Trabajo Fin de Máster

Cooperación en problemas de rutas de vehículos: estimación del valor de Shapley

Guido Ignacio Novoa Flores

Máster en Técnicas Estadísticas

Curso 2019-2020

Propuesta de Trabajo Fin de Máster

Título en galego: Cooperación en problemas de rutas de vehículos: estimación do valor de Shapley.

Título en español: Cooperación en problemas de rutas de vehículos: estimación del valor de Shapley.

English title: Cooperative vehicle routing problems: Shapley value estimation.

Modalidad: Modalidad A

Autor: Guido Ignacio Novoa Flores, Universidad de Santiago de Compostela

Directora: Balbina Casas Méndez, Universidad de Santiago de Compostela

Breve resumen del trabajo:

Dentro de las aplicaciones de los juegos cooperativos a los problemas de investigación operativa se encuentra el caso de los problemas de rutas de vehículos. La complejidad del cálculo de soluciones a los juegos cooperativos surgidos en este contexto, como el valor de Shapley, se podría aminorar usando procedimientos de estimación del mismo que han sido introducidos en artículos recientes. Se pretende revisar algunos trabajos relacionados con esta doble línea de investigación, que da título al TFM, e ilustrarlos empíricamente por medio de datos tomados de la literatura de los problemas de rutas.

Referencias:

1. J. Castro, D. Gómez and J. Tejada (2009) Polynomial calculation of the Shapley value based on sampling. Computers and Operations Research 36: 1726 – 1730.

2. S. Zibaei, A. Hafezalkotob and S. S. Ghashami (2016) Cooperative vehicle routing problem: An opportunity for cost saving. Journal of Industrial Engineering International 12: 271 – 286.

Otras observaciones: Se requerirá el uso de lenguajes propios de la programación matemática como AMPL y de bibliotecas de R como GameTheory.

Doña Balbina Casas Méndez, profesora titular del departamento de Estadística, Análisis Matemático y Optimización de la Universidad de Santiago de Compostela, informa que el Trabajo Fin de Máster titulado

Cooperación en problemas de rutas de vehículos: estimación del valor de Shapley

fue realizado bajo su dirección por don Guido Ignacio Novoa Flores para el Máster en Técnicas Estadísticas. Estimando que el trabajo está terminado, da su conformidad para su presentación y defensa ante un tribunal.

En Santiago de Compostela, a 8 de septiembre de 2020.

La directora:

Doña Balbina Casas Méndez

El autor:

NOVOA FLORES GUIDO IGNACIO - 49333849E

Firmado digitalmente por NOVOA FLORES GUIDO IGNACIO - 49333849E
Fecha: 2020.09.08 21:13:48 +02'00'

Don Guido Ignacio Novoa Flores

Índice general

Resumen

Resumen en español

En este trabajo revisamos los problemas de rutas de vehículos con múltiples depósitos. En este tipo de problemas una empresa cuenta con una flota de vehículos y varios puntos desde donde distribuir un determinado producto. Usaremos programación lineal para formular esta situación intentando minimizar los costes generados en dicha distribución. No obstante, esta formulación no permite resolver casos que involucren una gran cantidad de clientes al tratarse de problemas de gran tamaño. Por ese motivo, diseñamos un algoritmo metaheurístico que parte de una solución usando el algoritmo propuesto por Clarke y Wright (1964) adaptado a nuestro problema e incluye una fase de mejora basada en una técnica voraz.

Por otro lado, generalizamos el problema suponiendo que existe un conjunto de empresas que decide compartir sus recursos para intentar reducir sus costes individuales. En este nuevo modelo surge otro problema asociado al reparto del coste global si las empresas deciden cooperar. Para eso, recurrimos a reglas de reparto que aparecen en el contexto de los juegos cooperativos como el valor de Shapley.

English abstract

In this work we address the vehicle routing problem with multiple depots. In this type of problems a company owns a fleet of vehicles and several points from which to distribute a certain product. We will use linear programming to formulate this situation in order to minimize the costs generated by such distribution.However, this formulation cannot be used to solve cases involving a large number of customers because of its high computing demands. For this reason, we designed a metaheuristic algorithm that starts from a solution using the algorithm proposed by Clarke y Wright (1964) adapted to our problem and includes an improvement phase based on the iterated greedy algorithm.

Besides, we generalize the problem by assuming that there is a group of companies that decide to share their resources trying to reduce their individual costs. In this new model, another problem arises associated with the distribution of the overall cost if companies decide to cooperate. To do this, we use distribution rules that appear in the context of cooperative games such as the Shapley value.

Capítulo 1

Preliminares

1.1. Problemas de rutas de vehículos (VRP)

Una definición genérica de la familia de los problemas de rutas de vehículos (VRP, por sus siglas en inglés) puede ser la siguiente:

Dado un conjunto de clientes que demandan un producto o servicio y una flota de vehículos, se trata de determinar un conjunto de rutas que satisfaga la demanda de los clientes a coste mínimo; en particular, decidir para cada vehículo en qué orden tienen que visitar a los clientes de forma que las rutas se realicen al menor coste posible y de forma factible atendiendo a las diversas restricciones existentes, como por ejemplo la capacidad de los vehículos.

Esta familia de problemas es una de las más estudiadas y aplicadas en el ámbito de la Investigación Operativa. Originalmente, el VRP nace como una extensión del problema de viajante de comercio (TSP). En este problema, un agente de comercio parte de una ciudad origen a la que tiene que regresar tras visitar a un conjunto de clientes intentando que el coste total sea mínimo. En Dantzig y Ramser (1959), los autores propusieron la primera formulación matemática y una aproximación heurística del VRP para un problema de reparto de combustible desde un depósito a 12 estaciones de servicio. A diferencia del TSP, si suponemos que los vehículos que transportan combustible tienen una capacidad limitada, inferior a la demanda total de los clientes, es necesario utilizar más de un vehículo. Este trabajo se convirtió en la base para el desarrollo de otras formulaciones y generalizaciones que van incrementando su complejidad en número de variables y restricciones.

La familia de problemas del VRP es una de las más prolíficas tanto por su interés científico como por su aplicación a problemas reales. Entre las aplicaciones del VRP destacan las relacionadas con el campo de la logística: diseño de rutas escolares, reparto de mercancías y correos, sistemas de recogida de basura, industria marítima, etc. Actualmente, las empresas tratan de mejorar sus cadenas de suministro resolviendo simultáneamente algunas fases conjuntamente, e.g., producción, distribución o inventario.

Debido al gran número de aplicaciones industriales que posee, ha sido un problema enormemente estudiado, produciéndose una evolución constante en la calidad de las metodologías resolutivas usadas, tanto en las númericamente exactas como en las aproximadas.

Desde el punto de vista computacional, el VRP pertenece a la clase de problemas NP-completos (véase Lenstra y Kan (1981)) para los que no existe un algoritmo que pueda resolver el problema en

tiempo polinomial. Por tanto, para instancias grandes es necesario recurrir a técnicas heurísticas o metaheurísticas para obtener soluciones con rapidez y que se aproximen a la solución óptima.

Para una revisión exhaustiva de los VRP puede consultarse Toth y Vigo (2002).

1.1.1. Formulación del VRP

En un problema de distribución, un grupo de clientes solicita un producto a una empresa que tratará de satisfacer sus demandas ajustándose a las necesidades específicas de sus clientes usando los recursos de los que dispone. Por su parte, la empresa tratará de minimizar alguno de los costes derivados diseñando una política de trabajo óptima para su flota de vehículos que, además de las exigencias de los clientes, tendría que cumplir los requerimientos legales aplicables a cada situación.

En la literatura aparecen numerosas referencias sobre problemas de rutas ya que muchos surgen como necesidad a resolver un problema real. Por otro lado, el desarrollo de la industria y la tecnología computacional contribuyen a la aparición de modelos cada vez más precisos y adaptados a casos concretos dando lugar a una gran variedad de problemas. No obstante, los siguientes elementos son comunes a todos los problemas de rutas:

Red de rutas: Usada para el transporte de mercancías indica las posibles conexiones entre clientes y/o depósito(s) así como el coste de cada conexión. Este coste varía en función de los objetivos de la empresa y la capacidad que tengan estas de analizarlo. Actualmente, algunas empresas cuentan con sistemas de información geográfica (GIS) para obtener información detallada de las rutas en tiempo real como el estado de la red, existencia de peajes, etc. Así, las empresas pueden hacer una estimación más ajustada de los costes y una planificación más realista de las rutas.

Depósito(s): Lugar donde se almacenan los productos y suele estar establecida la flota. Una empresa puede contar con más de un depósito en cuyo caso pueden presentar distintas características como la composición de la flota (en número y tipo de vehículos) o capacidad de almacenamiento. La existencia de varios depósitos puede provocar que el diseño de las rutas incluya la decisión de qué depósito atiende a cada cliente si no existe una asignación previa.

Flota de vehículos: Usados para distribuir la mercancía, generalmente, desde los depósitos a los distintos clientes. Dependiendo de la empresa y el producto a repartir podría tratarse de una flota genérica con vehículos similares o de vehículos adaptados con unas características concretas como la existencia de múltiples compartimentos o distintas capacidades. Los responsables de cada vehículo suelen ser conductores cuyas exigencias están relacionadas con su contrato, i.e., duración y franja horaria en la que se tiene que realizar su jornada laboral, posibilidad de hacer descansos, etc. En caso de que los conductores estén asignados a un vehículo concreto, sus restricciones pueden integrarse con las correspondientes del vehículo.

Objetivo de la empresa: Aunque en líneas generales será minimizar el coste total del transporte, entendiendo este coste como la distancia recorrida o el tiempo de viaje, se pueden encontrar otros como seleccionar el número mínimo de vehículos necesarios o atender a tantos clientes como sea posible. Por otro lado, es posible incluir varios objetivos generando problemas más complejos.

A continuación, introduciremos los parámetros y la notación necesarios para presentar el VRP con capacidades (CVRP). Este modelo nos servirá como base para otros más generales que expondremos a lo largo de la memoria. El CVRP se ajusta a la siguiente situación:

Un **conjunto de clientes** representado por el conjunto $N = \{1, \dots, n\}$ solicitan una cantidad $d_i \geq 0$ ($i \in N$) de un producto que tendrá que ser entregado en un único envío. Para ello, la empresa cuenta

con una **flota de vehículos** idénticos $K = \{1, \ldots, |K|\}$ con una capacidad idéntica Q situados en su único **depósito** ($\{0\}$) desde el cual reparte el producto y donde los vehículos tienen que regresar una vez finalizadas sus rutas. Supondremos además que $d_i \leq Q$ y $d_0 = 0$. Para la distribución del producto, los vehículos usarán la **red de rutas** que puede ser vista como un grafo $G = (V, A)$ donde el conjunto de vértices $V = \{0\} \cup N$ representa el depósito y el conjunto de clientes, respectivamente; y el conjunto de arcos $A = \{(i, j) \in V \times V / i \neq j\}$ las conexiones disponibles entre cada par de vértices. Cada una de estas conexiones incurre en un coste que denotaremos por c_{ij} y que interpretaremos como la distancia entre cada par de vértices. El **objetivo de la empresa** será minimizar la distancia total recorrida por sus vehículos, de forma que se satisfagan las demandas y exigencias de los clientes. Para ello tendrá que seleccionar el conjunto de arcos que hagan mínima esta distancia.

La Figura 1.1 muestra un ejemplo de CVRP en el que en cada nodo representa un cliente con sus respectivas demandas. Además, supondremos que la flota dispone de tres camiones idénticos con capacidad $Q = 100$.

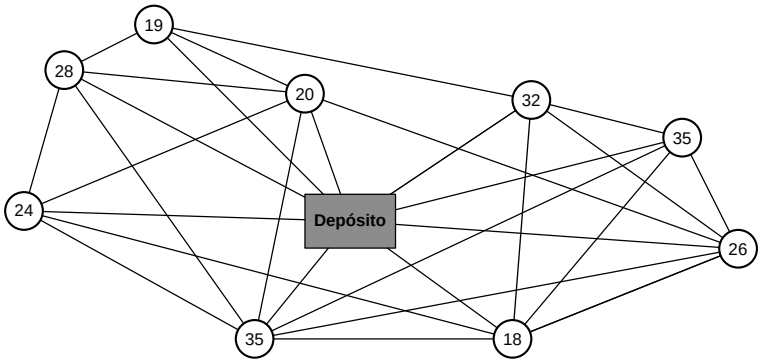


Figura 1.1: Ejemplo VRP con 9 clientes.

Para plantear un problema de programación lineal que modele esta situación, empezamos introduciendo las siguientes variables de decisión binarias:

$$x_{ij} = \begin{cases} 1, & \text{si el arco } (i,j) \in A \text{ es seleccionado;} \\ 0, & \text{en caso contrario.} \end{cases}$$

De esta forma, podemos modelar el CVRP como el siguiente problema de programación lineal binaria:

Modelo (VRP1)

$$\text{Min } z = \sum_{(i,j) \in A} c_{ij} x_{ij} \tag{1}$$

Sujeto a

$$\sum_{i \in V} x_{ij} = 1 \qquad \forall j \in N \tag{2}$$

$$\sum_{j \in V} x_{ij} = 1 \qquad \forall i \in N \tag{3}$$

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1 \qquad \forall S \subset N, |S| \geq 2 \tag{4}$$

$$\sum_{i \in N} x_{i0} \leq |K| \tag{5}$$

$$\sum_{j \in N} x_{0j} \leq |K| \tag{6}$$

$$\sum_{i \in S \cup \{0\}} \sum_{j \in S} d_j x_{ij} \leq Q \qquad \forall S \subset N, S \neq \emptyset \tag{7}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in V \tag{8}$$

En este modelo, la función objetivo (1) selecciona los arcos de la red de forma que la distancia total recorrida sea minimizada. En cuanto a las restricciones, como cada cliente solo puede ser visitado una vez, se tiene que por cada vértice del conjunto de clientes N solo pueden pasar dos arcos: uno de entrada (restricción (2)) y otro de salida (restricción (3)). No obstante, esta condición podría no ser suficiente ya que podrían formarse ciclos entre los clientes de forma que cada uno tuviese exactamente un arco entrante y otro saliente. En la Figura 1.2 vemos un ejemplo de ciclo entre 3 clientes unidos por 3 arcos. Para evitar esta situación, es necesario que el número de arcos en cualquier subconjunto de clientes $S \subset N$ sea inferior a $|S|$[1] como indica la restricción (4). En cuanto al depósito, vértice $\{0\}$, como cada vehículo comienza y finaliza su recorrido en él se tiene que el número de arcos entrantes (restricción (5)) y salientes (restricción (6)) no puede superar al número de vehículos. Por último, la restricción (7) indica que la demanda total de un subconjunto de clientes $S \subset N$ no puede exceder la capacidad Q de los vehículos.

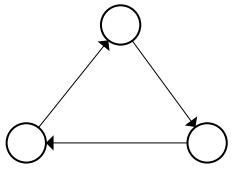


Figura 1.2: Ejemplo de ciclo con 3 clientes.

Si resolviéramos el ejemplo planteado en la Figura 1.1 mediante el modelo (VRP1) obtendríamos un problema de programación lineal con 1033 restricciones y 72 variables binarias. La Figura 1.3 representa una posible solución del problema indicando la ruta que realiza cada vehículo.

[1] Siendo $|S|$ el número de elementos en el conjunto S.

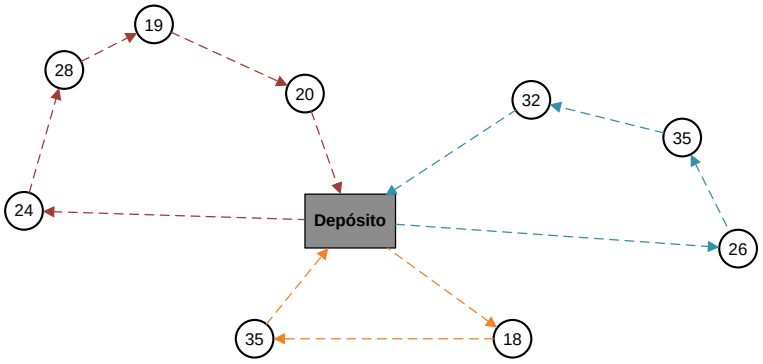


Figura 1.3: Posible solución al ejemplo de la Figura 1.1 considerando una flota homogénea.

Uno de los principales inconvenientes de este modelo es que el número de restricciones es del orden $\mathcal{O}(2^n)$ haciendo que no sea apropiado para resolver situaciones con un número elevado de clientes. Esta situación se puede evitar reescribiendo la restricciones (4) y (7). Para esto, adaptamos la propuesta de Miller, Tucker y Zemlin (1960) para las restricciones de rotura de ciclo en el TSP.

Si introducimos las variables $u_i \geq 0$ representando la demanda repartida después de visitar al cliente $i \in N$, el modelo (VRP1) puede reescribirse como el siguiente modelo, el cual denominamos (VRP2).

Modelo (VRP2)

$$\text{Min } z = \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{1}$$

Sujeto a

$$\sum_{i\in V} x_{ij} = 1 \qquad \forall j \in N \tag{2}$$

$$\sum_{j\in V} x_{ij} = 1 \qquad \forall i \in N \tag{3}$$

$$u_i + d_j x_{ij} - Q(1 - x_{ij}) \leq u_j \qquad \forall i, j \in N \tag{4}$$

$$\sum_{i\in N} x_{i0} \leq |K| \tag{5}$$

$$\sum_{j\in N} x_{0j} \leq |K| \tag{6}$$

$$d_i \leq u_i \leq Q \qquad \forall i \in N \tag{7}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in V \tag{8}$$

$$u_i \geq 0 \qquad \forall i \in N \tag{9}$$

Si en la restricción (4) del modelo (VRP2) suponemos $x_{ij} = 1$ obtendríamos que $u_i < u_i + d_j \leq u_j$, y, de haber un ciclo entre clientes($i \to j \to k \to \ldots \to i$) tendríamos que $u_i < u_j < u_k \ldots < u_i$ lo cual es una contradicción. En el caso $x_{ij} =$ obtendríamos $u_i - Q \leq u_j$ lo cual está garantizado por la restricción(7). Esta última restricción asegura que la demanda repartida nunca excede la capacidad de un vehículo.

Comparando ambos modelos, observamos que el modelo (VRP2) incluye $\mathcal{O}(n^2)$ variables binarias y $\mathcal{O}(n)$ variables continuas frente a las $\mathcal{O}(n^2)$ variables binarias del modelo (VRP1). No obstante, todas

las restricciones del modelo (VRP2) son de orden polinómico. En particular, el ejemplo representado en la Figura 1.1 daría lugar a un problema de programación lineal con 110 restricciones y 81 variables (72 binarias y 9 continuas).

Sin embargo, estos modelos presentan algunos inconvenientes al no saber qué vehículo pasa por cada ruta. Si los vehículos presentan distintas características como capacidad o imposibilidad de visitar a algún cliente estos modelos no serían apropiados. Para indicar qué vehículo realiza cada ruta habría que incluir esta información en las variables de decisión.

Supongamos que los vehículos presentan distintas capacidades y sea $Q_k \geq 0$ la capacidad del vehículo $k \in K$. Seguimos suponiendo que los clientes pueden ser visitados por cualquier vehículo al no exceder sus demandas las capacidades de los camiones, $\max(d_i) \leq \min(Q_k)$.

Extendiendo las variables de decisión del modelo (VRP2) introducimos las siguientes variables de decisión:

$$x_{ijk} = \begin{cases} 1, & \text{si el arco } (i,j) \in A \text{ es recorrido por el vehículo } k \in K; \\ 0, & \text{en caso contrario.} \end{cases}$$

$u_{ik} \geq 0$, demanda repartida por el vehículo $k \in K$ tras visitar al cliente $i \in N$.

De esta forma, podemos presentar un tercer modelo, denominado (VRP3), para el VRP en caso de que la flota de vehículos presente una capacidad heterogénea.

Modelo (VRP3)

$$\text{Min } z = \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} x_{ijk} \tag{1}$$

Sujeto a

$$\sum_{k \in K} \sum_{i \in V} x_{ijk} = 1 \qquad \forall j \in N \tag{2}$$

$$\sum_{k \in K} \sum_{j \in V} x_{ijk} = 1 \qquad \forall i \in N \tag{3}$$

$$u_{ik} + d_j x_{ijk} - Q_k(1 - x_{ijk}) \leq u_{jk} \qquad \forall i,j \in N, k \in K \tag{4}$$

$$\sum_{k \in K} \sum_{i \in N} x_{i0k} \leq |K| \tag{5}$$

$$\sum_{k \in K} \sum_{j \in N} x_{0jk} \leq |K| \tag{6}$$

$$d_i \leq u_{ik} \leq Q_k \qquad \forall i \in N, k \in K \tag{7}$$

$$x_{ijk} \in \{0,1\} \qquad \forall i,j \in V, k \in K \tag{8}$$

$$u_{ik} \geq 0 \qquad \forall i \in N, k \in K \tag{9}$$

Retomando el ejemplo de la Figura 1.1 si consideramos una flota de 3 vehículos con capacidades $Q_1 = 130$, $Q_2 = 120$ y $Q_3 = 50$ una posible solución sería la que muestra la Figura 1.4. Nótese que a diferencia de la Figura 1.3 no es preciso utilizar toda la flota lo cual, aunque no se indique en la función objetivo, conllevaría un ahorro en los costes operacionales.

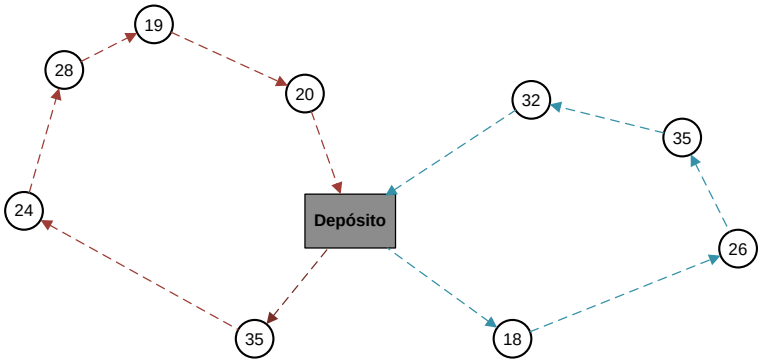


Figura 1.4: Posible solución al ejemplo de la Figura 1.1 considerando una flota heterogénea.

