Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

Optimización del multi-trip vehicle routing problem mediante el algoritmo de Clarke-Wright

Autor: Carlos Sánchez Gómez
Tutor: Alejandro Santana Escudero

Dpto. Organización Industrial y Gestión de Empresas II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Proyecto Fin de Grado
Ingeniería de las Tecnologías Industriales

Optimización del multi-trip vehicle routing problem mediante el algoritmo de Clarke-Wright

Autor:
Carlos Sánchez Gómez

Tutor:
Alejandro Santana Escudero
Profesor Contratado Doctor

Dpto. de Organización Industrial y Gestión de Empresas II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

A mi familia
A mis amigos
A María.

Agradecimientos

Me gustaría comenzar agradeciendo la posibilidad de realizar este proyecto a mi tutor, Alejandro. Ha sido un placer trabajar siguiendo tus indicaciones y directrices, sabiendo que tras ese gran profesional de la investigación hay un profesor que siempre busca lo mejor para sus alumnos.

También quiero agradecer a mis padres y mi hermana todo el apoyo mostrado durante el transcurso de este trabajo de fin de grado. Ha sido una etapa con idas y venidas y cariño ha sido fundamental para llegar a donde he llegado.

Por último, no puede faltar el agradecimiento a mi siempre leal compañera, María. No ha faltado un momento en el que no me hayas alentado a no rendirme y a conseguir mis objetivos. Gran parte de este proyecto lo he conseguido gracias a tu apoyo incondicional y paciencia.

Resumen

El incremento de las políticas de restricción del número de vehículos motorizados que pueden acceder al centro de las grandes ciudades del país plantea un nuevo campo de estudio para las compañías de transporte. Así, en este documento se aborda uno de los problemas más complejos en el campo de la logística y la optimización, el VRP o problema de rutado de vehículos. En concreto, se enfatiza en una variante que posibilita los viajes múltiples de mismo único vehículo en una sola jornada de reparto, el llamado Multi-trip VRP o MTVRP. El problema es tratado en el ámbito de la logística urbana como una alternativa de adaptación a los cambios que se están produciendo hoy día en el sector del transporte. Tras explicar los antecedentes y tipología del problema, se presentan los resultados obtenidos una vez ejecutado la adaptación del algoritmo de ahorros de Clarke y Wright al MTVRP para unas determinadas instancias de problemas, así como las conclusiones al respecto.

Abstract

The increase of restrictive policies about the number of motorized vehicles used to reach the city center of major cities of the country suggests a new research field for the transport companies. Thereby, this document is an approach to one of the most complex problems in the field of logistics and optimization, the VRP or Vehicles Routing Problem. More specifically, it is tackled a problem variant which enables a single vehicle to take multiple trips during the same delivery period, the Multi-trip VRP o MTVRP. The problem is presented in relation with urban logistics as an adaptative alternative to the multiple changes which have been producing nowadays in the transport area. Once explained the previous research and the typology of the problem, the results obtained, once the Clarke and Wright savings algorithm is run and modified in order to cover all the MTVRP restrictions for certain problem instances, will be shown and so will be the conclusions about them.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

1.1 Motivación

El concepto de "city logistics" o "logística urbana" se define en Taniguchi, et al. (1999) como "el proceso para la completa optimización de la logística y actividades de transporte llevadas a cabo por empresas privadas en áreas urbanas considerando las condiciones del tráfico, la congestión de este y el consumo de combustible en el marco de una economía de mercado".

En la actualidad, el crecimiento de la población en las grandes ciudades y zonas urbanas es cada vez más evidente. "El 55% de la población mundial vive en zonas urbanas, una proporción que se espera que aumente hasta el 68% en 2050" (UN DESA, 2018). Teniendo en cuesta estos datos, la logística urbana cobra especial importancia. Un 13% de aumento de población podría llegar a comprometer la sostenibilidad y la viabilidad del transporte en urbano por diferentes motivos, algunos de los cuales ya se producen hoy día:

- *Aumento del tráfico*, proporcional al aumento del número habitantes, que impediría ejecutar la actividad del transporte de forma eficiente, alargando así los tiempos de esta actividad.
- *Aumento de las necesidades de los ciudadanos*. Los bienes básicos de consumo y la demanda de mercancías en general tendrían un notado incremento, forzando así la creación de nuevos comercios, que requerirían a su vez de más capacidad de transporte para su adecuado abastecimiento. Además del comercio físico, habría que tener en cuenta el comercio electrónico, pues el aumento de demanda de este tipo de comercio conllevaría un incremento en el transporte de mercancías, estrechamente ligado al crecimiento de la población.
- *Empeoramiento de la calidad de vida de los ciudadanos*. Los dos puntos anteriores llevarían asociados estrictamente un deterioro de la calidad del aire debido a la contaminación, el gran aumento del ruido generado por los vehículos e incluso posibles dificultades a la hora del estacionamiento de particulares debido a la cantidad de medios de transportes que pudiera haber ya estacionados en la ciudad. Todos estos puntos mencionados actuarían en detrimento de la salud, el bienestar y la calidad de vida de los ciudadanos.

La dificultad de la situación actual planteada, así como el posible empeoramiento en el futuro de esta sugieren la necesidad de una optimización inmediata de la gestión del transporte en las zonas urbanas, ya sea mediante la disminución del número total de vehículos utilizados, mediante la reducción de la distancia total recorrida por estos o por ambas.

Sin perder de vista el objetivo principal de mejorar la calidad de vida de todos los habitantes de las zonas urbanas, el factor económico derivado de esta posible optimización es el más relevante y atractivo para las diferentes compañías, tanto proveedores de transporte como clientes de éstos. La mejora del rutado de los diferentes vehículos y disminución del número de estos llevaría a estas empresas proveedoras a reducir sus costes destinados a transporte de forma sustancial, con lo cual el interés en acometer el desarrollo de estos proyectos de mejora sería mayor, además de tener aportar un impulso a la mejora la viabilidad del transporte en las ciudades de aquí a unos años.

1.2 Problema

Teniendo en cuenta todo lo anterior, se plantea para su resolución en este documento un problema de enrutamiento de vehículos o VRP (Vehicles Routing Problem), cuyo concepto se introducirá en el capítulo 2. En concreto, se tratará la variante Multi-Trip VRP (MTVRP) de este problema, que ofrece la posibilidad de múltiples viajes por parte de un mismo vehículo durante el mismo día de reparto, lo cual ofrece una serie de beneficios notables en la gestión de la logística urbana que se desarrollarán posteriormente.

El objetivo principal del problema será mejorar la viabilidad del transporte en las zonas urbanas. Desde el punto de vista empresarial, el objetivo será minimizar costes totales de logística. La búsqueda de la reducción de costes llevará de forma lógica a la minimización de los kilómetros recorridos en el transporte, así como de los vehículos utilizados para ello. Por tanto, se podrán abordar los dos aspectos de mejora, humano y económico, de forma simultánea.

1.3 Resumen

La estructura seguida en el documento es la siguiente:

En el capítulo 2, se hará una contextualización del problema MTVRP que se va a abordar, comenzando por sus orígenes con el problema del viajante (TSP), a partir del cual surgiría con el problema VRP, el cual sentaría las bases del problema de rutado de vehículos. Una vez dado un breve contexto histórico de este problema, se detallará el modelo del problema VRP y un listado de las diferentes variantes existentes, dentro de las cuales se encuentra el MTVRP, el problema concreto a tratar. Este se definirá y desarrollará posteriormente, junto a sus modelos y posibles variantes, así como la justificación del uso de MTVRP en logística urbana.

La estructura del capítulo 3 es la que sigue. Una vez explicado el problema, se plantearán el método utilizado para la resolución del problema. Brevemente, se comentarán los métodos exactos existentes, y por qué no son eficientes para el problema a estudiar. Posteriormente, se centrará el foco en las distintas heurísticas y metaheurísticas. En concreto, dentro de las heurísticas y metaheurísticas analizadas, se destacarán el algoritmo de Clarke y Wright, metodología que será la elegida para la resolución del problema de rutado de vehículos con múltiples viajes.

En el capítulo 4, conocida ya la metodología a utilizar, se procederá al desarrollo en profundidad de esta, explicándose así su funcionamiento y adaptaciones sufridas para el problema en particular.

En el capítulo 5, se desglosarán los diferentes resultados para una batería de problemas elegidos, estudiándose los resultados para los dos casos anteriores propuestos, así como el tiempo computacional de cada uno de ellos. Posteriormente, se compararán los resultados finales con la resolución de la base Solomon, pudiendo así determinar si el procedimiento de resolución ha conseguido llegar al óptimo del problema. Se propondrán algunas variaciones en el algoritmo inicial tras las cuales se comprobará la mejora o el empeoramiento de los resultados.

Finalmente, en el último capítulo, una vez realizados los correspondientes cálculos, se procederá a su análisis y a las posteriores conclusiones. En concreto, se estudiará la eficiencia de cada una de las variantes propuestas, analizando así la viabilidad de aplicar el algoritmo de Clarke y Wright para generar soluciones iniciales, así como el impacto final que podría tener la resolución del problema en una compañía real.

2 ANTECEDENTES

A lo largo de este capítulo, se explicará el problema de rutado de vehículos desde su origen con el problema del viajante hasta llegar al Multi-Trip VRP. Una vez definido el problema del viajante, se expondrá el problema VRP junto a su modelo y variantes, dentro de las cuales se puede identificar el Multi-trip VRP. Una vez propuesto el modelo del Multi-trip VRP, se elegirá la variante usada en este proyecto, justificándose el uso de este procedimiento y las ventajas que ofrece en el entorno actual de la logística urbana.

2.1 El problema del viajante (TSP)

El problema del viajante (TSP, "Travel Sallesman Problem") se puede definir mediante la siguiente pregunta:

"Si un viajante parte de una ciudad y las distancias a otras ciudades son conocidas, ¿cual es la ruta óptima que debe elegir para visitar todas las ciudades y volver a la ciudad de partida?" (Calviño, 2011). Entiéndase por ruta óptima en este caso aquella que permite minimizar la distancia total recorrida por el viajante.

A pesar de la aparente simplicidad del problema, ha sido uno de los problemas más estudiados del siglo XX, debido principalmente a que se trata de un problema tipo *NP-Duro* y, por tanto, su resolución es altamente compleja.

Aunque los orígenes son difusos, parece que este problema fue planteado por primera vez en la Europa del siglo XIX en un libro llamado Der Handlungsreisende - Von einem alten Commis - Voyageur, según se explica en Cook (2012). No era más que un planteamiento conceptual del problema en un guía de viajes, por tanto, no contenía ningún aspecto técnico o matemático. Posteriormente, en ese mismo siglo, el problema fue planteado de otra forma, presentándose como una especie de "puzle para niños" por Thomas Kirkman y W.R. Hamilton, cuya resolución pasaba por encontrar un ciclo hamiltoniano entre los puntos de reparto. No fue hasta la época de 1930 cuando el problema fue formulado, tal y como se conoce hoy en día, por Karl Menger en Viena y Harvard, lo cual se detalla en Cook (2012).

La futura generalización y aparente simplificación de este problema daría origen a lo que hoy se conoce como problema de rutado de vehículos o VRP, lo cual se explica en el siguiente apartado.

2.2 VRP

En 1959, G.B. Dantzig y J.H. Ramser plantearon en Dantzig y Ramser (1959) un nuevo tipo de problema, usado en primera instancia para el reparto eficiente de gasolina, llamado "The truck dispatching problem". Posteriormente, este problema pasaría a conocerse como VRP (Vehicle Routing Problem). El origen de este problema era el TSP, descrito en el apartado 2.1, con evidentes modificaciones añadidas, como podían ser la posibilidad del uso de varios vehículos para la programación de rutas, lo cual simplificaba la resolución del problema. No obstante, seguía manteniendo su clasificación de *NP-Duro*. Dantzig y Ramser, propondrían así el primer algoritmo que trataría de obtener la resolución de este problema junto con el modelado matemático del mismo en Dantzig y Ramser (1959).

Posteriormente, en 1964, Clarke y Wright reformularían el problema en Clarke y Wright (1964), desarrollando el primer algoritmo de aproximación claramente eficiente, conocido como el

algoritmo de Clarke y Wright o algoritmo de ahorros. Este heurística en concreto ha sido una de las metodologías elegidas para su desarrollo en este documento. Se explica de forma más extensa en el capítulo 3, donde se planteará junto a otros métodos. Es en el capítulo 4 donde se profundiza en el modo de empleo de esta metodología. Han sido numerosos autores los que han estudiado hasta el día de hoy las posibles aplicaciones y variantes del VRP, las cuales también se desarrollan en este documento.

2.2.1 Definición

El VRP (Vehicule Routing Problem) o problema de rutado de vehículos tiene como fin transportar una serie de bienes o mercancías, dada una flota de vehículos, a un conjunto de localizaciones dispersas geográficamente, conocidas las distancias entre cada una de ellas, con una programación de rutas eficiente para cada uno de los vehículos.

Para este tipo de problema de rutado de vehículos habrá tres componentes relevantes que lo acabarán definiendo, que son: La flota de vehículos, los clientes y los depósitos. Según las características que se le asignen a cada uno de estos tres términos, surgirán unas variantes u otras del problema. A continuación, se sentarán las bases con el VRP general o CVRP, para posteriormente incidir en sus variantes.

El problema general del VRP tal y como se conoce hoy en día se corresponde con el CVRP (Capacited Vehicle Routing Problem) o rutado de vehículos con capacidad. Para su formulación, a partir de la cual derivan el resto de las variantes del problema, se tienen en cuenta una serie de consideraciones previas con respecto a la flota de vehículos, los clientes y los depósitos que definirán el problema.

- Los vehículos **solo pueden realizar un viaje de ida** y vuelta al depósito. Esta condición será clave posteriormente.
- Hay un número de clientes esparcidos geográficamente con una demanda diaria conocida.
- El número de viajes o rutas no está estipulado con anterioridad.
- En principio, solo existe un depósito o almacén del que parten todos los vehículos.
- Cada cliente puede ser visitado exclusivamente por un vehículo y solamente una vez.
- Hay un límite de capacidad conocido en cada uno de los vehículos.

El objetivo del problema es la optimización del rutado de los vehículos. Este pasará, según la variante del VRP que se esté tratando, por diferentes funciones objetivos, aunque en general, todos los problemas, independientemente de su función objetivo, tenderán a converger en valores mínimos o cerca de este.

- Minimizar el coste total del transporte.
- Minimizar la distancia total recorrida.
- Minimizar el tiempo total de reparto.

