera que la distancia recorrida sea mínima.

- El problema gráfico del viajante, **GTSP**, que es una variante del TSP en la que el grafo no tiene porqué ser completo y el vehiculo puede visitar más de una vez cada vértice del grafo.

- El problema de rutas de vehículos, **VRP**, que es una generalización del TSP en la que la demanda de los clientes requiere el uso de más de un vehículo. Este problema es más dificil que el TSP puesto que se hace necesario particionar el conjunto de clientes para que éstos puedan ser atendidos por los vehículos y después determinar el orden de servicio de cada cliente. Al igual que el TSP, estos vehículos salen de un punto inicial al que llamaremos depósito.
  Algunos casos particulares de este problema son:

  - El problema de rutas de vehículos con restricciones de distancia.
    Cada vehículo, además de tener una capacidad o carga máxima, también está limitado por un tiempo máximo que puede emplear, o una distancia máxima que puede recorrer.
  - El problema de rutas de vehículos con ventanas de tiempo, **VRPTW**.
    En este caso el servicio a cada cliente debe realizarse dentro de una horario o ventana de tiempo. Un cliente puede tener más de una ventana de tiempo. El vehículo no puede atender la demanda de un cliente fuera de su ventana horaria.
  - El problema de rutas de vehículos con demanda compartida.
    En este problema es posible atender la demanda de un cliente con más de un vehículo. Esta relajación del VRP puede significar la reducción del número de vehículos necesarios para atender la demanda.
  - El problema de rutas de vehículos con restricciones de precedencia.
    En ocasiones la colocación de la mercancía en el vehículo obliga a que unos clientes sean servidos antes que otros. Esto supone una secuenciación en el orden de visita de los clientes.
  - El problema del dial-a-ride.
    Este problema está definido por el hecho de que, además de que las rutas deben satisfacer unas ventanas de tiempo de recogida y entrega de los clientes, estos no deben permanecer en el vehículo más de un tiempo límite.
  - El problema de rutas de vehículos abierto.
    Se considera la posibilidad de que las rutas sean abiertas, es decir, caminos que visitan clientes sin necesidad de retornar al depósito.
  - El problema de rutas de vehículos con múltiples depósitos.
    En este caso, existen diversos depósitos que sirven de punto de partida y retorno a los vehículos.
  - El problema de recogida y entrega de mercancías (pick up and delivery). Aquí hay que encontrar un conjunto de rutas de mínimo coste para una flota de vehículos. Cada vehículo tiene una capacidad determinada, un punto de partida y un punto de finalización de la ruta. Cada orden de transporte especifica el tamaño de la carga a transportar, los puntos donde se debe recoger y aquellos a los que hay que llevarla. Cada carga tiene que ser transportada por un vehículo desde sus orígenes a sus destinos sin transbordos en otros puntos.

2.2. Problemas de rutas por arcos

En este tipo de problemas el vehículo ha de recorrer un conjunto de arcos y/o aristas del grafo, donde se encuentra el servicio a realizar. Según el tipo de grafo en el que se definen los problemas de rutas por arcos, éstos se clasifican en:

- **No dirigidos:** el grafo está formado únicamente por aristas. Estas suelen utilizarse para modelizar, por ejemplo, calles de doble sentido. El coste de atravesar una arista es el mismo en los dos sentidos.
- **Dirigidos:** el grafo está formado por arcos únicamente. Suelen utilizarse para modelizar calles de sentido único.
- **Mixtos:** el grafo contiene arcos y aristas.
- **Windy:** el grafo está formado por aristas, pero el coste de recorrerlas depende del sentido en que se haga.

Dentro de estos tipos de problemas, podemos destacar:

- El problema del cartero chino, **CPP**, que consiste en encontrar una ruta cerrada de coste mínimo que recorra cada arco o arista del grafo al menos una vez.
- El problema del cartero con viento, **WPP**, que es la versión del CPP, definido sobre un grafo windy.
- El problema del cartero rural, **RPP**, generalización del CPP donde el servicio a realizar es unicamente un subconjunto de aristas del grafo.
- El problema de rutas por arcos con capacidades, **CARP**. Consiste en encontrar un conjunto de rutas para una flota de vehículos, partiendo y volviendo al depósito, que recorran al menos una vez todas las aristas que requieren servicio de forma que el coste total sea mínimo.

Podemos decir que el trabajo de Bodin, Golden, Assad y Ball (1983) [1] fue uno de los primeros en proporcionar una panorámica general sobre los problemas de rutas por arcos. Hoy en día, la referencia más actualizada es el libro editado por Corberán y Laporte (2014) [4]

Capítulo 3

Preliminares matemáticos

En este capítulo resumimos algunos de los conceptos básicos de teoría de grafos, álgebra lineal y programación lineal.

3.1. Teoría de grafos

Los grafos suelen utilizarse para modelizar muchos problemas del mundo real, entre ellos los problemas de rutas de vehículos.
Gracias a los grafos, estos problemas se pueden representar de manera clara y precisa, aunque a priori el problema pueda parecer complicado y ambiguo.
En esta sección introduciremos algunos conceptos y resultados de la teoría de grafos que hemos empleado en el presente trabajo.
Algunas referencias clásicas en esta área son Harary (1969) [7] y Bondy y Murty (1976) [2].

Definición 3.1.1 *(Grafo)*

Un **grafo G** se define como un par (V,E), donde V es un conjunto cuyos elementos son denominados vértices, o nodos y E, un conjunto de pares de vértices que reciben el nombre de arista si el par no es ordenado y arco si el par es ordenado.

Existen tres tipos de grafos según sus aristas y árcos.

- **No dirigidos:** Definimos el conjunto E, como el conjunto de pares no ordenados, es decir, formado únicamente por aristas. Denotaremos G=(V,E).
- **Dirigidos:** Definimos el conjunto E, como el conjunto de pares ordenados, es decir, está formado únicamente por arcos. Normalmente en este caso se suele denotar a E por A. Denotamos así, G=(V,A).
- **Mixto:** En este caso el conjunto E está formado por pares ordenados y por pares no ordenados. En este caso se denota por E al conjunto de los pares no ordenados (aristas) y por A al conjunto de pares ordenados (arcos). Denotaremos por G=(V,E,A).

Representamos por $n = |V|$ al número de vértices, por $m_E = |E|$ al número de aristas y por $m_A = |A|$ al número de arcos. A los vértices los denotaremos por $v_0, v_1, ..., v_n$ y por (s, t) si nuestra intención es resaltar el vértice inicial o final respectivamente. En general, denotaremos a las aristas como $e_{ij} = \{v_i, v_j\}$ y a los arcos como $a_{ij} = (v_i, v_j)$.

Definición 3.1.2 ***(Subgrafo)***

Diremos que $G' = (V', E', A')$ es un **subgrafo** de G si $V' \subset V$, $E' \subset E$, $A' \subset A$.

En muchos problemas de grafos es interesante definir una función que asigne un coste no negativo a cada arista o arco del grafo, es decir, dado un grafo $G = (V, E, A)$ definimos:

$$C : E \cup A \longrightarrow \mathbb{R}$$

$$(i, j) \longrightarrow c_{i,j} \geq 0$$

Definición 3.1.3 ***(Grafo Completo)***

Decimos que un grafo no dirigido G con n vértices es un ***grafo completo*** si cada par de vértices está unido por una arista. A este tipo de grafos se le denota por K_n.

Un grafo dirigido es completo si desde cada vértice sale un arco a todos los demás vértices.

Definición 3.1.4 ***(Definición de camino)***

Un **camino** es una secuencia $w = (v_i, e_{ij}, v_j, ..., v_s, e_{sr}, v_r)$ de vértices y aristas (o arcos) alternativamente.

Si trabajamos con grafos completos, el camino puede ser representado exclusivamente por la secuencia de vértices $(v_i, ..., v_j)$.

Un camino en la que todos lo vértices son distintos recibe el nombre de **camino simple**. Definimos el coste de un camino como la suma de los costes asociados a las aristas (o arcos) que lo forman.

Dado un grafo G no dirigido, diremos que dos vértices están conectados si y solo si existe un camino entre ellos. En el caso de un grafo dirigido hemos de exigir que, además de un camino de v_i a v_j, exista otro de v_j a v_i.

Definición 3.1.5 ***(conexión de un grafo)***

Dado un grafo G no dirigido, decimos que es **conexo** si dado cualquier par de vértices del grafo existe al menos un camino entre ellos. Un grafo mixto, o dirigido, es **fuertemente conexo** si para cualquier par de vértices v_i y v_j existe un camino de v_i a v_j y otro de v_j a v_i. Un subgrafo fuertemente conexo es **maximal** si contiene todos los vértices del grafo o si al agregarle un vértice cualquiera deja de ser fuertemente conexo. Diremos que es **débilmente conexo** si el grafo no dirigido subyacente es conexo.

Definición 3.1.6 ***(componente conexa)***

Dado un grafo no dirigido, llamamos **componente conexa** de G a cada subgrafo conexo maximal de G. Las componentes conexas de G forman una partición de los vértices de V.

Definición 3.1.7 ***(tour)***

Dado un grafo G, llamaremos **tour** (o closed walk) a un camino que acabe y empieze en el mismo vértice.

Definición 3.1.8 ***(grafo Hamiltoniano)***

Decimos que un grafo G es **Hamiltoniano** si existe un tour que pasa por todos los vértices de G una única vez (tour Hamiltoniano).

3.2. Programación lineal y entera

El principal objetivo de un problema de optimización combinatoria (y, en particular, de una problema de rutas) es encontrar su solución óptima. Para ello, tenemos que formular el problema y ver si existe una solución factible.

3.2.1 Programación lineal

Llamaremos problema de **Programación Lineal**, abreviadamente PL, al problema de minimizar (maximizar) una función lineal $f(x) = c^T x,\ c, x \in \mathbb{R}^n$, que llamaremos función objetivo, sobre un poliedro $P \subset \mathbb{R}^n$, que vendrá definido por las restricciones del problema.

Tenemos así la siguiente formulación:

$$\begin{aligned} &\text{Min } f(x) \\ s.t.\ \ &A_1x = b_1, \\ &A_2x \leq b_2, \\ &A_3x \geq b_3, \\ &x_i \in \mathbb{R},\ \forall i \in \{1, ..., n\}. \end{aligned}$$

Donde $A_i \in \mathbb{R}^{m \times n}\ \forall i = 1, 2, 3$ y $b_i \in \mathbb{R}^m\ \forall i$, siendo m el número de restricciones y $f(x)$ una función lineal.

Llamaremos solución posible a cualquier $x \in P$ y llamaremos **solución óptima** a cualquier $x^* \in P$ tal que:

$$c^T x^* = min\{c^T x : x \in P\}\ \ (c^T x^* = max\{c^T x : x \in P\},\ \text{En caso de maximizar la función objetivo})$$

Teorema 3.2.1

Si P tiene al menos un vértice y $min\{c^T x :\ x \in P\}$ es finito, entonces existe al menos un vértice en P que es solución óptima.

Teorema 3.2.2

Para cada vértice x^* de P, existe un vector $c \in \mathbb{R}^n$ tal que $c^T x^* < c^T x$ para cada $x \in P \setminus \{x^*\}$

Colorario 3.2.1

Un poliedro P es entero si y solo si para cada $c \in \mathbb{R}^n$ tal que $min\{c^T x : x \in P\}$ es finito, existe un vector entero $x^* \in P$ tal que:

$$c^T x^* = min\{c^T x : x \in P \cap \mathbb{Z}^n\}$$

Dantzig desarrolló en 1945, un algoritmo para la resolución de problemas de Programación Lineal, llamadao método simplex (Dantzig, 1963) [5].