A la vista de estos modelos, hemos comprobado que la formulación del problema puede determinar su eficacia en términos computacionales y cómo extender modelos a situaciones más generales haciendo pequeñas modificaciones.

1.1.2. Técnicas de resolución

Desde que Dantzig y Ramser (1959) plantearon y resolvieron por primera vez un VRP con 12 clientes las técnicas de resolución han aumentado y evolucionado exponencialmente. En particular, hasta finales del siglo pasado los métodos exactos y heurísticos se desarrollaron de forma paralela. Desde principios de los 90 hasta nuestros días se han desarrollado muchos algoritmos basados en técnicas metaheurísticas. La Tabla 1.1 resume las técnicas de resolución para los problemas de rutas.

Métodos exactos

Algunos métodos exactos se basan en la resolución de un problema de programación lineal. Por tanto, su eficacia depende del modelado del problema. Por ejemplo, un VRP con flota homogénea puede plantearse como un modelo basado en redes (modelo (VRP1)) o usando la formulación de 2 índices propuesta en Miller, Tucker y Zemlin (1960) (modelo(VRP2)). La principal diferencia entre estos modelos reside en la restricción de rotura de ciclos. En los modelos basados en redes el número de restricciones de este tipo aumenta exponencialmente mientras que con la formulación de Miller-Tucker-Zemlin el aumento es polinómico aunque precisa de introducir nuevas variables en el problema. En la práctica esta segunda formulación ofrece mejores resultados computacionales.

Por otro lado, los algoritmos de partición de conjuntos y generación de columnas (Agarwal, Mathur y Salkin (1989)) han demostrado ser de gran utilidad para resolver el VRP y sus variantes. Este tipo de algoritmos son de gran utilidad en problemas de programación lineal con un número elevado de variables binarias. Partiendo de la premisa de que muchas de las variables no formarán parte de la solución óptima, no es necesario trabajar con todas las variables para encontrar una solución óptima. Por ejemplo, en un CVRP muchas de las rutas no son factibles por superar la capacidad del vehículo. En líneas generales, se divide el problema en dos subproblemas: el problema maestro en el que se obtiene una solución en la que todos los clientes sean visitados y un subproblema cuyo objetivo es identificar la ruta que reduzca más el coste total. En la actualidad, este tipo de algoritmos siguen siendo de gran utilidad para resolver problemas más generales combinándolos con técnicas más sofisticadas.

En general, los VRP suelen modelarse como problemas de programación lineal entera mixta en los que las variables de decisión involucradas son enteras o binarias. Otros métodos exactos de resolución comienzan obteniendo una solución del problema relajado, esto es, suponiendo que las variables son continuas y, a continuación, refinan esta solución hasta que sea óptima del problema original. Estos algoritmos se conocen como búsqueda directa de árbol ya que el problema se representa como un árbol en el que cada rama se identifica con los posibles valores que toma una variable no entera. Los principales algoritmos de este grupo son los de ramificación y acotación, y, ramificación y corte introducidos para el VRP por Christofides, Mingozzi y Toth (1981) y Araque et al. (1994), respectivamente.

Algoritmos heurísticos

Las técnicas heurísticas se caracterizan por obtener soluciones, sin garantía de optimalidad, en poco tiempo de ejecución. Suelen ser algoritmos de fácil implementación y adaptables tanto a generalizaciones del VRP como a otros problemas. Los algoritmos heurísticos para el VRP y sus variantes se desarrollaron de forma paralela a los algoritmos exactos.

A diferencia de los algoritmos aproximados, para los cuales es posible encontrar una cota de error con respecto a la solución óptima, para los algoritmos heurísticos no es posible determinar dicha cota.

Los algoritmos heurísticos para el VRP pueden clasificarse en tres grandes grupos:

- **Heurísticas constructivas:** Construyen una solución iterativamente añadiendo en cada itera-

ción el elemento con mayor valor según un criterio establecido.

Dentro de este grupo destaca el algoritmo de ahorros propuesto por Clarke y Wright (1964) en el cual se parte de que cada cliente es una ruta para, a continuación, ir uniendo los pares de rutas que conllevan un ahorro máximo.

Otros algoritmos constructivos son los de inserción en los que en cada iteración los clientes no visitados se colocan en la ruta en la que producen menor coste. Para ello, el algoritmo debe evaluar en cada iteración todas las posibles ubicaciones de un nuevo cliente. La inserción puede ser secuencial (Mole y Jameson (1976)) en el que se van añadiendo clientes a una ruta hasta que ésta se completa o en paralelo (Christofides, Mingozzi y Toth (1979)) en la que, en cada iteración, cada cliente es insertado en la ruta que más le conviene.

Los algoritmos constructivos se caracterizan por no tener una fase de mejora de la solución, es decir, una vez construidas las rutas no hay reordenamiento de los clientes. Estos algoritmos destacan por tener un bajo coste computacional aunque en la práctica ofrecen soluciones que podrían estar lejos del óptimo, en especial para problemas de gran tamaño. Por tanto, suelen usarse para obtener una solución de partida que será mejorada por algoritmos que sí incluyen una fase de mejora.

- **Heurísticas de dos fases:** Dividen el problema en dos subproblemas: asignación de clientes a una misma ruta y organización de las rutas.

Este apartado engloba una gran cantidad de algoritmos ya que algunos primero agrupan los clientes en distintas rutas para luego optimizarlas individualmente mientras que otros algoritmos parten de una ruta general que incluye a todos los clientes para luego descomponer esta ruta en otras más pequeñas.

Dentro del primer grupo se encuentran los algoritmos de barrido introducidos por Gillet y Miller (1974). Este algoritmo considera la ubicación de los clientes en coordenadas polares (r, θ) tomando como origen el depósito. A continuación, los clientes se van agrupando en función del ángulo θ hasta completar la capacidad de un vehículo. Una vez creadas todas las rutas, los clientes se conectan teniendo en cuenta que el punto de partida y de finalización es el depósito.

Beasley (1983) fue el primero en diseñar un algoritmo heurístico que primero diseña las rutas y luego asigna los clientes. Para ello parte de una única ruta que engloba a todos los clientes, resolviendo el problema como un TSP y, a continuación, los clientes se agrupan en sub-rutas teniendo en cuenta las distancias entre ellos y la capacidad de los vehículos.

- **Heurísticas de mejora:** Parten de una solución inicial factible y tratan de mejorar la solución introduciendo pequeñas perturbaciones.

Las heurísticas de mejora para el VRP están diseñadas para mejorar una solución inicial factible mediante un mecanismo de búsqueda. Estas heurísticas pueden clasificarse en intra-rutas, si solo se utiliza una única ruta para realizar la mejora, o entre-rutas si están involucradas varias rutas

Los métodos intra-rutas se centran individualmente en cada una de las rutas intentando optimizar la secuencia de esa ruta intercambiando el orden de los clientes. Los movimientos aplicados a cada ruta están inspirados en el TSP. Dentro de esta categoría podemos distinguir los métodos exactos (Tucker-Miller-Zemlin) o métodos heurísticos como los algoritmos $k-$opt y Or-opt.

Los intercambios $k-$opt consisten en eliminar $k > 1$ arcos de una ruta y evaluar todas las posibles inserciones de los arcos eliminados. Se puede demostrar que el coste computacional de calcular y

evaluar todas las posibles soluciones es de orden $\mathcal{O}(n^k)$ por ello, en la práctica se suele escoger $k = 2$ como propuso Croes (1958) o $k = 3$ propuesto por Lin (1965). Actualmente, el algoritmo que proporciona mejores resultados es el propuesto por Lin y Kernighan (1973) que en cada iteración alterna movimientos $2-$opt y $3-$opt.

Por otro lado, los algoritmos Or-opt introducidos por Or (1976) seleccionan un segmento de k nodos ($k \leq 3$) que será recolocado en ese mismo orden en otra parte de la ruta.

Los métodos entre-rutas involucran varias rutas para realizar una mejora. En este grupo, el primer trabajo se debe a Thompson y Psaraftis (1993) quienes definieron los $b-$ciclos de $k-$transferencia de forma que en cada iteración se selecciona una permutación circular de b rutas en las que se intercambian k clientes.

Algoritmos metaheurísticos

Las metaheurísticas son el desarrollo más reciente dentro de los algoritmos para resolver problemas complejos de optimización combinatoria. Se han desarrollado y extendido rápidamente desde comienzos de los años 80 para dar respuesta a una gran variedad de problemas. Dichas técnicas surgen por la necesidad de resolver aquellos problemas donde los métodos exactos y los heurísticos fallan en términos de eficiencia y eficacia. La definición de metaheurística enunciada en Osman y Kelly (1997) es la siguiente:

"Una metaheurística es un procedimiento iterativo que guía un método heurístico subordinado combinando inteligentemente diversos conceptos para explorar y explotar los espacios de búsqueda mediante estrategias aprendidas para conseguir soluciones casi-óptimas de manera eficiente".

En líneas generales, un algoritmo metaheurístico parte de una solución inicial y de un conjunto de reglas para generar nuevas soluciones. Por tanto, en cada iteración t del algoritmo se genera una vecindad $N(x_t)$ de la solución x_t. Las soluciones obtenidas son evaluadas según una función de coste, f que depende del problema estudiado. Además, los algoritmos metaheurísticos incluyen una fase de comparación de soluciones para decidir desde dónde seguir explorando el espacio de soluciones. Otra diferencia importante entre los algoritmos heurísticos y metaheurísticos es que estos últimos permiten la elección de soluciones peores o no factibles durante la exploración del espacio de soluciones. De esta forma se evita caer en óptimos locales.

Algunos de los algoritmos metaheurísticos más utilizados en la práctica son:

Recocido simulado:

Este algoritmo propuesto por Kirkpatrick, Gelatt y Vecchi (1983) debe su nombre a la analogía con el proceso físico de recocido al que se someten los sólidos para obtener estados de mínima entropía. Este algoritmo acepta las soluciones de peor calidad con una probabilidad $p = e^{\frac{-\Delta z}{T_k}}$ siendo Δz la distancia entre las soluciones y T_k la *temperatura* en la iteración k. Esta temperatura suele ser una función decreciente en k de forma que el algoritmo se *enfría* a lo largo de las iteraciones reduciendo la probabilidad p.

Búsqueda tabú:

Introducidos por Glover (1986) este tipo de algoritmos también permiten la aceptación de soluciones de peor calidad para explorar el espacio de soluciones. Además, incluyen una *lista tabú* que prohíbe visitar las soluciones calculadas recientementes para evitar caer movimientos cíclicos en el proceso de búsqueda.

Métodos exactos	Programación lineal y entera	Formulación de flujo de vehículos de 2 o 3 índices
		Partición de conjuntos y generación de columnas
	Búsqueda directa de árbol	Algoritmos de ramificación y acotación
		Algoritmos de ramificación y corte
Heurísticas	Métodos constructivos	Algoritmo de los ahorros
		Algoritmos de inserción
	Métodos de dos fases	Métodos de asignar pimero y rutear después
		Métodos de rutear pimero y asignar después
	Heurísticas de mejora	Mejoras ruta simple
		Mejoras multirruta
Metaheurísticas	Algoritmos genéticos	
	Recocido simulado	
	Búsqueda tabú	

Tabla 1.1: Resumen de los principales métodos de resolución en VRP.

1.2. Juegos Cooperativos

Dentro del campo de la teoría de juegos, los juegos cooperativos son aquellos en los que los jugadores disponen de mecanismos que les permiten tomar acuerdos vinculantes. El principal problema de estos juegos es cómo efectuar el reparto de los beneficios (o costes) que se generan con la cooperación de algunos (o todos) los jugadores. Si suponemos que los beneficios generados por las coaliciones pueden repartirse libremente estamos ante problemas de negociación coalicional con utilidad transferible, o,

en otras palabras, un juego cooperativo con utilidad transferible (abreviadamente juegos TU) que definimos formalmente a continuación.

Los juegos cooperativos han demostrado ser de gran utilidad para modelar diversas situaciones reales. González y Herrero (2004) definen un juego de coste asociado al uso de quirófanos compartidos por varias especialidades teniendo en cuenta el modelo de colas subyacente. Fiestras-Janeiro et al. (2014) proponen un juego cooperativo para repartir los costes generados cuando varios ganaderos comparten silos de almacenamiento. Para más aplicaciones de los juegos cooperativos puede consultarse Fiestras-Janeiro, García-Jurado y Mosquera (2011).

1.2.1. Conceptos básicos

Definición 1.2.1. *Un juego cooperativo con utilidad transferible (juego TU) es un par (P, f) donde $P = \{1, 2, \ldots, p\}$ representa el conjunto de jugadores y $f : \mathcal{P}(P) \to \mathbb{R}$, denominada función característica*[2]*, es una función verificando $f(\emptyset) = 0$.*

Dada una coalición $S \subset P$, interpretamos $f(S)$, como el beneficio que se pueden asegurar los jugadores de S, independientemente de cómo actúe el resto de los jugadores. Introducimos los siguientes ejemplos para ilustrar el concepto de juego TU.

Ejemplo 1.2.1. *(Reparto de un millón). Un hombre pretende dejar una herencia de un millón de euros a tres herederos, con la condición de que al menos dos de ellos lleguen a un acuerdo sobre cómo efectuar el reparto; si tal acuerdo no es alcanzado, el millón se donará a una institución benéfica. Esta situación puede ser representada como un juego TU tal que $P = \{1, 2, 3\}$ representa a cada uno de los herederos y $f(1) = f(2) = f(3) = 0$ y $f(1, 2) = f(2, 3) = f(1, 3) = f(P) = 1$ representaría el beneficio asociado a cada posible acuerdo entre ellos.*

Ejemplo 1.2.2. *(Juego del guante). Supongamos que reunimos a 3 personas de las cuales dos de ellas poseen un guante izquierdo y la otra un guante derecho. Si el objetivo del juego es repartir los beneficios derivados de la venta de un par de guantes, podríamos modelar esta situación como un juego TU (P, f) con $P = \{1, 2, 3\}$ y $f(1) = f(2) = f(3) = f(1, 2) = 0$ y $f(1, 3) = f(2, 3) = f(P) = 1$.*

Ejemplo 1.2.3. *(El profesor visitante). Tres grupos de investigación pertenecientes a las universidades de Milán (grupo 1), Génova (grupo 2) y Santiago de Compostela (grupo 3), planean invitar a un profesor japonés para impartir un curso de teoría de juegos. Para minimizar los costes, los grupos se organizan de forma que el profesor visite las tres ciudades en un único viaje y planean repartirse los costes generados. Tomando $P = \{1, 2, 3\}$ como el conjunto de ciudades y teniendo en cuenta que el coste mínimo asociado a cada una de las posibles combinaciones de viaje viene dado por: $c(1) = 1500$, $c(2) = 1600$, $c(3) = 1900$, $c(1, 2) = 1600$, $c(1, 3) = 2900$, $c(2, 3) = 3000$, $c(P) = 3000$ se tiene que (P, c) representa un juego TU.*

El juego del Ejemplo 1.2.3 suele denominarse juego de coste ya que para cada coalición S, $c(S)$ no representa los beneficios que S puede generar, sino el gasto que genera cada coalición. A cada juego de coste (P, c) se le puede asociar un juego de ahorro (P, f) donde, para cada $S \subset P$, $f(S) = \sum_{i \in S} c(i) - c(S)$ representaría lo que cada coalición se ahorra si decide cooperar.

El juego de ahorro asociado vendría dado por $f(1) = f(2) = f(3) = 0$, $f(1, 2) = 1500$, $f(1, 3) = f(2, 3) = 500$ y $f(P) = 2000$. Vemos que combinando cualquier subconjunto de dos o más grupos de investigación el beneficio total es mayor que la suma de los beneficios por separado, por tanto, los

[2] $\mathcal{P}(P) = \{S / S \subset P\}$ representa el conjunto de todos los subconjuntos de P.

grupos tendrán incentivos reales para coordinarse y planear una visita conjunta del profesor a las tres ciudades. Estos juegos son de especial interés y los denominaremos juegos superaditivos.

Definición 1.2.2. *Dado (P, f) un juego TU diremos que es superaditivo si para cada par de coaliciones $S, T \subset P$ disjuntas se verifica que $f(S \cup T) \geq f(S) + f(T)$.*

Puede comprobarse que los Ejemplos 1.2.1 y 1.2.2 son superaditivos, pero el Ejemplo 1.2.3 como juego de costes no lo es.

Una vez tenemos modelada una situación mediante un juego cooperativo el principal problema reside en buscar repartos (de los gastos o beneficios generados) que puedan ser aceptados por todos los jugadores involucrados.

Definición 1.2.3. *Dado (P, f) un juego TU, una solución (o reparto) es un vector $x = (x_1, \dots, x_p)$ en el que cada componente x_i representa el beneficio asignado al jugador $i \in P$.*

Esta tarea no es fácil ya que hay muchos factores externos como la capacidad de negociación de los jugadores o presiones de tipo social que dificultan la existencia de una regla general de reparto. En la literatura, podemos encontrar soluciones de tipo conjunto que plantean una serie de repartos posibles o las soluciones de tipo puntual que se reducen a un único reparto.

Intuitivamente, un reparto en el que ningún jugador (o coalición de jugadores) recibe menos de lo que puede asegurarse por su cuenta podría ser aceptado por todos los jugadores. Los repartos que cumplen esta propiedad constituyen el núcleo del juego, introducido por Gillies (1953). No obstante, la existencia del núcleo no está garantizada para todos los juegos, y, en algunos casos, no se reduce a un único punto.

Otro tipo de soluciones son las denominadas de tipo puntual, en las que se propone un único reparto de los beneficios (o costes) obtenidos por los jugadores atendiendo a una serie de propiedades. Dentro de este grupo, una de las reglas más utilizadas y extendidas a distintos tipos de juegos es el valor de Shapley (Shapley, 1953) el cual estudiaremos en este trabajo.

Definición 1.2.4. *Dado (P, f) un juego TU, se define el valor de Shapley de f, como el vector $\Phi(f) = (\Phi_1(f), \dots, \Phi_p(f))$ que asigna a cada jugador i el siguiente pago:*

$$\Phi_i(f) = \sum_{S \subset P \setminus \{i\}} \frac{|S|!(|P| - |S| - 1)!}{|P|!} \big(f(S \cup \{i\}) - f(S)\big).$$

Nótese que el valor de Shapley de un jugador i puede verse como una suma ponderada de lo que cada jugador i aporta al unirse a una coalición S de la que no formaba parte, $f(S \cup \{i\}) - f(S)$. Esta contribución se denomina contribución marginal del jugador i a la coalición S. A partir de esta idea, podemos dar una expresión equivalente del valor de Shapley teniendo en cuenta las siguientes consideraciones:

I) Todos los jugadores deciden cooperar y reunirse para realizar la negociación.

II) Todos los posibles órdenes de llegada a la reunión son equiprobables.

III) A su llegada, cada jugador recibe como pago su contribución marginal a la coalición previamente formada.

Entonces, el valor de Shapley puede calcularse para cada jugador $i \in P$ como:

$$\Phi_i(f) = \frac{1}{p!} \sum_{\pi \in \Pi(P)} \Big(f\big(B_\pi(i) \cup \{i\}\big) - f\big(B_\pi(i)\big) \Big),$$

donde $\Pi(P)$ representa el conjunto de permutaciones de P y $B_\pi(i) = \{j \in P / \pi(j) < \pi(i)\}$ el conjunto de jugadores que aparecen antes que i en la permutación $\pi \in \Pi(P)$. Con esta definición, podemos interpretar el valor de Shapley como la media de las contribuciones marginales de los jugadores a cada uno de los posibles órdenes de llegada $\pi \in \Pi(P)$.

En la Tabla 1.2 se calculan los denominados vectores de contribuciones marginales para los distintos órdenes de llegada de los jugadores para el Ejemplo 1.2.1. Finalmente, promediando por el número total de órdenes de llegada posible, se obtiene el valor de Shapley.

$\Pi(P)$ \ P	1	2	3
123	0	1	0
132	0	0	1
213	1	0	0
231	0	0	1
312	1	0	1
321	0	1	0
$\Phi(f)$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$

Tabla 1.2: Conjunto de permutaciones y vectores de contribuciones marginales en el Ejemplo 1.2.1.

Nótese que $\sum_{i \in P} \Phi_i(f) = \frac{1}{3} + \frac{1}{3} + \frac{1}{3} = f(P)$ de forma que los beneficios totales se reparten entre todos los jugadores. Puede probarse que para cualquier juego (P, f) se verifica que $\sum_{i \in P} \Phi_i(f) = f(P)$ por lo que decimos que el valor de Shapley es una regla de reparto *eficiente*.

Independientemente de la expresión utilizada, el valor de Shapley tiene complejidad computacional de orden $\mathcal{O}(2^p)$ lo cual dificulta su cálculo cuando el número de jugadores es grande.

1.2.2. Aproximación del valor de Shapley

En esta sección, presentamos el método propuesto por Castro, Gómez y Tejada (2009) que permite aproximar el valor de Shapley en tiempo polinomial y analizaremos su comportamiento en algunos ejemplos.

Como hemos visto anteriormente, el valor de Shapley puede interpretarse como la media de las contribuciones marginales de cada jugador a cada uno de los posibles órdenes del conjunto de jugadores P. No obstante, hay $p!$ órdenes posibles lo que dificulta la tarea cuando el número de jugadores es elevado. El algoritmo expuesto en el Cuadro 1.3 propone trabajar con una muestra (tomada con reemplazamiento) de todos los posibles órdenes.

Para estimar el valor de Shapley, seguiremos el procedimiento estándar usado en inferencia estadística. A partir de una muestra aleatoria simple de todos los posibles órdenes de llegada $\Pi(P)$ calcularemos, para cada jugador $i \in P$ un estadístico muestral $\hat{\Phi}_i(f)$ para estimar $\Phi_i(f)$. El procedimiento consiste en los siguientes pasos:

Paso 1:

Construir una muestra $M = \{\pi_1, \ldots, \pi_m\}$ de m posibles órdenes de $\Pi(P)$ seleccionados con probabilidad $1/p!$.