2.2.2 Modelo

A partir de las consideraciones anteriores se construirá el modelo matemático del problema VRP o CVRP, según se formula en Oliveira (2004). En este caso, la función objetivo será la minimización del coste de transporte.

La red de transporte del problema se modelará mediante el uso de un grafo $G = (V, E, C)$ donde, en V, el nodo 0 del problema representa al depósito o almacén y los nodos 1, …, n representan a los clientes, siendo n el número de clientes totales. Los arcos (i, j) pertenecientes a E denotarán

la mejor solución para ir desde el nodo i hasta el nodo j, definidos por las variables binarias x_{ij} que tomen valor 1. Cada camino tendrá asociado un coste asociado al transporte c_{ij} y un tiempo para recorrer la distancia que separa cada nodo t_{ij}.

Se denotarán $\Delta^+(i)$ y $\Delta^-(i)$ para el problema de la siguiente forma:

$\Delta^+(i) = \{j \in V | (i,j) \in E\}$

$\Delta^-(i) = \{j \in V | (j,i) \in E\}$

Cada cliente i perteneciente a V tiene una demanda d_i y una capacidad C.

Dados un número S de clientes a servir $d(S) = \sum_{i \in S} d_i$ representará la demanda total de todos los clientes y $r(S)$ el número de vehículos necesarios para servir esta demanda.

El modelo del problema VRP, también llamado modelo de "flujo de vehículos", es el siguiente:

$$\min \sum_{(i,j)\in E} c_{ij}x_{ij} \quad (1.1)$$

$Sujeto\ a$:

$$\sum_{j\in\Delta^+(0)} x_{0j} = m \quad (1.2)$$

$$\sum_{i\in\Delta^-(0)} x_{i0} = m \quad (1.3)$$

$$\sum_{j\in\Delta^+(i)} x_{ij} = 1, \quad \forall i \in V \setminus \{0\} \quad (1.4)$$

$$\sum_{j\in\Delta^-(j)} x_{ij} = 1, \quad \forall j \in V \setminus \{0\} \quad (1.5)$$

$$\sum_{i\in S, j\in\Delta^+(i)\setminus S} x_{ij} \geq r(S), \quad \forall S \subset V \setminus \{0\} \quad (1.6)$$

$$m \geq 1 \quad (1.7)$$

$$x_{ij} \in \{0,1\}, \quad \forall (i,j) \in E \quad (1.8)$$

Teniendo en cuenta que $r(S)$ se calcularía como:

$$r(S) = \min \sum_{k\in K} y_k \quad (1.9)$$

$s.a$:

$$\sum_{i\in S} d_i x_{ik} \leq C y_k, \quad \forall k \in S \quad (1.10)$$

$$\sum_{k\in K} x_{ik} = 1, \quad \forall i \in S \quad (1.11)$$

$$x_{ik} \in \{0,1\} \quad \forall i \in S, \forall k \in K \quad (1.12)$$

$$y_k \in \{0,1\} \quad \forall k \in K \quad (1.13)$$

En la función objetivo (1.1) se minimiza el coste total de las rutas realizadas por los diferentes vehículos. Las restricciones (1.2) y (1.3) indican que un número m de vehículos que parten del depósito han de regresar de forma obligatoria. En las restricciones (1.4) y (1.5) se puede asegurar que todo cliente es visitado en alguna de las rutas establecidas. En (1.6), además de eliminarse la posibilidad de realizar sub-rutas, se obliga al cumplimiento del límite de capacidad C en los vehículos. Para ello, se utiliza la $r(S)$, cuyo cálculo se desglosa en las expresiones (1.9) - (1.12), sabiendo que K representa a una flota que permita satisfacer la demanda total. Por último, en

(1.11) se impone a que el número de vehículos siempre sea mayor que uno y en (1.12) y (1.13) se le da el carácter binario a la variable x_{ij}.

2.2.3 Variantes

A partir de las consideraciones anteriormente establecidas para el CVRP como problema general, surgen una serie de variantes que, o bien añaden nuevas tesituras y escenarios al problema, o bien modifican algunas de las previamente establecidas.

- MDVRP (Multi-Depot VRP) (Hjorring, 1995)
  Esta variante rompe una de las suposiciones iniciales del VRP general, ampliando el número de depósitos a más una unidad, de los cuales pueden partir cualesquiera de los vehículos de la flota.
- PVRP (Periodic VRP) (Baptista, Oliveira y Zúquete, 2002)
  El VRP periódico amplía su periodo de planificación, considerando un número previamente establecido de jornadas como periodo de reparto. Además, los vehículos no tienen por qué regresar al depósito el mismo día que salen de este.
- SDVRP (Split Delivery VRP) (Dror, Laporte y Trudeau, 1994; Archetti, Mansini y Speranza, 2001)
  En esta variación de problema, el reparto a un cliente se puede llevar a cabo por varios vehículos siempre y cuando se reduzca el coste total de reparto.
- SVRP (Stochastic VRP) (Laporte y Louveaux, 1998)
  En esta variación del VRP se le añade un determinado grado de incertidumbre. La aleatoriedad vendrá dada o bien en la demanda o bien en la presencia de clientes, derivando en otros dos tipos de subproblemas.
- VRPB (VRP with Backhauls) (Ralphs, Hartman y Galati, 2001); (Jacobs-Blecha y Goetschalckx, 1992)
  A esta variante del problema se le añade la posibilidad de devolución de productos por parte del cliente. La devolución de mercancía el cliente en el instante posterior a la entrega del pedido en sus instalaciones. Las cantidades por repartir y a devolver son conocidas previamente.
- VRPPD (VRP with Pick-Up and Delivering) (Righini, 2000)
  Es un problema muy similar a VRPB, con la excepción de que las devoluciones solo se podrán realizar por parte del cliente cuando una vez se haya producido el reparto completo a los demás clientes.
- VRPSF (VRP with Satellite Facilities) (Bard et al., 1997)
  Este nuevo problema VRP se estudia considerando la carga o recogida de mercancía durante el desarrollo de la misma ruta sin necesidad de volver al depósito o almacén central.
- VRPTW (VRP with Time Windows) (Cordeau et al., 2002)
  En esta variación del problema se plantea una situación algo mas concreta: La entrega de mercancía se tiene que producir en unas ventanas temporales que vendrán dadas para cada uno de los destinos a repartir. En caso de no llegar dentro de este rango, el reparto no se puede llevar a cabo.
- **MTVRP (Multi-Trip VRP).**
  Esta variante rompe con otra de las bases establecidas antes. En este tipo de problema se permite la posibilidad de que un camión realice varios viajes de ida y vuelta al depósito en un mismo día. Será esta variante del VRP la que se tome como caso de estudio para el documento y el cual se analizará en profundidad en el siguiente punto 2.3.

2.3 MTVRP

Para definir el Multi-Trip VRP o MTVRP se usará el siguiente extracto:

"Este problema es una extensión del bien conocido Problema de Rutado de Vehículos, donde se les permite a los vehículos ser recargados y rutados de nuevo una vez han acabado su viaje en el depósito. Fue introducido hace más de 25 años, y desde entonces, investigadores han estado trabajando en él, pero ninguna investigación exhaustiva ha sido propuesta." (Cataruzza, 2014).

Al ser una variante del problema original VRP, mantiene su clasificación siendo un problema *NP-Duro*, por lo que su resolución es claramente compleja.

Según Cataruzza (2014), el primer planteamiento formal del Multi-Trip VRP vendría dado por Fleischmann en 1990 usando el sobrenombre de "Problema de rutado de vehículos con uso múltiple de vehículos". Durante estos años no se ha llegado a convenir un nombre concreto para este tipo de problema, apareciendo así en diferentes estudios bajo diferentes denominaciones: "Multiple trips VRP", "Multi-Travel VRP". En este texto, y como se ha venido haciendo hasta ahora, se usará el nombre de Multi-Trip VRP (MTVRP) para su tratamiento.

Las posibles aplicaciones de este nuevo tipo de problema a la logística urbana frente el VRP o CVRP al uso son muy numerosas, abriendo un abanico de posibilidades con respecto al tipo de vehículos utilizados en el reparto, pudiendo aumentar la eficiencia de la entrega. Todo ello se desarrollará en el punto 2.3.5.

Para la correcta formulación de este problema se tendrán en cuenta una serie de consideraciones que, claramente, ya aparecieron en el CVRP.

- Habrá un único depósito.
- Cada vehículo parte del depósito y ha de regresar a este al final del viaje.
- La demanda total de los clientes en cada viaje no excederá la capacidad Q.
- La duración total de los viajes asignados a un vehículo no superará el valor de T_H.

En este caso concreto se establecerá una limitación de horas de reparto el día de un mismo vehículo con la variable T_H, presentando así un escenario mucho más realista ajustado a las jornadas de trabajo diarias.

2.3.1 Modelo

Para la formulación del modelo, se utilizará la versión que aparece a Cataruzza (2014), que no es mas que una reformulación que mejora y agrupa los modelos desarrollados por diferentes autores que han estudiado previamente el Multi-Trip VRP.

El modelo se definirá por un grafo $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ donde $\mathcal{N} = \{0,1, \dots, N\}$ representa el conjunto de nodos del problema, siendo 0 el depósito y $\mathcal{A}$ el conjunto de arcos que los unen tales que $\mathcal{A} = \{i, j | (i, j) \in V, i < j\}$. Los nodos $1, \dots, N$ representan los clientes a servir, cuya demanda individual es de Q_i. Q, Q_i, T_H son enteros no negativos.

El modelo adaptado de Cataruzza incluye dos nuevos conjuntos:

$\bar{\mathcal{N}} = \mathcal{N} \cup \{N + 1\}$
$\bar{\mathcal{A}} = \{(i, j) \in \mathcal{N} \setminus \{0\}\} \cup \{(0, i) | i \in \mathcal{N} \setminus \{0\}\} \cup \{(i, N + 1) | i \in \mathcal{N}\}$

El nodo $(N + 1)$ hace las veces de depósito. Su demanda y entregas son nulas. $\bar{\mathcal{A}}$ contiene todos los arcos que conectan a todos los clientes por pares, así como los arcos que van desde cualquier cliente hasta el nodo $(N + 1)$, así como los arcos que son trazados desde el nodo 0 o depósito

hasta el nodo $(N+1)$. Además, se propone el espacio $\mathcal{R}$ donde se almacenan todos los viajes que forman una solución.

Las variables son las siguientes:

- x_{ij}^r: Variable binaria definida para cada par de nodos (i,j) pertenecientes a $\bar{\mathcal{N}}$. Indica si el arco (i,j) está cubierto por el viaje r.
- y_i^r: Variable binaria para cada i en $\mathcal{N}\backslash\{0\}\ y\ cada\ r\ en\ \mathcal{R}$. Indica si el cliente i ha sido servido por el viaje r.
- z_{rs}: Variable binaria Indica si el viaje r precede al viaje s en una jornada de trabajo.
- w_r: Variable binaria que indica si el viaje r es el primero de la jornada.
- t_i^r: Variable continua que indica el instante en el que servicio empieza en el nodo i cuando es visitado por el viaje r.

$$\min \sum_{r\in\mathcal{R}} \sum_{(i,j)\in\bar{\mathcal{A}}} D_{ij} x_{ij}^r \tag{2.1}$$

$Sujeto\ a$:

$$\sum_{j\in\bar{\mathcal{N}}\backslash\{0\}} x_{ij}^r = y_i^r, \quad \forall i \in \mathcal{N}\setminus\{0\}, \forall r \in \mathcal{R} \tag{2.2}$$

$$\sum_{r\in\mathcal{R}} y_i^r = 1, \quad \forall i \in \mathcal{N}\setminus\{0\} \tag{2.3}$$

$$\sum_{i\in\bar{\mathcal{N}}\backslash\{N+1\}} x_{ih}^r - \sum_{i,j\in\bar{\mathcal{N}}\backslash\{0\}\}} x_{hj}^r = 0, \quad \forall i \in \mathcal{N}\setminus\{0\}, \quad \forall h \in \mathcal{N}\setminus\{0\}, \forall r \in \mathcal{R} \tag{2.4}$$

$$\sum_{i\in\bar{\mathcal{N}}\backslash\{0\}} x_{0i}^r = 1, \quad \forall\, r \in \mathcal{R} \tag{2.5}$$

$$\sum_{i\in\bar{\mathcal{N}}\backslash\{N+1\}} x_{N+i,i}^r = 1, \quad \forall\, r \in \mathcal{R} \tag{2.6}$$

$$\sum_{i\in\bar{\mathcal{N}}\backslash\{0\}} Q_i y_i^r \le Q, \quad \forall r \in \mathcal{R} \tag{2.7}$$

$$t_i^r + S_i + T_{ij} - \alpha\left(1 - x_{ij}^r\right) \le t_j^r, \quad \forall i,j|(i,j) \in \bar{\mathcal{A}}, \forall r \in \mathcal{R} \tag{2.8}$$

$$t_{N+1}^r + S_0 \le t_0^s + \alpha(1 - z_{rs}), \quad \forall r,s\ \forall\mathcal{R}|r<s \tag{2.9}$$

$$t_{N+1}^r \le T_H, \quad \forall r \in \mathcal{R} \tag{2.10}$$

$$\sum_{r<s} z_{rs} = 1 - w_s\ , \quad \forall s \in \mathcal{R} \tag{2.11}$$

$$\sum_{r\in\mathcal{R}} w_r \le M \tag{2.12}$$

$$x_{ij}^r \in \{0,1\}, \quad \forall i,j|(i,j) \in \bar{\mathcal{A}}, \forall r \in \mathcal{R} \tag{2.13}$$

$$y_i^r \in \{0,1\}, \quad \forall i \in \mathcal{N}\setminus\{0\}, \forall r \in \mathcal{R} \tag{2.14}$$

$$z_{rs} \in \{0,1\}, \quad \forall r,s\ \forall\mathcal{R}|r<s \tag{2.15}$$

$$w_r \in \{0,1\}, \quad \forall r \in \mathcal{R} \tag{2.16}$$

$$t_i^r \ge 0, \qquad \forall i \in \bar{\mathcal{N}}, \forall r \in \mathcal{R} \tag{2.17}$$

En la función objetivo (2.1) se minimiza la distancia total recorrida y no el coste total como se hacía con anterioridad. Ambas son válidas pero el autor ha elegido esta opción. Las restricciones (2.2) y (2.3) aseguran el hecho de que cada cliente solo sea servido una vez. Las restricciones (2.4) - (2.6) aseguran que el flujo se conserve según se ha definido. En la restricción (2.7) se

restringe la capacidad total de cada vehículo. Las expresiones (2.8) - (2.10) aseguran que las restricciones temporales de cada viaje, día laboral y horizonte temporal se cumplan de forma efectiva. (2.11) garantiza el hecho de que a cada viaje le preceda única y exclusivamente otro viaje a excepción del primero. Y la última restricción (2.12) se encarga de limitar el uso de vehículos totales usados. El parámetro α tiene un valor alto arbitrario.