Este algoritmo no es polinómico y, para demostrar este resultado, se encontró un conjunto de instancias de PL para las que el simplex necesita hacer un número de operaciones que depende exponencialmente del tamaño de la instancia.

Por otro lado, Khachiyan (1979) [8] propuso un algoritmo polinómico para la resolución de los problemas de PL. La aportación más importante de este algoritmo es la constatación de que PL es un problema perteneciente a la clase P de complejidad. Karmarkar propuso el método de las elipsoides que, además de ser polinómico, ha demostrado ser eficiente en la resolución de instancias grandes de PL.

3.2.2 Programación Lineal Entera

Llamaremos **problema de Programación Lineal Entera** o, abreviadamente PLE, al problema de minimizar (maximizar) una función lineal sobre los vectores enteros de un poliedro $P \subset \mathbb{R}^n$:

$$\begin{aligned} &\text{Min } f(x) \\ s.t.\ \ &A_1 x = b_1, \\ &A_2 x \leq b_2, \\ &A_3 x \geq b_3, \\ &x_i \in \mathbb{Z},\ \forall i \in \{1, ..., n\} \end{aligned}$$

Donde $A_i \in \mathbb{R}^{m \times n}\ \forall i = 1, 2, 3$ y $b_i \in \mathbb{R}^m\ \forall i = 1, 2, 3$, siendo m el número de restricciones y $f(x)$ una función lineal. La diferencia con el anterior modelo es que la solución ha de ser entera, $x \in \mathbb{Z}^n$.

Definición 3.2.1 ***(Relajación lineal)***

Llamamos **relajación lineal** de un PLE al PL resultante de eliminar las restricciones de integridad en el PLE.

Definición 3.2.2 ***(branch and bound)***

Para resolver los problemas de ***PLE*** o problemas ***Mixtos***, el método más utilizado es el conocido como método de ***branch and bound***. Se basa en la enumeración implícita de todas las soluciones posibles enteras mediante la resolución de varios problemas relajados, a los cuales llamaremos nodos, en los que se ha particionado el problema original. El método trata de saturar dichos nodos (resolverlos) por haber encontrado la solución óptima del PLE definido en ese nodo, porque ese PLE es imposible o porque la solución que podríamos encontrar en ese nodo es (sería) peor que la solución óptima entera. El método sigue el siguiente esquema:

1. Calculamos la solución óptima del problema relajado.
2. Elegimos una de las variables con valor no entero en la solución del problema relajado y calculamos su parte entera por exceso y por defecto. Construimos dos nuevos problemas en los que, en uno de ellos, añadimos la restricción de que dicha variable valga menos o igual que el valor por defecto y, en el otro, que sea mayor o igual que su valor por exceso.
3. Seguimos haciendo 2) hasta que saturemos todos los nodos utilizando los criterios descritos anteriormente.

Observaciones generales:

a) En algunos casos es mejor hacer un algoritmo heurístico para determinar una solución factible y buena para la saturación de nodos. Esta saturación aplicará a los nodos cuya solución sea peor que la obtenida en el algoritmo heurístico.

b) Si los costes de las variables en la función objetivo son enteros, entonces podemos optimizar el algoritmo de la siguiente manera: Si tenemos una solución entera con coste z^* entero y en otro nodo tenemos una solución del problema relajado de $z^* + a$, siendo $a \in]0, 1[$ entonces ese nodo lo podemos saturar puesto que las soluciones enteras han de dar un coste entero en la función objetivo y por tanto este nodo, como mucho, podríamos hallar una solución entera de coste z^*.

Pongamos un ejemplo del algoritmo del ***branch and bound***. Tenemos el siguiente problema:

$$\text{Min } -4x_1 + x_2$$

sujeto a:

$$7x_1 - 2x_2 \leq 14 \tag{3.1}$$

$$x_2 \leq 3 \tag{3.2}$$

$$2x_1 - 2x_2 \leq 3 \tag{3.3}$$

$$x_1, x_2 \in \mathbb{Z}_+^2$$

Resolvemos el problema relajado. La solución es $x_1 = \frac{20}{7},\ x_2 = 3,\ x_3 = 0$, la función objetivo vale $-\frac{59}{7}$.
Cogemos una de las variables que deba ser entera. Hay muchas posibilidades a la hora de elegir la variable. Nuestro criterio ha sido coger la variable con valor fraccionario más cerca de un número entero. En este caso cogemos x_1. En el nodo 1 añadimos la restricción $x_1 \leq 2$ y al nodo 2 la restricción $x_1 \geq 3$. Resolvemos el problema relajado del nodo 1 y 2 quedando así lo siguiente:

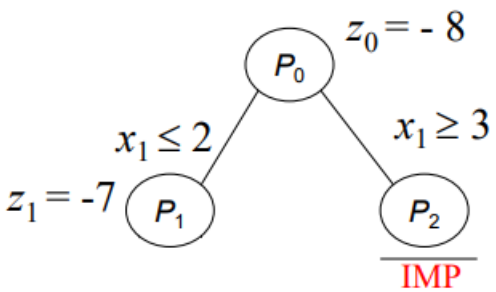

Figura 3.1: Nodos 1 y 2

La solución óptima de P_1 es $x_1 = 2,\ x_2 = \frac{1}{2}$, con un coste de $\frac{15}{2}$. En el nodo 2, la restricción 3.1 y $x_1 \geq 3$ hacen que P_2 sea imposible. Por tanto saturamos el nodo.
Seguimos así con el nodo 1. Tenemos x_1 entera en la solución de la relajación lineal, entonces hemos de restringir $x_2 \leq 0$ y $x_2 \geq 1$. Llamaremos 3 al nodo al que le añadimos la restricción $x_2 \leq 0$ y 4 al que le añadimos la restricción $x_2 \geq 1$. Resolvamos primero el problema relajado de P_3. La solución obtenida es: $x_1 = \frac{3}{2},\ x_2 = 0$ con un coste de -6. Resolvemos P_4. En este caso hallamos una solución entera, donde $x_1 = 2,\ x_2 = 1$ y con un coste de -7. Como estamos minimizando y en P_3 tenemos $-6 > -7$, entonces procedemos a saturar el nodo puesto que en sus hijos vamos a tener soluciones mayores a -6 y por tanto peores. Nos queda pues el siguiente árbol:

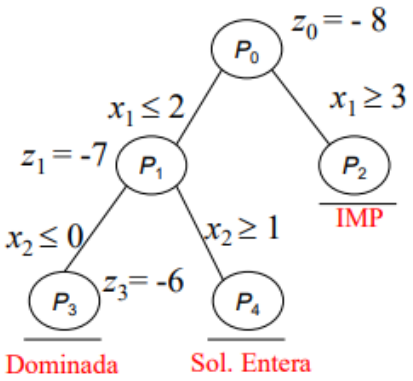

Figura 3.2: Árbol del problema

Quedando así la solución óptima $x_1 = 2,\ x_2 = 1$ y el coste de la función objetivo $z_{PLE} = -7$.

Definición 3.2.3 ***(branch and cut)***

Un algoritmo de branch and cut es, básicamente, un procedimiento que une la estructura de árbol de enumeración y el uso de las cotas propio de los algoritmos de branch and bound con el uso de desigualdades válidas (cortes) que permitan reforzar los problemas lineales que proporcionarán las cotas a usar en los nodos del árbol. Esas desigualdades válidas pueden ser "generalistas", como los cortes de Gomory, o específicas para el problema en estudio.

3.3. Algoritmos heurísticos y metaheurísticos

En esta sección discutiremos el problema de encontrar buenas soluciones posibles para un problema de optimización.
Recurriremos a estos métodos cuando el coste computacional para hallar la solución óptima sea excesivo para nuestra disponibilidad o sea prácticamente imposible encontrar la solución óptima en un tiempo razonable. También los usaremos para obtener una cota inferior (superior) que nos permita acortar el árbol de ramificación de un algoritmo exacto del tipo ***branch and bound*** o ***branch and cut***.

3.3.1 Algoritmos heurísticos

Un método heurístico es un algoritmo que proporciona una buena solución del problema, no necesariamente óptima, en poco tiempo. Un algoritmo heurístico es definido como un proceso inteligente, basado en la intuición, el contexto y la estructura del problema.
Lo que se busca de un algoritmo heurístico es que sea capaz de encontrar soluciones posibles, cercanas a la óptima para cualquier instancia del problema en un tiempo de computación razonable.

"Un proceso heurístico es una colección de reglas y pasos que guian a una solución que quizas, o quizás no, sea la mejor solución" Laguna M., Martí R. [9]

Básicamente, estos algoritmos pueden clasificarse en:

- **Métodos constructivos:** Son procedimientos iterativos que, en cada paso añaden un elemento hasta completar una solución. Usualmente son métodos deterministas y están basados en seleccionar, en cada iteración, el elemento con mejor evaluación.
- **Métodos de Búsqueda Local:** Basados en explorar el entorno de una solución y seleccionar una nueva solución en él (i.e., realizar el movimiento asociado). Desde la nueva solución se explora su entorno y se repite el proceso.
- **Métodos Combinados:** Un método combinado es el que junta un algoritmo constructivo con la búsqueda local.

3.3.2 Algoritmos metaheurísticos

Los algoritmos metaheurísticos son estrategias que guían y modifican un heurístico. Normalmente estos empiezan cuando el heurístico llega al óptimo local.

Entre los metaheurísticos más utilizados podemos mencionar la búsqueda tabú, los algoritmos genéticos, GRASP, la búsqueda dispersa, el simulated annealing, ...

Capítulo 4

El problema del viajante

El ***Problema del Viajante*** o ***Traveling Salesman Problem (TSP)*** es uno de los problemas más estudiados de optimización combinatoria. En el problema tenemos un grafo donde las aristas tienen un coste asociado. El objetivo es visitar todos los vértices exactamente una vez, minimizando el coste del tour.
El problema, aparentemente fácil, pertenece a la clase de complejidad NP-difícil.

4.1. Formulación

Consideramos el problema del viajante definido sobre un conjunto de vértices (ciudades) N, compuesto por $n+1$ vértices numerados de 0 a n. Las distancias entre ellos vienen dadas por $c_{ij} \geq 0,\ \forall i, j \in N$.
Si definimos las variables x_{ij} como: $x_{ij} = 1$ si el viajante va de i a j y 0 en otro caso, podemos formular el TSP de la siguiente forma:

$$\text{Min} \sum_{i,j=0}^{n} c_{ij} x_{ij} \tag{4.1}$$

s.t.:

$$\sum_{i=0}^{n} x_{ij} = 1,\ \forall j \in \{0, ..., n\}, \tag{4.2}$$

$$\sum_{j=0}^{n} x_{ij} = 1,\ \forall i \in \{0, ..., n\}, \tag{4.3}$$

$$\sum_{i,j \in S}^{n} x_{ij} \leq |S| - 1,\ \forall S \subset N, \tag{4.4}$$

$$x_{ij} \in \{0, 1\},\ \forall i, j \in N. \tag{4.5}$$

Las restricciones (4.2) aseguran que en cada vértice entra un único arco, mientras que las restricciones (4.3) aseguran que de cada vértice salga un único arco.

Las restricciones (4.4) establecen que no puede haber un subgrafo con un número mayor de aristas en la solución que el número de nodos menos uno. Esto permite que no se formen subtours y por tanto podamos empezar y acabar en el mismo vértice.

4.2. Un algoritmo heurístico para el TSP

El TSP es uno de los problemas más estudiados de la Investigación Operativa, por tanto se han propuesto muchos algoritmos para su resolución. Nosotros hemos utilizado una pequeña modificación del conocido algoritmo de los ahorros (savings) propuesto para la resolución del problema de rutas de vehículos con capacidades por Clarke and Wright (1964) [16].