Paso 2:

Calcular, para cada jugador $i \in P$, su contribución marginal $x_\pi(i) = f\big(B_\pi(i) \cup \{i\}\big) - f\big(B_\pi(i)\big)$ a cada orden $\pi \in M$

Paso 3:

Calcular, para cada jugador $i \in P$, el estadístico muestral $\hat{\Phi}_i(f) = \dfrac{1}{m} \sum\limits_{\pi \in M} x_\pi(i)$.

Cuadro 1.3: Procedimiento para calcular el valor de Shapley approximado.

A continuación, estudiaremos las propiedades teóricas de $\hat{\Phi}_i(f)$ como estimador del parámetro real $\Phi_i(f)$.

Teorema 1.2.1. *Para cada jugador $i \in P$ el estimador $\hat{\Phi}_i(f)$ es insesgado y y su varianza viene dada por $Var(\hat{\Phi}_i(f)) = \dfrac{\sigma^2}{m}$ siendo $\sigma^2 = \dfrac{1}{p!} \sum\limits_{\pi \in \Pi(P)} (x_\pi(i) - \Phi_i(f))^2$.*

Demostración. En primer lugar probaremos que $\hat{\Phi}_i(f)$ es un estimador insesgado de $\Phi_i(f)$.

$$\mathbb{E}(\hat{\Phi}_i(f)) = \mathbb{E}\Big(\frac{1}{m}\sum_{\pi\in M} x_\pi(i)\Big) = \frac{1}{m}\mathbb{E}\Big(\sum_{\pi\in M} x_\pi(i)\Big) = \frac{1}{m}\sum_{\pi\in M}\mathbb{E}(x_\pi(i)) = \frac{1}{m}\sum_{\pi\in M}\Phi_i(f) = \frac{1}{m}m\Phi_i(f) = \Phi_i(f).$$

Por otro lado, para la varianza se tiene:

$$\begin{aligned}
Var(\hat{\Phi}_i(f)) &= \mathbb{E}(\hat{\Phi}_i(f)^2) - \Big(\mathbb{E}(\hat{\Phi}_i(f))\Big)^2 \\
&= \mathbb{E}\left(\Big(\frac{1}{m}\sum_{\pi\in M} x_\pi(i)\Big)^2\right) - \Phi_i(f)^2 \\
&= \mathbb{E}\left(\frac{1}{m^2}\Big(\sum_{\pi\in M} x_\pi(i)\Big)^2\right) - \Phi_i(f)^2 \\
&= \mathbb{E}\left(\frac{1}{m^2}\Big(\sum_{\pi\in M} [x_\pi(i)]^2 + 2\sum_{\substack{\pi,\pi'\in M\\ \pi\neq\pi'}} x_\pi(i)x_{\pi'}(i)\Big)\right) - \Phi_i(f)^2 \\
&= \frac{1}{m^2}\mathbb{E}\left(\sum_{\pi\in M} [x_\pi(i)]^2 + 2\sum_{\substack{\pi,\pi'\in M\\ \pi\neq\pi'}} x_\pi(i)x_{\pi'}(i)\right) - \Phi_i(f)^2 \\
&= \frac{1}{m^2}\left(\mathbb{E}\Big(\sum_{\pi\in M} [x_\pi(i)]^2\Big) + 2\mathbb{E}\Big(\sum_{\substack{\pi,\pi'\in M\\ \pi\neq\pi'}} x_\pi(i)x_{\pi'}(i))\Big)\right) - \Phi_i(f)^2 \\
&= \frac{1}{m^2}\left(\Big(\sum_{\pi\in M} \mathbb{E}([x_\pi(i)]^2) + 2\sum_{\substack{\pi,\pi'\in M\\ \pi\neq\pi'}} \mathbb{E}(x_\pi(i))\mathbb{E}(x_{\pi'}(i))\Big)\right) - \Phi_i(f)^2 \\
&= \frac{1}{m^2}\Big(m\mathbb{E}([x_\pi(i)]^2) + 2\big(\frac{m^2-m}{2}\big)\Phi_i(f)^2\Big) - \Phi_i(f)^2 \\
&= \frac{1}{m}\mathbb{E}([x_\pi(i)]^2) + \Big(1-\frac{1}{m}\Big)\Phi_i(f)^2 - \Phi_i(f)^2 \\
&= \frac{1}{m}\mathbb{E}([x_\pi(i)]^2) - \frac{1}{m}\Phi_i(f)^2 \\
&= \frac{1}{m}\Big(\mathbb{E}([x_\pi(i)]^2) - \Phi_i(f)^2\Big) \\
&= \frac{1}{m}\Big(Var(x_\pi(i))\Big) \\
&= \frac{1}{m}\Big(\sum_{\pi\in\Pi(P)} \frac{1}{p!}\big(x_\pi(i) - \Phi_i(f)\big)^2\Big) \\
&= \frac{\sigma^2}{m}.
\end{aligned}$$

□

Como vemos, del hecho de que el estimador sea insesgado y que la varianza se aproxime a cero a medida que aumenta el tamaño muestral, m, se deduce que el estimador $\hat{\Phi}_i(f)$ es consistente en probabilidad. Además, al aumentar el tamaño muestral la varianza del estimador se aproxima a 0 reduciendo así su variabilidad. No obstante, en la práctica se aumenta el tiempo computacional.

A continuación, estudiaremos la bondad del estimador empíricamente comparando las soluciones exactas y aproximadas en los Ejemplos 1.2.4, 1.2.5 y 1.2.6. Consideraremos varios escenarios al variar el número de jugadores ($p = 6, 8, 10$) y el número de remuestras ($m = 10^2, 10^3, \ldots, 10^7$). Para cada escenario calcularemos el error máximo, $e_{\max} = \max_{i \in P} |\Phi_i(f) - \hat{\Phi}_i(f)|$; el error medio, $\bar{e} = \sum_{i \in P} \dfrac{|\Phi_i(f) - \hat{\Phi}_i(f)|}{p}$, y el tiempo computacional necesario para obtener cada solución. El objetivo es encontrar un número de remuestras que nos permita obtener buenas soluciones en poco tiempo.

El procedimiento para calcular el valor de Shapley descrito en está sección ha sido implementado en `R` y puede consultarse en el Anexo B.1.

Ejemplo 1.2.4. *(Reparto de un millón generalizado). Consideramos que un único heredero tiene que repartir un millón de euros entre p herederos de forma que, al menos, la mitad de ellos tienen que ponerse de acuerdo para que se efectúe el reparto. Así, para una coalición $S \subset P$ se tiene que:*

$$f(S) = \begin{cases} 1, & si\ |S| > \lceil \frac{p}{2} \rceil; \\ 0, & en\ caso\ contrario. \end{cases}$$

Puede probarse que para cualquier jugador $i \in P$ el valor de Shapley viene dado por $\Phi_i(f) = \frac{1}{p}$.

Ejemplo 1.2.5. *(Juego del guante generalizado). Partimos de un conjunto de jugadores en el que cada jugador posee un único guante. De esta forma, podemos escribir el conjunto de jugadores como $P = L \cup R$ siendo L el conjunto de personas con un guante izquierdo y R el conjunto de personas con un guante derecho. Así, toda coalición $S \subset P$ puede ser escrita como $S = L_S \cup R_S$ con $L_S \subset L, R_S \subset R$ y puede deducirse que $f(S) = min\{|L_S|, |R_S|\}$. Si suponemos que $|L| = |R| = \frac{p}{2}$ puede demostrarse que $\Phi_i(f) = \frac{1}{2}$, para cualquier jugador $i \in P$.*

Ejemplo 1.2.6. *(El profesor visitante generalizado). Supongamos que el profesor tiene que visitar un conjunto de universidades $P = E \cup A$ con E el conjunto de universidades en Europa y A el conjunto de universidades en América. Si por cada universidad visitada en Europa el coste es de 500 u.m., por cada universidad en América de 700 u.m. y hay un coste fijo de 1000 u.m. para trámites legales entonces la función de costes vendrá dada por $c(S) = 1000 + 500|E \cap S| + 700|A \cap S|$.*

El valor de Shapley del juego de costes vendrá dado por: $\Phi_i(c) = \begin{cases} x, & si\ i \in E; \\ x + 200, & si\ i \in A. \end{cases}$

Para calcular x, puede hacerse uso de la eficiencia del valor de Shapley por lo que $\sum_{i \in P} \Phi_i(c) = x|E| + (x + 200)|A| = c(P)$.

Los resultados obtenidos para cada ejemplo se muestran, respectivamente, en las Tablas 1.4, 1.5 y 1.6. En la Figuras 1.5 y 1.6 vemos como varían el error máximo y medio en cada ejemplo. Para ambos errores, e independientemente del número de jugadores, tomando $m \geq 10^5$ el error es próximo a 0. Por otro lado, observando como varía el tiempo de cómputo (Figura 1.7) descartamos la elección de $m = 10^7$ por ser éste muy elevado.

Podemos concluir que para obtener soluciones (casi)óptimas en poco tiempo computacional es suficiente escoger $m = 10^5$ o $m = 10^6$ como número de remuestras.

Número de jugadores (p)		Número de remuestras (m)					
		100	1000	10000	100000	1000000	10000000
6	$e_{\max}$	0.05667	0.01367	0.0056	0.0007	0.00059	0.00025
	$\bar{e}$	0.02778	0.00567	0.00294	0.00038	0.00032	0.00011
	Tiempo (s)	0.03	0.16	1.66	15.81	154.35	1553.44
8	$e_{\max}$	0.055	0.014	0.0064	0.0014	0.00036	0.00014
	$\bar{e}$	0.025	0.00775	0.00292	0.00072	0.00015	0.00008
	Tiempo (s)	0.01	0.33	2.28	22.88	228.69	2237.46
10	$e_{\max}$	0.04	0.027	0.0036	0.0017	0.00059	0.00021
	$\bar{e}$	0.02	0.0078	0.00136	0.00096	0.00031	0.00007
	Tiempo (s)	0.03	0.35	3.41	32.98	329.17	3276.95

Tabla 1.4: Resultados obtenidos para el juego del millón generalizado.

Número de jugadores (p)		Número de remuestras (m)					
		100	1000	10000	100000	1000000	10000000
6	$e_{\max}$	0.1	0.02	0.012	0.002	0.00074	0.00032
	$\bar{e}$	0.02333	0.00633	0.00247	0.001	0.00023	0.00008
	Tiempo (s)	0.02	0.14	1.56	15.44	152.99	1564.89
8	$e_{\max}$	0.08	0.023	0.0094	0.00226	0.00107	0.00048
	$\bar{e}$	0.03	0.00875	0.0036	0.00074	0.00047	0.00019
	Tiempo (s)	0.03	0.24	2.38	23.14	229.53	2291.02
10	$e_{\max}$	0.07	0.014	0.0065	0.00187	0.00053	0.00014
	$\bar{e}$	0.08	0.0094	0.00484	0.00069	0.00035	0.00019
	Tiempo (s)	0.04	0.34	3.52	33.94	342.72	3348.42

Tabla 1.5: Resultados obtenidos para el juego del guante generalizado.

Número de jugadores (p)		Número de remuestras (m)					
		100	1000	10000	100000	1000000	10000000
6	$e_{\max}$	66.67	13.67	6.13	1.313	0.4013	0.17547
	$\bar{e}$	36.66	86.67	3.0	0.77778	0.273	0.06858
	Tiempo (s)	0.01	0.14	1.7	15.55	153.57	1536.2
8	$e_{\max}$	45	21	8.6	2.51	0.709	0.1869
	$\bar{e}$	23.75	7.25	3.6	1.1225	0.309	0.08963
	Tiempo (s)	0.01	0.22	2.27	22.45	225.09	2237.08
10	$e_{\max}$	60	17	4.7	1.78	0.613	0.1715
	$\bar{e}$	22	5.8	2.04	0.77	0.2182	0.11492
	Tiempo (s)	0.04	0.32	3.25	32.64	325.72	3235.77

Tabla 1.6: Resultados obtenidos para el juego del profesor visitante generalizado.

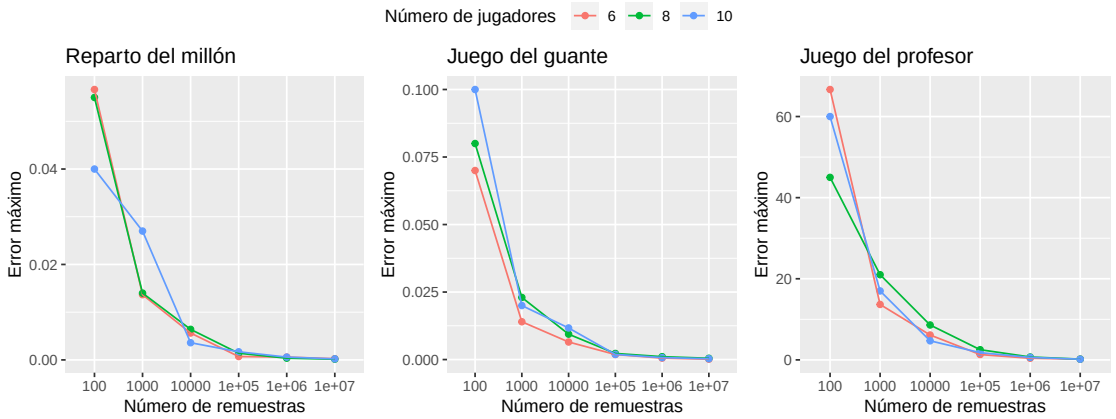


Figura 1.5: Comportamiento del error máximo para distintos tamaños de remuestra.

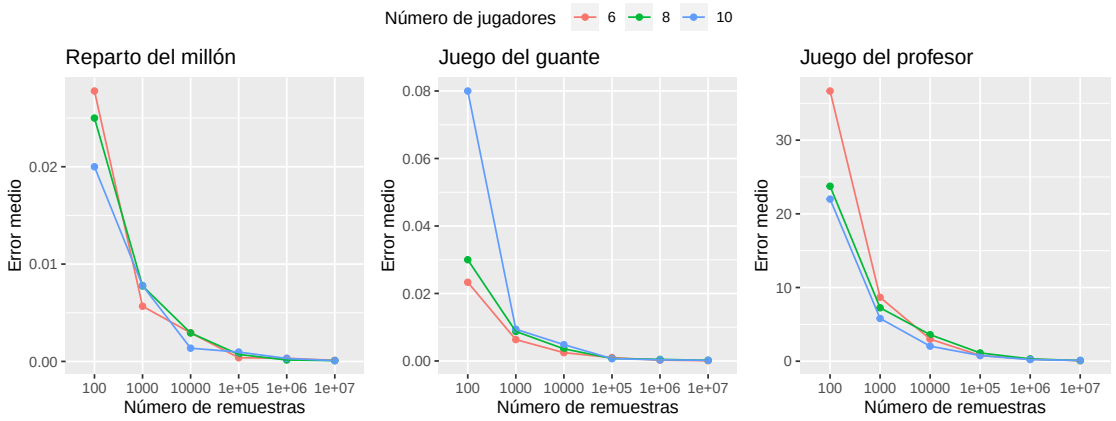


Figura 1.6: Comportamiento del error medio para distintos tamaños de remuestra.

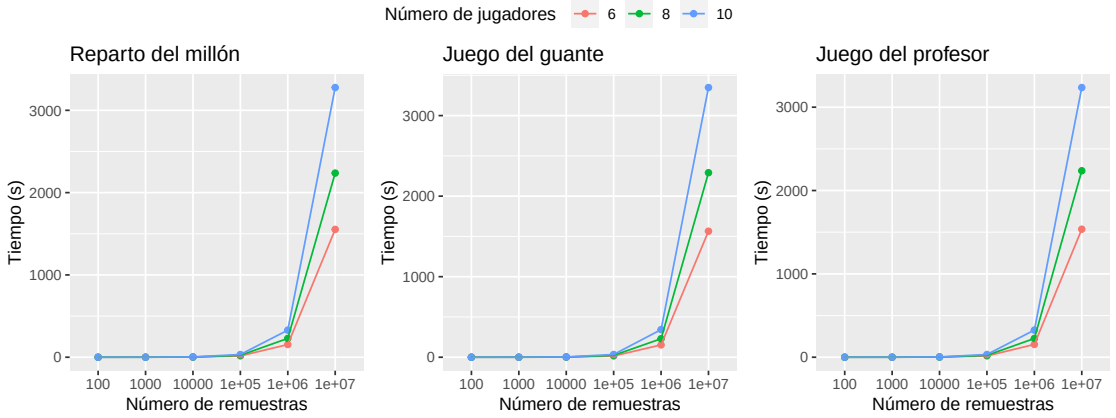


Figura 1.7: Comportamiento del tiempo computacional para distintos tamaños de remuestra.

Capítulo 2

Cooperación en problemas de rutas de vehículos con múltiples depósitos

En este capítulo, introduciremos los problemas de rutas de vehículos con múltiples depósitos (MDVRP) y la situación en la que los dueños de los depósitos deciden cooperar (CoMDVRP).

Este tipo de problemas, son una generalización del VRP presentado en la sección anterior suponiendo que existen varios depósitos desde los que se puede repartir (recibir) un producto. Si suponemos que los depósitos pertenecen a distintas empresas vale la pena estudiar el efecto de una posible cooperación entre ellas. Para eso, plantearemos un juego TU representando la situación y compararemos las soluciones en caso de que las empresas decidan cooperar o no.

Finalizamos el capítulo presentando un algoritmo heurístico para resolver el MDVRP.

2.1. Problemas de rutas de vehículos con múltiples depósitos (MDVRP)

En el presente trabajo nos centraremos en el MDVRP introducido por Kulkarni y Bhave (1985) y cuyas primeras soluciones fueron propuestas por Laporte, Nobert y Taillefer (1988). En este último artículo los autores proponen un algoritmo de ramificación y acotación inspirándose en el trabajo de Carpaneto et al. (1989). Una revisión completa de los modelos MDVRP y sus generalizaciones puede encontrarse en Montoya et al. (2015).

La principal diferencia entre el VRP y el MDVRP es la existencia de varios depósitos desde los que la empresa puede repartir sus pedidos. Además, supondremos que los de vehículos pueden ser utilizados por cualquiera de los depósitos. El número de vehículos disponibles y sus capacidades son conocidos de antemano.

Cabe destacar que si cada depósito tuviera su flota particular y los clientes estuvieran asignados previamente a un depósito concreto el problema podría resolverse mediante VRPs independientes para cada depósito.

En la Tabla 2.1 presentamos los elementos necesarios para formular un MDVRP basándonos en el artículo de Zibaei, Hafezalkotob y Ghashami (2016).

Conjuntos	
I	Conjunto de depósitos.
J	Conjunto de clientes.
K	Conjunto de vehículos.
$V = I \cup J$	Conjunto de nodos
Constantes y parámetros	
d	Número total de depósitos.
n	Número total de clientes.
$\|K\|$	Número total de vehículos.
c_{ij}	Coste[1] de viajar del nodo i al nodo j; $i, j \in V$.
A_i	Cantidad almacenada del producto en el depósito $i \in I$.
d_j	Demanda del cliente $j \in J$.
Q_k	Capacidad del vehículo $k \in K$.
Variables de decisión	
$x_{ijk} = \begin{cases} 1 & \text{si el nodo } i \in V \text{ precede al } j \in V \text{ en la ruta del vehículo } k \in K; \\ 0 & \text{en caso contrario.} \end{cases}$	
$z_{ij} = \begin{cases} 1 & \text{si el depósito } i \in I \text{ sirve al cliente } j \in J; \\ 0 & \text{en caso contrario.} \end{cases}$	
$u_{ik} =$ Número de clientes visitados previamente al $i \in I$ por el vehículo $k \in K$.	

Tabla 2.1: Elementos necesarios para la formulación del MDVRP.

[1] En el resto de la memoria, entenderemos el coste c_{ij} como la distancia entre los nodos i, j.

Modelo (MDVRP)

$$\min z = \sum_{i\in V}\sum_{j\in V}\sum_{k\in K} c_{ij}x_{ijk} \tag{1}$$

Sujeto a

$$\sum_{k\in K}\sum_{i\in V} x_{ijk} = 1 \qquad \forall j \in J \tag{2}$$

$$\sum_{j\in J} d_j \sum_{i\in V} x_{ijk} \leq Q_k \qquad \forall k \in K \tag{3}$$

$$u_{ik} - u_{jk} + n x_{ijk} \leq n - 1 \qquad \forall i, j \in J, k \in K \tag{4}$$

$$\sum_{i\in V} x_{ijk} = \sum_{i\in V} x_{jik} \qquad \forall j \in V, k \in K \tag{5}$$

$$\sum_{i\in I}\sum_{j\in J} x_{ijk} \leq 1 \qquad \forall k \in K \tag{6}$$

$$\sum_{j\in J} d_j z_{ij} \leq A_i \qquad \forall i \in I \tag{7}$$

$$-z_{ij} + \sum_{u\in J}(x_{iuk} + x_{ujk}) \leq 1 \qquad \forall i \in I, j \in J, k \in K \tag{8}$$

$$x_{ijk} \leq z_{ij} \qquad \forall i \in I, j \in J, k \in K \tag{9}$$

$$x_{ijk} \in \{0, 1\} \qquad \forall i, j \in V, k \in K \tag{10}$$

$$z_{ij} \in \{0, 1\} \qquad \forall i \in I, j \in J \tag{11}$$

$$u_{ik} \geq 0 \qquad \forall i \in J, k \in K \tag{12}$$

La función objetivo (1) minimiza la distancia total recorrida por los vehículos. En cuanto a las restricciones, la restricción (2) garantiza que cada cliente es visitado una única vez, mientras que la restricción (3) no permite que se exceda la capacidad los vehículos. La rotura de ciclos está asegurada por la restricción (4) y la conservación de flujo mediante la restricción (5). La restricción (6) indica que cada camión será utilizado como mucho una vez y la restricción (7) que no se excede la capacidad de los depósitos. La restricción (8) permite que un cliente sea asignado a un depósito solamente si existe una ruta que los una y la restricción (9) que cada cliente sea atendido por un único depósito. El resto de restricciones hacen referencia al dominio de las variables de decisión.