2.3.2 Variantes. MTVRPTW

Una vez establecido el problema MTVRP, se han definido numerosas variantes de este tratando de ofrecer múltiples posibilidades a la hora de su desarrollo. En Prins (2002), por ejemplo, el autor trata de minimizar la distancia total recorrida por los vehículos y de forma secundaria, el número de vehículos utilizados. También se ha estudiado el Periodic MTVRP en otras investigaciones, así como el MTVRPTW (Multi-Trip VRP with Time Windows), que se podría definir como la variante más interesante de este problema y que será la que se considerará para su resolución.

En el MTVRP con ventanas temporales a cada cliente se le asocia una ventana temporal $[E_i, L_i]$ dentro de la cual se tendrá que producir el reparto. Se puede llegar antes de que comience la ventana temporal, aunque el transportista tendrá que esperar hasta el momento de apertura E_i. No obstante, la llegada más tarde de L_i no está permitida por parte del cliente. El resto de los supuestos realizados para el problema MTVRP antes planteado en el punto 2.3 se siguen manteniendo para esta variante. Así, se podría modelar de forma similar añadiendo estas dos nuevas restricciones de temporalidad al modelo anterior.

2.3.3 Discusión de los aspectos de interés para el proyecto

Este problema, aunque no ha sido estudiado de forma tan profunda y exhaustiva como lo ha sido el VRP a lo largo de los siglos XX y XXI, presenta una casuística que le da una gran proyección para su futuro desarrollo en la sociedad actual. El hecho de, en un mismo día de reparto, que un vehículo tenga la posibilidad u obligación de volver al depósito en múltiples ocasiones podría tener numerosas ventajas en los siguientes escenarios:

- La compañía distribuidora dispone de una flota de vehículos pequeños para poder acceder a calles estrechas y a zonas con limitaciones de peso de vehículos en el centro de las ciudades.
- La empresa dispone de una flota de furgonetas eléctricas con dimensiones reducidas para acceder por las zonas anteriormente comentadas. Además, pueden acceder a los núcleos urbanos limitados exclusivamente a los vehículos con cero emisiones. Una vez realizado el reparto, los vehículos podrían volver al almacén central para recoger nueva mercancía y también aprovechar para recargar la batería eléctrica de que disponen.
- La compañía en cuestión dedica el grueso de su actividad a repartos de última milla. En una zona urbana, el reparto de pedidos con pequeño volumen exige el uso claro de un vehículo de menor capacidad, como podría ser una furgoneta eléctrica o vehículos de bajo consumo.

En cualesquiera de los tres casos planteados, es evidente que hacer solo un viaje de reparto sería siempre ineficiente para el conductor, dado que la capacidad de su vehículo limitaría el número de clientes a los que pudiera asistir, ya sean particulares o comercios y grandes compañías. En el caso de los vehículos eléctricos, el factor limitante podría ser, además de la capacidad, la batería del vehículo. Asimismo, es prácticamente una necesidad para la empresa de reparto el hecho de considerar que un vehículo realice más de un viaje de ida y vuelta, por lo que el problema MTVRP se vuelve un planteamiento básico del que partir para el reparto eficiente de este tipo de compañías de transporte.

Este nuevo planteamiento del problema no ofrecerá siempre una mejora en términos de distancia recorrida y coste de transporte con respecto al problema VRP clásico donde se contaba con altas capacidades en los vehículos, pero si disminuirá previsiblemente el número total de vehículos utilizado. No obstante, si se considera el escenario de logística urbana planteado anteriormente, no utilizar un Multitrip – VRP podría llevar incluso a la pérdida de clientes por falta de adaptación por parte del proveedor de transporte teniendo en cuenta las limitaciones de dimensiones y emisiones de los vehículos utilizados para el acceso a las zonas de reparto.

En este sentido, sería interesante destacar las llamadas entregas de última milla y las múltiples posibilidades de mejora que el MTVRPTW puede ofrecer a los proveedores de transporte con estas vías de negocio.

Las entregas de última milla son aquellas que se entregan directamente al consumidor final, y vienen determinadas en la mayoría de los casos por el comercio electrónico. El coste de este tipo de entregas asciende a un 53% del coste total logístico de las compañías según Honeywell (2016). A pesar de que en la mayoría de los casos se delega este coste (o parte de él) en el cliente con los llamados "gastos de envío", hay una necesidad evidente de optimizar cuanto antes este tipo de entregas de cara al ahorro final de costes de la compañía. Por tanto, una empresa que dedique gran parte o la totalidad de su actividad a este tipo de entregas y cuya flota de vehículos sea eléctricos o de bajo consumo con poca capacidad para poder acceder a cualquier lugar, tiene la necesidad de optimizar su reparto de mercancías con un MTVRP y cualquiera de sus variantes.

En este caso, el MTVRPTW sería clave para la satisfacción del cliente, determinante para sus futuras compras. Un cliente con un margen más estrecho y concreto de horas en las que puede recibir un paquete estará, por norma, más satisfecho que aquel que tiene que esperar todo un día para el recibo de su mercancía.

En resumen, el endurecimiento de las limitaciones actuales relacionadas con el acceso de las compañías de transporte a las ciudades y, especialmente, al centro de estas, establece un nuevo escenario en la logística urbana. En este escenario, el planteamiento original del VRP o alguna de las variantes que no incluyen viajes múltiples no cubriría las necesidades de muchos de los clientes, dado que, sin los vehículos apropiados, algunos de ellos podrían llegar a perder la opción de reparto.

Por tanto, en términos de competitividad, sostenibilidad y eficiencia, de cara al futuro, un sistema de rutado de vehículos Multitrip – VRP con ventanas temporales será, junto con las diferentes variantes que se desarrollen, la opción más viable a tomar por las compañías de transporte.

No obstante, el modelo CVRP o VRPTW para los repartos de larga distancia sigue siendo totalmente factible y eficiente. Así, una vez entregada la mercancía en el nodo logístico de la ciudad, para el reparto de última milla es para el cual debería llevarse a cabo el MTVRPTW, siendo una consecuencia del reparto realizado posteriormente con un VRP sin viajes múltiples.

3 METODOLOGÍAS

A lo largo de las últimas décadas, el problema de optimización VRP y sus múltiples variantes han ido cobrando cada vez más relevancia tanto en el ámbito de la investigación como en el ámbito empresarial. Se han propuesto numerosas metodologías para su resolución, que según su tipo se clasifican en tres grupos: métodos exactos, heurísticas y metaheurísticas. Las más usadas son, por su eficiencia, las heurísticas y metaheurísticas, las cuales se desarrollarán con más profundidad en este capítulo.

3.1 Métodos exactos

Los métodos exactos, según Lüer, Buenavente, Lustos y Venegas (2009) son "aquellos que parten de una formulación como modelos de programación lineal (enteros) o similares, y llegan a una solución factible (entera) gracias a algoritmos de acotamiento del conjunto de soluciones factibles."

Los métodos exactos, debido a que el problema a resolver es *NP-Duro*, han demostrado ser altamente ineficientes para la resolución de instancias superiores a 50 clientes, debido al elevado tiempo de computación necesario para la obtención del óptimo de problema. Sin embargo, para números reducidos de clientes sí se han desarrollado e implementado algunos métodos. Entre los más usados se encuentra el método de Branch & Bound, así como el método de Brunch & Cut y la programación dinámica. En el caso que se estudia, estos métodos tomarían como modelo de entrada para su resolución el definido en el punto 2.3.2.

3.1.1 Branch & Bound

La finalidad del método de ramificación y poda o "Branch and Bound", Land and Doig (1960) se describe en Canca y González (2015) de la siguiente forma:

"La idea básica de estos métodos es la de eliminar zonas completas de la región de admisibilidad sin tener que explorar las soluciones de su interior. Para ello se lleva a cabo una comparación de los valores esperados de la función objetivo en el interior de estas, trabajando con cotas de la función objetivo."

El objetivo de esta metodología es eliminar de las soluciones posibles aquellas que, antes de la realización de diferentes combinaciones, ya permitan asegurar que no serán óptimas, dejando ramificar así únicamente aquellas soluciones que puedan conducir al valor óptimo del problema. De esta forma, se reducirá el número de combinaciones posibles del problema facilitando su resolución (siempre con instancias de pocos clientes).

3.1.2 Branch & Cut

En este método, además de utilizar "Branch and Bound", se añade el método de planos de corte al ejercicio, que se definen en Canca y González (2015) de la siguiente forma:

"En optimización, los métodos de planos de corte son procedimientos para encontrar soluciones enteras de un problema lineal entero. (…) Este tipo de técnicas funcionan sobre la resolución de los problemas lineales relajados /no enteros) mediante la inclusión de restricciones, generadas de forma automática que van eliminando porciones de la región factible sin eliminar ninguna de las posibles soluciones enteras del problema."

Branch and Cut es, por tanto, un método híbrido en el que se aplica primero el método de Branch and Bound y una vez se ha llegado a un óptimo no entero, requiriendo el problema soluciones enteras, se aplica el método de plano de cortes para llegar a su solución final.

3.2 Heurísticas

El término "heurística" se asocia a la proposición de estrategias para llegar al descubrimiento como concepto. Hablando en términos de programación, un algoritmo heurístico es aquel que, teniendo en cuenta lo conocido sobre el problema, la solución se determina mediante pruebas, ensayos y errores, hasta llegar a la mejor solución posible, que no siempre es el óptimo real del problema. Se desarrollan a continuación las heurísticas más conocidas en el VRP.

3.2.1 Algoritmo de Clarke y Wright

Esta metodología fue desarrollada en el año 1964 por Clarke y Wright y es una de las heurísticas más difundidas para la resolución del VRP, siendo la primera que llegaba a ser realmente efectiva tras el planteamiento del problema de Danztig y Ramser (1959). También se conoce como *Algoritmo de ahorros* y tiene numerosas variantes. En concreto, este será la metodología elegida para llevar a cabo el proyecto y se desarrollará con más profundidad en el capítulo 4.

La base principal de esta metodología es calcular el "ahorro" que supone realizar una nueva combinación de soluciones y elegir aquellos que mejoren la solución dada. Se suponen un número de vehículos no finito y un solo depósito.

1. Se parte de una solución inicial para cada cliente i con un número no limitado de vehículos tipo $(0, i, 0)$. Se hacen todas las posibles combinaciones por parejas de cada uno de los clientes y se calcula el llamado ahorro, definido por la siguiente fórmula.

   $$s_{ij} = 2 * (c_{0i} + c_{0j}) - (c_{0i} + c_{ij} + c_{0j})$$

   Siendo c_{0i} y c_{0j} el coste del desplazamiento desde el depósito a cada cliente y c_{ij} el coste de desplazamiento de cada pareja de clientes.

2. Una vez calculado el ahorro de cada pareja, se seleccionan aquellos cuyo ahorro es mayor y se valora el cumplimiento de las restricciones del problema, dado que las condiciones supuestas inicialmente no son las reales.

3.2.2 Búsqueda Local (Local Search)

"Para cada solución s se define un conjunto de soluciones vecinas N(s). Un procedimiento de Búsqueda Local parte de una solución s, la reemplaza por una solución s* ∈ N(s) de menor costo y repite el procedimiento hasta que la solución no pueda ser mejorada. Al terminar, se obtiene una solución localmente ´optima respecto a la definición de la vecindad. Para obtener s* puede buscarse la mejor solución de N(s) (estrategia best improvement) o simplemente tomar la primera solución de N(s) que mejore el costo (estrategia first improvement)" (Oliveria, 2004).

Dentro de la metodología de búsqueda local hay numerosas variantes, entre las que destacan el operador λ -intercambio o el algoritmo de Lin-Kernigham.

El algoritmo suele finalizar en un tiempo computacional breve, eligiéndose la falta de mejora de las nuevas soluciones vecinas encontradas durante un periodo de tiempo como criterio de parada.

3.3 Metaheurísticas

"Las Metaheurísticas son procedimientos genéricos de exploración del espacio de soluciones para problemas de optimización y búsqueda. Proporcionan una línea de diseño que, adaptada en cada contexto, permite generar algoritmos de solución. En general, las metaheurísticas obtienen mejores resultados que las heurísticas clásicas, pero incurriendo en mayores tiempos de ejecución (que, de todos modos, son inferiores a los de los algoritmos exactos)." (Oliveira, 2004)

Un procedimiento genérico llevado a cabo para la resolución de problemas de rutado de vehículos, y el cual se usará en este documento, es el uso de una heurística para determinar las llamadas soluciones iniciales y una metaheurística para afinar dichas soluciones obtenidas. En concreto, se presentan tres metaheurísticas: el algoritmo genético, el recocido simulado y la búsqueda tabú.

3.3.1 Algoritmo Genético

Metodología implementada por Holland en la época de 1970. Ha sido fuente de inspiración para el desarrollo de los conocidos algoritmos evolutivos. El algoritmo genético tiene su base, como su propio nombre indica, en la evolución genética de las especies según la ley de selección natural, donde los mejores (o más adaptados) prosperan y los peores dejan de existir. Al tener una semejanza con el comportamiento de la naturaleza, se denomina algoritmo bioinspirado.

Cada solución se suele asociar con los cromosomas de un individuo y así los componentes del vector solución se comparan con los genes del cromosoma. El procedimiento es el siguiente:

1. Generación de la población inicial. Hay que generar una población lo suficientemente grande como para que no se vea afectada en gran medida por los pequeños cambios en ella. El número n de individuos suele ser determinado tras varias pruebas.
2. Evaluación. Se evalúa cada una de las soluciones según la función de fitness, que normalmente suele coincidir con la función objetivo.
3. Seleccionar los individuos con mejor características. Los individuos con una mejor función de fitness tienen más probabilidad de ser seleccionados. Teniendo en cuenta esta probabilidad e implementando un método tipo juego de la ruleta, se deciden los individuos que pasan.
4. Cruzar los individuos seleccionados. Se define un método de cruce, que no suele ser único y se procede.
5. Mutar. Además de los cruces naturales producidos, se producen cambios o mutaciones en las soluciones como ocurriría en un entorno natural.
6. Repetir. Una vez se ha formado la nueva población, se repite el proceso de forma iterativa hasta que se alcanza un criterio previamente establecido.