4.2.1 Algoritmo de los ahorros (savings)

Definición 4.2.1 ***(entorno de cero en la ruta)***

Diremos que un elemento i es **entorno de cero en la ruta**, si tenemos la ruta (0,i,...) o (...,i,0), es decir, i es el primer vértice visitado o el último, respectivamente.

El algoritmo que se ha utilizado es el siguiente:

1. Se calculan los ahorros, $s_{ij} = c_{i0} + c_{0j} - c_{ij}$, $\forall i, j = 1, ..., n$ con $i \neq j$. Se crean n rutas $(0, i, 0)$ que parten del depósito, van a un cliente i y vuelven al depósito, para $i = 1, ..., n$. Por último se ordenan los ahorros de mayor a menor.
2. Cogemos la arista asociada al ahorro que esté en la primera posición y hacemos lo siguiente: Dada la arista asociada con el saving que está en la primera posición, determinamos si puede existir la ruta, es decir, tenemos que ver que j e i sean entornos de ceros en la ruta [1]teniendo así algunas de las dos siguientes rutas:

   $a)$ $(0 - i - a)$ y $(0 - j - b)$
   $b)$ $(0 - i - a)$ y $(b - j - 0)$

   donde a y b son conjuntos de vértices. Dependiéndo de qué ruta tengamos, se pueden unir de la siguiente forma $(0, a, i, j, b, 0)$ si tenemos la ruta $2a$, o bien $(0, b, j, i, a, 0)$ si tenemos la ruta $2b$.
3. Se elimina el saving utilizado.
4. Se repite 2 y 3 hasta tener una única ruta.

Se ha modificado el algoritmo añadiéndole una búsqueda local para conseguir mejorar la solución construida y, además, se ha introducido aleatoriedad en el algoritmo constructivo, lo que permite generar distintas soluciones posibles del problema.

[1]En el TSP la única restricción es que no puede pasar dos veces por el mísmo vértice

4.2.2 Un procedimiento de búsqueda local

Para hallar un óptimo local en una instancia del TSP hemos realizado el reordenamiento de los vértices visitados en una solución del TSP. Este procedimiento consiste en tomar dos vértices e intercambiarlos en el tour.
Antes de efectuar el cambio entre dos vértices, hemos de explorar todo el entorno y quedarnos con el intercambio que más aumente la función objetivo. Este procedimiento se le denomina, procedimiento "best", pues se prueba con todo el entorno antes de realizar el intercambio. Una vez teminado con todo par de vértices, si hemos conseguido mejorar nuestra función objetivo, volvemos a realizar el procedimiento de búsqueda local. Esto lo haremos hasta que no se produzca mejora en la función objetivo, es decir, que ningún intercambio se efectúe.

Notar que en nuestro algoritmo [15] no calculamos cada vez el valor de la función objetivo, pues basta con probar el intercambio y ver cuánto disminuye (o aumenta) la función objetivo con este intercambio. Lo hacemos de la siguiente manera:

Tenemos la ruta $(0 - ... - a - i - b - ... - a' - j - b' - ... - 0)$, donde a, b, a', b' son los vértices adyacentes a i y j respectivamente. Queremos intercambiar i con j, por tanto, si efectuamos el cambio quedaría la ruta siguiente: $(0 - ... - a - j - b - ... - a' - i - b' - ... - 0)$. Para evaluar si esta nueva ruta mejora a la anterior tenemos que hacer el siguiente cálculo: $C_n = c_{aj} + c_{jb} + c_{a'i} + c_{ib'}$ y $C_v = c_{ai} + c_{ib} + c_{a'j} + c_{jb'}$, siendo C_n el coste cuando hemos intercambiado las rutas y C_v el coste antigüo de la ruta. Si $C_v \leq C_n$ entonces la función objetivo a la hora de intercambiar estos dos vértices, empeora o no cambia, y por tanto no intercambiamos los vértices. En cambio, si $C_n < C_v$ entonces nos guardamos el valor de C_n puesto que antes de efectuar ningún cambio tenemos que mirar los demás entornos de i.

Esto nos devuelve una solución óptima local. Gracias al proceso que describimos a continuación podemos hallar varios óptimos locales.

4.2.3 Aleatoriedad en la elección del s_{ij}

Para poder usar el algoritmo descrito de manera que genere distintas soluciones, introduciremos la aleatorización en la elección del ahorro a utilizar. Para ello, teniendo la lista ordenada de los savings, extraemos un número aleatorio entre 1 y α, siendo α un parámetro que determinaremos en el capítulo 7, y, en lugar de utilizar siempre el saving de la primera posición, utilizaremos el saving del número aleatorio.

Como hemos comentado, esto nos ayudará a explorar distintas soluciones factibles de nuestro problema. Sin la aleatorización, tendríamos siempre la misma solución. Esto nos permite ejecutar nuestro algoritmo el tiempo de computación que nosotros deseemos. Si tenemos 10 segundos para obtener una solución de nuestro problema, podemos emplear esos 10 segundos en buscar nuevas soluciones factibles y quedarnos con la mejor. Si no aleatorizamos, el programa tardaría menos de 1 segundo en hallar una solución pero no obtendríamos la diversidad de soluciones que posiblemente nos proporcionará una mejor solución final.

Hallar el valor de α es un aspecto importante puesto que de su valor puede depender la calidad de las soluciones que obtendremos. Normalmente se tendrá un tiempo de computación limitado. A la hora de resolver la instancia no podemos estudiar el α que funciona mejor, puesto que estaremos perdiendo posibles mejoras de nuestra función objetivo.

Capítulo 5

El problema de rutas de vehículos con capacidades

El problema de rutas de vehículos con capacidades (capacitated vehicle routing problem, CVRP) es un problema difícil y muy importante por el gran número de aplicaciones reales que tiene. Su aplicación no se limita a los problemas que conciernen al envio de bienes, también se puede aplicar a problemas como el de limpieza de calles, rutas de los autobuses escolares, etc. [2]

5.1. Definición del problema

En este problema se considera que hay una cantidad de bienes que se tienen que repartir a ciertos clientes (vértices de un grafo) con una flota de vehículos, los cuales tienen una capacidad máxima. Esta flota parte de un depósito (en otras versiones puede haber más de un depósito) y, al terminar, los vehículos han de volver al mismo. Hay distintos objetivos, por lo que la función a optimizar varía según lo que se pretenda hacer. Algunos de los objetivos pueden ser:

- Minimizar la distancia total recorrida por los vehículos.
- Minimizar la longitud de la ruta más larga.
- Minimizar el número de vehículos empleados.

Las restricciones del problema son las siguientes:

- Demanda: La suma de demandas de los vértices de una ruta no ha de exceder la capacidad del camión que hace dicha ruta.
- Cliente: Cada vértice ha de ser visitado exáctamente por un vehículo.
- Depósito: En cada ruta el vehículo ha de partir y volver al depósito.
- Capacidad: Todos los camiones tendrán la misma capacidad.

[2] Notemos que el problema de rutas de vehículos es una generalización del TSP

Se tienen n vértices (clientes) que definen el conjunto V. Cada cliente tiene una demanda, d_i, no negativa y el depósito tendrá una demanda ficticia $d_0 = 0$. Dado un subconjunto de vértices $S \subseteq V$, $d(S) = \sum_{i \in S} d_i$ denotará la demanda total del subconjunto S. Existe un coste no negativo, c_{ij}. Este coste hace referencia al coste que tiene ir del vértice i al vértice j, $\forall i, j, i < j$.

Sea K el conjunto de los vehículos, con capaciad C, que hay disponibles en el depósito. Para asegurar que el problema es posible, asumimos que $d_i \leq C$ para cada $i = 1, ..., n$. Cada vehículo hará como mucho una ruta. Asumimos que K no puede ser más pequeño que K_{min}, donde K_{min} es el mínimo número de vehículos que necesitamos para servir a todos los clientes.

Aunque el problema de determinar exactamente K_{min} es un problema NP-difícil, una cota superior puede determinarse fácilmente como $\lceil d(V)/C \rceil$. Dado un subconjunto $S \subseteq V \setminus \{0\}$, se denotará por $r(S)$ el número mínimo de vehículos que se necesitarán para servir a todos los clientes de S que, al igual que antes, puede ser estimado por $\lceil d(S)/C \rceil$.

Si tuvieramos un único camión con capacidad suficiente para servir a todos los clientes, es decir, si $\sum_{i=1}^{n} d_i \leq C$, entonces tendríamos un problema del viajante (4), ya que con un solo camión tendríamos suficiente para hacer el tour que pase por todas las ciudades. Esto nos servirá para explicar el algoritmo de Ulusoy (6.2).

En las formulaciones de los problemas usaremos directamente K refiriéndonos a K_{min}.

5.2. Formulación del problema

Existen distintas formulaciones del problema, y es importante resaltar que algunas formulaciones hacen que el problema sea menos costoso a la hora de resolverlo computacionalmente. Una de estas formulaciones es la siguiente:

$$\text{Min} \quad \sum_{i \in V \setminus \{n\}} \sum_{j > i} c_{ij} x_{ij} \tag{5.1}$$

Sujeto a

$$\sum_{h < i} x_{hi} + \sum_{j > i} x_{ij} = 2, \quad \forall i \in V \setminus \{0\}, \tag{5.2}$$

$$\sum_{j \in V \setminus \{0\}} x_{0j} = 2K, \tag{5.3}$$

$$\sum_{i \in S} \sum_{\substack{h < i \\ h \notin S}} x_{hi} + \sum_{i \in S} \sum_{\substack{j > i \\ j \notin S}} x_{ij} \geq 2r(S), \ \forall S \subseteq V \setminus \{0\}, \ S \neq \emptyset, \tag{5.4}$$

$$x_{ij} \in \{0, 1\}, \ \forall i, j \in V \setminus \{0\}, i < j, \tag{5.5}$$

$$x_{0j} \in \{0, 1, 2\}, \ \forall j \in V \setminus \{0\}. \tag{5.6}$$

Las variables x_{ij} dan la dirección que ha de tomar el vehículo para efectuar la ruta óptima[3]. Esta variable es binaria, valiendo 0 cuando no usamos esa arista y 1 cuando lo hacemos (5.5). Al tener un grafo no dirigido,

[3]En nuestro caso si tenemos una ruta $(0, a_1, ..., a_n, 0)$, la podemos hacer en el sentido que nos dicta el conjunto o podemos recorrerla en el sentido contrario $(0, a_n, ..., a_1, 0)$ y sigue teniendo el mismo coste

tenemos que $x_{ij} = x_{ji}$, por tanto reducimos x a una matriz triangular superior ($i < j$).

La restriccion (5.2) impone que cada vértice tenga *exactamente* una arista que entra y otra que sale, haciendo así que cada vértice solo pueda ser visitado por un vehículo.

La restricción (5.3) obliga a que todo vehículo que sale del depósito vuelva a este. La variable x_{0j} será igual a 2 (5.6) cuando exista la ruta (0,j,0), en cualquier otro caso el valor de la variable será binario. En nuestro problema aceptamos que puedan haber rutas de un solo cliente, en caso de no aceptarlo no tendríamos esta restricción.

La restricción (5.4) garantiza la conectividad de las rutas, ya que hay que entrar en cualquier subconjunto de vértices ($\sum_{i\in S}\sum_{\substack{h<i\\ h\notin S}} x_{hi}$) y salir de él ($\sum_{i\in S}\sum_{\substack{j>i\\ j\notin S}} x_{ij}$), ya que estas dos sumas han de ser ≥ 2. El motivo por el que multiplicamos por $r(S)$, es enviar el número de vehículos necesarios para abastecer la demanda que hay en el subconjunto.