El MDVRP ha sido aplicado con éxito para resolver situaciones que provienen de problemas reales. En esta línea, Pathumnakul (1996) desarrolló un algoritmo tabú aplicado a la recogida de papel reciclado en Iowa, Shi et al. (2020) abordan el problema de recogida de residuos y Du et al. (2017) estudian la aplicación del MDVRP al transporte de materiales peligrosos.

Otras formulaciones más generales combinan el MDVRP con otras particularidades que pueden aparecen en los problemas de rutas: existencia de ventanas de tiempo (VRPTW), posibilidad de satisfacer la demanda en varios envíos (SDVRP), posibilidad de que los clientes reciban y entreguen pedidos al mismo tiempo (VRPPD), etc. Por ejemplo, Wasner y Zäpfel (2004) estudian un problema de ubicación de depósitos y distribución conjuntamente para un servicio de paquetería en Austria. En este problema los clientes pueden tanto enviar o recibir paquetes desde su domicilios. Por su parte, Flisberg, Lidén y Rönnqvist (2009) diseñan una política óptima de rutas aplicada a un problema en la industria forestal en Suecia. Este problema incluye algunas de las características antes mencionadas. Crevier, Cordeau y

Laporte (2007) se inspiran en un problema real en la industria alimentaria para presentar una extensión del MDVRP en la que los vehículos pueden reponer su carga en depósitos intermedios.

En este trabajo, estudiaremos una extensión del MDVRP en elz caso de que varias empresas deciden estudiar el efecto de una posible cooperación combinando sus respectivos recursos.

2.2. Cooperación en problemas de rutas de vehículos con múltiples depósitos (CoMDVRP)

La situación que vamos a analizar parte del siguiente supuesto: un grupo de empresas, cada cual con sus respectivos recursos, deciden cooperar para intentar reducir los costes totales de distribución. Para ello, plantearemos un juego TU en el que cada jugador representa una empresa y la función característica se obtiene resolviendo una generalización del MDVRP. Esta generalización se recoge en Zibaei, Hafezalkotob y Ghashami (2016).

Cada jugador $i \in P$ manejará una cierta cantidad de depósitos y una flota de vehículos para atender a sus respectivos clientes. Cuando se forma una coalición $S \subset P$, los agentes involucrados unifican sus recursos de forma que cualquier miembro de la coalición pueda hacer uso de ellos y es preciso atender a los clientes asociados a cada jugador de la coalición S.

Por su parte, la función de coste, $c(S)$ viene dada por la resolución del MDVRP obtenido al considerar todos los depósitos, vehículos y clientes de S conjuntamente. En particular, si denotamos por $[S_m]$ la posición que ocupa la coalición S_m de entre todos las posibles coaliciones de P la función de coste $c(S_m)$ se obtiene al resolver el siguiente modelo de programación lineal que denominamos (CoMDVRP).

Conjuntos	
$I_{[S_m]}$	Conjunto de depósitos de la coalición S_m.
$J_{[S_m]}$	Conjunto de clientes de la coalición S_m.
$K_{[S_m]}$	Conjunto de vehículos de la coalición S_m.
$V_{[S_m]} = I_{[S_m]} \cup J_{[S_m]}$	Conjunto de nodos considerando la coalición S_m.
Constantes y parámetros	
$d_{[S_m]}$	Número total de depósitos de la coalición S_m.
$n_{[S_m]}$	Número total de clientes de la coalición S_m.
$\|K\|_{[S_m]}$	Número total de vehículos de la coalición S_m.
$c_{ij[S_m]}$	Coste de viajar del nodo i al nodo j considerando la coalición S_m; $i, j \in V_{[S_m]}$.
A_i	Cantidad almacenada del producto en el depósito $i \in I$.
d_j	Demanda del cliente $j \in J$.
$Q_{k[S_m]}$	Capacidad del vehículo $k \in K$ considerando la coalición S_m.
Variables de decisión	

$$x_{ijk[S_m]} = \begin{cases} 1 & \text{si el nodo } i \in V_{[S_m]} \text{ precede al } j \in V_{[S_m]} \text{ en la ruta del vehículo } k \in K_{[S_m]} \\ & \text{considerando la coalición } S_m; \\ 0 & \text{en caso contrario.} \end{cases}$$

$$z_{ij[S_m]} = \begin{cases} 1 & \text{si el depósito } i \in I_{[S_m]} \text{ sirve al cliente } J_{[S_m]} \text{ considerando la coalición } S_m; \\ 0 & \text{en caso contrario.} \end{cases}$$

$u_{ik[S_m]} =$ Número de clientes visitados previamente al $i \in I_{[S_m]}$ por el vehículo $k \in K_{[S_m]}$ considerando la coalición S_m.

Tabla 2.2: Elementos necesarios para la formulación del CoMDVRP.

Modelo (CoMDVRP)

$$c(Sm) := \min\ z = \sum_{i \in V_{[S_m]}} \sum_{j \in V_{[S_m]}} \sum_{k \in K_{[S_m]}} c_{ij[S_m]} x_{ijk[S_m]} \tag{1}$$

Sujeto a

$$\sum_{k \in K_{[S_m]}} \sum_{i \in V_{[S_m]}} x_{ijk[S_m]} = 1 \qquad \forall j \in J_{[S_m]} \tag{2}$$

$$\sum_{j \in J_{[S_m]}} d_j \sum_{i \in V_{[S_m]}} x_{ijk[S_m]} \leq Q_k[S_m] \qquad \forall k \in K_{[S_m]} \tag{3}$$

$$u_{ik[S_m]} - u_{jk[S_m]} + n_{[S_m]} x_{ijk[S_m]} \leq n_{[S_m]} - 1 \qquad \forall i, j \in J_{[S_m]}, k \in K_{[S_m]} \tag{4}$$

$$\sum_{i \in V_{[S_m]}} x_{ijk[S_m]} = \sum_{i \in V_{[S_m]}} x_{jik[S_m]} \qquad \forall j \in V_{[S_m]}, k \in K_{[S_m]} \tag{5}$$

$$\sum_{i \in I_{[S_m]}} \sum_{j \in J_{[S_m]}} x_{ijk[S_m]} \leq 1 \qquad \forall k \in K_{[S_m]} \tag{6}$$

$$\sum_{j \in J_{[S_m]}} d_j z_{ij[S_m]} \leq A_i \qquad \forall i \in I_{[S_m]} \tag{7}$$

$$-z_{ij[S_m]} + \sum_{u \in J_{[S_m]}} (x_{iuk[S_m]} + x_{ujk[S_m]}) \leq 1 \qquad \forall i \in I_{[S_m]}, j \in J_{[S_m]}, k \in K_{[S_m]} \tag{8}$$

$$x_{ijk[S_m]} \leq z_{ij[S_m]} \qquad \forall i \in I_{[S_m]}, j \in J_{[S_m]}, k \in K_{[S_m]} \tag{9}$$

$$\sum_{k \in K_{[S_m]}} \sum_{j \in J_{[S_m]}} x_{ijk[S_m]} = 1 \qquad \forall i \in I_{[S_m]} \tag{10}$$

$$x_{ijk[S_m]} + x_{j'i'k[S_m]} \leq 1 \qquad \forall i, i' \in I_{[S_m]}, i \neq i', j, j' \in J_{[S_m]}, j \neq j', k \in K_{[S_m]} \tag{11}$$

$$x_{ijk[S_m]} \in \{0, 1\} \qquad \forall i, j \in V_{[S_m]}, k \in K_{[S_m]} \tag{12}$$

$$z_{ij[S_m]} \in \{0, 1\} \qquad \forall i \in I_{[S_m]}, j \in J_{[S_m]} \tag{13}$$

$$u_{ik[S_m]} \geq 0 \qquad \forall i \in J_{[S_m]}, k \in K_{[S_m]} \tag{14}$$

Este modelo es una extensión del (MDVRP) presentado en la Sección 2.1 adaptando las restricciones a cada coalición S_m y añadiendo dos nuevas restricciones: la restricción (10) restringe la asignación de cada cliente a un único depósito y la restricción (11) asegura que las rutas que empiezan en un depósito no terminan en otro.

A continuación, ilustramos las ventajas de la cooperación entre los dueños de los depósitos adaptando el ejemplo presentado en Zibaei, Hafezalkotob y Ghashami (2016). Para ello, compararemos las soluciones obtenidas mediante los modelos (MDVRP) y (CoMDVRP).

Consideraremos tres empresas, cada una de ellas controlando dos depósitos con dos vehículos. La Tabla 2.3 indica los clientes, y sus respectivas demandas, asociados a cada empresa; y la información relativa a depósitos y vehículos. Las distancias entre cualquier par de nodos se recogen en la matriz de costes de la Tabla 2.4. La ubicación de los depósitos y los clientes asociados a cada empresa se representan en la Figura 2.1.

Empresa	Clientes J	Demanda d_j
1	1	30
	2	50
	3	30
	4	40
	5	70
	6	40
2	7	40
	8	30
	9	70
	10	50
	11	30
	12	40
3	13	50
	14	40
	15	30
	16	40
	17	30
	18	70

Empresa	Depósitos I	Cantidad almacenada A_i	Vehículos K	Capacidad Q_k
1	1	260	1	120
	2	260	2	140
2	3	130	3	120
	4	150	4	150
3	5	150	5	150
	6	130	6	150

Tabla 2.3: Datos del ejemplo.

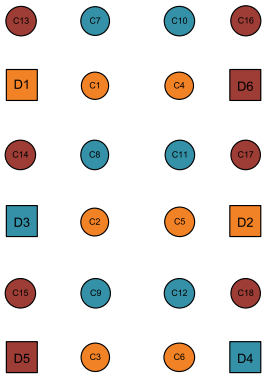


Figura 2.1: Ubicación de depósitos y clientes de cada empresa.

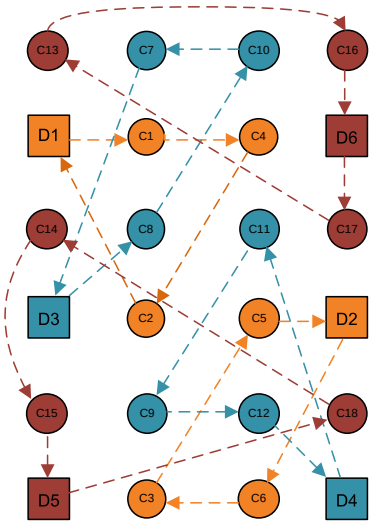
	D_1	D_2	D_3	D4	D5	D6	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}	C_{12}	C_{13}	C_{14}	C_15	C_16	C_17	C_18
D_1	–	–	–	–	–	–	0.25	1.03	2.02	0.75	1.25	2.14	0.35	0.79	1.77	0.79	1.06	1.9	0.25	0.75	1.75	1.03	1.25	2.02
D_2	–	–	–	–	–	–	1.25	0.75	1.25	1.03	0.25	1.03	1.46	0.79	1.06	1.27	0.35	0.79	1.60	1.03	1.25	1.25	0.25	0.75
D_3	–	–	–	–	–	–	1.03	0.25	1.03	1.25	0.75	1.25	1.27	0.35	0.79	1.46	0.79	1.06	1.25	0.25	0.75	1.6	1.03	1.25
D4	–	–	–	–	–	–	2.14	1.25	0.75	2.02	1.03	0.25	2.37	1.46	0.79	2.26	1.27	0.35	2.46	1.6	1.03	2.25	1.25	0.25
D5	–	–	–	–	–	–	2.02	1.03	0.25	2.14	1.25	0.75	2.26	1.27	0.35	2.37	1.46	0.79	2.25	1.25	0.25	2.46	1.6	1.03
D6	–	–	–	–	–	–	0.75	1.25	2.14	0.25	1.03	2.02	0.79	1.06	1.9	0.35	0.79	1.77	1.03	1.25	2.02	0.25	0.75	1.75
C_1	0.25	1.25	1.03	2.14	2.02	0.75	–	1.00	2.00	0.5	1.12	2.06	0.25	0.75	1.75	0.56	0.9	1.82	0.35	0.79	1.77	0.79	1.06	1.90
C_2	1.03	0.75	0.25	1.25	1.03	1.25	1.00	–	1.00	1.12	0.5	1.12	1.25	0.25	0.75	1.35	0.56	0.9	1.27	0.35	0.79	1.46	0.79	1.06
C_3	2.02	1.25	1.03	0.75	0.25	2.14	2.00	1.00	–	2.06	1.12	0.5	2.25	1.25	0.25	2.3	1.35	0.56	2.26	1.27	0.35	2.37	1.46	0.79
C_4	0.75	1.03	1.25	2.02	2.14	0.25	0.5	1.12	2.06	–	1.00	2.00	0.56	0.9	1.82	0.25	0.75	1.75	0.79	1.06	1.9	0.35	0.79	1.77
C_5	1.25	0.25	0.75	1.03	1.25	1.03	1.12	0.5	1.12	1.00	–	1.00	1.35	0.56	0.9	1.25	0.25	0.75	1.46	0.79	1.06	1.27	0.35	0.79
C_6	2.14	1.03	1.25	0.25	0.75	2.02	2.06	1.12	0.5	2.00	1.00	–	2.3	1.35	0.56	2.25	1.25	0.25	2.37	1.46	0.79	2.26	1.27	0.35
C_7	0.35	1.46	1.27	2.37	2.26	0.79	0.25	1.25	2.25	0.56	1.35	2.3	–	1.00	2.00	0.5	1.12	2.06	0.25	1.03	2.02	0.75	1.25	2.14
C_8	0.79	0.79	0.35	1.46	1.27	1.06	0.75	0.25	1.25	0.9	0.56	1.35	1.00	–	1.00	1.12	0.5	1.12	1.03	0.25	1.03	1.25	0.75	1.25
C_9	1.77	1.06	0.79	0.79	0.35	1.9	1.75	0.75	0.25	1.82	0.9	0.56	2.00	1.00	–	2.06	1.12	0.5	2.02	1.03	0.25	2.14	1.25	0.75
C_{10}	0.79	1.27	1.46	2.26	2.37	0.35	0.56	1.35	2.3	0.25	1.25	2.25	0.5	1.12	2.06	–	1.00	2.00	0.75	1.25	2.14	0.25	1.03	2.02
C_{11}	1.06	0.35	0.79	1.27	1.46	0.79	0.9	0.56	1.35	0.75	0.25	1.25	1.12	0.5	1.12	1.00	–	1.00	1.25	0.75	1.25	1.03	0.25	1.03
C_{12}	1.9	0.79	1.06	0.35	0.79	1.77	1.82	0.9	0.56	1.75	0.75	0.25	2.06	1.12	0.5	2.00	1.00	–	2.14	1.25	0.75	2.02	1.03	0.25
C_{13}	0.25	1.60	1.25	2.46	2.25	1.03	0.35	1.27	2.26	0.79	1.46	2.37	0.25	1.03	2.02	0.75	1.25	2.14	–	1.00	2.00	1.00	1.41	2.24
C_{14}	0.75	1.03	0.25	1.6	1.25	1.25	0.79	0.35	1.27	1.06	0.79	1.46	1.03	0.25	1.03	1.25	0.75	1.25	1.00	–	1.00	1.41	1.00	1.41
C_15	1.75	1.25	0.75	1.03	0.25	2.02	1.77	0.79	0.35	1.9	1.06	0.79	2.02	1.03	0.25	2.14	1.25	0.75	2.00	1	–	2.24	1.41	1.00
C_16	1.03	1.25	1.6	2.25	2.46	0.25	0.79	1.46	2.37	0.35	1.27	2.26	0.75	1.25	2.14	0.25	1.03	2.02	1.00	1.41	2.24	–	1.00	2.00
C_17	1.25	0.25	1.03	1.25	1.6	0.75	1.06	0.79	1.46	0.79	0.35	1.27	1.25	0.75	1.25	1.03	0.25	1.03	1.41	1.00	1.41	1.00	–	1.00
C_18	2.02	0.75	1.25	0.25	1.03	1.75	1.90	1.06	0.79	1.77	0.79	0.35	2.14	1.25	0.75	2.02	1.03	0.25	2.24	1.41	1.00	2.00	1.00	–

Tabla 2.4: Matriz de costes (C_{ij}) para el ejemplo.

Para aplicar el modelo (MDVRP) tenemos que considerar las empresas individualmente de forma que cada una atiende a sus respectivos clientes haciendo uso de su flota. La solución óptima para cada empresa se recoge en la Tabla 2.5. La Tabla 2.5a muestra las rutas óptimas y la distancia total recorrida para cada empresa, la representación de estas rutas se indica en la Tabla 2.5b. Con este modelo, las empresas tendrían que recorrer un total de $z = z_1 + z_2 + z_3 = 19.38$ kilómetros.

Empresa	Rutas	Distancia recorrida
1	D_1–c_1–c_4–c_2–D_1 D_2–c_6–c_3–c_5–D_2	$z_1 = 5.80$
2	D_3–c_8–c_{10}–c_7–D_3 D_4–c_{11}–c_9–c_{12}–D_4	$z_2 = 6.48$
3	D_5–c_{18}–c_{14}–c_{15}–D_5 D_6–c_{17}–c_{13}–c_{16}–D_6	$z_3 = 7.10$

(a) Rutas óptimas para cada empresa.



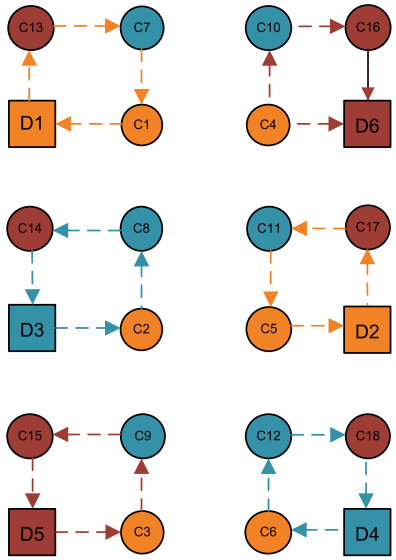
(b) Representación rutas óptimas.

Tabla 2.5: Solución óptima para cada empresa usando el modelo MDVRP.

Si consideramos que las empresas pueden cooperar podemos modelar la situación como un juego TU donde cada empresa representa un jugador del conjunto $P = \{1, 2, 3\}$ y la función característica se obtiene al resolver el modelo (CoMDVRP) para cada coalición $S_m \subset P$. La solución óptima para cada coalición se recoge en la Tabla 2.6. La Tabla 2.6a muestra las rutas óptimas y la distancia total recorrida para cada coalición y la Tabla 2.6b representa la solución óptima para la coalición de todos los jugadores. En este caso, la distancia total recorrida es de $z_{S_7} = 6.00$ lo cual indica una reducción del 70 % con respecto al modelo anterior.

Coalición	Rutas	Distancia recorrida
$S_1 = \{1\}$	D_1–c_1–c_4–c_2–D_1 D_2–c_6–c_3–c_5–D_2	$z_{S_1} = 5.80$
$S_2 = \{2\}$	D_3–c_8–c_{10}–c_7–D_3 D_4–c_{11}–c_9–c_{12}–D_4	$z_{S_2} = 6.48$
$S_3 = \{3\}$	D_5–c_{18}–c_{14}–c_{15}–D_5 $D6$–c_{17}–c_{13}–c_{16}–D_6	$z_{S_3} = 7.10$
$S_4 = \{1, 2\}$	D_2–c_9–c_5–D_2 D_2–c_{10}–c_4–c_{11}–D_2 D_3–c_7–c_1–c_8–c_2–D_3 D_4–c_{12}–c_3–c_6–D_4	$z_{S_4} = 7.59$
$S_5 = \{1, 3\}$	D_1–c_{14}–c_{15}–c_1–c_{13}–D_1 D_2–c_5–c_2–c_{17}–D_2 D_5–c_{18}–c_6–c_3–D_5 D_6–c_4–c_{16}–D_6	$z_{S_5} = 8.89$
$S_6 = \{2, 3\}$	D_3–c_8–c_{13}–c_{14}–D_3 D_4–c_{18}–c_{17}–c_{11}–D_4 D_5–c_{15}–c_9–c_{12}–D_5 D_6–c_7–c_{10}–c_{16}–D_6	$z_{S_6} = 8.98$
$S_7 = \{1, 2, 3\}$	D_1–c_{13}–c_7–c_1–D_1 D_2–c_{17}–c_{11}–c_5–D_2 D_3–c_2–c_8–c_{14}–D_3 D_4–c_6–c_{12}–c_{18}–D_4 D_5–c_3–c_9–c_{15}–D_5 D_6–c_4–c_{10}–c_{16}–D_6	$z_{S_7} = 6.00$

(a) Rutas óptimas para cada coalición $S_m \subset P$.



(b) Representación rutas óptimas para la gran coalición.

Tabla 2.6: Solución óptima para cada coalición usando el modelo CoMDVRP.

Es inmediato comprobar que se verifica la propiedad de subaditividad (mencionada en la Sección 1.2.1) por lo que los jugadores tendrán incentivos para cooperar. El valor de Shapley del juegos de costes es $\Phi(c) = (1.42, 1.81, 2.77)$ el cual asegura que aquellos jugadores que recorren más distancia individualmente se les asigne una mayor parte de los costes.

Con respecto a la complejidad, la Tabla 2.7 recoge para cada coalición las dimensiones y el tiempo de resolución de cada problema de programación lineal. Nótese que a medida que aumenta el número de jugadores en cada coalición el tiempo de resolución tiende a aumentar. Además, el número de problemas de programación lineal a resolver aumenta de forma exponencial con el número de jugadores.

Para hallar un reparto exacto de los costes en un modelo CoMDVRP estamos ante una situación doblemente compleja. Por un lado hay que resolver un número exponencial de problemas exactos de programación lineal y, por otro lado, calcular el valor de Shapley que también tiene complejidad exponencial.

Eso justifica el diseño de un algoritmo heurístico que permita resolver los problemas de rutas en poco tiempo y el cálculo del valor de Shapley mediante simulación presentado en la Sección 1.2.2.