3.3.2 Método de Búsqueda Global. Recocido Simulado.

El recocido simulado o "Simulated Annealing" es una metodología de optimización de búsqueda aleatoria. Sus autores originales fueron Kirkpatrick, Gelatt y Vecchi en 1983 aunque Cerny de forma paralela e independiente lo describía en 1985.

Según Canca y González (2016), el recocido simulado es un algoritmo que intenta recrear el proceso de recocido del acero, calentándose a una temperatura T y enfriándose posteriormente de forma lenta a con una tasa de enfriamiento α. Inicialmente, la gran energía de los átomos permite que realicen grandes saltos, los cuales se ven reducidos conforme se va enfriando el material.

La particularidad de este algoritmo es que no permite establecer la solución de un problema en torno a un óptimo local. Es decir, una vez ha llegado a una solución y el material no se "ha enfriado" lo suficiente aún, podría optar por una solución diferente, aunque eso empeorase la función objetivo en primera instancia.

La forma de establecer los posibles cambios en la estructura del material viene determinada por la siguiente probabilidad, denominada velocidad de transición:

$$p = e^{-\Delta E / K_B T}$$

El algoritmo, por tanto, funciona de la siguiente manera.

1. Inicialización del problema.
2. Establecimiento de parámetros: La temperatura inicial la temperatura final, la tasa de enfriamiento y el número máximo de iteraciones.
3. Evaluación de la función objetivo
    a. Se genera una solución aleatoria vecina.
        i. Si mejora la función objetivo, se toma.
        ii. Si empeora la función objetivo, se elige un número r aleatorio y se calcula p. Si p > r, se acepta el cambio,
    b. Se actualiza la temperatura y la iteración.

3.3.3 Búsqueda Tabú (Tabu Search)

La búsqueda tabú fue desarrollada en primera instancia por Glover en 1986 y se basa, a grandes rasgos, en una búsqueda local con la excepción de permitir soluciones que aumenten el costo en primera instancia. Se considera que ofrece uno de los mejores resultados de cara a la resolución de problemas tipo VRP.

La metodología emplea, siguiendo las bases de la búsqueda local, una memoria de corto plazo donde va registrando los atributos de las soluciones obtenidas sin importar necesariamente que mejoren la función objetivo. Durante un número determinado de iteraciones, evita las soluciones con los mismos atributos que las soluciones registradas en la memoria a corto plazo. Estas soluciones no permitidas se denominan *soluciones tabú.* Aún así, se usa un criterio llamado "criterio de aspiración" para aceptar soluciones a pesar de que sean tabú. A estas soluciones tabú aceptadas se les da el sobrenombre de soluciones admisibles.

Dentro del tratamiento de la resolución de problema VRP han sido numerosos autores los que han propuesto diferentes variantes de la búsqueda tabú. Al ser la segunda metodología elegida para refinar las soluciones de el algoritmo de ahorros, en el siguiente capítulo se profundizará sobre las diferentes variantes de la búsqueda tabú, su ejecución y la elección final de la variante usada para la resolución del problema que se estudiará.

4 ALGORITMO DE CLARKE Y WRIGHT

Como se ha comentado a lo largo del documento, la metodología empleada como base para la resolución del problema Multi-trip VRP será el algoritmo de ahorros de Clarke y Wright. A continuación, se procede a la explicación detallada de la heurística, así como las diferentes modificaciones llevadas a cabo sobre la misma para la resolución efectiva del problema VRP con múltiples viajes planteado.

4.1 Algoritmo de Clarke y Wright

Según se explicita en el punto 3.2.1, el algoritmo de Clarke y Wright o *algoritmo de ahorros* fue una de las primeras técnicas empleadas para la resolución del VRP por los autores Clarke y Wright (1964), que dieron nombre al método.

El objetivo de esta heurística es disminuir la duración total, así como la distancia total recorrida para el reparto a cada uno de los clientes o localizaciones. Esta heurística consigue la disminución de la distancia mediante la combinación de rutas cuya distancia conjunta es menor que la suma de las individuales. Se parte de la solución de reparto individual a cada uno de los clientes y se aplica la fórmula de ahorros, que se explicará posteriormente.

A pesar de las múltiples variantes existentes para esta heurística, en este documento se partirá de la variante más sencilla de problema CVRP detallada a continuación. Una vez construido el problema básico se irán incluyendo diferentes restricciones hasta llegar a resolver el problema Multitrip - VRP con ventanas temporales.

4.1.1 Consideraciones previas

Para la correcta ejecución del algoritmo de ahorros, la definición del problema VRP deberá ser concisa y concreta. Para la versión base de la heurística de un CVRP, se tomarán como premisa los siguientes puntos, previamente indicados en este documento en la definición de problema:

- Hay un único depósito.
- Cada vehículo parte del depósito y ha de regresar a este al final de la jornada.
- Cada vehículo solo podrá realizar un único viaje de ida y vuelta desde la base.
- Existe un número de clientes esparcidos geográficamente cuyas coordenadas son conocidas.
- La demanda total de los clientes seleccionados para el reparto de cada vehículo no excederá la capacidad Q en cada viaje.
- La duración total de los viajes asignados a un vehículo no superará un valor establecido T_H.

Si bien estas premisas serán tomadas como base para la definición inicial de la heurística utilizada, en las modificaciones llevadas a cabo sobre la misma se indicará cuáles estas han sido modificadas con el fin de ampliar el espectro de tipos de problemas posibles a resolver, llegando a dar una solución completa para el problema MTVRPTW.

4.1.2 Algoritmo Ahorros para CVRP

El procedimiento que se debe seguir para la correcta ejecución de este algoritmo, una vez se dispone de todos los datos necesarios, se detalla a continuación:

1. Cálculo de matriz de distancias. Dadas las coordenadas de las localizaciones de los clientes, se calcula la distancia euclídea entre cada par de localizaciones, obteniéndose así una matriz de distancias simétrica. Se considera c_{ij} como la distancia o coste de reparto entre la localización i y la localización j.

2. Cálculo de matriz de ahorros. Una vez calculadas las distancias, se procede al cálculo del ahorro entre cada par de clientes. Se calcula la diferencia entre la distancia total o coste del reparto a dos clientes individuales, presentado en la Figura 1, y la distancia al combinar el reparto a ambos clientes, presentado en la figura 2. Las distancias totales de cada uno de los viajes individuales a los clientes i y j, indicados en la serán respectivamente d_i y d_j.

$$d_i = c_{0i} + c_{i0}$$
$$d_j = c_{0j} + c_{j0}$$

La distancia de la ruta combinada de la figura 2 se identifica con el término d_{ij}.

$$d_{ij} = c_{0i} + c_{i0}$$

Así, la fórmula de ahorros, diferencia entre las situaciones presentadas, es la siguiente:

$$s_{ij} = d_i + d_j - d_{ij} = c_{0i} + c_{i0} + c_{0j} + c_{j0} - (c_{0i} + c_{ij} + c_{j0})$$

Sabiendo que $c_{0i} = c_{i0}$ y que $c_{0j} = c_{j0}$, se obtiene la expresión final tradicional del cálculo de ahorros de Clarke & Wright.

$$\boldsymbol{s_{ij} = 2 * (c_{0i} + c_{0j}) - (c_{0i} + c_{ij} + c_{0j})}$$

Simplificando términos de cada unas de las dos opciones de reparto, se podría llegar a obtener la siguiente expresión:

$$\boldsymbol{s_{ij} = c_{0i} + c_{0j} - c_{ij}}$$

Ambas expresiones son totalmente válidas y se deben usar en función del sentido que se le quiera dar a la misma, expresando en ella la diferencia entre los dos viajes o tratando de simplificarla al máximo.

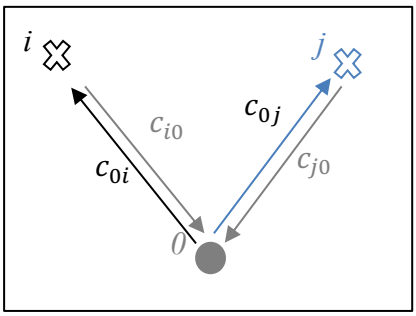


Figura 1. Reparto individual.

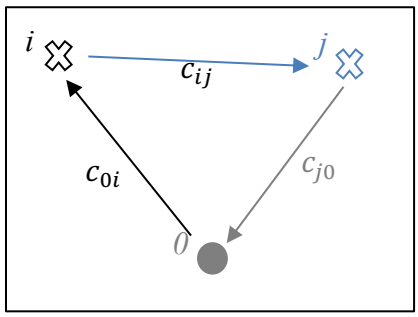
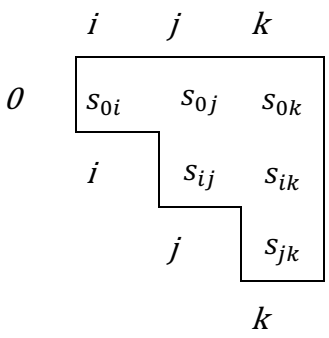


Figura 2. Reparto conjunto.

Una vez conocida la expresión, se aplica para el total de localizaciones de los clientes obteniéndose una matriz simétrica de la cual solo se tendrá en cuenta la forma de la matriz diagonal superior. Los términos de la diagonal principal serán nulos, dado que el ahorro entre el reparto una localización y la combinación de rutas con ella misma siempre será inexistente.

Se obtiene, por tanto, una matriz s como la indicada a continuación, donde la localización 0 corresponderá al depósito y el resto a los diferentes clientes. Solo se considerarán aquellos ahorros no negativos. A los ahorros menores que 0 se les asignará el número 0 automáticamente.

Matriz de ahorros.



3. Búsqueda del máximo ahorro y formación de nuevas rutas. Se busca el componente con el máximo valor en la matriz de ahorros. Sus índices permitirán identificar los clientes cuyas localizaciones forman una nueva combinación con ahorro máximo.
4. Comprobación capacidad. Se comprueba que la combinación de clientes es factible en términos de capacidad teniendo en cuenta la demanda de cada uno de ellos. Si la suma de demandas no excede del valor establecido, se procede a establecer la nueva ruta. En caso contrario se desecha ese máximo ahorro y se busca el siguiente en orden descendente, volviendo al punto 3.
5. Nueva ruta. Se establece la nueva ruta tras la comprobación de capacidad. Si los clientes no están aún en ninguna ruta combinada ya establecida, se crea una nueva. Si, por el contrario, uno de los dos clientes se encuentra en una nueva ruta, se añadirá el otro a la ruta combinada ya existente.
6. Actualización matriz de ahorros. Una vez usada dicha componente de la matriz y formada la ruta, se le asigna un número negativo alto con el fin de que nunca se vuelva a elegir esa componente como máximo ahorro de la matriz y se vuelve al punto 3.

Este algoritmo se aplica de forma reiterativa hasta que no queden clientes con rutas para asignar o bien la matriz de ahorros se haya desechado por completo, quedando aquellos clientes que no han encontrado una ruta de ahorro factible con un viaje individual de ida y vuelta a su localización.

4.1.3 Modificaciones de la heurística

Si bien se han planteado las bases del algoritmo de Clarke y Wright según lo definieron sus autores, el Multi-trip VRP con ventanas temporales, problema que se ha presentado para su resolución en este documento, contiene una serie de restricciones adicionales que no se consideran en la heurística original planteada por Clarke y Wright.

Por tanto, dado que el problema a tratar no solo contiene limitaciones de capacidad en el reparto, sino también restricciones de temporalidad y se añade la posibilidad de realizar multiviaje, se ha desarrollado una adaptación del algoritmo de ahorros original con el fin de cubrir todos estos aspectos no contemplados previamente. No obstante, la base de la heurística se ha mantenido en torno al cálculo de ahorros.

4.1.3.1 Limitaciones de ventanas temporales

A la hora de establecer una solución, es interesante considerar las ventanas temporales dentro de las cuales es posible el reparto en cada uno de los clientes. Esto acerca aún más el problema tratando a un entorno realista donde cada cliente tiene un horario de apertura y cierre diferentes. Así, además de tener un máximo ahorro los dos clientes que vayan a conformar la futura nueva ruta, esta deberá ser viable en términos temporales, directamente ligados a la distancia a recorrer y la ordenación del reparto a clientes.

Para ello, según el procedimiento llevado a cabo por la heurística definido en el apartado 4.1.2, tras el punto de comprobación de la limitación de capacidad (4), debería añadirse una comprobación adicional de ventanas temporales. Es decir, el algoritmo simularía los tiempos de llegada y salida de los diferentes hipotéticos clientes antes de establecer las modificaciones en las rutas existentes de forma definitiva.

Así, en caso de que los dos índices pertenecientes al máximo ahorro de la matriz no estuviesen incluidos en ninguna ruta ya establecida y se quisiera formar una nueva ruta, también se comprobaría la viabilidad temporal, probándose a invertir el orden de reparto de los clientes en caso de que no existiese factibilidad en un sentido. Si no fueran viables ninguno de las dos opciones, se descartaría esta ruta de ahorro.

4.1.3.2 Opción Multitrip

Una vez definido en problema tipo CVRPTW, se pasaría al desarrollo de la opción de multi-trip o multiviaje, obteniendo un MTVRPTW. La posibilidad de añadir o no un viaje múltiple llevado a cabo por un mismo vehículo, volviendo al depósito a recoger mercancía en una o más de una ocasión vendría forzada por la falta de capacidad de un vehículo y el exceso de tiempo restante en el periodo temporal definido para el reparto.

Los pasos por seguir serían los mismos que los establecidos en el punto 4.1.2 hasta la comprobación de capacidad. En caso de superar el límite establecido de capacidad al añadir el nuevo cliente a una ruta existente, se abriría la posibilidad de añadir un viaje adicional por parte del mismo vehículo, antes o después del ya establecido, para repartir al cliente que sobrepasaba la capacidad. Las ventanas temporales se comprobarían de forma idéntica a como se explicaba en el punto anterior. Se comprobaría que el coste de volver al depósito a recoger mercancía para repartir al nuevo cliente fuese menor que un coste por vehículo previamente establecido.

Fijada la ruta con el multiviaje, se trataría igual que el principio o final de una ruta simple, pudiéndose añadir nuevos clientes a la misma mediante el cálculo de ahorros.

4.2 Algoritmo

En el punto 4.1 se ha expuesto en profundidad la estructura y el funcionamiento de la heurística final elegida con sus modificaciones pertinentes. En este apartado, por tanto, se indicará el funcionamiento general del algoritmo a partir de un pseudocódigo donde se explicitará, grosso modo, como actúa la heurística desarrollada en cada caso.