El modelo simétrico normalmente se suele formular usando variables con un único índice, $e \in E$. Definimos $\delta(i)$ al conjunto de las aristas incidentes con el vértice i.
Si las rutas de un solo cliente no están permitidas, todas las variables serán binarias. Si permitimos rutas de un solo cliente tendremos que si $e \notin \delta(0)$ entonces $x_e \in \{0,1\}$ y si $x_e \in \delta(0)$ entonces $x_e \in \{0,1,2\}$.
Quedando así la siguiente formulación:

$$\text{Min} \quad \sum_{e\in E} c_e x_e \tag{5.7}$$

Sujeto a

$$\sum_{e\in\delta(i)} x_e = 2, \quad \forall i \in V \setminus \{0\}, \tag{5.8}$$

$$\sum_{e\in\delta(0)} x_e = 2K, \tag{5.9}$$

$$\sum_{e\in\delta(S)} x_e \geq 2r(S),\ \forall S \subseteq V \setminus \{0\},\ S \neq \emptyset, \tag{5.10}$$

$$x_e \in \{0,1\},\ \forall e \notin \delta(0), \tag{5.11}$$

$$x_e \in \{0,1,2\},\ \forall e \in \delta(0). \tag{5.12}$$

Nótese que en ambas formulaciones hemos considerado el objetivo más común de minimizar el coste o distancia total. La resolución de cualquiera de las dos formulaciones requiere un tiempo computacional muy alto. Nótese que las restricciones (5.4) y (5.10) están definidas para cualquier subconjunto S de vértices, lo que implica que tenemos del orden de 2^{n-1} restricciones de ese tipo.

Puesto que la resolución exacta del problema requiere un tiempo de computación muy alto, vamos a centrarnos en el uso de algoritmos heurísticos que nos permitan tener una solución factible, si es posible cercana a la óptima, en un tiempo razonable.

Capítulo 6

Dos algoritmos heurísticos para la resolución del CVRP

Para la resolución del problema se han programado dos algoritmos en Java que se pueden consultar en el siguiente enlance [15]. Uno de ellos es el algoritmo de los ahorros (savings), descrito para el TSP en la sección 4.2.1, y el algoritmo heurístico extendido de Ulusoy [13].

6.1. Algoritmo de los ahorros para el CVRP

La diferencia con el algoritmo de los ahorros para el TSP es que aquí debemos considerar la restricción de capacidad de los camiones. Debido a esta restricción, no tiene porqué haber una única ruta.

El algoritmo es el siguiente:

1. Se calculan los savings, $s_{ij} = c_{i0} + c_{0j} - c_{ij}$, $\forall i, j = 1, ..., n$ con $i \neq j$, y se ordenan de mayor a menor. Se crean n rutas $(0, i, 0)$ para $i = 1, ..., n$.
2. Cogemos la arista asociada al ahorro que esté en la primera posición y hacemos lo siguiente: Dada la arista asociada con el saving que está en la primera posición, determinamos si puede existir una ruta que resulte de unir el vértice i con el vértice j. La existencia de esta nueva ruta depende de si los vértices i y j son entornos de cero en una ruta y si, al unir estas dos rutas, la demanda de esta nueva ruta es menor que la capacidad del camión, es decir, $\sum_{i \in \Omega} d_i \leq C$, siendo Ω el conjunto de vértices de la unión entre las dos rutas.
3. Se elimina el saving utilizado.
4. Repetir los pasos 2 y 3 hasta que no se tengan más savings o hasta que las capacidades de los camiones estén completas.

6.1.1 Ejemplo del algoritmo

A modo de ilustración, en esta sección resolveremos, con el algoritmo de los ahorros, una instancia con 9 clientes y camiones de capacidad de 40.

La matriz de costes (simétrica) asociada al problema es:

c_{ij}	0	1	2	3	4	5	6	7	8	9
0	0	12	11	7	10	10	9	8	6	12
1	12	0	8	5	9	12	14	16	17	22
2	11	8	0	9	15	17	8	18	14	22
3	7	5	9	0	7	9	11	12	12	17
4	10	9	15	7	0	3	17	7	15	18
5	10	12	17	9	3	0	18	6	15	15
6	9	14	8	11	17	18	0	16	8	16
7	8	16	18	12	7	6	16	0	11	11
8	6	17	14	12	15	15	8	11	0	10
9	12	22	22	17	18	15	16	11	10	0

La demanda de cada cliente se visualiza en la siguiente tabla:

i	1	2	3	4	5	6	7	8	9
d_i	10	15	18	17	3	5	9	4	6

Calculamos los savings, obteniendo la siguiente matriz triangular superior:

s_{ij}	1	2	3	4	5	6	7	8	9
1		15	14	13	10	7	4	1	2
2			9	6	4	12	1	3	1
3				10	8	5	3	1	2
4					17	2	11	1	4
5						1	12	1	7
6							1	7	5
7								3	9
8									8

La unión de dos vértices cualesquiera depende de que ambos sean entornos de cero en la ruta. Esto corresponde, por el propio cálculo del savings, al ahorro que tenemos al juntar ambos vértices. Así pues, el saving s_{12} correspondería al ahorro que obtendríamos al unir, en caso de ser posible, lo vértices 1 y 2.

Hay tres posibilidades que nos imposibilitan unir dos rutas.

1. Que ambos vértices ya estén en la misma ruta.
2. Que ambos vértices no sean entornos de cero en la ruta.
3. Si la suma de las demandas de la nueva ruta es mayor que la capacidad del camión.

Se procede a ordenar los savings de mayor a menor. Las aristas asociadas a estos savings son las siguientes: (4,5), (1,2), (1,3), (1,4), (2,6), (5,7), (4,7), (1,5), (3,4), (2,3), (7,9), (3,5), (8,9), (1,6),(5,9), (6,8), (2,4),...

Partiendo de la solución inicial: (0-1-0), (0-2-0), ... , (0-9-0), al considerar el primer saving, nos planteamos usar la arista (4,5) para unir las rutas (0-4-0) y (0-5-0). Así obtenemos la ruta (0-4-5-0), que es posible porque la demanda cargada es: $d_4 + d_5 = 20 < C = 40$.

Se sigue con el siguiente saving, planteándonos así usar la arista (1,2). Las rutas (0-1-0) y (0-2-0), estas dos rutas forman la nueva ruta (0-1-2-0), cuya demanda carga es: $d_1 + d_2 = 25 < C = 40$.

El siguiente saving a utilizar corresponde a la arista (1,3) y tenemos las rutas (0-1-2-0) y (0-3-0). Tanto el 1 como el 3 son entornos del 0 y no están en la misma ruta, pero al unir el vértice 3 a la ruta nos queda la siguiente ruta: (0-3-1-2-0), con una carga de $d_1+d_2+d_3 = 43 > C = 40$. Al tener exceso de carga, no podemos formar esta ruta.

Pasa igual con la arista asociada al siguiente saving, (1,4). Intentamos unir las rutas (0-1-2-0) con la ruta (0-4-5-0) mediante la arista (1,4). Tanto el 1 como el 4 son entornos del 0 en la ruta y no están en la misma ruta, pero la capacidad que la ruta, (0-5-4-1-2-0), sería de $d_1 + d_2 + d_4 + d_5 = 45 > C = 40$.

La siguiente arista a tener en cuenta es (2,6), tomamos las rutas (0-1-2-0) y (0-6-0) e intentamos unirlas mediante la arista (2,6). Ambos vértices cumplen el estar en rutas distintas y el de ser entornos de cero en la ruta. Además, la carga que nos deja la ruta, (0-1-2-6-0), es de $d_1+d_2+d_6 = 30 < C = 40$, formando así una nueva ruta.

Cogemos la arista (5,7). La ruta correspondiente al 5 es (0-4-5-0) y la ruta correspondiente al 7 es (0-7-0). Ambos vértices son entornos de cero en la ruta y no pertenecen a la misma ruta. Al unir estas dos rutas mediante la arista (5,7), nos queda la nueva ruta (0-4-5-7-0), con la demanda cargada: $d_4 + d_5 + d_7 = 29 < C = 40$.

El siguiente saving viene asociado con la arista (4,7) y tenemos la ruta (0-4-5-7-0), donde ambos vértices pertenecen. Por tanto es imposible unirlos.

La siguiente arista ha utilizar es (1,5) y tenemos las rutas (0-1-2-6-0) y (0-4-5-7-0). Notamos que el vértice 5 no corresponde a un entorno del cero en la ruta, por tanto no se pueden unir ambas rutas.

El siguiente saving tiene asociado la arista (3,4). Si intentamos unir las rutas (0-3-0) y (0-4-5-7-0), quedando la ruta (0-3-4-5-7-0), tendremos un exceso de capacidad, puesto que la carga de esta ruta es: $d_3 + d_4 + d_5 + d_7 = 47 > C = 40$.

Con la ruta (0-1-2-6-0) y sabiendo que la arista asociada al siguiente saving es (2,3), es imposible unir dos rutas, puesto que el vértice 2 no es un entorno de cero en la ruta.

Se siguie así sucesivamente hasta terminar todos los savings.

La solución es la siguiente:

Ruta	Carga del camión	Coste
(0-3-0)	18	14
(0-1-2-6-0)	30	37
(0-4-5-7-9-8-0)	39	46

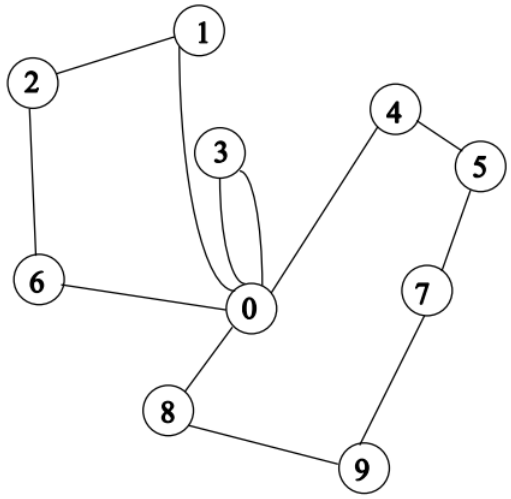

Figura 6.1: Ejemplo algoritmo de los ahorros

Se tienen 3 rutas con un coste total 97. Nos damos cuenta de que no llenamos al máximo el camión, puesto que lo máximo que llega a llenar es 39 de 40. Además, tenemos una ruta simple, (0,3,0).

Esta solución se puede mejorar mediante una búsqueda local y aleatorizando el orden de los savings. Ambos procesos se explican a continuación.

6.1.2 Un procedimiento de búsqueda local

El procedimiento utilizado para la búsqueda local ha sido coger dos rutas e intentar intercambiar un vértice de cada una de ellas. Este intercambio se puede hacer si la suma de las demandas de las nuevas rutas, no excede la capacidad del camión, es decir, $\sum_{i \in r_1} d_i < C$ y $\sum_{i \in r_2} d_i < C$, siendo r_1 y r_2, las nuevas rutas. Hacemos esto con todos los vértices. Efectuaremos el intercambio que produzca una mayor disminución en el valor de la función objetivo. Se vuelve a repetir el procedimiento de búsqueda local mientras queden vértices por estudiar.

También se ha realizado el intercambio intrarutas, es decir, dentro de una misma ruta intentar cambiar el orden de los vértices. Este intercambio no crea problema con las capacidades de los vehículos puesto que la suma de las demandas de los clientes no depende del orden en el que se recorre la ruta, $\sum_{i \in r_1} d_i = \sum_{i \in r_2} d_i$ siendo r_1 la

ruta original y r_2 una reordenación de la ruta.

Si se consigue mejorar al explorar todos los entornos de la ruta, se vuelve a realizar la búsqueda local, garantizando así, llegar al óptimo local.