Coalición	Dimensiones del problema Variables × Restricciones	Tiempo (s)
$S_1 = \{1\}$	136×136	0.09
$S_2 = \{2\}$	136×136	0.11
$S_3 = \{3\}$	136×136	0.16
$S_4 = \{1, 2\}$	1056×1000	23.53
$S_5 = \{1, 3\}$	1056×1000	180.90
$S_6 = \{2, 3\}$	1056×1000	69.69
$S_7 = \{1, 2, 3\}$	3312×3528	20.99

Tabla 2.7: Dimensión y tiempo de resolución de cada problema de programación lineal.

2.3. Heurística para el MDVRP

En esta sección presentamos el algoritmo (meta)heurístico diseñado para este problema. En primer lugar, obtendremos una solución inicial del problema adaptando el algoritmo de Clarke y Wright (1964) con la propuesta de Tillman (1969). Acto seguido, presentamos un procedicimiento para adaptar el número de rutas obtenidas al número de vehículos disponible para obtener una solución inicial válida que luego optimizaremos en la fase de mejora haciendo uso de un algoritmo voraz. Finalmente, aplicamos el algoritmo heurístico a un ejemplo.

2.3.1. Solución inicial

El algoritmo propuesto por Clarke y Wright (1964) es una de las heurísticas más utilizadas para problemas de rutas de vehículos por su sencilla implementación y su buen rendimiento computacional. Como ya hemos mencionado, se trata de una heurística constructiva en la que una solución es obtenida iterativamente siguiendo un criterio de ahorros.

En el caso del VRP en el que existe un único depósito y el número de vehículos no está prefijado, el algoritmo parte de que cada cliente representa una única ruta de ida y vuelta al depósito. A continuación, las rutas se van uniendo considerando el ahorro que supondría la fusión de dos rutas y respetando la capacidad máxima de los vehículos. En la Figura 2.2 representamos las rutas iniciales de dos clientes, i, j cuyo coste total sería de $z_1 = c_{0i} + c_{i0} + c_{0j} + c_{j0}$. En caso de unir ambas rutas (Figura 2.3) el coste total sería $z_2 = c_{0i} + c_{ij} + c_{j0}$, con lo cual, el ahorro total sería de $s_{ij} = z_2 - z_1 = c_{i0} + c_{0j} - c_{ij}$.

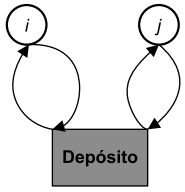


Figura 2.2: Solución inicial algoritmo de Clarke y Wright.

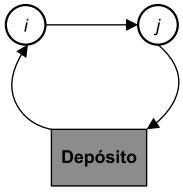


Figura 2.3: Ruta resultante de unir los clientes i y j.

En el Cuadro 2.8 enumeramos los pasos a seguir por el algoritmo de Clarke y Wright para el caso del VRP.

Paso 1:

Crear n rutas de la forma $(0, j, 0)$ entre cada cliente j y el depósito 0.

Paso 2:

Calcular la matriz de ahorros $s_{ij} = c_{i0} + c_{0j} - c_{ij}$ para cada par de clientes i, j y ordenar los ahorros de forma decreciente.

Paso 3:

Seleccionar los clientes i, j tales que $s_{ij} = \max_{i',j' \in J} \{s_{i'j'}\}$ y ver si podemos unir sus respectivas rutas. Para eso tienen que darse las siguientes condiciones:

* $s_{ij} > 0$.
* Los clientes i, j no pertenecen a la misma ruta, i es el último cliente de su ruta y j el primero de la suya.
* La unión de las rutas conteniendo a i y a j no excede la capacidad del vehículo.

Si la unión es factible, eliminar los arcos $(i, 0)$, $(0, j)$ y añadir el arco (i, j) a la solución. Independientemente de si se produce la unión o no, se elimina s_{ij} de la lista de ahorros.

Paso 4:

Repetir el Paso 3 hasta que no haya más uniones factibles o no se produzcan ahorros positivos al unir nuevas rutas.

Cuadro 2.8: Algoritmo de Clarke y Wright para el VRP.

En el caso del MDVRP contamos con varios depósitos para atender a los clientes sin una asignación previa. Por tanto, empezaríamos asignando cada cliente al depósito más cercano. Por otro lado, los ahorros producidos al unir dos rutas varían en función del depósito al que pueden ser asignadas.

Si calculamos los ahorros de la misma forma que en el VRP podríamos asignar dos clientes cercanos a un depósito a otro mucho más lejano como se muestra en la Figura 2.4 en la cual $s^2_{jk} = c_{j2} + c_{2k} - c_{jk} > s^1_{jk} = c_{j1} + c_{1k} - c_{jk}$.

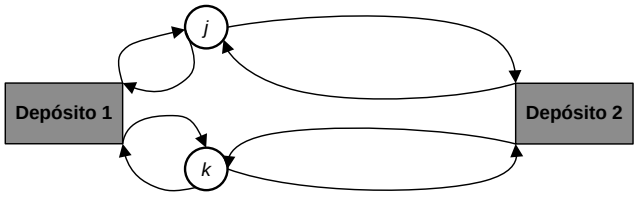


Figura 2.4: Solución inicial para el algoritmo de Clarke y Wright multidepósito.

Para corregir esta situación recurrimos a la propuesta de Tillman (1969) para adaptar el algoritmo de Clarke y Wright al MDVRP.

En primer lugar, el autor proponer recalcular la distancia entre cada depósito $i \in I$ y cada cliente $j \in J$ como $\hat{c}_{ij} = \min_{i' \in I}\{c_{i'j}\} - (c_{ij} - \min_{i' \in I}\{c_{i'j}\}) = 2\min_{i' \in I}\{c_{i'j}\} - c_{ij}$.

Si aplicamos esta expresión al cliente j y a cada uno de los depósitos de la Figura 2.4 obtendríamos:

- Depósito 1: $\hat{c}_{1j} = \min\{c_{1j}, c_{2j}\} - (c_{1j} - \min\{c_{1j}, c_{2j}\}) = c_{1j}$
- Depósito 2: $\hat{c}_{2j} = \min\{c_{1j}, c_{2j}\} - (c_{2j} - \min\{c_{1j}, c_{2j}\}) = 2c_{1j} - c_{2j} < 0$

Vemos que la nueva distancia entre el cliente j y su depósito más cercano no varía. Para el depósito 2, la nueva distancia con el cliente j es negativa por ser más corto realizar la ruta de ida y vuelta entre el cliente j depósito 1 que la distancia original entre el cliente j y el depósito 2.

De esta forma, penalizamos aquellos depósitos que están muy lejos y para los que no vale la pena considerar una posible unión. A continuación procederíamos a calcular los ahorros entre cada par de clientes y cada depósito usando esta nueva distancia y construiríamos las rutas de forma similar al caso del VRP.

En el Cuadro 2.9 presentamos el algoritmo de Clarke y Wright para el MDVRP.

Paso 1:

Crear n rutas de la forma (i_j, j, i_j) siendo $i_j \in I$ el depósito más cercano al cliente $j \in N$.

Paso 2:

Calcular la matriz de ahorros $\hat{s}^i_{jk} = \hat{c}_{ij} + \hat{c}_{ik} - c_{jk}$ para cada depósito $i \in I$ y cada par de clientes $j, k \in N$.

Paso 3:

Seleccionar los el depósito i clientes $j, k \in N$ tales que $\hat{s}^i_{jk} = \max_{i' \in I}\{\hat{s}^{i'}_{j'k'} / j', k' \in J\}$ y ver si podemos unir sus respectivas rutas con el depósito $i \in I$. Para eso tienen que darse las siguientes condiciones:

* $\hat{s}^i_{jk} > 0$.
* Los clientes j, k no pertenecen a la misma ruta, ambos están asignados al mismo depósito i, j es el último cliente de su ruta y k el primero de la suya
* La unión de las rutas conteniendo a j y a k no excede la capacidad del vehículo más grande.

Si la unión es factible se eliminan los arcos (j, i), (i, k) y se añade el arco (i, j) a la solución. Independientemente de si se produce la unión o no, se elimina $\hat{s}^i_{jk}$ para cada depósito $i \in I$.

Paso 4:

Repetir el Paso 3 hasta que no haya más uniones factibles o no se produzcan ahorros positivos al unir nuevas rutas.

Cuadro 2.9: Algoritmo de Clarke y Wright para el MDVRP.

Una vez finalizado el algoritmo de Clarke y Wright obtenemos un conjunto de rutas sin vehículos asignados. Además, no podemos asegurar que dispongamos de vehículos suficientes para cubrir todas esas rutas ya que en nuestro caso el número de vehículos está fijado previamente. Por tanto, hemos diseñado un procedimiento para asignar los vehículos disponibles a las rutas obtenidas por el algoritmo de Clarke y Wright. Este procedimiento se recoge en el Cuadro 2.10.

Paso 1: Comprobar si hay vehículos suficientes para cubrir las rutas.

Sea $|K|$ el número de vehículos disponible y r el número de rutas.

* Si $r \leq |K|$ ir al Paso 2.
* Si $r > |K|$ fusionar las $r - |K| + 1$ rutas con menor demanda.

Paso 2:

Ordenar las rutas según su carga en orden decreciente.

Paso 3:

Asignar, en orden decreciente, los vehículos con mayor capacidad a las rutas con mayor carga.

Cuadro 2.10: Procedimiento para asignar los vehículos a las rutas obtenidas por el algoritmo de Clarke y Wright MDVRP.

Independientemente del número de de rutas originales, la solución podría ser infactible al superarse la capacidad de algún vehículo. Nótese que en el Paso 3 del algoritmo de Clarke y Wright MDVRP se considera únicamente el vehículo de mayor capacidad como limitante para crear las rutas.

Por tanto, hemos diseñado un procedimiento para corregir la posible no factibilidad de la solución obtenida después de asignar los vehículos. Este procedimiento se expone en el Cuadro 2.11.

Paso 1:

Calcular el espacio libre en cada vehículo $\ell_k = Q_k - \sum_{i \in R_k} d_i$ siendo R_k la ruta asignada al vehículo $k \in K$.

* Si $\ell_k < 0$ para algún $k \in K$ ir al Paso 2.
* Si $\ell_k \geq 0$ en todo vehículo $k \in K$ la solución es factible y finalizamos el proceso.

Paso 2:

En la ruta que más excede se la capacidad del vehículo que tiene asignado:

1. Ordenar los clientes por demanda en orden decreciente.
2. Eliminar tantos clientes como sea necesario hasta obtener una ruta factible, comenzando por los clientes con menor demanda.

Paso 3:

Cada cliente eliminado es recolocado en alguna ruta en dos fases:

* Fase I: Elegir la posición óptima del cliente eliminado en cada ruta.

  Para eso, ubicamos el cliente eliminado en todas las posibles posiciones de cada ruta y seleccionamos aquella posición que genere una ruta con menor coste.

* Fase II: Elegir la ruta definitiva en la que estará el cliente eliminado.

  Para eso, calculamos las soluciones obtenidas al sustituir cada ruta por las ruta óptima generada en la Fase I y seleccionamos la solución con menor coste que sea factible. En caso de que ninguna solución sea factible, seleccionamos aquella con menor coste.

Repetir el Paso 3 hasta que no quede ningún cliente por recolocar.

Paso 4: Volver al Paso 1.

Cuadro 2.11: Procedimiento para corregir la no factibilidad de las rutas.

El Paso 3 de este procedimiento se corresponde con una estrategia de inserción voraz ya que, en primer lugar, es necesario evaluar todas las posibles ubicaciones de un cliente en cada ruta, y, a continuación, evaluar las soluciones obtenidas al considerar cada una de estas nuevas rutas.

Una vez finalizado el procedimiento descrito en el Cuadro 2.11 obtenemos una solución inicial factible de nuestro problema. Esta solución será utilizada como punto de partida en la fase de mejora en la que se intentará optimizar esta solución inicial.

2.3.2. Fase de mejora

La fase de mejora incluye el cálculo de nuevas soluciones siguiendo la estrategia de *Ruin & Recreate* propuesta originalmente por Schrimpf et al. (2000).

En cada iteración, el algoritmo genera una nueva solución eliminando algunos (Fase *Ruin*) y reubicándolos de forma voraz (Fase *Recreate*) de forma que la nueva solución tenga el menor coste posible. La nueva solución es aceptada como solución de partida de la siguiente iteración siguiendo un criterio basado en el recocido simulado.

Esta parte del algoritmo, incluye dos parámetros que tienen que ser fijados previamente:

- `MaxDestroy:` Influye en la generación de nuevas soluciones e indica el número máximo de clientes

a eliminar en cada iteración.

- `T:` Influye en la comparación de soluciones en la probabilidad de aceptación de soluciones de peor calidad.

En el Cuadro 2.12 detallamos los pasos a seguir en cada iteración de la fase de mejora.

- **Generación de nuevas soluciones**

  Partimos de una solución (`currentSol`) a partir de la cual generamos una nueva solución (`newSol`) siguiendo la estrategia *Ruin & Recreate*:

  * Fase *Ruin:*
    - Seleccionamos el número de clientes a eliminar al azar en el intervalo $[1, \texttt{MaxDestroy}]$ Denotamos por `n.eliminar` el número seleccionado.
    - Eliminamos al azar `n.eliminar` clientes de la `currentSol`.
      Denotamos por `auxSol` la solución obtenida tras eliminar los clientes seleccionados.
  * Fase *Recreate:*

    Generamos la nueva solución (`newSol`) reconstruyendo la (`auxSol`) recolocando, secuncialmente. los clientes eliminados en la mejor ubicación posible de forma análoga a la descrita en el Paso 3 del Cuadro 2.11.

- **Comparación nuevas soluciones**

  Comparamos las soluciones `currentSol` y `newSol` para decidir la solución de partida en la próxima iteración como sigue:

  * Si `newSol` no es factible la descartamos y la siguiente iteración vuelve a partir de `currentSol`
  * Si `newSol` es factible comparamos la distancia total recorrida por cada solución
    - Si $z_{\texttt{newSol}} < z_{\texttt{currentSol}}$ la siguiente iteración parte de `newSol` por tratarse de una solución de mejor calidad.
    - Si $z_{\texttt{newSol}} \geq z_{\texttt{currentSol}}$ aceptaremos `newSol` con probabilidad $p = e^{\frac{-\Delta z}{\texttt{T}}} = e^{\frac{-(z_{\texttt{newSol}} - z_{\texttt{currentSol}})}{\texttt{T}}}$

Cuadro 2.12: Fase de mejora del algoritmo.

El algoritmo calcula nuevas soluciones hasta que se verifica un criterio de parada como puede ser el número de iteraciones o el tiempo máximo de ejecución y devuelve la mejora solución generada en todo el proceso. En nuestro caso, el criterio de parada se determina fijando un tiempo máximo de ejecución.

Nótese que el algoritmo permite la aceptación de nuevas soluciones como punto de partida en la siguiente iteración aunque ésta sea peor que la solución actual. De esta forma, evitamos que el algoritmo quede atrapado en óptimos locales. La probabilidad de aceptación $p = e^{\frac{-\Delta z}{\texttt{T}}}$ depende de la diferencia entre la distancia recorrida por cada solución, Δz, y el parámetro `T`. En la Figura 2.5 vemos como varía esta probabilidad para distintos valores de Δz y $\texttt{T} \in [0, 100]$.

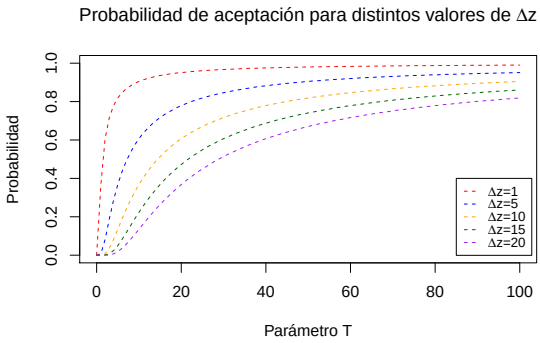


Figura 2.5: Probabilidad de aceptación para distintos valores de Δz y $\mathtt{T}$.

Vemos que a medida que aumenta Δz, la probabilidad de aceptación se reduce ya que estaríamos aceptando soluciones de peor calidad. La elección del parámetro $\mathtt{T}$ se escoge en función de la estrategia que queramos seguir para explorar nuevas soluciones. Una estrategia conservadora ($\mathtt{T} \approx 0$) no permitiría aceptar soluciones de peor calidad, con el correspondiente riesgo de quedar atrapado en un mínimo local. Por su parte, un valor muy grande de $\mathtt{T}$ permitiría la aceptación de soluciones aunque estas sean mucho peores que la actual.

Finalmente, resumimos los pasos a seguir por nuestro algoritmo en el Cuadro 2.13.

Paso 1:

Construcción de una primera solución usando el algoritmo de Clarke y Wright para el MDVRP (Cuadro 2.9).

Paso 2:

Asignación de vehículos a las distintas rutas calculadas en el Paso 1 (Cuadro 2.10).

Paso 3:

Corrección de la factibilidad de la solución obtenida en el Paso 2 (Cuadro 2.11).

Paso 4:

Generación de nuevas soluciones partiendo de la solución obtenida en el Paso 3 hasta que se alcanza un tiempo máximo de ejecución (Cuadro 2.12)

Cuadro 2.13: Resumen de nuestro algoritmo heurístico para el MDVRP.

2.3.3. Ilustración algoritmo heurístico en un ejemplo

Ilustramos nuestro algoritmo heurístico aplicando los pasos indicados en el Cuadro 2.13 en un ejemplo.

Consideramos el caso de una empresa que cuenta con $d = 3$ depósitos y 3 vehículos con capacidades $Q_k = (145, 145, 140)$ para atender a $n = 8$ clientes con demandas $d_j = (30, 55, 40, 40, 20, 45, 30, 60)$. Las distancias (en km) entre clientes y depósitos se recogen en la matriz de costes de la Tabla 2.14.

	D_1	D_2	D_3	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
D_1	–	–	–	3	4	38	8	6	32	41	41
D_2	–	–	–	33	38	4	38	35	9	8	32
D_3	–	–	–	41	40	41	31	40	32	25	2
C_1	3	33	41	–	37	41	1	44	44	34	32
C_2	4	38	40	37	–	33	5	11	32	6	33
C_3	38	4	41	33	43	–	43	32	32	15	44
C_4	8	38	31	1	5	43	–	2	32	35	44
C_5	6	35	40	44	37	32	2	–	38	32	34
C_6	32	9	32	44	32	14	32	38	–	7	32
C_7	41	8	25	34	6	15	35	32	7	–	3
C_8	41	32	2	32	33	44	44	34	32	3	–

Tabla 2.14: Distancia (en km) entre clientes y depósitos del ejemplo.

Paso 1: Algoritmo de Clarke y Wright MDVRP.

Empezamos *recalculando* la distancia entre depósitos y clientes $\hat{c}_{ij} = \min_{i' \in I}\{c_{i'j}\} - (c_{ij} - \min_{i' \in I}\{c_{i'j}\})$ para cada depósito $i \in I$ y cada cliente $j \in J$. Las distancias modificadas se recogen en la Tabla 2.15.

	C_1	C_2	C_3	C_3	C_5	C_6	C_7	C_8
D_1	3	4	-30	8	6	-14	-25	-37
D_2	-27	-30	4	-22	-23	9	8	-28
D_3	-35	-32	-33	-15	-28	-14	-9	2

Tabla 2.15: Distancia modificada entre clientes y depósitos del ejemplo.

Las matrices de ahorro (Tabla 2.16) se calculan para cada depósito $i \in I$ y cada par de clientes $j, k \in N$ como $\hat{s}^i_{jk} = \hat{c}_{ij} + \hat{c}_{ik} - c_{jk}$.

$\hat{s}^1_{jk}$	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
C_1	–	-30	-60	10	-35	-55	-56	-66
C_2	-30	–	-69	7	-1	-42	-27	-66
C_3	-60	-69	–	-65	-56	-76	-70	-111
C_4	10	7	-65	–	12	-38	-52	-73
C_5	-35	-27	-56	12	–	-46	-51	-65
C_6	-55	-42	-58	-38	-46	–	-46	-83
C_7	-56	-27	-70	-52	-51	-46	–	-65
C_8	-66	-66	-111	-73	-65	-83	-65	–

$\hat{s}^2_{jk}$	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
C_1	–	-94	-56	-50	-94	-62	-53	-87
C_2	-94	–	-69	-57	-64	-53	-28	-91
C_3	-56	-69	–	-61	-51	-19	-3	-68
C_4	-50	-57	-61	–	-47	-45	-49	-94
C_5	-94	-90	-51	-47	–	-52	-47	-85
C_6	-62	-53	-1	-45	-52	–	10	-51
C_7	-53	-28	-3	-49	-47	10	–	-23
C_8	-87	-91	-68	-94	-85	-51	-23	–

$\hat{s}^3_{jk}$	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
C_1	–	-104	-101	-51	-107	-93	-78	-65
C_2	-104	–	-108	-52	-71	-78	-47	-63
C_3	-101	-108	–	-91	-93	-79	-57	-75
C_4	-51	-5	-91	–	-45	-61	-59	-57
C_5	-107	-97	-93	-45	–	-80	-69	-60
C_6	-93	-78	-61	-61	-80	–	-30	-44
C_7	-78	-47	-57	-59	-69	-30	–	-10
C_8	-65	-63	-75	-57	-60	-44	-10	–

Tabla 2.16: Matrices de ahorro $\hat{s}^i_{jk}$ para cada depósito i.

A continuación, el algoritmo comienza a fusionar las rutas de los distintos clientes siempre que la

unión entre clientes conlleve un ahorro positivo y la unión sea factible, para eso limitamos la capacidad de las rutas considerando el vehículo de mayor capacidad. Además, para unir dos clientes j, k tiene que verificarse que j sea el último cliente de su ruta, k el primero de la suya. En la Tabla 2.17 se muestran las iteraciones que realiza el algoritmo indicando, para cada iteración:

- Ahorro S_m: Máximo ahorro posible calculado como $S_m = \max_{i \in I}\{\hat{s}^i_{jk}/j, k \in N\}$.
- Depósito i: Depósito correspondiente al ahorro S_m.
- Clientes $j \to k$: Clientes correspondientes al ahorro S_m.
- Ruta r_j: Ruta actual a la que pertenece el cliente j.
- Ruta r_k: Ruta actual a la que pertenece el cliente k.
- Unión: Indica si es posible realizar la unión entre las rutas r_j y r_k.
- Coste: Coste de la solución al finalizar la iteración.
- Rutas: Rutas resultantes al finalizar la iteración.