El código completo del algoritmo que se ve reflejado en el pseudocódigo que se expondrá a continuación se ha desarrollado en *Pycharm Community 2018.2.* en diferentes scripts que, en conjunción, permiten el correcto funcionamiento de la heurística con sus diferentes modificaciones. Para más detalle acerca del código y su desarrollo, ver el apartado *Anexo* al final de este documento.

Pseudocódigo.

1. *Cálculo matriz de distancias* $\boldsymbol{distancia}$
2. *Cálculo matriz de ahorros* $\boldsymbol{s}$
3. *Definición de la variable clientes pendientes* ($\boldsymbol{clientes\_ptes}$)
4. *Definición de parámetros*: *número máximo de vehículos, coste de nuevo vehículo y capacidad máxima* ($\boldsymbol{numero\_vehiculos}, \boldsymbol{coste\_nuevo\_vehiculo}, \boldsymbol{capacidad\_maxima}$)
5. Definición del vector solución **ruta**.
6. *While* $\boldsymbol{clientes\_ptes} = ! \emptyset$ *or* $it < 1000$:
   6.1. *Calcular máximo de* $\boldsymbol{s}$
   6.2. *Calcular los índices del máximo de* $\boldsymbol{s}$. ($\boldsymbol{indice0}, \boldsymbol{indice1}$)
   6.3. $\boldsymbol{s}[\boldsymbol{indice0}, \boldsymbol{indice1}] = \boldsymbol{-10000}$
   6.4. *Identificar si los clientes identificados existen ya en alguna de las rutas, así como la posición de estas con las variables* $\boldsymbol{inicio0}, \boldsymbol{inicio1}, \boldsymbol{fin0}, \boldsymbol{fin1}, \boldsymbol{rutanueva}, \boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$.

```
if inicio0 = True or inicio1 = True:
    comprob_capacidad = Comprobante de limite de capacidad
    comprob_temporal = Comprobante de límite de ventanas temporales
        if comprob_capacidad and comprob_temporal
            Actualizar ruta en i, j, k con cliente (indice0 o indice1) correspondiente al inicio
        else:
            comprob_multitrip = Comprobante booleano del multitrip
            if comprob_multitrip
                Actualizar ruta con multitrip al inicio
elif fin0 = True or fin1 = True:
    comprob_capacidad = Comprobante de limite de capacidad
    comprob_temporal = Comprobante de límite de ventanas temporales
        if comprob_capacidad and comprob_temporal
            Actualizar ruta en i, j, k con cliente (indice0 o indice1) correspondiente al final
        else:
            comprob_multitrip = Comprobante booleano del multitrip
            if comprob_multitrip
                Actualizar ruta con multitrip al final
elif rutanueva = True:
    Comprobar el mejor orden de los índices y su factibilidad
    Crear una nueva ruta con indice0 e indice1.
elif it > param:
    Crear rutas individuales con los clientes restantes de clientes_ptes.
```

   6.5. *Actualizar matriz de ahorros* $\boldsymbol{s}$ *eliminando las filas y columnas de los clientes que no estén ni al prinicpio ni al final de una ruta.*

4.3 Implicaciones del procedimiento diseñado en los resultados

La elección del algoritmo de ahorros de Clarke y Wright modificado para el cálculo de la mejor ruta posible en el problema Multi-trip VRP permite intuir los resultados que se obtendrán en el apartado posterior. Al tratarse de una heurística no combinatoria, se puede prever un tiempo de ejecución del algoritmo relativamente reducido, así como una solución que, pese a ser completamente factible, distará en mayor o menor medida de los valores encontrados como óptimos o cercanos a este por diferentes investigadores para ciertas instancias de problemas.

No obstante, el diseño de las modificaciones implementada para posibilitar los viajes múltiples y tener en consideración las restricciones temporales dará total libertad de decidir el uso o no de estos, ofreciendo una heurística versátil diseñada para resolver tanto un VRPTW como un MTVRPTW según las necesidades del problema. Así, el algoritmo también permitirá hacer comparativas entre el comportamiento y la mejora del Multi-trip VRP con respecto al VRP estándar.

Por tanto, a pesar de utilizar un método relativamente sencillo como es el cálculo de los ahorros, se prevén resultados razonables que, posiblemente, con pequeñas alteraciones en el cálculo de ahorros puedan mejorar notablemente.

5 RESULTADOS

Se ha planteado en el apartado anterior la heurística adaptada a partir del algoritmo de ahorros de Clarke y Wright que se empleará para la resolución del problema multi-trip VRP.
Se procede a continuación a la resolución de un problema concreto de rutado de vehículos con datos tomados de una instancia de problemas VRP. Para ello, se explicará detenidamente los tipos de problemas existentes en la instancia y los resultados obtenidos para cada tipo de problema planteado.

5.1 Instancias utilizadas

Las instancias utilizadas de datos serán las generadas por el profesor Marius M. Solomon en su página web [11]. Los seis tipos de problemas generados por Solomon comparten la misma localización de los hipotéticos clientes, cambiando así las ventanas temporales de disponibilidad del reparto en cada tipo de problema de la siguiente forma:

- Tipo R1 y tipo R2. Las ventanas temporales de cada uno de los clientes están dispuestas de forma aleatoria, tanto su duración como su instante de apertura y cierre.
- Tipo C1 y tipo C2. Las ventanas temporales están agrupadas en función de la cercanía de los clientes entre sí, de forma que las aperturas y cierres de los clientes vecinos serán relativamente próximas en el tiempo.
- Tipo RC1 y RC2. Las ventanas temporales se disponen como una combinación de ventanas aleatorias y ventanas temporales adecuadas a la proximidad entre los clientes.

Todos los tipos de instancias incluyen la capacidad del vehículo, coordenadas de cada cliente, demanda, ventanas temporales y tiempo de servicio. Para la correcta resolución del problema, serán necesarios dos datos adicionales: el límite horario de trabajo por camión y el coste de contratación de un nuevo vehículo. En concreto, para la resolución de este problema, se usarán exclusivamente las instancias tipo C1 y C2.

Si bien la capacidad viene indicada en la instancia, en los seis tipos de problema, esta tendrá un valor alto o muy alto (tipo 1 o 2) en relación con la demanda de cada cliente. Por tanto, según el diseño del algoritmo, que se detallará posteriormente, será imprescindible que un vehículo agote su capacidad para que se plantee la posibilidad de realizar un viaje múltiple o multitrip volviendo al depósito tras un primer reparto para recoger más mercancía. Así, se usarán diferentes valores de capacidad además de los indicados en la instancia con el fin de forzar al algoritmo a utilizar viajes múltiples y así analizar la variación del óptimo encontrado por el problema en función del número de vehículos, distancia total recorrida y tiempo total utilizado.

Se plantearán dos escenarios: la resolución del problema VRP al uso y la solución del problema para hipotéticos vehículos con baja capacidad o autonomía, como se comentaba al principio del documento.

5.2 Resultados

Para la ejecución de la heurística diseñada, se ha utilizado un equipo con las siguientes características:

Tabla 1. Características del equipo informático.

Modelo	Macbook Pro (13-inch, 2017, Two Thunderbolt 3 ports)
Procesador	2,3 Ghz Intel Core i5
Memoria	8 GB 2133 MHz LPDDR3
Gráficos	Intel Iris Plus Graphics 640 1536 MB

Los resultados se dividirán según el tipo de heurística utilizada, el número de clientes totales elegidos, y los parámetros de entrada elegidos (número máximo de vehículos, coste de vehículo adicional, capacidad de cada vehículo). Así, se podrá ver los resultados del VRPTW estándar en comparación con los del MTVRPTW. Además, se incluirá una comparativa con los resultados que ofrece la mejora de la heurística utilizada en cada uno de los casos mencionados.

5.2.1 Resolución VRP sin multiviaje

A continuación, se ejecuta el algoritmo tomando como datos de entrada los de la instancia tipo C, estableciendo los parámetros de entrada de forma que el algoritmo nunca decida tomar la opción de multiviaje como óptima. Esto es, establecer un coste de un nuevo vehículo nulo.

Con estos datos, se pone en marcha la heurística para 25, 50 y 100 clientes con el fin de interpretar la optimalidad de los diferentes resultados en función de la dimensión del problema.

5.2.1.1 Instancias tipo C1. Algoritmo Original

Se procede a resolver los problemas de la instancia tipo C1 usando el algoritmo de ahorros original con unos parámetros de entrada tales que no exista multitrip, como se ha explicado antes. Se utilizará un parámetro auxiliar denominado param que indicará para qué número de iteraciones el algoritmo debe crear rutas individuales con los clientes restantes que no han encontrado ruta de ahorro. Tras varias pruebas, se ha determinado 900 como un parámetro que permite al algoritmo funcionar adecuadamente. No se limitará también el número de vehículos utilizado valores pequeños para que se pueda obtener una solución completa del reparto.

Tabla 2. Parámetros de entrada de la heurística para la instancia C1. Problema VRP.

Capacidad	200
Coste nuevo vehículo	0
Número máx. de vehículos	50
Param	900

En primer lugar, por simplicidad a la hora de la presentación de resultados, se ejecutará el algoritmo para un único problema de la instancia con un total de 25 clientes. Los datos de este problema clasificado en la instancia de Solomon como *c101* se adjuntan a modo de ejemplo de estructura de datos:

CUST NO.	XCOORD.	YCOORD.	DEMAND	READY TIME	DUE DATE	SERVICE TIME
0	40	50	0	0	1236	0
1	45	68	10	912	967	90
2	45	70	30	825	870	90
3	42	66	10	65	146	90
4	42	68	10	727	782	90
5	42	65	10	15	67	90
6	40	69	20	621	702	90
7	40	66	20	170	225	90
8	38	68	20	255	324	90
9	38	70	10	534	605	90
10	35	66	10	357	410	90
11	35	69	10	448	505	90
12	25	85	20	652	721	90
13	22	75	30	30	92	90
14	22	85	10	567	620	90
15	20	80	40	384	429	90
16	20	85	40	475	528	90
17	18	75	20	99	148	90
18	15	75	20	179	254	90
19	15	80	10	278	345	90
20	30	50	10	10	73	90
21	30	52	20	914	965	90
22	28	52	20	812	883	90
23	28	55	10	732	777	90
24	25	50	10	65	144	90
25	25	52	40	169	224	90

La ejecución de la heurística que conforman la heurística da como resultado la ruta, formada como una variable tipo lista con dos listas embebidas en su interior. La lista global del primer nivel representa la solución completa, el segundo nivel, los viajes de un vehículo completo y el último nivel, los diferentes multiviajes que puede realizar un mismo vehículo durante un mismo periodo de reparto.

En este caso elegido como ejemplo, no se encontrarían multiviajes dado que se ha forzado al algoritmo para ello. Además, se obtiene la dimensión de esta ruta solución, que corresponde al número de vehículos utilizado; los clientes pendientes de repartir, cuyo reparto sería subcontratado en caso de que se limitara el número máximo de vehículos por debajo de lo necesario; la distancia total recorrida por todos los vehículos utilizados así como el tiempo total transcurrido en el reparto de todos los vehículos de forma individual y, por último, el tiempo de ejecución de la heurística que proporciona la solución.

La ruta obtenida como solución se ha representado en un mapa de puntos en la Figura 4. Las localizaciones de los clientes están marcadas con aspas negras junto al número de identificación correspondiente, de forma que cada color señalará el viaje realizado por cada vehículo, indicado por la unión de las diferentes aspas negras de cada cliente. Observando la leyenda, se puede ver el número de vehículos utilizados para el reparto.

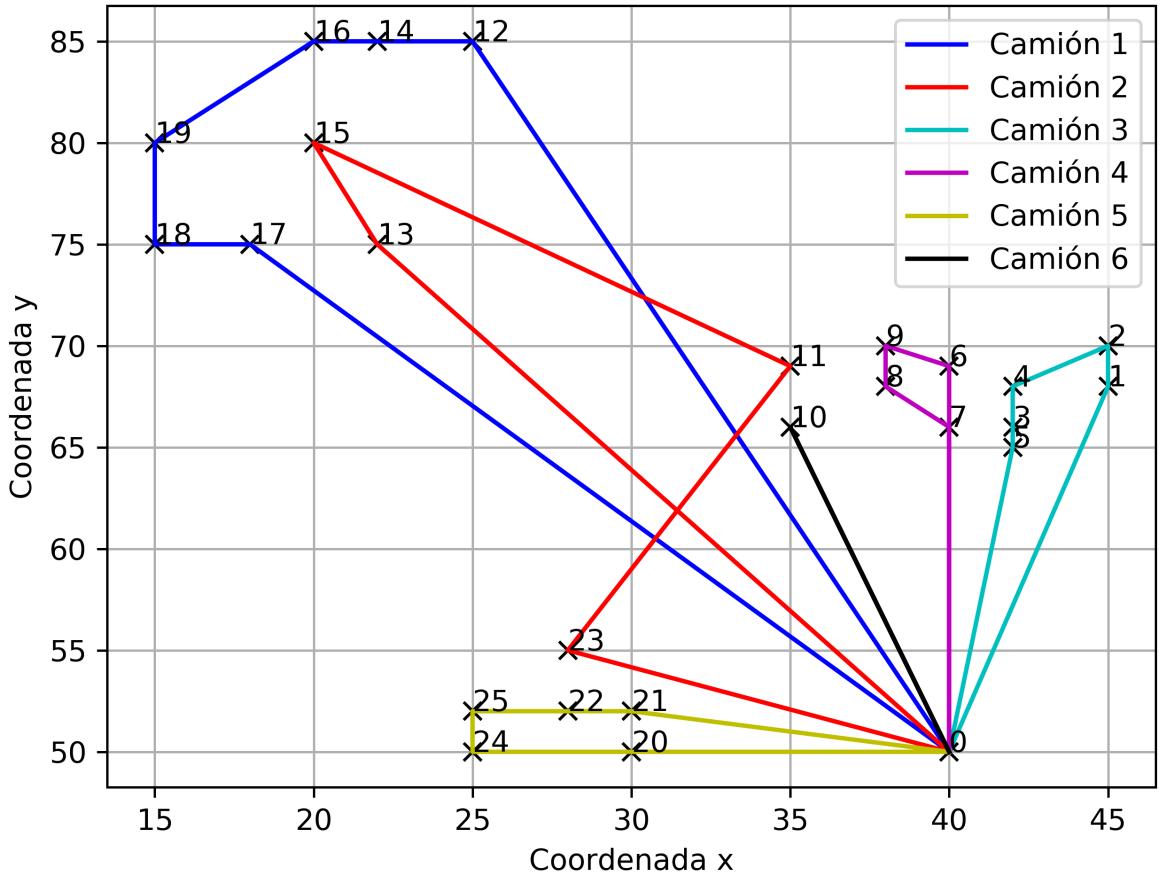


Figura 3. Representación de la ruta solución de un VRPTW con 25 clientes.