6.1.3 Aleatorización de los savings

Al igual que en el TSP (4.2.3), reordenamos los savings de una manera aleatoria. Esta aleatoriedad viene marcada por el parámetro $\alpha \in \mathbb{N}$.

Una manera de aleatorizar el algoritmo de los ahorros es, una vez teniendo ordenados los savings, reordenarlos de una manera aleatoria de la siguiente manera:

1. Tomamos un número aleatorio, a, entre el 1 y α, con $\alpha > 1$.
2. En el vector de los savings cogemos el $a - simo$ elemento y lo movemos a un nuevo vector y lo situamos en la última posición de este nuevo vector.
3. Realizamos 1 y 2 hasta que nos quedemos sin savings.

Al tener el algoritmo una componente aleatoria exploramos soluciones distintas cada vez que este se ejecuta.

Al unir la búsqueda local con la aleatorización de los savings, podremos obtener distintos óptimos locales, quedándonos así con el mejor de ellos.

6.2. Algoritmo heurístico extendido de Ulusoy (Extended Ulusoy's heuristic, EUH)

La idea general del algoritmo es eliminar temporalmente la restricción de capacidad y resolver el problema como si tuviéramos un único vehículo, teniendo así el problema del viajante. Se enumera, empezando desde el depósito, el tour que proporciona la solución del TSP y se "trocea.[en] rutas que cumplan la restricción de capacidad. Obtenemos así una solución para el problema de rutas con capacidades.

Una parte fundamental del algoritmo de Ulusoy es el cálculo de un camino más corto en un grafo auxiliar. Pasamos a describir brevemente el algoritmo del camino más corto (CMC).

6.2.1 Camino más corto (CMC)

Sea $G = (V, E)$ un grafo dirigido en el que cada arco (i, j) tiene asociado un coste c_{ij}, que supondremos no negativo. Cabe la posibilidad de que no exista un arco entre i y j, dejando así $c_{ij} = \infty$. El coste de un camino entre dos vértices s y t, es la suma de los costes de los arcos que lo contiene. Llamaremos camino más corto de s a t a aquél camino con coste mínimo (o longitud mínima). Usaremos el algoritmo de **Dijkstra** (1959) [6] para hallar este camino más corto.

Representamos por $l(x_i)$ la etiqueta del vértice x_i. Los pasos a seguir para el algoritmo de **Dijkstra** son los siguientes:

1. Definimos $l(s) = 0$ y lo marcamos como etiqueta permanente. Le damos el valor $l(x_i) = \infty \ \forall x_i \neq s$ y marcamos esta etiqueta como temporal. Hacemos $p = s$.
2. Para todo vértice x_i adyacente a p y que tenga etiqueta temporal actualizamos su etiqueta de la siguiente manera:
$$l(x_i) = \min\{l(x_i), l(p) + c(p, x_i)\}$$
3. De todos los vértices con etiquetas temporales, escogemos x_i^* como el vértice con etiqueta menor, $l(x_i^*) = \min\{l(x_i)\}$.
4. Marcamos la etiqueda de la variable x_i^* como permanente y redefinimos $p = x_i^*$.
5. *a*) Si $p = t$, $l(p)$ es el coste del camino más corto. Paramos el algoritmo.
   b) Si $p \neq t$ volvemos al paso 2.

Ahora podemos describir el algoritmo de Ulusoy [13] como sigue:

1. Obviamos la restricción de capacidad y resolvemos el TSP asociado a nuestro problema con el algoritmo de los savings (4.2.1). En nuestro problemas tenemos $n + 1$ vértices, de los cuales n son clientes y el restante es el depósito.
2. Dada la ruta del TSP, la reenumeramos dando al primer vértice que visitamos partiendo del depósito el número 1, al segundo el número 2, y así sucesivamente.
3. Construimos un grafo dirigido auxiliar con n vértices y los siguientes arcos: los vértices i e $i + 1$ se unen con un arco para todo $i = 1, ..., n$. El coste del arco i-ésimo será $c_{0,i} + c_{i,0} = 2 \cdot c_{0,i}$.
4. Añadiremos un arco entre el vértice i-ésimo y el vértice $(i + j)$-ésimo si se cumple que $\sum_{k=i+1}^{i+j} d_k \leq C$. El coste de ese arco es $c_{0,(i+1)} + \sum_{k=i+1}^{(i+1+j)-2} c_{k,k+1} + c_{(i+j),0}$
5. Por último calculamos, con el algoritmo de los caminos más cortos, la ruta más corta entre los vertices 1 y n.

La solución del algoritmo nos dará una partición óptima de la solución del TSP.

6.2.2 Ejemplo del algoritmo de Ulusoy

Consideremos una instancia con 13 clientes con las demandas siguientes:

i	1	2	3	4	5	6	7	8	9	10	11	12	13
d_i	30	15	24	30	77	5	8	6	4	14	24	16	30

Supongamos que los camiones de nuestra flota tienen una capacidad de 100 unidades ($C = 100$).

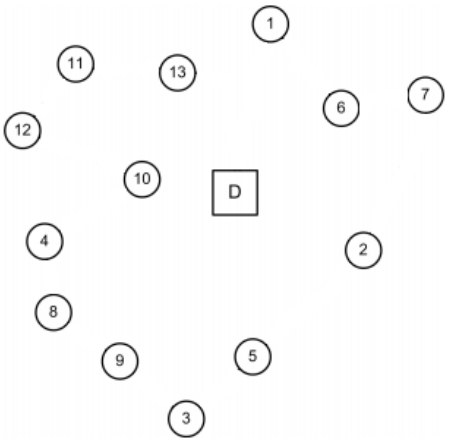

Figura 6.2: Instancia con 13 clientes

Procedemos a resolver el problema sin la restricción de capacidad, es decir, resolvemos el TSP de esta instancia y reordenamos los vértices según el orden de visita del tour formado por la solución del TSP.

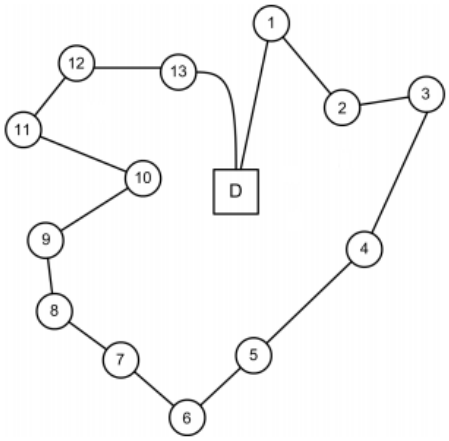

Figura 6.3: Solución del TSP (1) y reordenación de los vértices (2)

La matriz de costes (distancias) asociada a los vértices reordenados es la siguiente:

c_{ij}	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0		5	4	6	4	5	6	6	6	5	2	6	6	4
1			3	5	7	10	13	11	10	9	7	10	8	3
2				2	3	7	9	9	10	10	7	10	9	8
3					4	10	12	11	12	12	9	12	10	10
4						4	6	6	8	8	6	10	11	11
5							2	3	4	5	6	8	10	11
6								2	4	5	7	9	11	13
7									2	3	4	6	8	10
8										1	4	5	7	9
9											2	2	4	6
10												2	3	2
11													1	3
12														2

Construimos a continuación el grafo auxiliar.

Arista D-2: [4] $d_1+d_2 \leq C$, se conectan los vértices D y 2 con un arco. El coste de este arco es: $c_{01}+\sum_{k=1}^{1} c_{k(k+1)}+c_{20} = c_{01}+c_{12}+c_{02} = 5+3+4 = 12$
Arista D-3: [5] $d_1+d_2+d_3 \leq C$, se conectan los vértices D y 3 con un arco. El coste de este arco es: $c_{01}+\sum_{k=1}^{2} c_{k(k+1)}+c_{30} = c_{01}+c_{12}+c_{23}+c_{03} = 5+3+2+6 = 16$
Arista D-4: [6] $d_1+d_2+d_3+d_4 \leq C$, se conectan los vértices D y 4 con un arco. El coste de este arco es: $c_{01}+\sum_{k=1}^{3} c_{k(k+1)}+c_{40} = c_{01}+c_{12}+c_{23}+c_{34}+c_{04} = 5+3+2+4+4 = 18$.
Arista D-5: [7] $d_1+d_2+d_3+d_4+d_5 > C$ no se conectan los vértices D y 5. Se pasa al siguiente vértice.
Arista 1-3: [8] $d_2+d_3 \leq C$, se conectan los vértices 1 y 3 con un arco. El coste de este arco es: $c_{02}+\sum_{k=2}^{2} c_{k(k+1)}+c_{30} = c_{02}+c_{23}+c_{03} = 4+2+6 = 12$.

[4] Se utiliza la fórmula que hay en 4 con $i = 0, j = 2$
[5] $i = 0, j = 3$
[6] $i = 0, j = 4$
[7] $i = 0, j = 5$
[8] $i = 1, j = 2$

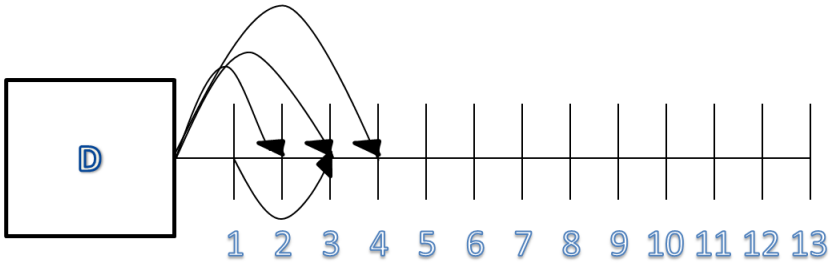

Figura 6.4: Construcción el grafo (3)

Así sucesivamente hasta construir el grafo. Una vez construido, ejecutamos el algoritmo del los camino más corto desde el vértice D hasta el vértice 13.

En este caso el camino más corto en el grafo auxiliar desde D a 13 es: (D,4,9,13). Al "deshacer" la representación de las aristas (D,4), (4,9), (9,13) en rutas, obtenemos la solución siguiente:

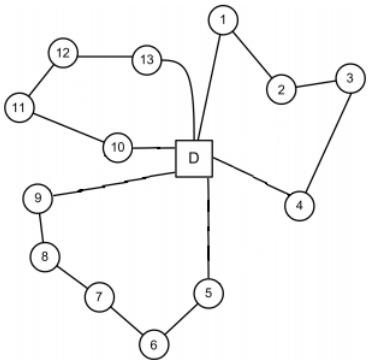

Figura 6.5: Solución proporcionada por el algoritmo de Ulusoy (5)

Como la partición depende de la solución del TSP, si partimos de otra solución del TSP podríamos obtener una solución del CVRP diferente. Podemos pues, repetir el proceso (con una condición de parada) para así explorar más soluciones. Al calcular la solución del TSP también tenemos en cuenta el parámetro de aleatorización α. Con él obtendremos diferentes soluciones del TSP.

Capítulo 7

Resultados computacionales

Este capítulo lo dedicaremos a determinar cuál es el mejor valor de α (parámetro de aleatorización) para nuestros dos algoritmos y a la comparación entre estos. Con este objetivo hemos construido 360 instancias. El tiempo de computación ha sido el mismo para cada instancia y para cada algoritmo.

7.1. Instancias utilizadas

Se han construido 360 instancias en una malla de 50x50. Los vértices se han generado de una manera aleatoria siendo el primero en generarse el depósito y después los clientes. Al generar todos los vértices tenemos que construir la matriz de costes, formada por las distancias Euclídeas entre ellos. Obviamente, la matriz es simétrica y cumple la desigualdad triangular.