Iter	S_m	Depósito i	Clientes $j \to k$	Ruta r_j	Ruta r_k	Unión	Coste	Rutas
0	–	–	–	–	–	–	88	D_1–c_1–D_1 D_1–c_2–D_1 D_2–c_3–D_2 D_1–c_4–D_1 D_1–c_5–D_1 D_2–c_6–D_2 D_2–c_7–D_2 D_3–c_8–D_3
1	12	1	$5 \to 4$	D_1–c_5–D_1	D_1–c_4–D_1	Sí	76	D_1–c_1–D_1 D_1–c_2–D_1 D_2–c_3–D_2 D_1–c_5–c_4–D_1 D_2–c_6–D_2 D_2–c_7–D_2 D_3–c_8–D_3
2	10	1	$4 \to 1$	D_1–c_5–c_4–D_1	D_1–c_1–D_1	Sí	66	D_1–c_2–D_1 D_2–c_3–D_2 D_1–c_5–c_4–c_1–D_1 D_2–c_6–D_2 D_2–c_7–D_2 D_3–c_8–D_3
3	10	2	$7 \to 6$	D_2–c_7–D_2	D_2–c_6–D_2	Sí	56	D_1–c_2–D_1 D_2–c_3–D_2 D_1–c_5–c_4–c_1–D_1 D_2–c_7–c_6–D_2 D_3–c_8–D_3
4	7	1	$4 \to 2$	D_1–c_5–c_4–c_1–D_1	D_1–c_2–D_1	No	56	D_1–c_2–D_1 D_2–c_3–D_2 D_1–c_5–c_4–c_1–D_1 D_2–c_7–c_6–D_2 D_3–c_8–D_3
5	0	–	–	–	–	–	56	D_1–c_2–D_1 D_2–c_3–D_2 D_1–c_5–c_4–c_1–D_1 D_2–c_7–c_6–D_2 D_3–c_8–D_3

Tabla 2.17: Iteraciones del algoritmo Clarke y Wright multidepósito.

Nótese que la iteración 0, se corresponde con las rutas resultantes de unir cada cliente con el depósito más cercano.

Vemos que en la iteración 5, no se produce ningún ahorro ($S_m = 0$) por tanto el algoritmo finaliza ya que la unión de nuevas rutas no reduciría la función objetivo. De esta forma, obtenemos $r = 5$ rutas con un coste total de $z = 56$.

Paso 2: Asignación de vehículos.

En este caso, el número de vehículos disponibles ($|K| = 3$) es inferior al número de rutas ($r = 5$) obtenidas en el paso anterior. Por tanto, es necesario fusionar algunas rutas para igualarlas al número de vehículos. En primer lugar, ordenamos las rutas por su carga de forma decreciente 2.18a), y fusionamos las $r - |K| + 1 = 3$ rutas con menor carga (Tabla 2.18b) en una única ruta.

Ruta	Carga
D_1–c_5–c_4–c_1–D_1	90
D_2–c_7–c_6–D_2	75
D_3–c_8–D_3	60
D_1–c_2–D_1	55
D_2–c_3–D_2	40

(a) Rutas ordenadas por carga.

Ruta	Carga
D_3–c_8–c_2–c_3–D_3	155
D_1–c_5–c_4–c_1–D_1	90
D_2–c_7–c_6–D_2	75

(b) Rutas ordenadas por carga después de la fusión.

Tabla 2.18: Procedimiento para ajustar el número de rutas al número de vehículos disponibles.

Finalmente, empezamos asignando los vehículos de mayor capacidad a las rutas con mayor carga como se indica en la Tabla 2.19.

Vehículo	Capacidad	Ruta	Carga	Disponible	Coste Ruta
1	145	D_3–c_8–c_2–c_3–D_3	155	-10	119
2	145	D_1–c_5–c_4–c_1–D_1	90	55	12
3	140	D_2–c_7–c_6–D_2	75	65	24

Tabla 2.19: Asignación de vehículos a cada ruta.

Vemos que esta solución es infactible al superarse la capacidad del primer vehículo en 10 unidades.

Paso 3: Corrección infactibilidad

En este caso, seleccionamos la ruta asignada al vehículo 1 y eliminamos el cliente 3 (Tabla 2.20) por ser el de menor demanda en esa ruta ($d_8 = 60, d_2 = 55, d_3 = 40$). De esta forma, la ruta asociada al vehículo 1 pasa a ser factible.

Vehículo	Capacidad	Ruta	Carga	Libre
1	145	D_3–c_8–c_2–D_3	114	30
2	145	D_1–c_5–c_4–c_1–D_1	90	55
3	140	D_2–c_7–c_6–D_2	75	65

Tabla 2.20: Rutas resultantes al eliminar al cliente 3.

Para recolocar al cliente 3, empezamos insertándolo en todas las posiciones de cada ruta (Tabla 2.21) seleccionando la posición que genere el menor coste en cada una de ellas.

Vehículo	Ruta	Coste Ruta
1	D_3–c_3–c_8–c_2–D_3	158
	D_3–c_8–c_3–c_2–D_3	129
	D_3–c_8–c_2–c_3–D_3	**119**
2	D_1–c_3–c_5–c_4–c_1–D_1	**76**
	D_1–c_5–c_3–c_4–c_1–D_1	85
	D_1–c_5–c_4–c_3–c_1–D_1	87
	D_1–c_5–c_4–c_1–c_3–D_1	88
3	D_2–c_3–c_7–c_6–D_2	35
	D_2–c_7–c_3–c_6–D_2	64
	D_2–c_7–c_6–c_3–D_2	**33**

Tabla 2.21: Fase I: Posibles ubicaciones del cliente 3 en cada ruta.

Cada ruta obtenida en la Fase I da lugar a una posible solución. Esta solución se obtiene al sustituir cada una de las rutas de la Tabla 2.20 por la correspondiente ruta óptima de la Fase I. En la Tabla 2.22 indicamos cada una de las posibles soluciones. Para cada solución indicamos el coste de cada ruta, el coste total de la solución y el espacio libre en cada uno de los camiones para ver si la solución es factible.

Finalmente, colocamos al cliente 3 y en la ruta asignada al vehículo 3 por tratarse de la solución con menor coste y ser factible.

Soluciones posibles	Vehículo	Capacidad	Rutas	Carga	Libre	Coste Rutas	Coste Solución
1	1	145	D_3–c_8–c_2–c_3–D_3	155	-10	119	155
	2	145	D_1–c_5–c_4–c_1–D_1	90	55	12	
	3	140	D_2–c_7–c_6–D_2	75	65	24	
2	1	145	D_3–c_8–c_2–D_3	115	30	75	175
	2	145	D_1–c_3–c_5–c_4–c_1–D_1	130	15	76	
	3	140	D_2–c_7–c_6–D_2	75	65	24	
3	1	145	D_3–c_8–c_2–D_3	115	30	75	**120**
	2	145	D_1–c_5–c_4–c_1–D_1	90	55	12	
	3	140	D_2–c_7–c_6–c_3–D_2	115	25	33	

Tabla 2.22: Fase II: Soluciones obtenidas al considerar las rutas óptimas de la Fase I.

Paso 4: Fase de mejora.

Partimos de la siguiente solución `currentSol`: $\begin{cases} \text{Vehículo 1: } D_3\text{–}c_8\text{–}c_2\text{–}D_3 \\ \text{Vehículo 2: } D_1\text{–}c_5\text{–}c_4\text{–}c_1\text{–}D_1 \\ \text{Vehículo 3: } D_2\text{–}c_7\text{–}c_6\text{–}c_3\text{–}D_2 \end{cases}$ obtenida en el paso anterior cuyo coste es de $z_{\texttt{currentSol}} = 120$. A continuación, calcularíamos una nueva solución aplicando las fases *Ruin* y *Recreate*. En la Tabla 2.23 resumimos una iteración completa de la fase de mejora.

La Fase *Ruin* comienza generando el número de clientes a eliminar de forma aleatoria. Supongamos que hemos fijado `MaxDestroy`=3, en ese caso, el algoritmo selecciona un número de clientes a eliminar aleatoriamente en el intervalo $[1, 3]$, por ejemplo, 2. A continuación, seleccionamos 2 clientes al azar y los eliminamos de la solución, e.g., eliminamos los clientes 7 y 8.

La Fase *Recreate* calcula una nueva solución recolocando, secuencialmente, los clientes 7 y 8 en la mejor inserción factible posible.

Partimos de la solución obtenida al eliminar los clientes 7 y 8. En primer lugar, consideramos la solución auxiliar resultante de eliminar los clientes $7, 8$ e insertaríamos el cliente 7, obteniendo una nueva solución auxiliar. A continuación, insertaríamos el cliente 8 en la solución anterior. Las soluciones auxiliares se indican en la tercera columna de la Tabla 2.23.

Soluciones	Fase *Ruin*		Fase *Recreate*						
	Clientes a recolocar	Solución a reconstruir	Fase I: Mejor inserción en cada ruta			Fase II: Mejor solución factible			
Inicial: `currentSol`		D_3–c_2–D_3	Vehículo	Ruta	Coste Ruta	Solución	Coste Ruta	Libre	Coste Solución
D_3–c_8–c_2–D_3	7, 8	D_1–c_5–c_4–c_1–D_1	1	D_3–c_7–c_2–D_3	71	D_3–c_7–c_2–D_3	71	30	
D_1–c_5–c_4–c_1–D_1		D_2–c_6–c_3–D_2	2	D_1–c_5–c_7–c_4–c_1–D_1	77	D_1–c_5–c_4–c_1–D_1	12	55	110
D_2–c_7–c_6–c_3–D_2			3	D_2–c_7–c_6–c_3–D_2	33	D_2–c_6–c_3–D_2	27	25	
Final: `newSol`		D_3–c_7–c_2–D_3	Vehículo	Ruta	Coste Ruta	Solución	Coste Ruta	Libre	Coste Solución
D_3–c_8–c_7–c_2–D_3	8	D_1–c_5–c_4–c_1–D_1	1	D_3–c_8–c_7–c_2–D_3	51	D_3–c_8–c_7–c_2–D_3	51	0	
D_1–c_5–c_4–c_1–D_1		D_2–c_6–c_3–D_2	2	D_1–c_8–c_5–c_4–c_1–D_1	81	D_1–c_5–c_4–c_1–D_1	12	55	90
D_2–c_6–c_3–D_2			3	D_2–c_8–c_6–c_3–D_2	82	D_2–c_6–c_3–D_2	27	55	

Tabla 2.23: Ejemplo de iteración del algoritmo en la fase de mejora.

Una vez reubicados los clientes 7 y 8 el coste de la nueva solución $z_{\texttt{newSol}} = 90$ se reduce comparando con el coste de la solución de partida $z_{\texttt{currentSol}} = 120$ y la nueva solución es factible. Por tanto, en la siguiente iteración el algoritmo partiría de `newSol` para continuar con la fase de mejora.

De esta forma, el algoritmo seguiría calculando soluciones hasta que se alcanzase el tiempo máximo de ejecución fijado.

Capítulo 3

Resultados numéricos

En esta sección presentamos los resultados numéricos obtenidos usando el modelo exacto y el algoritmo heurístico para el MDVRP. En primer lugar, describimos los ficheros de datos utilizados y las muestras construidas a partir de estos. A continuación, comparamos las soluciones obtenidas por el modelo exacto y el algoritmo heurístico en las muestras. Finalizamos la sección usando un enfoque cooperativo en dos de los ficheros y calculando el valor de Shapley por simulación como se describe en la Sección 1.2.2

Todos los resultados presentados en esta sección han sido evaluados en un ordenador con procesador Intel Core i7-8559 de 2.7GHz de frecuencia y 16Gb de memoria RAM. Todos los códigos utilizados se incluyen en el Anexo.

3.1. Descripción de los datos

Para validar el modelo exacto (MDVRP) y el algoritmo heurístico descritos en esta memoria utilizaremos las 33 instancias recogidas en Cordeau, Gendreau y Laporte (1997) disponibles para consulta en `https://neo.lcc.uma.es/vrp/vrp-instances/multiple-depot-vrp-instances/`. La Tabla 3.1 resume la información relevante de cada instancia como el número de clientes, el número de depósitos, el número de vehículos disponibles y la capacidad de cada vehículo (en este caso todas las instancias presentan una flota homogénea). En la Tabla 3.1 también se indica un identificador para cada instancia que será usado en en el resto de esta sección.

Para calcular la matriz de costes, cada instancia incluye las coordenadas de clientes y depóstios. En nuestro caso, hemos calculado la distancia entre cada par de nodos $i, j \in I \cup J$ usando la distancia euclídea $c_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ siendo (x_i, y_i) y (x_j, y_j) las coordenadas de los nodos i, j respectivamente.

Ante la imposibilidad de usar el modelo exacto en estas instancias por tener un número elevado de clientes, hemos extraído muestras de algunas instancias seleccionando clientes al azar. En concreto, hemos seleccionado 10 clientes de la instancia `p04`, 20 del `p06` y 30 del `p08`. Para cada grupo de clientes, hemos variado el número de depósitos de entre todos los disponibles y hemos seleccionado el número mínimo de vehículos necesarios para atender a cada grupo. De esta forma, hemos creado 9 ficheros de muestra que presentamos en la Tabla 3.2.

Instancia	Clientes	Depósitos	Camiones	Capacidad	Instancia	Clientes	Depósitos	Vehículos	Capacidad
p01	50	4	4	160	p18	240	6	5	80
p02	50	4	2	160	p19	240	6	5	60
p03	75	5	3	140	p20	240	6	5	60
p04	100	2	8	100	p21	360	9	5	60
p05	100	2	5	200	p22	360	9	5	60
p06	100	3	6	100	p23	360	9	5	60
p07	100	4	4	100	pr01	48	4	1	200
p08	249	2	14	500	pr02	96	4	2	195
p09	249	3	12	500	pr03	144	4	3	190
p10	249	4	8	500	pr04	192	4	4	185
p11	249	5	6	500	pr05	240	4	5	180
p12	80	2	5	60	pr06	288	4	6	175
p13	80	2	5	60	pr07	72	6	1	200
p14	80	2	5	60	pr08	144	6	2	190
p15	160	4	5	60	pr09	216	6	3	180
p16	160	4	5	60	pr10	288	6	4	170
p17	160	4	5	60					

Tabla 3.1: Resumen de las instancias recogidas en Cordeau, Gendreau y Laporte (1997).

Muestra	Instancia	Clientes	Depósitos	Vehículos	Capacidad
m1	p04	10	2	2	80
m2	p04	10	3	2	80
m3	p04	10	4	2	80
m4	p06	20	4	2	60
m5	p06	20	5	2	60
m6	p06	20	6	2	60
m7	p08	30	6	3	60
m8	p08	30	7	3	60
m9	p08	30	8	3	60

Tabla 3.2: Resumen de las muestras generadas.

3.2. Aplicación del modelo exacto MDVRP

Presentamos los resultamos obtenidos al aplicar el modelo exacto a los ficheros de muestra presentados en la Tabla 3.2. El modelo exacto ha sido implementado en lenguaje AMPL (`www.https://ampl.com/`) y resuelto usando el solver Gurobi Optimizer v9.0.3 (`https://www.gurobi.com/`). Los ficheros necesarios para la ejecución del modelo exacto pueden encontrarse en el Anexo A.

En la Tabla 3.3 adjuntamos los resultados obtenidos por el modelo exacto tras 3 horas de ejecución en cada muestra. En cada caso, indicamos las dimensiones del problema de programación lineal, el coste en (km) de la mejor solución alcanzada, el tiempo de ejecución necesario para alcanzar esa solución y el Gap en cada solución.

Muestra	Dimensiones Variables × Restricciones	Coste (km)	Tiempo (s)	Gap
`m1`	304×300	149.02	1.67	0.0 %
`m2`	362×343	146.52	0.187	0.0 %
`m3`	424×386	236.33	0.687	0.0 %
`m4`	1224×1156	1089.07	800.98	0.0 %
`m5`	1340×1239	921.03	36.01	0.0 %
`m6`	1460×1322	1166.98	306.48	0.0 %
`m7`	4050×3840	1767.42	10800.00	19.0 %
`m8`	4296×4024	1518.89	10800.00	10.0 %
`m9`	4548×4208	1742.77	10800.00	22.6 %

Tabla 3.3: Resultados obtenidos por el modelo exacto en los ficheros de muestra.

Vemos que el modelo exacto es capaz de obtener soluciones óptimas en casos de hasta $n = 20$ clientes en poco tiempo de ejecución. No obstante, a partir de $n = 30$ clientes sería preciso ejecutar el modelo durante más de 3 horas para intentar conseguir una solución óptima. Por tanto, descartamos su uso para casos con más de 30 clientes.

3.3. Aplicación del algoritmo heurístico

Para estudiar la calidad del algoritmo heurístico a la hora de calcular soluciones óptimas compararemos las soluciones obtenidas por el modelo exacto y el algoritmo heurístico.

Como hemos mencionado en la Sección 2.3.2 el algoritmo cuenta con dos parámetros: `MaxDestroy` y `T` que tienen que ser fijados previamente:

Para seleccionar estos parámetros haremos un estudio empírico aplicando el algoritmo heurístico a distintos ficheros de muestra para distintas combinaciones de estos parámetros. Una vez seleccionados unos parámetros óptimos ejecutaremos el algoritmo heurístico en los distintos ficheros de muestra para comparar los resultados obtenidos con el modelo exacto. Finalmente, ejecutaremos el algoritmo heurístico en las instancias presentadas en la Tabla 3.1 de las que disponemos de la mejor solución conocida hasta el momento.

El algoritmo heurístico ha sido implementado en `R`, los códigos necesarios para la ejecución de cada paso recogido en el Cuadro 2.13 están disponibles en el Anexo B.2. Además, se incluyen las funciones auxiliares necesarias para el correcto funcionamiento del algoritmo.

3.3.1. Selección de parámetros `MaxDestroy` y `T`

Esta sección incluye el estudio empírico llevado a cabo para seleccionar unos valores concretos de `MaxDestroy` y `T`. El objetivo es encontrar una combinación de estos parámetros que nos proporcione soluciones (casi)óptimas en el menor tiempo posible.

Para la selección de estos parámetros, hemos ejecutado el algoritmo heurístico en los ficheros de muestra `m3`, `m6` y `m9` considerando los siguientes valores para cada parámetro: `MaxDestroy`$\in \{3, 4, \ldots, 8\}$ y `T`$\in \{20, 30, 40, 50\}$.

Para cada combinación de estos parámetros, hemos realizado 5 ejecuciones durante un tiempo máximo de 5 minutos. En las Tablas 3.4, 3.5 y 3.6 presentamos los resultados obtenidos para las muestras `m3`, `m6` y `m9` respectivamente. En cada caso, indicamos la mejor solución obtenida, la media de las 5 soluciones y el tiempo mínimo necesario para alcanzar la mejor solución.

Para las muestras `m3` y `m6` podemos asegurar que el algoritmo heurístico calcula la solución óptima en todas las ejecuciones independientemente de los parámetros seleccionados. Para la muestra `m9`, aunque no podemos garantizar la optimalidad, observamos que el algoritmo heurístico siempre obtiene mejores resultados que la solución obtenida por el modelo exacto (Tabla 3.3).

Para la muestra `m9` hay cierta variabilidad en las soluciones obtenidas. Por tanto, analizaremos los resultados en detalle para fijar los valores de `MaxDestroy` y `T`.

En la Figura 3.1 representamos el coste de la mejor solución obtenida en función del parámetro `MaxDestroy` para los distintos valores de `T`. Vemos que, en general, la elección de `T` $= 40$ produce soluciones de mejor calidad que el resto de parámetros. De hecho, las mejores soluciones en todo el proceso se obtienen tomando `T` $= 40$ y `MaxDestroy=3` o `MaxDestroy=6`.

Si observamos el comportamiento medio de las soluciones (Figura 3.2) podemos asegurar que la elección de `T` $= 40$ y `MaxDestroy` $= 6$ produce soluciones de mejor calidad que el resto de combinaciones de estos parámetros.

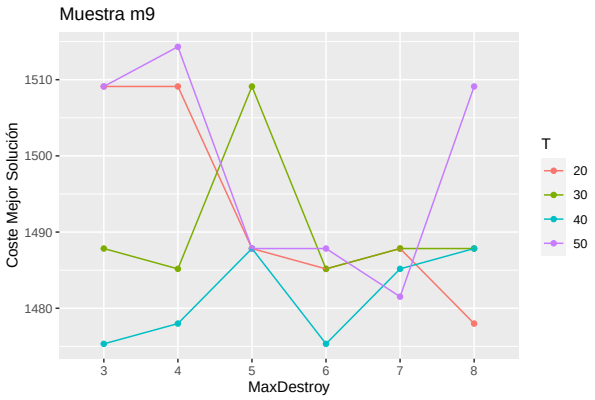


Figura 3.1: Evolución del coste de la mejor solución en función de `MaxDestroy`.

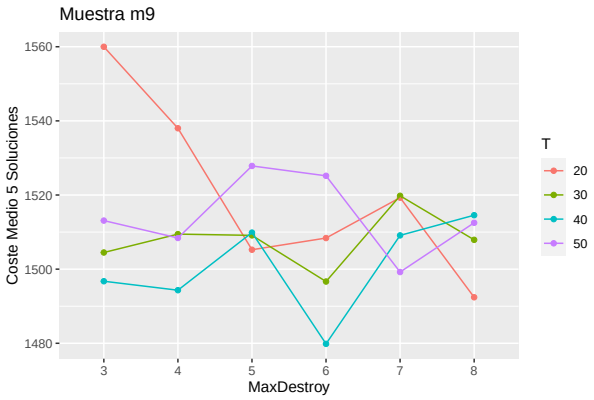


Figura 3.2: Evolución del coste medio de las soluciones en función de `MaxDestroy`.