Además, a continuación, se muestra una figura que muestra información sobre los tiempos de llegada y salida a los clientes en relación con las ventanas temporales. Así, se indicará el instante de llegada con un punto verde y el instante de salida con un aspa roja. Las ventanas temporales están marcadas con franjas de colores idénticos a los utilizados para cada uno de los camiones de la Figura 4. A pesar de que un vehículo llegue a un cliente en un determinado momento, la descarga no podrá empezar a producirse hasta que comience la ventana temporal, por tanto, el intervalo de estancia en el cliente será mayor. Dicho intervalo se marca con una línea discontinua gris.

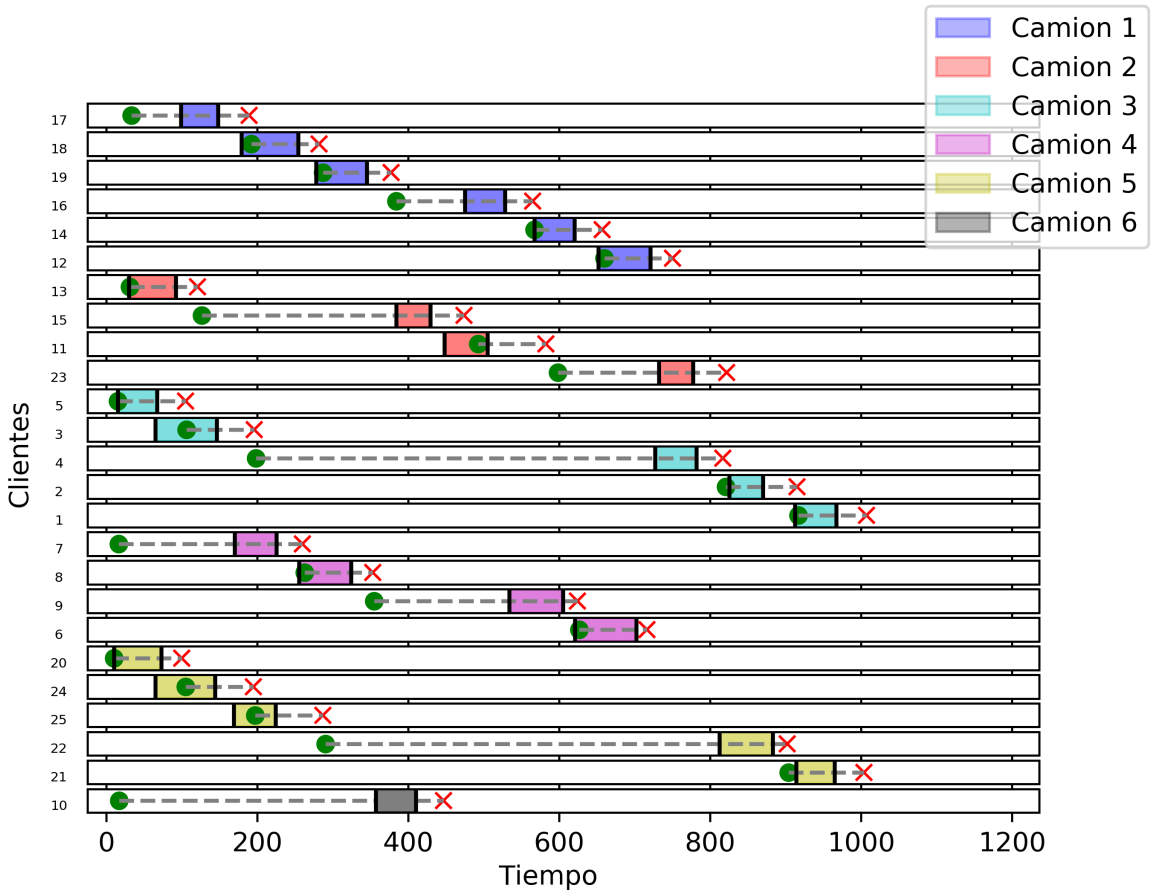


Figura 4. Representación de los instantes de llegada y salida de los vehículos a los clientes en función de sus ventanas temporales para un VRPTW con 25 clientes.

Si bien estos son los resultados obtenidos para un solo problema de la instancia tipo C1 a modo de ejemplo, a continuación, se procederá a mostrar los resultados obtenidos para toda la serie de problemas C1 y C2 para 25, 50 y 100 clientes con los parámetros de entrada indicados al principio.

Tabla 3. VRPTW con N=25 para problemas tipo C1.

Algoritmo Original. N=25.				
Problema	Distancia	N. Vehículos	Tiempo Reparto	Tiempo Ejecución (s)
c101	325,10	6	4861,96	0,142
c102	303,56	5	4387,87	0,021
c103	320,94	5	4157,44	0,018
c104	317,24	5	4038,39	0,019
c105	321,17	6	4566,28	0,143
c106	334,48	6	4929,39	0,136
c107	321,17	6	4346,88	0,147
c108	291,95	5	3813,80	0,020
c109	283,87	5	3819,40	0,021

Tabla 4. VRPTW con N=50 para problemas tipo C1.

Algoritmo Original. N=50.				
Problema	Distancia	N. Vehículos	Tiempo Reparto	Tiempo Ejecución (s)
c101	584,01	10	8624,66	0,296
c102	568,42	9	8146,74	0,070
c103	590,79	9	7690,26	0,071
c104	522,30	7	6456,84	0,062
c105	584,01	10	8320,66	0,295
c106	593,39	10	8387,49	0,282
c107	584,01	10	7945,27	0,299
c108	555,08	9	6807,58	0,071
c109	520,96	8	5789,78	0,068

Tabla 5. VRPTW con N=25 para problemas tipo C1.

Algoritmo Original. N=100.				
Problema	Distancia	N. Vehículos	Tiempo Reparto	Tiempo Ejecución
c101	1049,70	15	13409,39	0,66
c102	1132,00	16	13865,65	0,39
c103	1177,49	15	13401,84	0,40
c104	1168,82	13	12023,29	0,39
c105	1049,70	15	13108,39	0,68
c106	1059,08	15	13227,62	0,69
c107	1049,70	15	12726,39	0,70
c108	1020,77	14	11724,53	0,39
c109	986,65	13	10751,73	0,36

Como se puede observar, a medida que aumenta el número de clientes tomados de la instancia, aumentan también los campos dispuestos en las tablas de resultados. Debido a que la resolución de este problema se ha estandarizado por la mayoría de los autores con un total de 100 clientes, esta será la dimensión usada.

Estos valores de distancia total y tiempo transcurrido para el reparto obtenidos distan de los valores óptimos encontrados en la página web de Solomon [11], no obstante, conforman una solución factible al problema que se está buscando en un corto periodo de tiempo. Aún así, existe un margen de mejora, el cual se tratará de acortar en el próximo apartado.

Si bien, en apartados próximo donde se expondrán las soluciones obtenidas para el problema usando el multiviaje, se volverán a tener en cuenta problemas con un número de clientes menor a 100, esto solo será a modo explicativo para que las ilustraciones mostradas se puedan llegar a comprenderse con mayor facilidad y claridad. Así, el estándar de resolución se mantendrá en los 100 clientes.

Por tanto, a continuación, se adjuntan las figuras homólogas a las utilizadas para 25 clientes respecto al problema c101, pero esta vez calculadas para un total de 100 clientes.

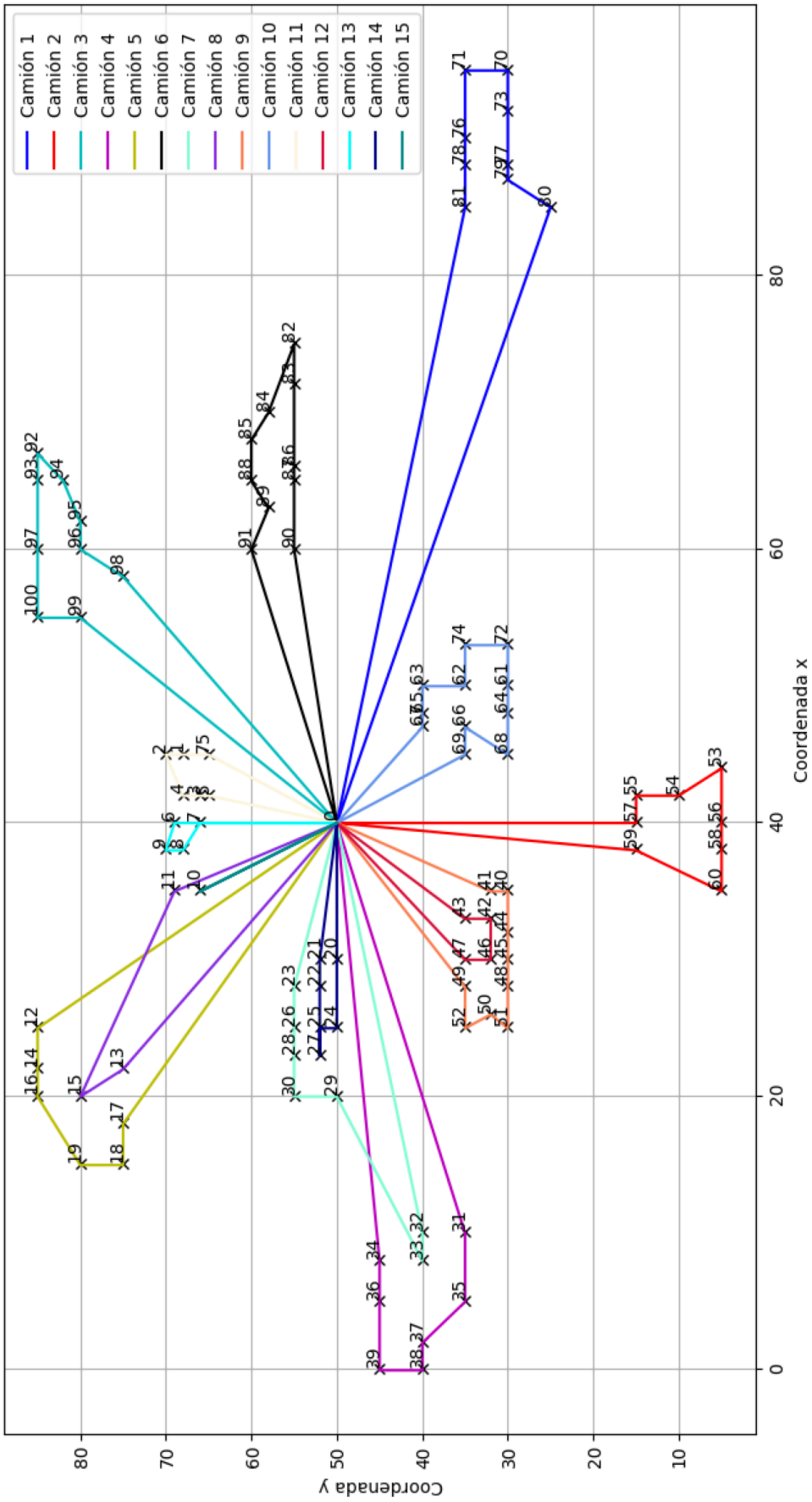


Figura 5. Solución en un mapa puntos para un VRPTW con 100 clientes.

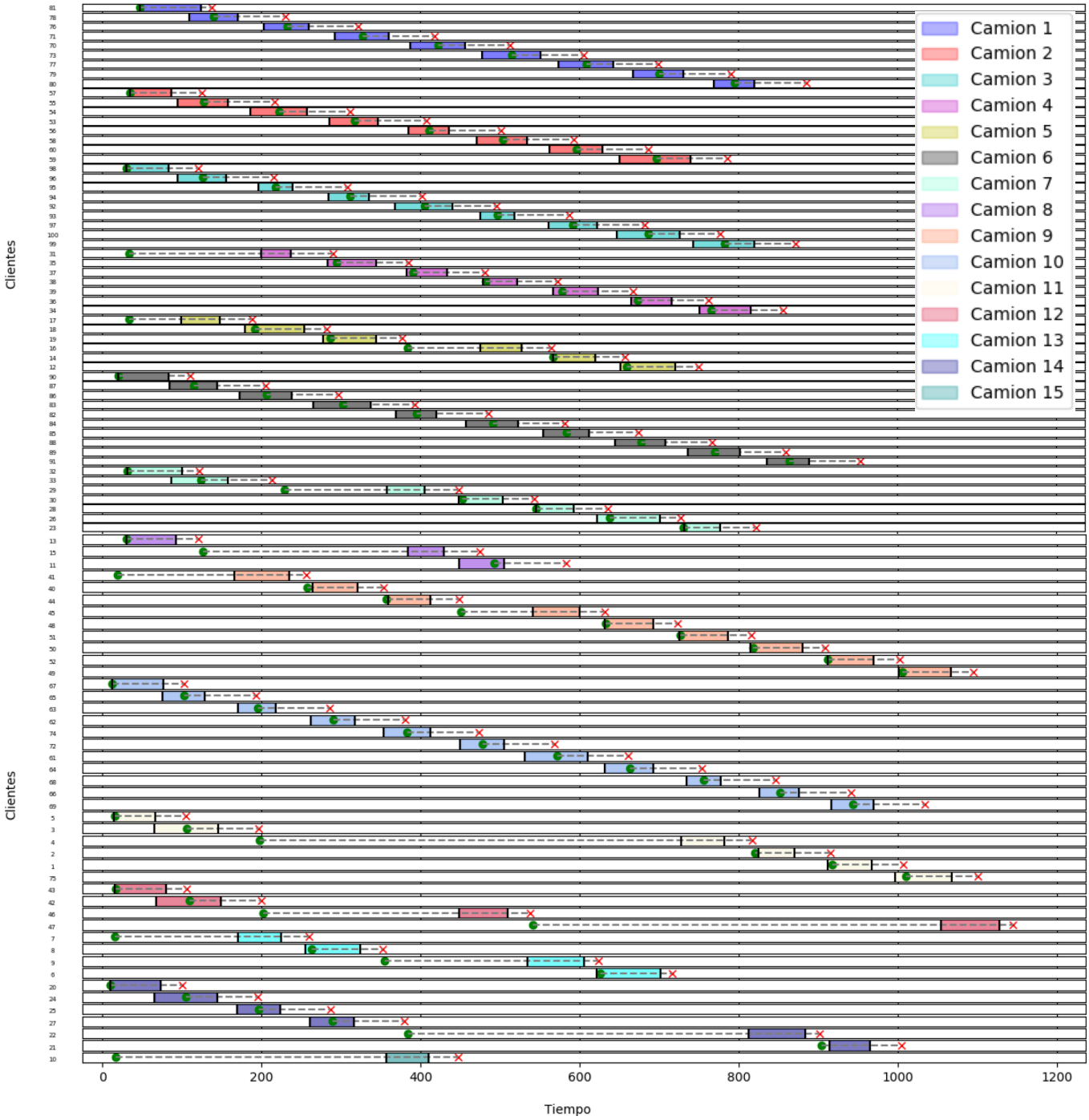


Figura 6. Representación tiempos de llegada y salida según las ventanas temporales paraa un VRPTW con 100 clientes.