Hay tres conjuntos de 120 instancias con 35, 50 y 65 vértices cada uno. También se han clasificado las instancias según los camiones de los que disponemos. Se han usado de 3 a 8 camiones. Tenemos pues 60 instancias con cada número de camiones disponibles. Estas 60 instancias se dividen en: 20 instancias con 35 ciudades, 20 con 50 ciudades y las últimas 20 instancias con 65 ciudades. Se han elegido así para poder hacer un análisis más detallado respecto de las diferentes características de las instancias. Obviamente, la suma de las capacidades de los camiones es mayor que la suma de las demandas ($\sum_{i=1}^{n} d_i \leq \sum_{i=1}^{K} C_i$, siendo K el número de camiones disponibles). El cálculo de la capacidad del camión ha sido el siguiente: calculamos la demanda total ($D = \sum_{i=1}^{n} d_i$), la dividimos por K y, al resultado, le añadimos un 10 % de D.
Así pues, la capacidad del camión queda como: $C = \lceil \frac{D}{K} + 0{,}1 \cdot D \rceil$.

Con este valor para la capacidad de los vehículos esperamos que con K camiones podamos abastecer a nuestros clientes sin tener que hacer rutas simples.

El hecho de haber generado instancias con distintas características se explica por nuestro interés en determinar qué algoritmo proporciona mejores resultados dependiendo de la instancia en la que estemos trabajando.

Las instancias se pueden consultar el repositorio [16]

7.2. Ajuste del parámetro α

El estudio para el ajuste del parámetro es un pilar fundamental en el algoritmo ya que, cuando vayamos a ejecutar nuestro algoritmo, le tenemos que introducir el valor α y el resultado que obtendremos dependerá del valor escogido para α. Por ejemplo, si cogemos $\alpha = 3$ y $\alpha = 10$ en el algoritmo de los ahorros y lo ejecutamos en la instancia 235, observamos una diferencia de 69 puntos sobre el coste de la función objetivo. Una diferencia de 69 puntos en la función objetivo en la práctica, puede llevarnos a una situación precaria o incluso a la quiebra.

Para ajustar α para el algoritmo de los ahorros, hemos ejecutado el algoritmo en las instancias generadas usando valores para α desde 2 hasta 10. Los resultados de las tablas que siguen, equivalen a los valores de la función objetivo, es decir, al coste de la solución.

Mostramos a continuación los resultados obtenidos con el algoritmo de los ahorros. La tabla siguiente proporciona las medias y varianzas de los costes de las soluciones obtenidas con cada valor probado de α, donde α_i representa $\alpha = i$.

	α_2	α_3	α_4	α_5	α_6	α_7	α_8	α_9	α_{10}
Media	435.66	409.85	411.83	416.05	419.10	423.07	426.18	428.72	432.19
Varianza	3905.77	3167.11	3206.67	3268.11	3320.16	3529.45	3573.73	3845.89	3819.61

Cuadro 7.1: Medias y varianzas asociadas a cada valor de α al utilizar el algoritmo de los ahorros

Cuanto menor sea la media mejor es el α, puesto que nuestro objetivo es minimizar el coste de la ruta. Notamos que hay dos posibles buenos candidatos, α_3 y α_4, puesto que son los dos valores con medias más bajas y menor varianza. Vamos a hacer un diagrama de cajas para los resultados obtenidos con estos dos valores del parámetro.

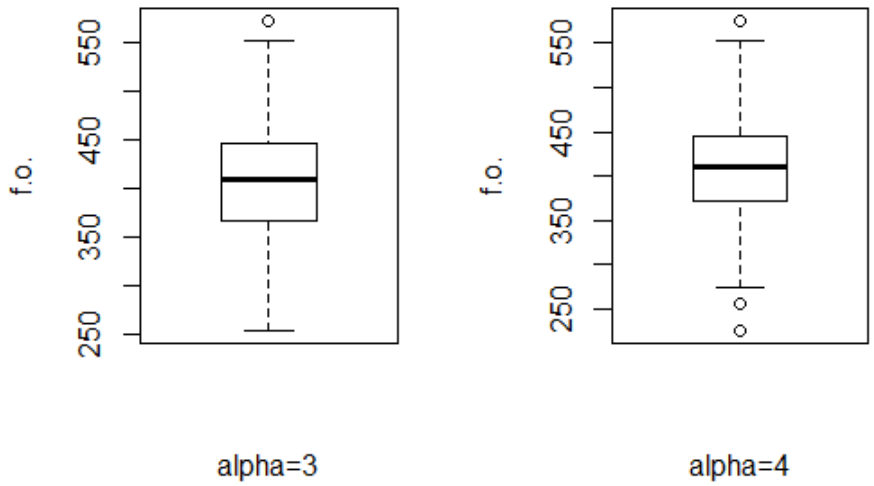

Figura 7.1: Diagrama de cajas para α_3 y α_4 en el algoritmo de los ahorros

Tenemos que α_3 tiene una mejor media que α_4, pero la varianza de α_3 es mayor que la de α_4.

Antes de realizar una comparación con el test de Friedman, vamos a comparar los resultados obtenidos con los distintos valores de α según el número de ciudades, es decir, vamos a seleccionar las instancias según el número de ciudades. Con estas tablas estudiaremos el comportamiento de los α según el tamaño de la instancia.

Para las instancias que tienen 35 ciudades, los resultados obtenidos son los siguientes:

	α_2	α_3	α_4	α_5	α_6	α_7	α_8	α_9	α_{10}
Media	389.57	369.14	373.25	378.58	381.02	383.12	384.10	384.08	384.71
Varianza	2862.99	2401.42	2590.69	2613.67	2554.18	2574.46	2497.44	2534.21	2446.34

Cuadro 7.2: Medias y varianzas con los distintos α en las instancias con 35 ciudades

Vamos a estudiar las instancias con 50 ciudades:

	α_2	α_3	α_4	α_5	α_6	α_7	α_8	α_9	α_{10}
Media	442.57	411.58	413.45	418.03	422.43	428.16	432.77	435.88	442.73
Varianza	2863.34	2152.61	2293.09	2507.53	2669.83	2839.36	2891.32	2900.21	2795.83

Cuadro 7.3: Medias y varianzas con los distintos α en las instancias con 50 ciudades

Por último vamos a ver las instancias que tienen 65 ciudades.

	α_2	α_3	α_4	α_5	α_6	α_7	α_8	α_9	α_{10}
Media	474.36	448.38	448.33	450.99	453.31	457.29	461.03	465.51	468.38
Varianza	2310.49	1784.07	1897.46	2030.74	2085.84	2341.82	2262.30	2656.15	2471.54

Cuadro 7.4: Medias y varianzas con los distintos α en las instancias con 65 ciudades

Los valores α_3 y α_4 tienen medias muy parecidas en muchas situaciones. La suma de las medias de las tablas anteriores es de 1229.1 para α_3 y 1235.03 para α_4. La diferencia entre estas dos medias es de 5.92458. El α_3 tiene una mejor media cuando clasificamos las instancias por su número de ciudades.

Vamos a desglosar un poco más las instancias para hacer un mejor análisis. Vamos a estudiar ahora las medias y varianzas de los α según los camiones que disponemos. Para ello ejecutaremos el algoritmo de los ahorros con los distintos valores de α clasificando las instancias en 6 bloques, desde las que tienen 3 camiones disponibles hasta las que tienen 8 camiones. Seguidamente calcularemos la sumas de las medias y varianzas de las tablas y calcularemos su diferencia. Así pues, veremos las distintas medias de α según esta clasificación.

	α_2	α_3	α_4	α_5	α_6	α_7	α_8	α_9	α_{10}
Media	396.97	364.28	364.22	365.63	368.18	370.50	372.43	373.32	378.00
Varianza	3839.25	2140.55	2194.99	1944.00	1921.61	1918.69	1953.06	1872.46	1973.97

Cuadro 7.5: Medias y varianzas asociadas a cada valor de α para las instancias con 3 camiones

	α_2	α_3	α_4	α_5	α_6	α_7	α_8	α_9	α_{10}
Media	402.95	378.87	378.97	382.12	384.43	388.30	391.22	394.40	397.08
Varianza	1918.01	1315.47	1352.20	1255.05	1127.54	1143.23	1158.17	1194.68	1289.74

Cuadro 7.6: Medias y varianzas asociadas a cada valor de α para las instancias con 4 camiones

	α_2	α_3	α_4	α_5	α_6	α_7	α_8	α_9	α_{10}
Media	432.88	405.57	407.83	411.63	414.97	417.73	421.40	423.50	429.53
Varianza	2842.75	1982.08	1880.11	1735.22	1824.71	1905.42	1916.24	2114.49	2202.69

Cuadro 7.7: Medias y varianzas asociadas a cada valor de α para las instancias con 5 camiones

	α_2	α_3	α_4	α_5	α_6	α_7	α_8	α_9	α_{10}
Media	439.32	412.68	415.15	420.15	423.55	427.87	431.08	432.80	435.92
Varianza	2802.36	2061.03	1909.01	1851.69	1903.40	2070.96	2086.99	2370.06	2362.45

Cuadro 7.8: Medias y varianzas asociadas a cada valor de α para las instancias con 6 camiones

	α_2	α_3	α_4	α_5	α_6	α_7	α_8	α_9	α_{10}
Media	469.40	446.75	450.08	456.42	460.43	463.95	469.35	471.42	473.83
Varianza	3041.74	2603.55	2500.96	2521.47	2511.54	2832.96	2871.11	3229.47	3114.82

Cuadro 7.9: Medias y varianzas asociadas a cada valor de α para las instancias con 7 camiones

	α_2	α_3	α_4	α_5	α_6	α_7	α_8	α_9	α_{10}
Media	471.44	450.08	453.90	459.41	462.10	469.00	470.47	475.68	477.41
Varianza	4164.80	2953.29	2887.23	3132.21	3324.78	3551.03	3521.67	4024.19	4024.25

Cuadro 7.10: Medias y varianzas asociadas a cada valor de α para las instancias con 8 camiones

La diferencia de la suma de las medias de las tablas anteriores entre α_4 y α_3 es de 11.91356, es decir, que α_3 tiene una media más baja en el conjunto de instancias clasificadas por los camiones disponibles. Al igual que antes, calculamos también la diferencia de la suma de varianzas de las tablas anteriores para los valores α_4 y α_3. Esta diferencia es de -331.4685, por tanto, a pesar de que las medias son menores con α_3, también tenemos una gran varianza entre los costes en comparación con el α_4.

Vamos a usar el test de Friedman para el análisis estadístico de nuestros datos. El test de Friedman es un test no paramétrico que compara si los conjuntos de datos son o no iguales, es decir, si existe dependencia entre los costes dependiendo del α que usemos.

Los datos usados en el test son los costes de las 360 instancias. El p-valor asociado al test es de $6.3 \cdot 10^{-11}$. El p-valor es lo suficientemente significativo como para admitir diferencias entre los costes según el α utilizado. Elegimos $\alpha = 3$ para el algoritmo de los ahorros, puesto que es el que menor media de costes ha tenido y, por ende, el que tiende a tener mejores resultados en comparación con los obtenidos con $\alpha = 4$.

Pasemos a ver ahora el ajuste para el algoritmo de Ulusoy (6.2). Al igual que en el algoritmo de los ahorros, vamos a ejecutar el algoritmo con los distintos valores de α y calcular la media y la varianza de la función objetivo en cada una de las instancias.

	α_2	α_3	α_4	α_5	α_6	α_7	α_8	α_9	α_{10}
Media	470.68	452.39	553.89	610.24	640.32	670.73	696.04	721.23	748.11
Varianza	4813.53	4350.83	9807.20	18228.06	23738.47	32569.74	43321.96	56338.54	73505.67

Cuadro 7.11: Medias y varianzas asociadas a cada valor de α al utilizar el algoritmo de Ulusoy

Hacemos un diagrama de cajas con los costes asociados a nuestras 360 instancias con los parámetros de aleatorización α_2 y α_3. Estos dos valores de α parecen ser los mejores, puesto que proporcionan mejores medias que los demás.