Fichero m3		T							
		20		30		40		50	
Max.Destroy		Mejor	Media	Mejor	Media	Mejor	Media	Mejor	Media
3	Distancia	236.33	236.33	236.33	236.33	236.33	236.33	236.33	236.33
	Tiempo (s)	7.59		5.51		0.18		5.16	
4	Distancia	236.33	236.33	236.33	236.33	236.33	236.33	236.33	236.33
	Tiempo (s)	12.24		9.39		0.54		5.57	
5	Distancia	236.33	236.33	236.33	236.33	236.33	236.33	236.33	236.33
	Tiempo (s)	10.64		1.31		0.45		3.13	
6	Distancia	236.33	236.33	236.33	236.33	236.33	236.33	236.33	236.33
	Tiempo (s)	34.95		0.64		0.25		8.99	
7	Distancia	236.33	236.33	236.33	236.33	236.33	236.33	236.33	236.33
	Tiempo (s)	16.97		2.6		2.19		7.16	
8	Distancia	236.33	236.33	236.33	236.33	236.33	236.33	236.33	236.33
	Tiempo (s)	1.48		1.13		8.5		3.17	

Tabla 3.4: Resultados obtenidos por el algoritmo heurístico en 5 ejecuciones para la muestra m3.

Fichero m6		T							
		20		30		40		50	
Max.Destroy		Mejor	Media	Mejor	Media	Mejor	Media	Mejor	Media
3	Distancia	1166.98	1166.98	1166.98	1166.98	1166.98	1166.98	1166.98	1166.98
	Tiempo (s)	0.53		0.43		0.44		0.52	
4	Distancia	1166.98	1166.98	1166.98	1166.98	1166.98	1166.98	1166.98	1166.98
	Tiempo (s)	0.53		0.59		0.04		0.86	
5	Distancia	1166.98	1166.98	1166.98	1166.98	1166.98	1166.98	1166.98	1166.98
	Tiempo (s)	0.26		0.48		0.46		0.15	
6	Distancia	1166.98	1166.98	1166.98	1166.98	1166.98	1166.98	1166.98	1166.98
	Tiempo (s)	0.28		0.61		1.84		0.76	
7	Distancia	1166.98	1166.98	1166.98	1166.98	1166.98	1164.69	1166.98	1166.53
	Tiempo (s)	1.13		0.16		0.27		13.27	
8	Distancia	1166.98	1166.98	1166.98	1166.98	1166.98	1166.98	1166.98	1166.98
	Tiempo (s)	0.73		75.68		0.41		62.46	

Tabla 3.5: Resultados obtenidos por el algoritmo heurístico en 5 ejecuciones para la muestra m6.

Fichero m9		T							
		20		30		40		50	
Max.Destroy		Mejor	Media	Mejor	Media	Mejor	Media	Mejor	Media
3	Distancia	1509.11	1559.99	1487.84	1504.49	1475.33	1496.74	1509.11	1513.1
	Tiempo (s)	32.05		31.82		144.76		62.44	
4	Distancia	1509.11	1538.01	1485.17	1509.45	1477.99	1494.32	1487.84	1508.41
	Tiempo (s)	13.19		16.33		43.66		13.27	
5	Distancia	1487.84	1505.24	1509.11	1509.11	1487.84	1509.86	1487.84	1527.84
	Tiempo (s)	15.76		96.12		5.87		17.16	
6	Distancia	1485.17	1508.39	1485.17	1496.66	1475.33	1479.875	1487.84	1525.18
	Tiempo (s)	32.05		42.98		13.79		8.09	
7	Distancia	1487.84	1519.29	1487.84	1519.79	1485.17	1509.11	1481.52	1499.22
	Tiempo (s)	43.66		8.24		12.85		23.41	
8	Distancia	1477.99	1492.41	1487.84	1507.91	1487.84	1514.55	1509.11	1512.51
	Tiempo (s)	21.75		43.66		20.44		29.23	

Tabla 3.6: Resultados obtenidos por el algoritmo heurístico en 5 ejecuciones para la muestra m9.

3.3.2. Aplicación del algoritmo heurístico en muestras e instancias

Una vez seleccionados los parámetros del algoritmo, procedemos a comparar el modelo exacto y el algoritmo heurístico. Para eso, ejecutamos el algoritmo en cada fichero de muestra durante un máximo de 5 minutos. En la Tabla 3.7 comparamos los resultados obtenidos por ambos métodos usando el error absoluto $e_{\text{abs}} = z_{\text{exacto}} - z_{\text{heur}}$ y el error relativo $e_{\text{rel}} = 100 \cdot e_{\text{abs}}/z_{\text{exacto}}$. Adjuntamos también el tiempo mínimo que requiere el algoritmo para llegar a su solución.

Muestra	Exacto		Heurístico		Errores	
	Coste	Tiempo (s)	Coste	Tiempo (s)	Absoluto	Relativo
`m1`	149.02	1.67	149.02	0.865	0	0
`m2`	146.52	0.187	146.52	1.16	0	0
`m3`	236.33	0.687	236.33	4.36	0	0
`m4`	1089.07	800.98	1089.07	24.14	0	0
`m5`	921.03	36.01	921.03	0.46	0	0
`m6`	1166.98	306.48	1166.98	1.12	0	0
`m7`	1767.42	10800	1611.39	295.85	-156.03	-8.83
`m8`	1518.89	10800	1611.48	254.62	92.59	6.09
`m9`	1742.77	10800	1479.87	43.44	-262.9	-13.41

Tabla 3.7: Comparación modelo exacto y algortimo heurístico en los ficheros de muestra.

Para las muestras de las que se dispone de solución óptima (muestras `m1`, ..., `m6`), el algoritmo heurístico es capaz de calcular dicha solución en un tiempo inferior a 30 segundos. En las muestras `m7`, `m9` el algoritmo mejora los resultados ofrecidos por el modelo exacto. No obstante, no podemos garantizar la optimalidad en ningún caso. En cuanto a la muestra `m8` el algoritmo exacto ofrece mejores resultados que el heurístico.

En cualquier caso, vemos que el algoritmo heurístico es capaz de calcular soluciones óptimas en poco tiempo computacional. No obstante, para ficheros con más de 30 clientes es necesario incrementar el tiempo de ejecución para asegurar soluciones de calidad.

Finalmente, ejecutamos el algoritmos heurístico en las instancias presentadas en la Tabla 3.1 y comparamos las soluciones obtenidas con las mejores soluciones de las que se tiene constancia. En este caso, hemos ejecutado cada instancia durante 15 minutos.

En la Tabla 3.8 presentamos los resultados obtenidos para cada instancia. En cada caso, indicamos el coste de la solución inicial factible desde la que comienza el algoritmo, la mejor solución obtenida y el tiempo mínimo necesario para alcanzar la solución final. Comparamos los resultados obtenidos con la mejor solución conocida calculando el error absoluto y relativo en cada caso.

Instancia	Heurístico			Mejor	Errores	
	Solución Inicial	Solución Final	Tiempo (s)	Solución	Absoluto	Relativo
p01	744.97	576.87	410.64	576.87	0	0
p02	758.6	473.53	833.94	473.53	0	0
p03	906.37	641.19	219.77	641.19	0	0
p04	1284.14	1057.12	594.8	1001.59	55.53	5.25
p05	1143.34	757.49	405.15	750.08	7.41	0.98
p06	1220.68	916.52	343.4	876.5	40.02	4.37
p07	1200.46	946.48	896.15	885.8	60.68	6.41
p08	6217.77	4746.65	541.25	4437.68	308.97	6.51
p09	6058.3	4314.14	717.2	3900.12	414.02	10.61
p10	5196.5	3960.66	460.18	3663.02	297.64	7.51
p11	10405.14	3857.47	424.72	3554.18	303.29	7.86
p12	3271.32	1346.6	664.87	1318.95	27.65	2.05
p13	3271.32	1346.6	668.62	1318.95	27.65	2.05
p14	2086.37	1318.95	572.96	1360.12	-41.17	-3.12
p15	8613.87	2753.66	823.38	2505.42	248.24	9.01
p16	8613.87	2753.66	818.43	2572.23	181.43	6.59
p17	8613.87	2753.66	820.43	2709.09	44.57	1.62
p18	7996.68	4264.77	909.5	3702.85	561.92	13.18
p19	7996.68	4237.59	904.41	3827.85	409.74	9.67
p20	16249.35	4360.93	873.94	4058.07	302.86	6.94
p21	28585.89	7101.05	511.02	5474.84	1626.21	22.9
p22	28585.89	7101.05	506.74	5702.16	1398.89	19.7
p23	28585.89	7101.05	502.27	6095.46	1005.59	14.16
pr01	1376.17	870.59	23.49	861.32	9.27	1.06
pr02	3207.04	1332.87	667.92	1307.61	25.26	1.9
pr03	5134.11	1925.01	84.97	1806.6	118.41	6.15
pr04	5985.54	2238.17	474.33	2072.52	165.65	7.4
pr05	5776.7	2589.85	506.85	2385.77	204.08	7.88
pr06	8272.55	2910.48	827.84	2723.27	187.21	6.43
pr07	1620.61	1084.88	167.43	1089.56	-4.68	-0.43
pr08	4729.84	1753.53	402.43	1666.6	86.93	4.96
pr09	7000.85	2336.77	856.27	2153.1	183.67	7.86
pr10	5426.26	3176.62	697.14	2921.85	254.77	8.02

Tabla 3.8: Resultados obtenidos por el algoritmo heurístico en las instancias.

El algoritmo heurístico solo es capaz de calcular la solución óptima en las instacias p01, p02 y p03 en las que el número de clientes es de 50 (instancias p01 y p02) y 75 para la p03. Cuando el número de clientes aumenta el algoritmo no es capaz de alcanzar la mejor solución en el tiempo establecido, no obstante, el error relativo suele ser inferior al 10 % salvo para las instancias p21 (22.9 %), p22 (19.7 %) y p23 (14.6 %) las cuales cuentan con 360 clientes y 9 depósitos cada una.

Cabe remarcar que para las instancias pr07 y p14 el algoritmo es capaz de mejorar la mejor solución conocida ligeramente. No obstante, estas instancias presentan una restricción adicional que nosotros no consideramos. Esta restricción limita el tiempo máximo que puede durar una ruta.

Observando el coste de la solución inicial, vemos que en algunos casos es considerablemente superior al de la solución final (en particular, en las instancias p21, p22 y p23 la solución inicial tiene un coste 400 % superior al de la solución final). Esto nos hace pensar que una primera corrección del algoritmo incluiría la revisión y mejora de la construcción de la solución inicial.

3.4. Aplicación del algoritmo: un enfoque cooperativo

Finalizamos la memoria aplicando el algoritmo heurístico a un MDVRP en el que se permite la cooperación entre entre empresas. Para ello, hemos seleccionado las instancias `p01` y `p19` de forma independiente.

Para cada instancia, supondremos que cada depósito pertenece a una empresa distinta las cuales deciden cooperar para intentar abaratar los costes globales de reparto de un determinado producto. En cada caso, asignamos cada cliente a la empresa que esté más próxima y asignamos los vehículos necesarios para satisfacer la demanda. En la Tabla 3.9 indicamos los recursos de los que dispone cada empresa y las necesidades de los clientes a los que tiene que atender.

Instancia	Empresas	Número clientes	Demanda total	Número vehículos	Capacidad
p01	1	13	205	4	80
	2	17	262	4	80
	3	11	177	4	80
	4	9	133	4	80
p19	1	40	216	5	60
	2	40	216	5	60
	3	40	216	5	60
	4	40	216	5	60
	5	40	216	5	60
	6	40	216	5	60

Tabla 3.9: Datos CoMDVRP.

Para cada instancia, calculamos el juego TU asociado suponiendo que cada jugador $p \in P$ representa una empresa. La función característica se obtiene al resolver el modelo MDVRP para cada coalición $S \subset P$ suponiendo que cuando dos jugadores forman una coalición los recursos y necesidades a atender son comunes.

Para calcular la función característica, hemos ejecutado el algoritmo heurístico para cada una de las posibles coaliciones $S \subset P$. Para cada coalición, hemos fijado el tiempo de ejecución de forma proporcional al número de clientes que tenga que atender dicha coalición. En particular, hemos ejecutado el algoritmo por un minuto por cada 10 clientes que tengan que ser atendidos. Así, para la gran coalición de todos los jugadores en la instancia `p19` hemos ejecutado el algoritmo durante 24 minutos.La función característica obtenida para cada juego se recoge en las Tablas 3.10 y 3.11.

Finalmente, para cada juego hemos aproximado el valor de Shapley usando el procedimiento descrito en la Sección 1.2.2 obteniendo

- Instancia `p01`: $\Phi_{\texttt{p01}}(c) = (142.49, 159.47, 118.59, 153.32)$
- Instancia `p19`: $\Phi_{\texttt{p19}}(c) = (660.06, 655.53, 655.18, 603.68, 660.63, 816.06)$.

Coalición	Coste
$\{1\}$	280.07
$\{2\}$	270.23
$\{3\}$	232.98
$\{4\}$	292.93
$\{1,2\}$	407.59
$\{1,3\}$	347.46
$\{1,4\}$	386.74
$\{2,3\}$	403.07
$\{2,4\}$	426.4
$\{3,4\}$	376.73
$\{1,2,3\}$	479.85
$\{1,2,4\}$	500.82
$\{1,3,4\}$	470.08
$\{2,3,4\}$	484.84
$\{1,2,3,4\}$	576.87

Tabla 3.10: Función característica del juego asociado a la instancia `p01`.

Coalición	Coste	Coalición	Coste	Coalición	Coste
$\{1\}$	809.23	$\{1,2,3\}$	2140.76	$\{1,2,3,4\}$	2738.46
$\{2\}$	807.86	$\{1,2,4\}$	2228.93	$\{1,2,3,5\}$	3019.5
$\{3\}$	805.94	$\{1,2,5\}$	2307.33	$\{1,2,3,6\}$	2861.67
$\{4\}$	805.94	$\{1,2,6\}$	2322.82	$\{1,2,4,5\}$	2907.52
$\{5\}$	809.23	$\{1,3,4\}$	2315.62	$\{1,2,4,6\}$	3017.34
$\{6\}$	1195.8	$\{1,3,5\}$	2471.96	$\{1,2,5,6\}$	2987.06
$\{1,2\}$	1466.69	$\{1,3,6\}$	2390.84	$\{1,3,4,5\}$	2945.78
$\{1,3\}$	1617.1	$\{1,4,5\}$	2283.35	$\{1,3,4,6\}$	3033.89
$\{1,4\}$	1604.85	$\{1,4,6\}$	2350.6	$\{1,3,5,6\}$	2929.64
$\{1,5\}$	1616.94	$\{1,5,6\}$	2323.54	$\{1,4,5,6\}$	2888.42
$\{1,6\}$	1767.98	$\{2,3,4\}$	2162.02	$\{2,3,4,5\}$	2769.4
$\{2,3\}$	1489.66	$\{2,3,5\}$	2325.99	$\{2,3,4,6\}$	2966.92
$\{2,4\}$	1613.81	$\{2,3,6\}$	2395.33	$\{2,3,5,6\}$	3004.94
$\{2,5\}$	1676.89	$\{2,4,5\}$	2276.02	$\{2,4,5,6\}$	2932.43
$\{2,6\}$	1783.66	$\{2,4,6\}$	2448.28	$\{3,4,5,6\}$	2855.9
$\{3,4\}$	1475.71	$\{2,5,6\}$	2316.24	$\{1,2,3,4,5\}$	3500.67
$\{3,5\}$	1615.18	$\{3,4,5\}$	2131.75	$\{1,2,3,4,6\}$	3566.63
$\{3,6\}$	1767.24	$\{3,4,6\}$	2326.24	$\{1,2,3,5,6\}$	3712.37
$\{4,5\}$	1441.96	$\{3,5,6\}$	2360.01	$\{1,2,4,5,6\}$	3516.93
$\{4,6\}$	1784.46	$\{4,5,6\}$	2271.17	$\{1,3,4,5,6\}$	3534.36
$\{5,6\}$	1781.01			$\{2,3,4,5,6\}$	3594.71
				$\{1,2,3,4,5,6\}$	4051.33

Tabla 3.11: Función característica del juego asociado a la instancia `p19`.

Apéndice A

Códigos AMPL

En esta sección incluimos el código `AMPL` necesario para ejecutar el modelo (MDVRP) presentado en la Sección 2.1.

El modelo de programación lineal se incluye en el fichero `MDVRP.mod` y para su aplicación es necesario ejecutar el fichero `MDVRP.run` el cual guarda los resultados. Incluimos un fichero de datos en el fichero `MDVRP.dat`.

MDVRP.mod

```
set I;  # Depots
set J;  # Customers
set K;  # Vehicles

param N; # Number of customers
param C{i in I union J, j in I union J} >= 0; # Travel cost from i to j
param V{i in I}; # Maximum throughput at i
param d{j in J}; # Demand of customer j
param Q{k in K}; # Capacity of vehicle (route) k

var x{i in I union J,j in I union J,k in K:i<>j} binary;
var z{i in I, j in J} binary;
var U{i in J, k in K}>=0;

minimize TotalCost:
sum{i in I union J,j in I union J,k in K:i<>j} C[i,j]*x[i,j,k];

subject to Constraint1 {j in J}:
sum{k in K, i in I union J:i<>j}x[i,j,k] = 1;

subject to Constraint2 {k in K}:
sum{j in J}d[j]*sum{i in I union J:i<>j}x[i,j,k] <= Q[k];

subject to Constraint3 {i in J, j in J, k in K:i<>j}:
U[i,k] + U[j,k] + N*x[i,j,k] <= N+1;

```

```
subject to Constraint4 {k in K, i in I union J}:
sum{j in I union J:i<>j}x[i,j,k]+sum{j in I union J:i<>j}x[j,i,k]=0;

subject to Constraint5 {k in K}:
sum{i in I,j in J}x[i,j,k] <=1;

subject to Constraint6 {i in I}:
sum{j in J}d[j]*z[i,j]<=V[i];

subject to Constraint7 {i in I, j in J, k in K}:
+z[i,j] + sum{u in I union J:u<> i and u<>j}(x[i,u,k]+x[u,j,k])<=1;

subject to Constraint9 {i in I, j in J, k in K}:
+z[i,j] + x[i,j,k]<=0;
```

MDVRP.run

```
reset;
model MDVRP_run.txt;
data MDVRP_dat.dat;
option solver gurobi_ampl;
solve;
display _varname, _var;

display TotalCost;
display _nvars;
display _ncons;
```

MDVRP.dat

```
set I := D1 D2;
set J := C1 C2 C3 C4 C5 C6;
set K := K1 K2;

param N := 6;

param C: D1 D2  C1  C2  C3  C4  C5  C6:=
D1  100 100 0.25    1.03    2.02    0.75    1.25    2.14
D2  100 100 1.25    0.75    1.25    1.03    0.25    1.03
C1  0.25    1.25    100 1.00    2.00    0.5 1.12    2.06
C2  1.03    0.75    1.00    100 1.00    1.12    0.5 1.12
C3  2.02    1.25    2.00    1.00    100 2.06    1.12    0.5
C4  0.75    1.03    0.5 1.12    2.06    100 1.00    2.00
C5  1.25    0.25    1.12    0.5 1.12    1.00    100 1.00
C6  2.14    1.03    2.06    1.12    0.5 2.00    1.00    100;


param V:=
D1 260
D2 260;

```

```
22 param d:=
23 C1 30
24 C2 50
25 C3 30
26 C4 40
27 C5 70
28 C6 40;
29
30 param Q:=
31 K1 120
32 K2 140;
```

Apéndice B

Códigos R

Esta sección incluye los códigos de R utilizados en la memoria para calcular el valor de Shapley aproximado como se describe en la Sección 1.2.2 (Anexo B.1) y el algoritmo heurístico presentado en la Sección 2.3.2 (Anexo B.2)

B.1. Valor de Shapley

La función `Shapley_est(v,m)` incluida en el fichero `Shapley_est.R` permite calcular el valor de Shapley de un juego cooperativo v usando m remuestras.

Las funciones auxiliares para el correcto funcionamiento se incluyen al final de la sección en el fichero `aux_FUN_Shapley.R`

Shapley_est.R

```r
#-- Shapley_est --#

Shapley_est <- function(v,m){

  n <- log(length(v), 2)
  if (n == round(n)) {
    v <- v[-1]
  }
  n <- round(n)

  res <- matrix(nrow=m,ncol=n)

  # Obtencion coaliciones
  aux <- lapply(1:n, function(x){combn(n,x)})      # Combinaciones de i jugadores, i = 1..n
  coalitions <- unlist(lapply(aux,aux.coalitions)) # Coaliciones posibles ordenadas y en texto

  # Valor de Shapley estimado para cada permutacion

  for(j in 1:m){
    O <- sample(n)
    res[j,] <- aux.Shapley(O,v,coalitions)
  }
  return(colMeans(res))
}

###############
#-- Ejemplo --#
###############


# # Definicion del juego
# n <- 8
# m <- 1e3
# players <- 1:n
# v <- numeric(2^n-1)
# v[(sum(choose(n,1:(n/2)))+1):length(v)] <- 1
# sol <- Shapley_est(v,m)
```

aux_FUN_Shapley.R

```r
##################################
##################################
library(cmpfun)
aux.coalitions <- function(S){
res <- numeric(ncol(S))
for(i in 1:ncol(S)){
res[i] <- paste(S[,i],sep="",collapse=",")
}
return(res)
}
aux.coalitions <- cmpfun(aux.coalitions)

####################################
####################################

aux.Shapley <- function(O,v,coalitions){

n <- log(length(v), 2)
if (n == round(n)) {
v <- v[-1]
}
n <- round(n)

marg <- numeric(n)
players <- c()
for(i in 1:n){
players <- sort(c(players,O[i]))
players.aux <- paste(players,sep="",collapse=",")
marg[O[i]] <- v[which(players.aux==coalitions)] - sum(marg)
}
return(marg)
}
aux.Shapley <- cmpfun(aux.Shapley)
####################################
####################################
```

B.2. Algoritmo heurístico

La función `IG_MDVRP(File,Param)` incluida en el archivo `IG_MDVRP.R` ejecuta el algoritmo heurístico descrito en la Sección 2.3.2.