5.2.1.2 Instancias tipo C1. Algoritmo mejorado.

Como se ha explicado previamente, es evidente que existe un margen de mejora en los resultados obtenidos. Por tanto, tratando de obtener una mejor solución, se ha implementado una propuesta de mejora al algoritmo original, reformulándose el cálculo de los ahorros teniendo en cuenta los tiempos de apertura y cierre tratando que tanto la distancia recorrida por todos los vehículos como el tiempo total de trayecto se reduzcan.

Partiendo de la expresión del cálculo de ahorros original:

$$s_{ij} = c_{0i} + c_{0j} - c_{ij}$$

Se propone la siguiente expresión:

$$s_{ij} = c_{0i} + c_{0j} - c_{ij} + coef * ((L_i - L_j) - (E_i - E_j))$$

Siendo $coef$ un parámetro que tomará valores positivos, L_k el tiempo límite de entrega a un cliente k y E_k el instante de apertura.

Se procederá a la obtención de los resultados de ejecución con diferentes valores de $coef$ tratando de buscar aquel que mejore los resultados globales. Para ello, además de los campos anteriores obtenidos en la tabla del algoritmo original, se añadirán unos campos de comparación entre este y el algoritmo mejorado, pudiéndose así valorar la mejora global en todos los problemas de la instancia tipo C1.

Tabla 6. VRPTW con algoritmo mejorado para N=100 y Coef =0,001. Problema tipo C1.

Algoritmo Mejorado. N=100. Coef = 0,001							
Problema	Distancia	N. Vehículos	T. Reparto	T. Ejecución	Dif. Distancia	Dif. Vehículos	Dif. Reparto
c101	1049,70	15	13409,39	0,59	0,00	0	0,00
c102	1137,77	15	13984,92	0,56	-5,78	1	-119,27
c103	1189,69	15	13509,99	0,39	-12,20	0	-108,14
c104	1272,53	14	12649,61	0,42	-103,72	-1	-626,32
c105	1049,70	15	13108,39	0,70	0,00	0	0,00
c106	1059,08	15	13227,62	0,68	0,00	0	0,00
c107	1049,70	15	12726,39	0,71	0,00	0	0,00
c108	1022,07	14	11693,19	0,38	-1,30	0	31,34
c109	986,65	13	10751,73	0,37	0,00	0	0,00
Mejora total					-123,00	0	-822,40

Tabla 7. VRPTW con algoritmo mejorado para N=100 y Coef =0,005. Problema tipo C1.

Algoritmo Mejorado. Coef = 0,0005							
Problema	Distancia	N. Vehículos	T. Reparto	T. Ejecución	Dif. Distancia	Dif. Vehíc.	Dif. Reparto
c101	1049,70	15	13409,39	0,62	0,00	0	0,00
c102	1137,17	15	13984,32	0,56	-5,18	1	-118,68
c103	1130,58	14	13398,73	0,39	46,91	1	3,11
c104	1211,72	14	12706,03	0,42	-42,90	-1	-682,74
c105	1049,70	15	13108,39	0,70	0,00	0	0,00
c106	1059,08	15	13227,62	0,71	0,00	0	0,00
c107	1049,70	15	12726,39	0,72	0,00	0	0,00
c108	1022,07	14	11693,19	0,38	-1,30	0	31,34
c109	986,65	13	10751,73	0,36	0,00	0	0,00
Mejora total					-2,47	1	-766,97

Tabla 8. VRPTW con algoritmo mejorado para N=100 y Coef =0,0001. Problema tipo C1.

Algoritmo Mejorado. Coef = 0,0001							
Problema	Distancia	N. Vehículos	T. Reparto	T. Ejecución	Dif. Distancia	Dif. Vehíc	Dif. Reparto
c101	1049,70	15	13409,39	0,64	0,00	0	0,00
c102	1132,00	16	13865,65	0,40	0,00	0	0,00
c103	1114,99	14	13115,17	0,39	62,50	1	286,68
c104	1158,75	13	12105,53	0,41	10,06	0	-82,24
c105	1049,70	15	13108,39	0,70	0,00	0	0,00
c106	1059,08	15	13227,62	0,70	0,00	0	0,00
c107	1049,70	15	12726,39	0,69	0,00	0	0,00
c108	1020,77	14	11724,53	0,37	0,00	0	0,00
c109	986,65	13	10751,73	0,36	0,00	0	0,00
Mejora total					72,57	1	204,44

Tabla 9. VRPTW con algoritmo mejorado para N=100 y Coef =0,0005.

Algoritmo Mejorado. Coeficiente = 0,0005							
Problema	Distancia	N. Vehículos	T. Reparto	T. Ejecución	Dif. Distancia	Dif. Vehíc	Dif. Reparto
c101	1049,70	15	13409,39	0,593922	0,00	0	0,00
c102	1132,00	16	13865,65	0,295756	0,00	0	0,00
c103	1114,99	14	13115,17	0,289125	62,50	1	286,68
c104	1158,75	13	12105,53	0,322168	10,06	0	-82,24
c105	1049,70	15	13108,39	0,603871	0,00	0	0,00
c106	1059,08	15	13227,62	0,606917	0,00	0	0,00
c107	1049,70	15	12726,39	0,609354	0,00	0	0,00
c108	1020,77	14	11724,53	0,288253	0,00	0	0,00
c109	986,65	13	10751,73	0,265511	0,00	0	0,00
Mejora total					72,57	1	204,44

Como se puede apreciar en las tablas, el resultado converge a valores mejorados a partir del valor de $coef = 0{,}0001$. Se determina así que valores de $coef < 0{,}0001$ no mejorarán el resultado existente. Por tanto, se tomará como estándar de mejor valor para el parámetro para el resto de los problemas que se planteen en el documento.

Se observan en el resultado final tanto una disminución global de la distancia recorrida en los problemas c103 y c104 como la disminución de los vehículos utilizados en el problema c103. Si bien el tiempo de reparto aumenta en el problema c104, este se produce a raíz de la disminución de distancia recorrida. Por tanto, se considerará que el problema mejora y que el valor del parámetro $coef$ es óptimo.

5.2.1.3 Instancias tipo C2. Algoritmo Original y Algoritmo Mejorado.

La diferencia entre los problemas tipo C1 y C2 es, además de la distribución de clientes y sus ventanas temporales, la capacidad de los vehículos. En este caso, los parámetros de entrada del problema serían los siguientes:

Tabla 10. Parámetros de entrada de la heurística para la instancia C2. Problema VRP.

Capacidad	700
Coste nuevo vehículo	0
Número máx. de vehículos	50
Param	900

Se procede a mostrar los resultados al igual que en los apartados anteriores, en este caso tanto del algoritmo original como la mejora con el valor del parámetro establecido para la batería de problemas de la instancia en cuestión.

Tabla 11. VRPTW con algoritmo original para N=100. Problemas tipo C2.

Algoritmo Original. N=100.				
Problema	Distancia	Número Vehículos	Tiempo Reparto	Tiempo Ejecución
c201	972,51	10	25104,58	0,257
c202	962,86	10	21817,68	0,334
c203	1003,51	10	21478,95	0,349
c204	944,70	8	18658,15	0,320
c205	855,00	9	20761,27	0,312
c206	854,81	9	20170,74	0,318
c207	882,53	9	18918,71	0,306
c208	915,13	10	22459,14	0,329

Tabla 12.VRPTW con algoritmo mejorado para N=100 y Coef =0,001. Problema tipo C2.

Algoritmo Mejorado. N= 100. Coef = 0,0001							
Problema	Distancia	N. Vehículos	T. Reparto	T. Ejecución	Dif. Distancia	Dif. Vehíc.	Dif. reparto
c201	972,51	10	25104,58	0,265	0,00	0	0,00
c202	965,46	10	21832,24	0,347	-2,59	0	-14,56
c203	1007,81	10	21494,37	0,351	-4,30	0	-15,42
c204	981,42	9	21462,28	0,316	-36,72	-1	-2804,12
c205	855,00	9	20761,27	0,321	0,00	0	0,00
c206	854,81	9	20170,74	0,326	0,00	0	0,00
c207	881,79	9	18827,28	0,331	0,74	0	91,43
c208	915,13	10	22459,14	0,342	0,00	0	0,00
Mejora total					-42,87	-1	-2742,67

En este caso, el algoritmo con el cálculo de ahorros propuesto no mejora la solución del algoritmo original para ninguno de los aspectos reflejados en la tabla.

El único factor diferencial de este tipo de problemas con respecto al tipo C1 es la capacidad, que es más de tres veces mayor. Por tanto, se usará como parámetro de entrada una capacidad menor igual a la de los problemas tipo C1 y se estudiará su posible mejora. Los parámetros de entrada del problema pasan a ser los siguientes:

Tabla 13.Parámetros de entrada de la heurística para la instancia C1. Problema VRP. Modificación.

Capacidad	200
Coste nuevo vehículo	0
Número máx. de vehículos	50
Param	900

Tabla 14.VRPTW con algoritmo original para N=100. Problemas tipo C2. Capacidad modificada.

Algoritmo Original				
Problema	Distancia	Número Vehículos	Tiempo Reparto	Tiempo Ejecución
c201	1150,00	14	30461,51	0,392
c202	1152,13	13	30183,69	0,391
c203	1136,77	12	25463,61	0,392
c204	1249,24	11	25316,34	0,399
c205	1068,70	13	26747,25	0,362
c206	1083,34	13	26196,24	0,373
c207	1053,18	12	24080,68	0,358
c208	1127,66	14	28281,09	0,392

Tabla 15. VRPTW con algoritmo mejorado para N=100 y Coef =0,001. Problema tipo C2. Capacidad modificada.

Algoritmo Mejorado. N=100. Coef = 0,0001							
Problema	Distancia	N. Vehículos	T. Reparto	T. Ejecución	Dif. Distancia	Dif. Vehíc.	Dif. reparto
c201	1150,00	14	30461,51	0,395	0,00	0	0,00
c202	1152,31	13	28864,28	0,391	-0,18	0	1319,40
c203	1135,43	12	25559,58	0,387	1,35	0	-95,98
c204	1178,42	10	22695,10	0,367	70,82	1	2621,24
c205	1068,70	13	26747,25	0,379	0,00	0	0,00
c206	1083,34	13	26196,24	0,373	0,00	0	0,00
c207	1052,45	12	23989,24	0,374	0,74	0	91,43
c208	1127,66	14	28281,09	0,398	0,00	0	0,00
Mejora total					72,73	1	3936,10

Con la capacidad máxima de 200 unidades por vehículo, el algoritmo mejorado vuelve a dar unos valores de salida reducidos en comparación al algoritmo original, si se estudia los resultados de todos los problemas con una perspectiva global, a pesar de que algunos empeoren mínimamente en algún factor.

Por tanto, se puede afirmar que, para valores de capacidad altos el algoritmo mejorado no proporciona buenos resultados. En el tipo de problema estudiado en este documento, MTVRPTW, no tiene sentido tomar como capacidad máxima por vehículo un número alto por la tipología del problema.

5.2.1.4 Comparación de resultados

Los valores obtenidos parecen concluyentes para considerar que la nueva expresión del cálculo de ahorros propuesta en este documento mejora o mantiene las soluciones de los problemas de rutado de vehículos con capacidad reducida y con localizaciones de clientes agrupadas por zonas, como ocurre en las instancias de Solomon tipo C1 y tipo C2.

Las siguientes gráficas expresan la diferencia en los valores obtenidos de salida para el algoritmo VRPTW en problemas de tipo C1 y tipo C2. En concreto, para la capacidad máxima de 200 unidades por vehículo, que es el valor para el cual también existía mejora en el nuevo cálculo de ahorros propuesto para los problemas tipo C2.

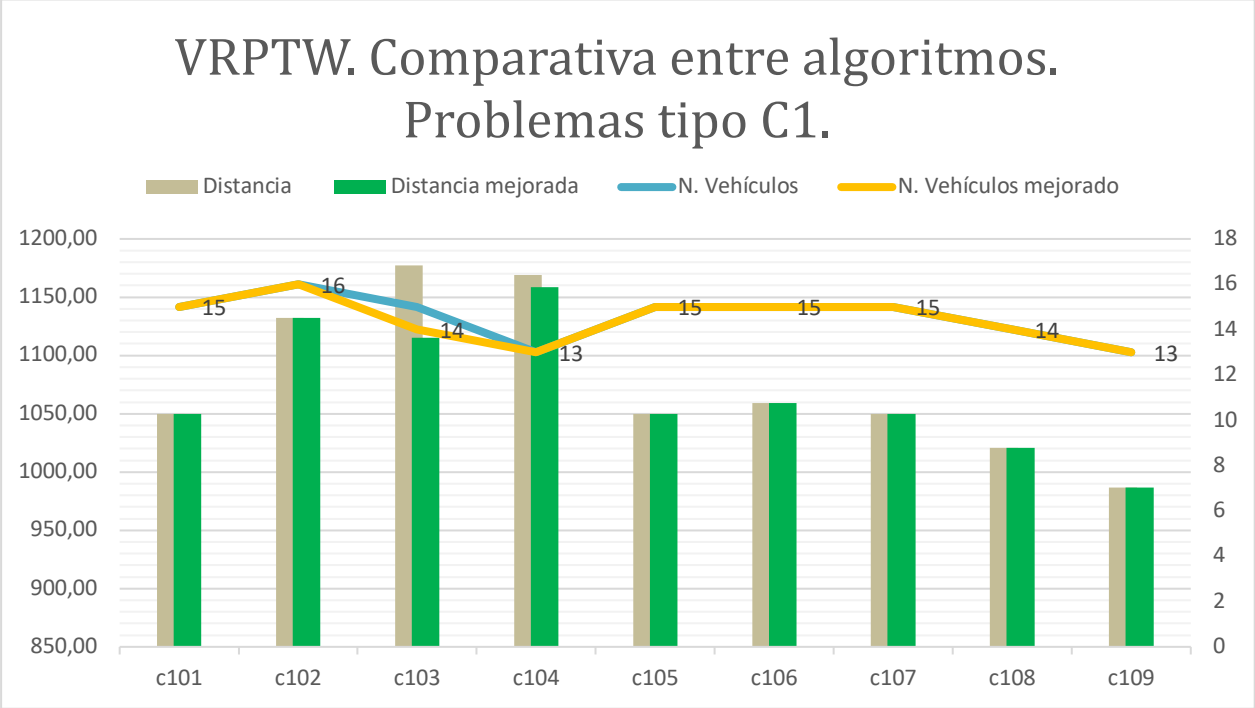


Figura 7. VRPTW. Comparativa entre algoritmos. Problemas tipo C1.