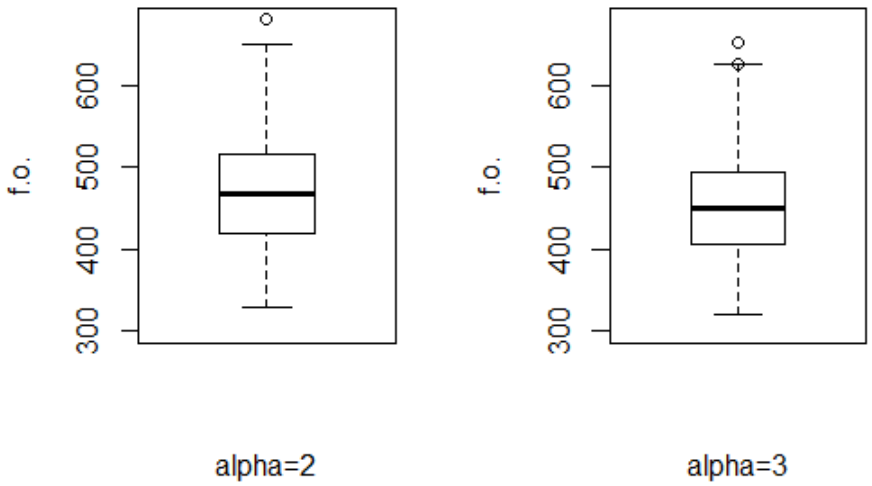

Figura 7.2: Diagrama de cajas para α_2 y α_3 en el algoritmo de Ulusoy

Vemos que la varianza es mucho mayor con α_2 que con α_3. Estudiando las medias obtenidas con esos dos valores de α, parecen no muy diferentes una de otra, aunque en el cómputo global exista una diferencia de 18.29. Analizaremos nuestros datos en profundidad. Al igual que para el algoritmo de los ahorros, empezaremos dividiendo nuestro conjunto de instancias según su número de ciudades.

	α_2	α_3	α_4	α_5	α_6	α_7	α_8	α_9	α_{10}
Media	427.88	412.35	634.56	723.04	727.03	733.34	729.74	726.63	725.35
Varianza	4813.53	4350.83	9807.20	18228.06	23738.47	32569.74	43321.96	56338.54	73505.67

Cuadro 7.12: Medias y varianzas con los distintos α en las instancias con 35 ciudades

	α_2	α_3	α_4	α_5	α_6	α_7	α_8	α_9	α_{10}
Media	454.12	438.73	437.54	436.86	436.47	435.75	435.31	436.15	436.07
Varianza	4813.53	4350.83	9807.20	18228.06	23738.47	32569.74	43321.96	56338.54	73505.67

Cuadro 7.13: Medias y varianzas con los distintos α en las instancias con 50 ciudades

	α_2	α_3	α_4	α_5	α_6	α_7	α_8	α_9	α_{10}
Media	529.30	505.80	589.34	670.54	757.33	843.29	923.19	1000.87	1083.29
Varianza	4813.53	4350.83	9807.20	18228.06	23738.47	32569.74	43321.96	56338.54	73505.67

Cuadro 7.14: Medias y varianzas con los distintos α en las instancias con 65 ciudades

La diferencia de la suma de las medias de las tablas anteriores entre α_2 y α_3 es de 44.62654 y el haciendo el mismo cálculo para la varianza, nos sale una diferencia de 1417.375. Con estos dos datos podemos asegurar que α_3 tiende a dar mejores soluciones en las instancias clasificadas por el número de ciudades. La diferencia entre las medias para los dos valores de α es cercana a 45 puntos en la función objetivo, lo que parece una diferencia bastante significativa.

Por último, vamos a hacer el estudio atendiendo al número de camiones en cada instancia.

	α_2	α_3	α_4	α_5	α_6	α_7	α_8	α_9	α_{10}
Media	417.67	397.28	507.92	566.82	594.38	626.07	653.25	677.45	708.67
Varianza	3145.45	2400.71	9871.57	19365.81	24632.24	33608.27	46026.02	57830.59	80166.09

Cuadro 7.15: Medias y varianzas asociadas a cada valor de α para las instancias con 3 camiones

	α_2	α_3	α_4	α_5	α_6	α_7	α_8	α_9	α_{10}
Media	438.73	421.38	532.65	585.18	615.70	648.58	675.25	703.25	731.28
Varianza	2408.30	2128.04	9894.81	17504.05	22376.89	30895.43	41663.48	55230.60	74407.43

Cuadro 7.16: Medias y varianzas asociadas a cada valor de α para las instancias con 4 camiones

	α_2	α_3	α_4	α_5	α_6	α_7	α_8	α_9	α_{10}
Media	456.77	440.83	538.92	599.15	629.90	656.77	686.33	714.33	731.58
Varianza	3243.81	3100.79	8281.60	17881.32	23660.57	33151.47	45752.36	62323.58	75487.70

Cuadro 7.17: Medias y varianzas asociadas a cada valor de α para las instancias con 5 camiones

	α_2	α_3	α_4	α_5	α_6	α_7	α_8	α_9	α_{10}
Media	487.63	467.80	564.00	622.62	653.90	687.08	706.50	735.53	766.22
Varianza	4440.24	3818.03	7999.86	16659.26	22826.67	33352.79	42578.29	60964.02	79745.19

Cuadro 7.18: Medias y varianzas asociadas a cada valor de α para las instancias con 6 camiones

	α_2	α_3	α_4	α_5	α_6	α_7	α_8	α_9	α_{10}
Media	500.37	485.20	580.45	638.08	669.65	700.12	720.72	741.00	770.32
Varianza	2796.85	2659.38	8050.35	16904.25	23125.99	31056.55	40010.07	50361.69	66704.53

Cuadro 7.19: Medias y varianzas asociadas a cada valor de α para las instancias con 7 camiones

	α_2	α_3	α_4	α_5	α_6	α_7	α_8	α_9	α_{10}
Media	521.31	501.17	599.12	648.66	676.78	703.81	731.24	751.59	776.22
Varianza	5139.42	4387.59	9974.55	17650.30	22636.52	31301.64	43169.15	51743.97	66665.28

Cuadro 7.20: Medias y varianzas asociadas a cada valor de α para las instancias con 8 camiones

La suma para las tablas anteriores de las diferencias entre las medias de los valores obtenidos con α_2 y α_3 es de 89.20339 y de 2880.043 en el caso de las varianzas. Por tanto, α_3 es el mejor valor si consideramos las instancias clasificadas según el número de camiones.

Queda claro que el mejor valor para α es $\alpha = 3$ para el algoritmo de Ulusoy, puesto que tiene menor media y menor varianza. Aún así, vamos a hacer el test de Friedman para concluir que los valores obtenidos con α_2 y α_3 son distintos. El p-valor del test es menor que $2 \cdot 10^{-16}$. Por tanto podemos afirmar que las los valores obtenidos con α_2 y α_3 son distintos. Fijamos pues, α_3 como parámetro de aleatorización para el algoritmo de Ulusoy.

7.3. Comparación de los algoritmos propuestos

Una vez decidido cuál es el mejor valor α para cada uno de nuestros algoritmos, vamos a analizar qué algoritmo funciona mejor. Esto nos servirá para invertir el tiempo de computación en el algoritmo que mejor funcione.

En la sección 7.2 hemos decidido fijar $\alpha = 3$ en ambos algoritmos. Son los valores que finalmente hemos utilizado en su comparación. Comenzamos mostrando una tabla y un diagrama de cajas de los valores obtenidos. Hacemos un resumen de los valores de la función objetivo obtenidos. Este resumen consiste en el mínimo, primer cuartil, mediana, media, tercer cuartil y el máximo de esos valores.

Resumen / Algoritmo	Mínimo	Q_1	Mediana	Media	Q_3	Maximo
Ahorros	255	367.8	409.5	446	375	572
Ulusoy	320	406	449.5	452.4	494.2	654

Cuadro 7.21: Resumen de cada algoritmos

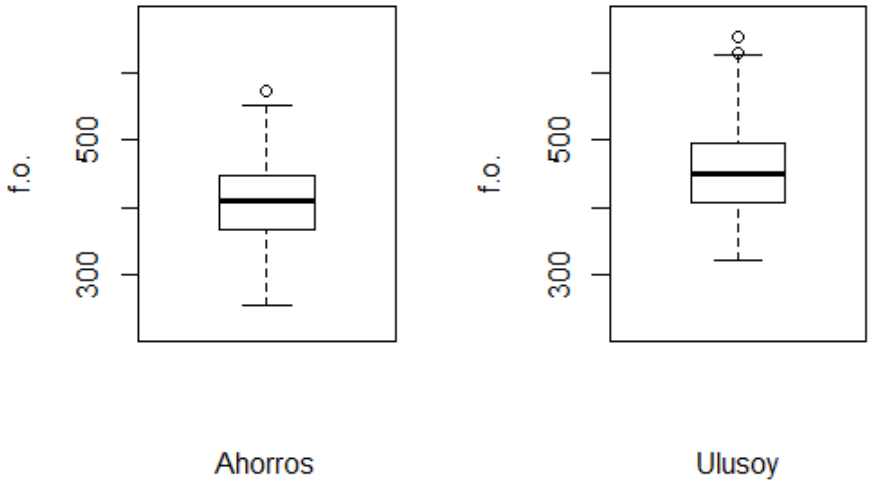

Figura 7.3: Diagrama de cajas para el algoritmo de los ahorros y de Ulusoy

Se observa claramente que el algoritmo que mejor funciona de los dos es el de los ahorros. De la tabla anterior (7.21) podemos ver que la varianza del algoritmo de los ahorros es de 3167.112, mientras que la varianza del algoritmo de Ulusoy es 4350.829. Queda así una diferencia de 1183.717 entre varianzas, que, a efectos prácticos, se relaciona con la diversidad de los resultados que podemos encontrar dependiendo de qué algoritmo estemos usando. Así pues, cuanto menor sea la varianza, menor dispersión de costes tendremos.

Vamos a ver cómo se comportan los algoritmos según las ciudades y los camiones disponibles.

Resumen / Algoritmo	Mínimo	Q_1	Mediana	Media	Q_3	Maximo
Ahorros	255	336	362	369.14	394.5	501
Ulusoy	320	371	407	412.35	449.75	549

Cuadro 7.22: Resultados de los algoritmos en las instancias con 35 ciudades

Resumen / Algoritmo	Mínimo	Q_1	Mediana	Media	Q_3	Maximo
Ahorros	322	375	408	411.58	440.25	572
Ulusoy	328	404	432.00	438.73	472.25	610

Cuadro 7.23: Resultados de los algoritmos en las instancias con 50 ciudades

Resumen / Algoritmo	Mínimo	Q_1	Mediana	Media	Q_3	Maximo
Ahorros	369	417	441	448.38	487.50	551
Ulusoy	394	465.50	501	505.80	538.50	654

Cuadro 7.24: Resultados de los algoritmos en las instancias con 65 ciudades

Hay una clara evidencia de que el algoritmo de los ahorros da mejores resultados que el de Ulusoy según el tamaño de la instancia. En las tablas observamos la gran diferencia en las medias que hay entre los algoritmos.