Para eso, los datos correspondientes al problema de rutas deberán estar en un fichero (`File`) conteniendo la siguiente información:

- `V:` Vector de tamaño d cuya componente $i-$ésima representa la capacidad del depósito $i \in I$.
- `Q:` Vector de tamaño $|K|$ cuya componente $k-$ésima representa la capacidad del vehículo $k \in K$.
- `demands:` Vector de tamaño $n+d$ cuyas d primerascomponentes son 0 por tratarse de los depósitos y el resto indican la demanda de cada cliente.
- `matrix.cost:` Matriz de tamaño $(n+d) \times (n+d)$ cuya componente i, j representa el coste de ir del nodo $i \in I \cup J$ al nodo $j \in I \cup J$

El fichero `Param` incluye la siguiente información necesaria para ejecutar el algoritmo:

- `MaxTime:` Tiempo máximo permitido para que se ejecute el algoritmo.
- `Max.Destroy:` Parámetro que controla el número máximo de clientes a eliminar durante la fase *Ruin*.
- `T:` Temperatura usada para calcular la probabilidad de aceptación.

Los archivos `ClarkeWrightMDVRP.R`,`asignacion.vehiculos.R` ,`Correct.Feasibility.R` contienen las funciones necesarias para calcular la solución inicial del algoritmo.

Las funciones auxiliares para el correcto funcionamiento de estas funciones se incluyen en el archivo `aux_FUN_IG.R`

IG_MDVRP.R

```
IG_MDVRP <- function(File,Param){

 # Datos del fichero
V <<- File$V
Q <<- File$Q
demands <<- File$demands
matrix.cost <<- File$matrix.cost
Q <<- sort(Q,decreasing=T)
d <<- length(V)
n <<- length(demands)-d
k <<- length(Q)

# Datos de los parametros
MaxTime <- Param$MaxTime
 Max.destroy <- Param$ Max.destroy
Temp <- Param$Temp

# Obtencion solucion inicial CW
ini1 <- proc.time()[3]
sol.initial.CW <- ClarkeWrightMDVRP(File)
fin1 <- proc.time()[3]-ini1

# Correccion factibilidad
ini2 <- proc.time()[3]
sol.initial <- correct.feasibility(sol.initial.CW)
fin2 <- proc.time()[3]-ini2

# IG

currentSol <- bestSol <- sol.initial

ini3 <- proc.time()[3]
iter <- iter2 <- 0
repeat{
#--- Fase Ruin
aux.Sol <- currentSol
eliminar <- sample((d+1):(n+d), Max.destroy)
currentSol <- lapply(currentSol,eliminate.clients,eliminated=eliminar)

#--- Fase Recreate
newSol <- reconstruct2(currentSol,eliminar)
currentSol <- aux.Sol
#--- Aceptacion
aux1 <- min(Q-sapply(newSol, carga.ruta))
if(aux1>=0){
a1 <- sum(sapply(newSol,coste.ruta))
b1 <- sum(sapply(currentSol,coste.ruta))
dif <- a1-b1
# newSol vs. currentSol

cond.cambio <- dif<=0

# Solo en caso de que newSol sea mejor que currentSol se podra actualizar la bestSol
if (cond.cambio) {
currentSol <- newSol
#  newSol vs. bestSol
c1 <- sum(sapply(bestSol,coste.ruta))

if(a1-c1<0){bestSol <- newSol;iter2 <- iter}
}
# En caso contrario de podemos actualizar la currentSol con una probabilidad exp(-beta/T)

if (!cond.cambio){if(runif(1)<=exp(-dif/Temp)){currentSol <- newSol}}
}
iter <- iter+1
end <- proc.time()[3]
if((end-ini3)>MaxTime){break}
return(bestSol)
}
```

ClarkeWrightMDVRP.R

```
ClarkeWrightMDVRP <- function(File){
  V <- File$V
  Q <- File$Q
  Q <- sort(Q,decreasing=T)
  demands <- File$demands
  matrix.cost <- File$matrix.cost

  d <- length(V)
  n <- length(demands)-d
  k <- length(Q)

  ## PreAsignacion de clientes a depositos
  pre.assign <- numeric(n)
  cost.ini <- numeric(n)
  for(i in 1:n){
    pre.assign[i] <- which.min(matrix.cost[i+d,1:d])
    cost.ini[i] <- 2*min(matrix.cost[i+d,1:d])
  }

  ## Distancia "real" clientes-depositos

  d.hat <- matrix(0,nrow=d,ncol=n) # Distancia real clientes-deposiitos
  for(i in 1:d){
    for(j in 1:n){
      d.hat[i,j] <- 2*min(matrix.cost[1:d,j+d])-matrix.cost[i,j+d]
    }
  }

  S.depots <- list()
  for(i in 1:d){
    S.depots[[i]] <- matrix(0,nrow=n,ncol=n)
    for(j in 1:n){
      for(k in 1:n){
        if(j!=k){
          S.depots[[i]][j,k] <- d.hat[i,j]+d.hat[i,k]-matrix.cost[j+d,k+d]
        }
      }
    }
  }

  # Rutas originales (0,i,0) considerando que cada cliente va al deposito mas cercano
  R <- matrix(0,nrow=n,ncol=3)
  R[,2] <- (1:n)+d
  R[,c(1,3)] <- pre.assign
  R.orig <- R

  # Clarke and Wright MDVRP

  Sm <- 1
  while(Sm>0){
    max.savings <- sapply(S.depots,max)

    Sm <- max(max.savings)
    if(Sm>0){
      depot.prov <- which.max(max.savings)

      client.i <- matxMax(S.depots[[depot.prov]])[1]
      client.j <- matxMax(S.depots[[depot.prov]])[2]

      CargaT <- demands[client.i+d]+demands[client.j+d] # Demanda de los clientes i y j

      { #Indicamos a que cliente visitamos antes de ir a i y despues de ir a j
        if(R[client.i,3]==depot.prov && R[client.j,1]==depot.prov && CargaT<=max(Q) && CargaT<=V[
            depot.prov]){
          client.i_new<-client.i
          client.j_new<-client.j
          x<-0 #Evitamos ciclos
          while(R[client.i_new,1]!=depot.prov){ #Sumamos la carga de los clientes anteriores a i
            CargaT<-CargaT+demands[R[client.i_new,1]]
            client.i_new<-R[client.i_new,1]-d
            if(client.i_new==client.j) x<-x+1
          }
          while(R[client.j_new,3]!=depot.prov){ #Sumamos la carga de los clientes posteriores a j
            CargaT<-CargaT+demands[R[client.j_new,3]]
            client.j_new<-R[client.j_new,3]-d
```

```
            if(client.j_new==client.i) x<-x+1
          }
          if(CargaT<=max(Q) && CargaT <= V[depot.prov] && x==0){ #A~nadimos la ruta si es factible
            R[client.i,3]<-client.j+d
            R[client.j,1]<-client.i+d
          }

          for(i in 1:d){ #Borramos ahorros utilizados para evitar ciclos
            S.depots[[i]][c(client.i,client.j),c(client.i,client.j)]<-0
          }
        }
      }
      for(i in 1:d){ #Borramos ahorros utilizados para evitar ciclos
        S.depots[[i]][c(client.i,client.j),c(client.i,client.j)]<-0
      }
    }
  }

  # Organizacion de las rutas
  rutas <- vector("list",length=d)
  for(i in 1:d){
    rutas[[i]][1] <-i
  }

  ind <- rep(2,d)

  for(i in 1:d){
    for(j in 1:n){
      if(R[j,1]==i){
        rutas[[i]][ind[i]] <- j+d

        while(rutas[[i]][ind[i]]!=i){
          rutas[[i]][ind[i]+1]<-R[rutas[[i]][ind[i]]-d,3]
          ind[i]<-ind[i]+1
        }
        ind[i]<-ind[i]+1
      }
    }
  }
  # Pasamos las rutas de los depositos a los camiones

  u <- lapply(rutas,eliminate.depot)
  rutas.vehiculos <- do.call(list, unlist(u, recursive=F))
  sol <- asignacion.vehiculos(rutas.vehiculos,Q)

  return(sol)
}
```

asignacion.vehiculos.R

```r
asignacion.vehiculos <- function(rutas.prev,Q){
  n.rutas <- length(rutas.prev)
  n.camiones <- length(Q)

  rutas.vehiculos <- vector("list",length=n.camiones)
  rutas.aux <- rutas.prev

  routes.demands <- sapply(rutas.prev,carga.ruta)
  rutas.ord <- rutas.prev[order(routes.demands,decreasing = T)]

  if(n.rutas>n.camiones){
    route.fusion <- unlist(rutas.ord[n.camiones:n.rutas])
    route.fusion <- c(route.fusion[1],route.fusion[!route.fusion %in% 1:d],route.fusion[1])
    rutas.vehiculos[1:(n.camiones-1)] <- rutas.ord[1:(n.camiones-1)]
    rutas.vehiculos[[n.camiones]] <- route.fusion
  } else{
    rutas.vehiculos[1:n.rutas] <- rutas.ord[1:n.rutas]
  }

  res <- rutas.vehiculos[order(sapply(rutas.vehiculos,carga.ruta),decreasing = T)]
  return(res)
}
asignacion.vehiculos <- cmpfun(asignacion.vehiculos)
```

Correct.Feasibility.R

```
correct.feasibility <- function(sol.initial){
  currentSol <- sol.initial
  bestSol <- sol.initial
  auxIni <- min(Q-sapply(currentSol,carga.ruta))

  if(auxIni<0){ # Si la solucion inicial de partida no es factible
    repeat{
      aux <- Q-sapply(currentSol,carga.ruta)
      r.1 <- nth(aux,1,index.return = T)

      clients.1 <- currentSol[[r.1]]
      aux2 <- sort(demands[clients.1],decreasing=T)
      pc <- max(which(cumsum(aux2) <= Q[r.1]))
      v <- sapply((pc+1):(length(clients.1)-2),nth,x=demands[clients.1],index.return = T,descending=T
          )
      eliminar <- clients.1[v]

      currentSol[[r.1]] <- eliminate.clients(currentSol[[r.1]],eliminar)
      newSol <- reconstruct2(currentSol,eliminar)
      currentSol[[r.1]] <- clients.1

      auxIni2 <- min(Q-sapply(newSol, carga.ruta))
      if(auxIni2>=0){currentSol <- newSol;bestSol <- newSol;sol.initial2 <- newSol;break
      }else{
        x1 <- auxIni2-auxIni
        if(x1>=0){
          currentSol <- newSol
          aux.bestSol <- sapply(bestSol, coste.ruta)
          auxIni3 <-  min(aux.bestSol)
          y1 <- auxIni2-auxIni3
          if(y1>=0){bestSol <- newSol}
        }
      }
    }
  }
  return(bestSol)
}
```

aux_FUN_IG.R

```
1  ##############################################################################
2  ##############################################################################
3
4  resamp <- function(x,alpha){if(length(x)==1) x else sample(x,size=alpha)} #FUNCION SAMPLE SIN
       SORPRESAS
5  resamp <- cmpfun(resamp)
6
7  ##############################################################################
8  ##############################################################################
9
10 insertElems = function(vect, pos, elem) {
11   l = length(vect)
12   if (pos==1)
13   vect = c(elem, vect)
14   else if (pos == l+1)
15   vect = c(vect, elem)
16   else
17   vect = c(vect[1:(pos-1)], elem, vect[pos:l])
18   return(vect)
19 }
20 insertElems <- cmpfun(insertElems)
21
22 ##############################################################################
23 ##############################################################################
24
25 reconstruct <- function(ruta, eliminated){
26   while(length(eliminated)>0){
27     j <- eliminated[1]
28     aux <- carga.ruta(ruta)
29
30     if(aux==0){
31       rutas.posibles <- lapply(1:d,function(x){c(x,j,x)})
32       idx <- which.min(sapply(rutas.posibles,coste.ruta))
33       ruta <- rutas.posibles[[idx]]
34     }
35
36     if(aux!=0){
37       n.clients <- length(ruta)
38
39       # Todas las inserciones posibles
40       pos.possibles <- 2:n.clients
41       rutas.posibles <- vector("list",n.clients-1)
42       for(i in 1:(n.clients-1)){
43         rutas.posibles[[i]] <- insertElems(ruta,pos.possibles[i],j)
44       }
45
46       # Elegimos la ruta mas barata
47       ruta.elegida <- which.min(sapply(rutas.posibles,coste.ruta))
48       ruta <- rutas.posibles[[ruta.elegida]]
49     }
50     eliminated <- setdiff(eliminated,j)
51   }
52   return(ruta)
53 }
54 reconstruct <- cmpfun(reconstruct)
55
56 ##############################################################################
57 ##############################################################################
58
59 carga.ruta <- function(x){sum(demands[x])}
60 carga.ruta <- cmpfun(carga.ruta)
61
62 ##############################################################################
63 ##############################################################################
64
65 eliminate.depot <- function(x){
66   depot <- x[1]
67   pos <- which(x==depot)
68   n.visits <- length(pos)
69   routes <- vector("list",length=(length(pos)-1))
70
71   if(n.visits>2){
72     for(i in 1:(n.visits-1)){
73       routes[[i]] <- c(depot,x[(pos[i]+1):(pos[i+1]-1)],depot)
74     }
```

```
  } else{
    routes <- list(x)
  }
  return(routes)
}
eliminate.depot <- cmpfun(eliminate.depot)

#############################################################################
#############################################################################

coste.ruta <- function(ruta){
  n <- length(ruta)
  coste <- 0
  if(n>0){
    for(i in 1:(n-1)){
      coste <- coste + matrix.cost[ruta[i],ruta[i+1]]
    }
  }
  return(coste)
}
coste.ruta <- cmpfun(coste.ruta)
#############################################################################
#############################################################################

matxMax <- function(matrix){
  m <- nrow(matrix)
  aux <- which(matrix == max(matrix))

  col <- ((aux-1) %/% m)[1]
  fil <- (aux%% m)[1]
  colmn <- col+1

  if(col==m){colmn <- m}
  if(fil==0){fil <- m}

  return( matrix(c(fil, colmn), 1))
}
matxMax <- cmpfun(matxMax)

#############################################################################
#############################################################################

reconstruct2 <- function(solution,eliminated){

  while(length(eliminated)>0){
    j <- eliminated[1]
    n <- length(solution)

    parent.aux <- solution
    u <- numeric(n)
    v <- numeric(n)
    possible.insertions <- lapply(solution,reconstruct,j)

    for(i in 1:n){
      parent.aux[[i]] <- possible.insertions[[i]]
      u[i] <- sum(sapply(parent.aux,coste.ruta))
      v[i] <- min(Q-sapply(parent.aux, carga.ruta))
      parent.aux <- solution
    }

    idx.fact <- which(v>=0)

    if(length(idx.fact>1)){
      idx.best <- idx.fact[which.min(u[idx.fact])]
    }else{
      idx.best <- which.max(v)
    }

    solution[[idx.best]] <- possible.insertions[[idx.best]]
    eliminated <- setdiff(eliminated,j)
  }

  return(solution)
}
reconstruct2 <- cmpfun(reconstruct2)
```

```
152 ##############################################################################
153 ##############################################################################
154
155 eliminate.clients <- function(x,eliminated){
156   x <- x [! x %in% eliminated]
157   return(x)
158 }
159
160 eliminate.clients <- cmpfun(eliminate.clients)
161
162 ##############################################################################
163 ##############################################################################
164
165 organizar.rutas <- function(ruta,d){
166   n <- length(ruta)
167   if(n>2){
168     depot <- ruta[1]
169     clients <- ruta[2:(n-1)]
170     sol <- paste(c(paste("D",depot,sep=""),paste("C",clients-d,sep=""),paste("D",depot,sep="")),
          collapse="-")
171   }
172   if(n<=2){
173     sol <- "NULL"
174   }
175   return(sol)
176 }
177 organizar.rutas <- cmpfun(organizar.rutas)
```

Bibliografía

Agarwal, Y., Mathur, K., Salkin, H. M. (1989). "A set-partitioning-based exact algorithm for the vehicle routing problem". *Networks*, 19 (7), 731–749.

Araque G., J. R., Kudva, G., Morin, T. L., Pekny, J. F. (1994). "A branch-and-cut algorithm for vehicle routing problems". *Annals of Operations Research*, 50 (1), 37–59.

Beasley, J. (1983). "Route first–Cluster second methods for vehicle routing". *Omega*, 11 (4), 403–408.

Castro, J., Gómez, D., Tejada, J. (2009). "Polynomial calculation of the Shapley value based on sampling". *Computer and Operations Research*, 36 (5), 1726–1730.

Carpaneto, G., Dell'Amico, M., Fischetti, M., Toth, P. (1989). "A branch and bound algorithm for the multiple depot vehicle scheduling problem". *Networks*, 19 (5), 531–548.

Christofides, N., Mingozzi, A., Toth, P. (1979). "The vehicle routing problem". Combinatorial Optimization, Wiley, Chichester, UK.

Christofides, N., Mingozzi, A., Toth, P. (1981). "Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations". *Mathematical Programming*, 20 (1), 255–282.

Clarke, G., Wright, J. R. (1964). "Scheduling of vehicle routing problem from a central depot to a number of delivery points". *Operations Research*, 12, 568–581.

Cordeau, J.F., Gendreau, M., Laporte, G. (1997). "A tabu search heuristic for periodic and multi-depot vehicle routing problems". *Networks*, 30 (2), 105–119.

Crevier, B., Cordeau, J. F., Laporte, G. (2007). "The multi-depot vehicle routing problem with inter-depot routes". *European Journal of Operational Research*, 176 (2), 756–773.

Croes, G. A. (1958). "A method for solving traveling-salesman problems. *Operations Research*, 6 (6), 791–812.

Dantzig, G. B., Ramser, R. H. (1959). "The truck dispatching problem". *Management Science*, 6 (1), 80–91.

Du, J., Li, X., Yu, L., Dan, R., Zhou, J. (2017). "Multi-depot vehicle routing problem for hazardous materials transportation: A fuzzy bilevel programming". *Information Sciences*, 399, 201–218.

Fiestras-Janeiro, M. G., García-Jurado, I., Meca, A., Mosquera, M. A. (2014). "Centralized inventory in a farming community." *Journal of Business Economics*, 84, 983–997.

Fiestras-Janeiro, M. G., García-Jurado, I., Mosquera, M. A. (2011). "Cooperative games and cost allocation problems". *TOP*, 19, 1–22.

Flisberg, P., Lidén, B., Rönnqvist, M. (2009). "A hybrid method based on linear programming and tabu search for routing of logging trucks". *Computers & Operations Research*, 36 (4), 1122–1144.

Gillies, D. B. (1953). "Some theorems on n-person games", Ph.D. Dissertation, Princeton University Press, New Jersey.

Gillet, B. E., Miller, L. R. (1974). "A heuristic algorithm for the vehicle-dispatch problem". *Operations Research*, 22 (2), 340–349.

González, P., Herrero, C. (2004). "Optimal sharing of surgical costs in the presence of queues." *Mathematical Methods of Operations Research*, 59, 435–446.

Glover, F. (1986). "Future paths for integer programming and links to artificial intelligence". *Computers & Operations Research*", 13 (5), 533–549.

Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P. (1983). "Optimization by Simulated Annealing". *Science*, 220 (4598), 671–680.

Kulkarni, R. V., Bhave, P. R. (1985). "Integer programming formulations of vehicle routing problems". *European Journal of Operational Research*, 20 (1), 58–67.

Laporte G., Nobert, Y., Taillefer, S. (1988). "Solving a family of multi-depot vehicle routing and location-routing problems". *Transportation Science*, 22 (3), 161–172.

Lenstra, J. K., Kan, A. H. G. R. (1981). "Complexity of vehicle routing and scheduling problems". *Networks*, 11 (2), 221–227.

Lin, S. (1965). "Computer solutions of the traveling salesman problem". *Bell System Technical Journal*, 44 (10), 2245–2269.

Lin, S., Kernighan, B. W. (1973). "An effective heuristic algorithm for the traveling-salesman problem". *Operations Research*, 21 (2), 498–516.

Miller, C. E., Tucker, A. W., Zemlin, R. A. (1960). "Integer programming formulation of traveling salesman problems". *Journal of the ACM*, 7 (4), 326–329.

Mole, R. H., Jameson, S. R. (1976). "A sequential route-building algorithm employing a generalised savings criterion". *Journal of the Operational Research Society*, 27 (2), 503–511.

Montoya-Torres, J. R., López Franco, J., Nieto Isaza, S., Felizzola Jiménez, H., Herazo-Padilla, N. (2015). "A literature review on the vehicle routing problem with multiple depots". *Computers & Industrial Engineering*, 79, 115–129.

Or, I. (1976). "Travelling salesman-type combinatorial problems and their relation to the logistics of blood-banking", Ph.D. Dissertation, Northwest University, Evanston.

Osman, I. H., Kelly, J. P. (1997). "Meta-Heuristics: Theory & Applications". Springer, US.

Pathumnakul, S. (1996). "Solving multi depot vehicle routing problem for Iowa recycled paper by tabu search heuristic". Ph.D. Dissertation, Iowa State University.

Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H., Dueck, G. (2000). "Record Breaking Optimization Results Using the Ruin and Recreate Principle". *Journal of Computational Physics*, 159 (2), 139–171.

Shapley, L.S. (1953). "A value for n-person games". *Annals of Mathematics Studies*, 28, 307–317.

Shi, Y., Lv, L., Hu, F., Han, Q. (2020). "A heuristic solution method for multi-depot vehicle routing-based waste collection problems". *Applied Sciences*, 10 (7), 2403.

Tillman, F. A. (1969). "The Multiple Terminal Delivery Problem with Probabilistic Demands". *Transportation Science*, 3 (3), 192–204.

Thompson, P. M., Psaraftis, H. N. (1993). "Cyclic transfer algorithms for the multivehicle routing and scheduling problems", *Operations Research*, 41, 935–946.

Toth, P., Vigo, D. (2002). "The Vehicle Routing Problem". Society for Industrial and Applied Mathematics, New Jersey.

Wasner, M., Zäpfel, G. (2004). "An integrated multi-depot hub-location vehicle routing model for network planning of parcel service". *International Journal of Production Economics*, 90 (3), 403–419.

Zibaei, S., Hafezalkotob, A., Ghashami, S. S. (2016). "Cooperative vehicle routing problem: An opportunity for cost saving". *Journal of Industrial Engineering International*, 12, 271–286.