Vamos a estudiar cómo se comportan los algoritmos con respecto a los camiones disponibles:

Resumen / Algoritmo	Mínimo	Q_1	Mediana	Media	Q_3	Maximo
Ahorros	255	329.25	364	364.28	402.25	461
Ulusoy	320	355.50	395	397.28	429.25	491

Cuadro 7.25: Resultados de los algoritmos en las instancias con 3 camiones disponibles

Resumen / Algoritmo	Mínimo	Q_1	Mediana	Media	Q_3	Maximo
Ahorros	299	352	380	378.87	409	446
Ulusoy	322	386	420.50	421.38	448.25	535

Cuadro 7.26: Resultados de los algoritmos en las instancias con 4 camiones disponibles

Resumen / Algoritmo	Mínimo	Q_1	Mediana	Media	Q_3	Maximo
Ahorros	317	373.75	412.50	405.57	437.25	509
Ulusoy	348	394.75	438	440.83	477.50	570

Cuadro 7.27: Resultados de los algoritmos en las instancias con 5 camiones disponibles

Resumen / Algoritmo	Mínimo	Q_1	Mediana	Media	Q_3	Maximo
Ahorros	302	385.75	413.50	412.68	443	523
Ulusoy	354	418	462.50	467.80	504.25	625

Cuadro 7.28: Resultados de los algoritmos en las instancias con 6 camiones disponibles

Resumen / Algoritmo	Mínimo	Q_1	Mediana	Media	Q_3	Maximo
Ahorros	336	404.25	446	446.75	488.50	551
Ulusoy	381	456.50	485.00	485.20	514.50	627

Cuadro 7.29: Resultados de los algoritmos en las instancias con 7 camiones disponibles

Resumen / Algoritmo	Mínimo	Q_1	Mediana	Media	Q_3	Maximo
Ahorros	325	417	448	450.08	489.50	572
Ulusoy	375	457	503	501.17	539.50	654

Cuadro 7.30: Resultados de los algoritmos en las instancias con 8 camiones disponibles

Viendo los resultados, podemos afirmar que el algoritmo de los ahorros es mejor que el algoritmo de Ulusoy, independientemente de las características principales de las instancias (número de ciudades y de camiones). Esto se debe a que el algoritmo de los ahorros proporciona soluciones con menor coste que el algoritmo de Ulusoy en la mayoría de las instancias generadas. Exactamente, el algoritmo de Ulusoy solo proporciona mejores soluciones en 76 de las 360 instancias, lo que representa el 21.11 % de las mismas.

Hacemos el test de Friedman para confirmar estadísticamente esta suposición. El p-valor que nos da el test es menor a $2.2 \cdot 10^{-16}$. Podemos asegurar pues que los dos algoritmos proporcionan soluciones con costes diferentes. El algoritmo que mejores resultados proporciona es el de los ahorros con parámetro de aleatorización α_3.

7.4. Comparación del óptimo de las instancias con la solución computada

Las instancias que vamos a utilizar están en el banco de datos de la Universidad de Málaga [17].

Representaremos por $z(H)$ al ***valor de la solución obtenida con el algoritmo H*** y por z^* al ***valor de la solución óptima***.

Para comparar los valores de las soluciones óptimas con las obtenidas con nuestros algoritmos, calcularemos $\frac{z(H)-z^*}{z^*} \cdot 100$. Los valores resultantes se conocen como desviaciones o gaps. El tiempo de computación que se ha utilizado ha sido de 30 segundos. Se ha utilizado α_3 en ambos algoritmos (7.2).

Los costes óptimos y los obtenidos con nuestros algoritmos son los siguientes:

Instancia	Óptimo	Ahorros	Ulusoy		Instancia	Óptimo	Ahorros	Ulusoy
1	784	807	834		11	937	979	1028
2	661	678	738		12	944	978	1074
3	742	763	816		13	1146	1183	1289
4	778	797	820		14	914	947	1042
5	799	807	871		15	1073	1128	1200
6	669	694	759		16	1010	1057	1211
7	949	986	1072		17	1167	1207	1302
8	730	757	802		18	1073	1106	1251
9	822	858	973		19	1035	1087	1222
10	831	855	941		20	1177	1225	1390

Cuadro 7.31: Comparación de resultados

Seguidamente vamos a ver las desviaciones obtenidas entre las soluciones óptimas y las generadas por nuestros procedimientos:

Instancias	Ahorros	Ulusoy		Instancias	Ahorros	Ulusoy
1	2.93	6.37		11	4.48	9.71
2	2.57	11.64		12	3.60	13.77
3	2.83	9.97		13	3.22	12.47
4	2.44	5.39		14	3.61	14.00
5	1.00	9.01		15	5.12	11.83
6	3.73	13.45		16	4.65	19.90
7	3.89	12.96		17	3.42	11.56
8	3.69	9.86		18	3.07	16.58
9	4.37	18.36		19	5.02	18.06
10	2.88	13.23		20	4.07	18.09

Cuadro 7.32: Desviaciones respecto del valor óptimo (gap)

Notamos que la desviación del algoritmo de los ahorros es mucho menor que la desviación del algoritmo de Ulusoy, lo que confirma el estudio realizado en la sección anterior (7.3).

La propia base de datos [17] proporciona algunas soluciones gráficas. Representamos a continuación la solución del algoritmo de los ahorros y la solución óptima, respectivamente, para la instancia 7.

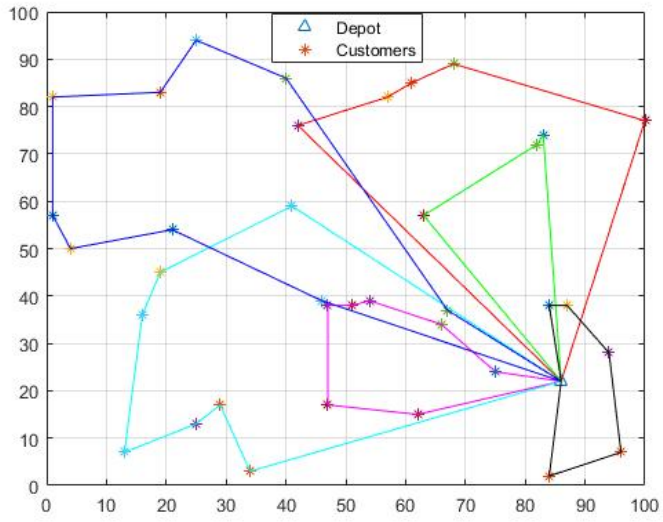

Figura 7.4: Solución computada

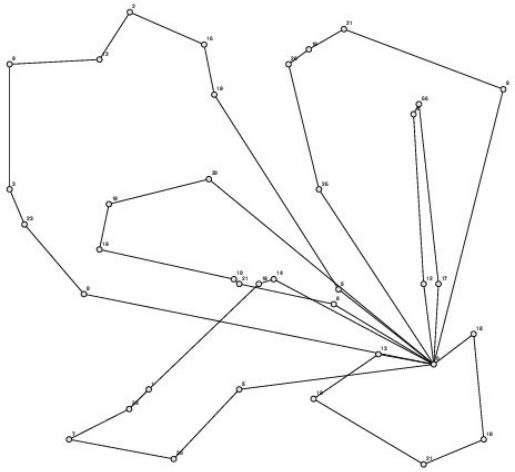

Figura 7.5: Solución óptima

Podemos observar que hay mucha semejanza en algunas de las rutas de estas dos representaciones gráficas. Cuantitativamente, la desviación entre la solución óptima y la proporcionada por el algoritmo de los ahorros es 3.89 (Tabla 7.32).

Capítulo 8

Conclusiones

En esta memoria hemos estudiado el problema de rutas de vehículos con capacidades. Para este problema hemos propuesto dos algoritmos heurísticos, uno basado en el algoritmo de los ahorros de Clarke and Wright y el otro en el algoritmo de Ulusoy.

Hemos realizado una comparación en profundidad de ambos algoritmos sobre 360 instancias con distintas características (7.1), las cuales se han utilizado para:

1. determinar el parámetro de aleatorización, α, de cada uno de los algoritmos (7.2), y
2. la comparación entre algoritmos (7.3).

La determinación del valor para el parámetro α se realizó mediante un análisis estadístico. En el análisis clasificamos las instancias según sus características, es decir, respecto del número de ciudades y el número de camiones disponibles. El parámetro α_3 proporcionó mejores resultados en ambos algoritmos.

Posteriormente, comparamos ambos algoritmos para determinar cuál de ellos proporciona mejores resultados y en qué situación conviene utilizar cada uno de ellos. Los datos muestran que el algoritmo de los ahorros es mejor en prácticamente todas las instancias probadas, aunque en 76 instancias el algoritmo de Ulusoy proporciona una solución mejor que el algoritmo de los ahorros.

También hemos utilizado un conjunto de instancias con solución óptima conocida. Estas instancias se han descargado de la base de datos [17]. Hemos ejecutado nuestros algoritmos en estas instancias durante 30 segundos de computación para ver qué desviaciones mostraban respecto de las soluciones óptimas. (7.4). En la Tabla (7.31) observamos que ningún valor del algoritmo de Ulusoy es mejor que los valores obtenidos con el algoritmo de los ahorros, dejando claro así que el algoritmo de los ahorros tiende a proporcionar mejores soluciones. En la Tabla (7.32) podemos ver que las desviaciones no son muy grandes, siendo la mayor 5.12 y la menor 1.00.

Tenemos pues un algoritmo que, con un bajo tiempo computacional, proporciona soluciones buenas al problema de rutas de vehículos con capacidades en instancias grandes. Este algoritmo se puede combinar con un B&B (3.2.2) para poder saturar nodos por dominancia, ya que tenemos una solución posible con un coste cercano al valor óptimo.

Bibliografía

[1] L. Bodin, B. Golden, A. Assad and M.Ball (1983). Routing and scheduling of vehicles and crews: The state of the art, Comp. Ops Res., 10, 63-211.

[2] J. A. Bondy and U.S.R. Murty (1976). Graph Theory with Applications, NY.

[3] G. Clarke and J.W. Wright (1964). Scheduling of vehicles from a central depot to a number of delivery points, Operations Research 12, 568-581.

[4] Á. Corberán and G. Laporte (2014). Arc Routing: Problems, Methods and Applications. MOS-SIAM Series on Optimization. Philadelphia.

[5] G.B.Dantzig (1963). Linear Programming and Extensions, Princeton University Press, Princeton.

[6] E. W. Dijkstra (1959) A note on two problems in connexion with graphs, Numerische Mathematik 1, 269-271.

[7] F. Harary (1969). Graph Theory, Addison-Wesley, Reading MA.

[8] L. G. Khachiyan (1979). A Polinomial Algorithm in Linear Programming, Soviet Mathematics Dolady 20, 191-194.

[9] M. Laguna and R. Martí (2013). Heuristics. In: Gass and Fu (eds). Encyclopedia of Operations Research and Management Science. Springer, Boston, MA.

[10] G. Laporte (2004). Introduction to Logistics Systems Planning and Control. John Wiley & Sons, 23-25.

[11] F. Marqués (2017). R en profundidad. Programación, gráficos y estadística, RCLIA, 300-307.

[12] P. Toth and D. Vigo (2014). Vehicle Routing: Problems, Methods, and Applications. MOS-SIAM Series on Optimization. Philadelphia.

[13] G. Ulusoy (1985). The fleet size and mix problem for capacitated arc routing. European Journal of Operational Research 22 (3), 329-337

[14] H. P. Williams (1993) Model solving in mathematical programming, Wiley Chichester.

[15] Algoritmos en Java: Actualmente a fecha de 21 de febebro de 2020, se está trabajando en un proyecto de mejora del

[16] Repositorio de instancias:Actualmente a fecha de 21 de febebro de 2020, se está trabajando en un proyecto de mej

[17] Instancias con la solución óptima: http://neo.lcc.uma.es/vrp/vrp-instances/capacitated-vrp-instances/

[18] Programas de R: https://mega.nz/#F!XMkFAIyK!tAr7EVzQmtbgONtX2Y3XIg