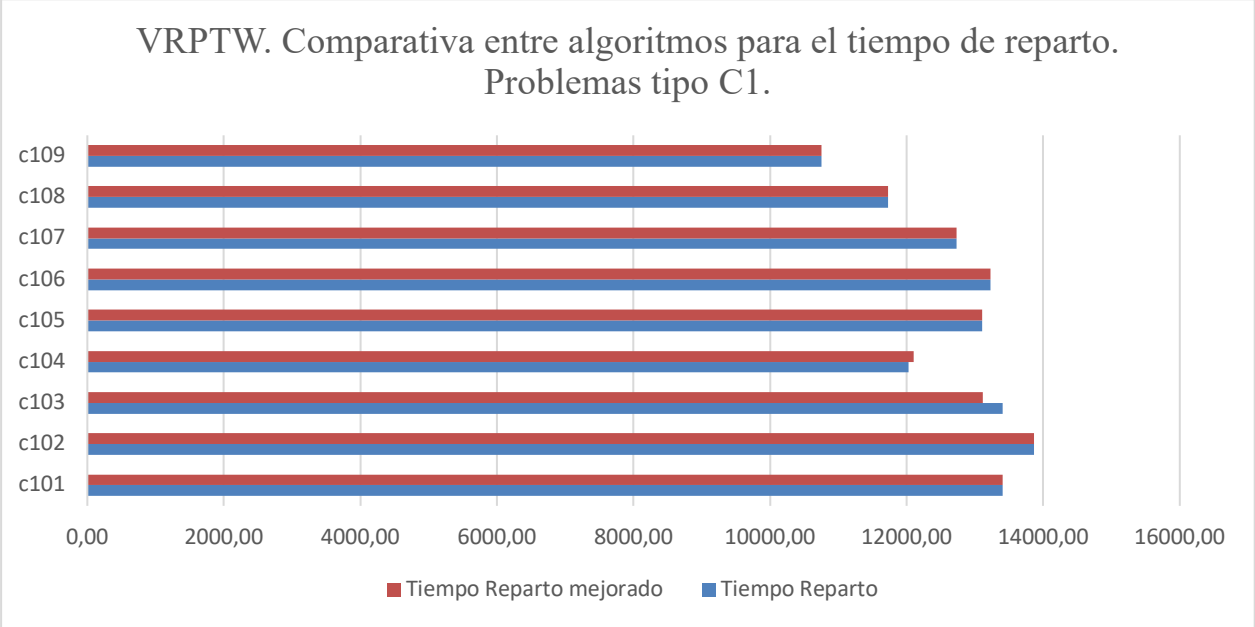


Figura 8. VRPTW. Comparativa entre algoritmos para el tiempo de reparto. Problemas tipo C1.

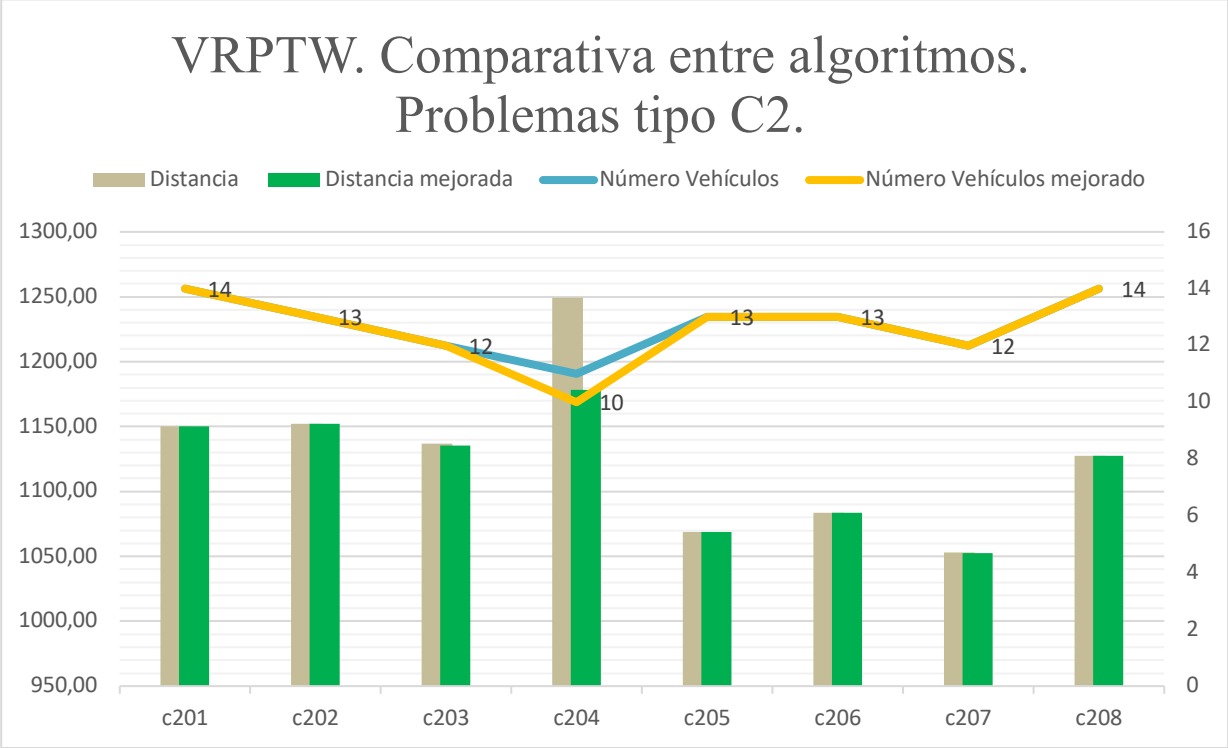


Figura 9. VRPTW. Comparativa entre algoritmos. Problemas tipo C2.

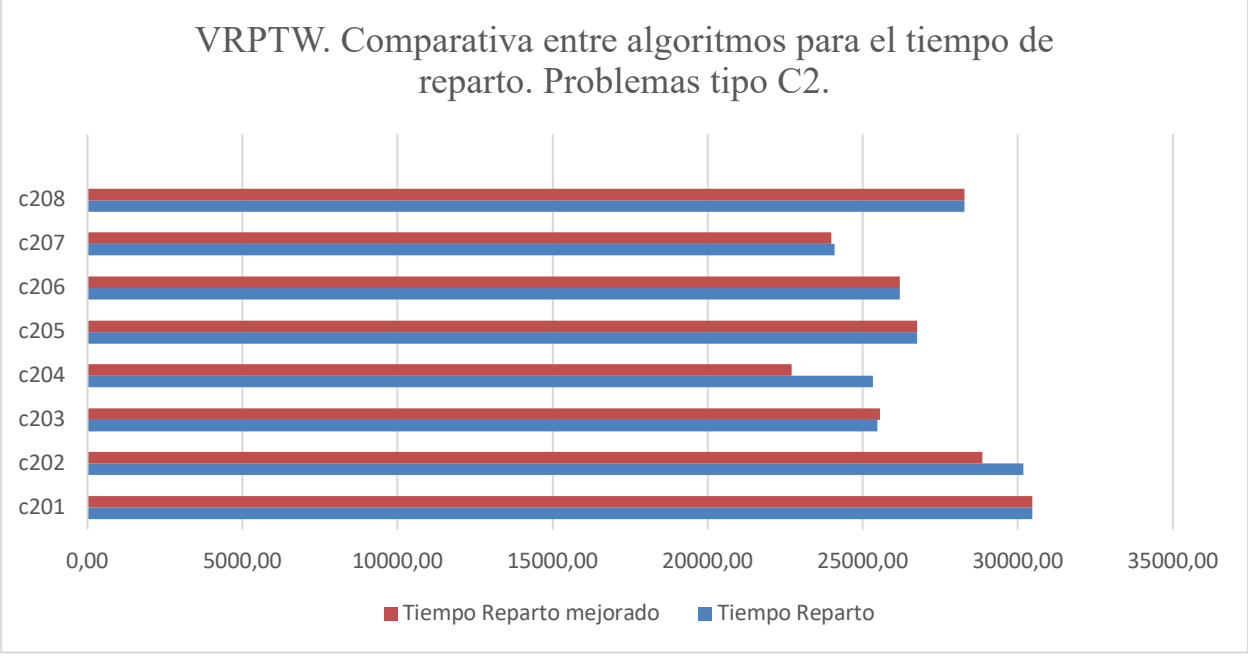


Figura 10. VRPTW. Comparativa entre algoritmos para el tiempo de reparto. Problemas tipo C2.

Por tanto, evidenciadas ya los resultados mejorados que proporciona el nuevo cálculo de ahorros para capacidades bajas, este será utilizado también en el cálculo de resultados del multi-trip VRP y se usará para obtener comparaciones entre el algoritmo multitrip sin modificar y este.

5.2.2 Resolución del Multi-trip VRP

En el apartado 5.4.1 se han mostrado los resultados obtenidos para un CVRP sin viajes múltiples. Se tomará este apartado como referencia para determinar las posibilidades de mejora que puede llegar a ofrecer el Multi-trip VRP en términos de distancia, tiempo total o número de vehículos.

Una vez más, con el fin de tratar de explicar los resultados y por simplicidad a la hora de mostrarlos en las diferentes imágenes obtenidas, se elegirán a modo de ejemplo dimensiones del problema pequeñas, aunque el estándar se mantenga para el cálculo de resultados en 100 clientes.

5.2.2.1 MTVRP. Ejemplo de resultados con algoritmo original.

Para este ejemplo se elegirá el problema c103 con un total de 25 clientes. Los parámetros de entrada, con el fin de forzar al algoritmo a usar multiviajes y que se pueda ejemplificar claramente, serán los siguientes:

Tabla 16. Parámetros de entrada. Ejemplo MTVRPTW.

Capacidad	40
Coste nuevo vehículo	1000
Número máx. de vehículos	15
Param	900

Tratando de mejorar la comprensión acerca del funcionamiento del multiviaje según se ha establecido en la ruta solución, se procede a mostrar las figuras ya generadas para el apartado de VRPTW anterior: Mapa de puntos con las rutas y gráfico con los intervalos temporales de reparto.

En la imagen se puede observar como en el multiviaje existen tanto clientes individuales como combinaciones de rutas. Todo ello dependerá de la factibilidad en términos de capacidad establecida y ventanas temporales planteadas por el problema de la instancia.

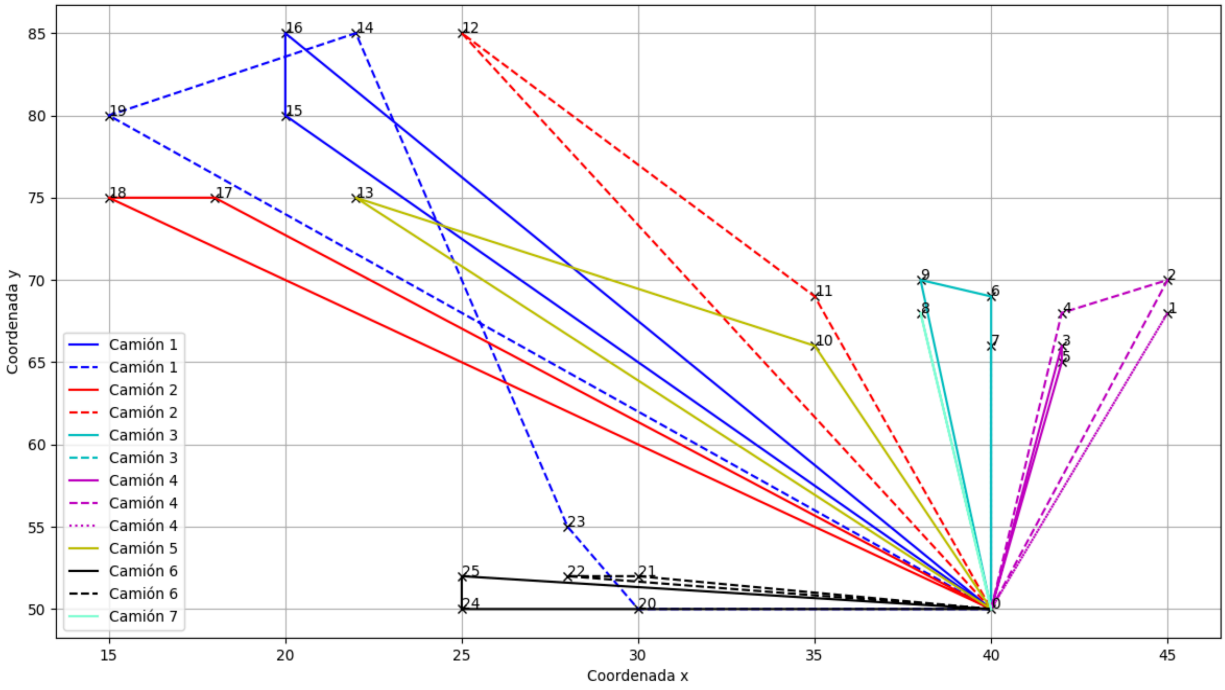


Figura 11. Representación en un mapa de puntos de la ruta obtenida de un MTVRPTW para 25 clientes. Algoritmo original.

Como se puede observar en la leyenda de la imagen superior, hay diferentes tipos de trazo en las líneas que conforman las rutas de los vehículos. Son los viajes múltiples los que se han representado con líneas diferentes a la continua. Así, por ejemplo, el camión 4 de color violeta, debido a los parámetros introducidos previamente en el problema, hará 3 viajes en su jornada de reparto, volviendo al depósito a recoger mercancía dos veces. Su primer viaje lo formará la línea lisa, el segundo la línea discontinua y el último la línea punteada.

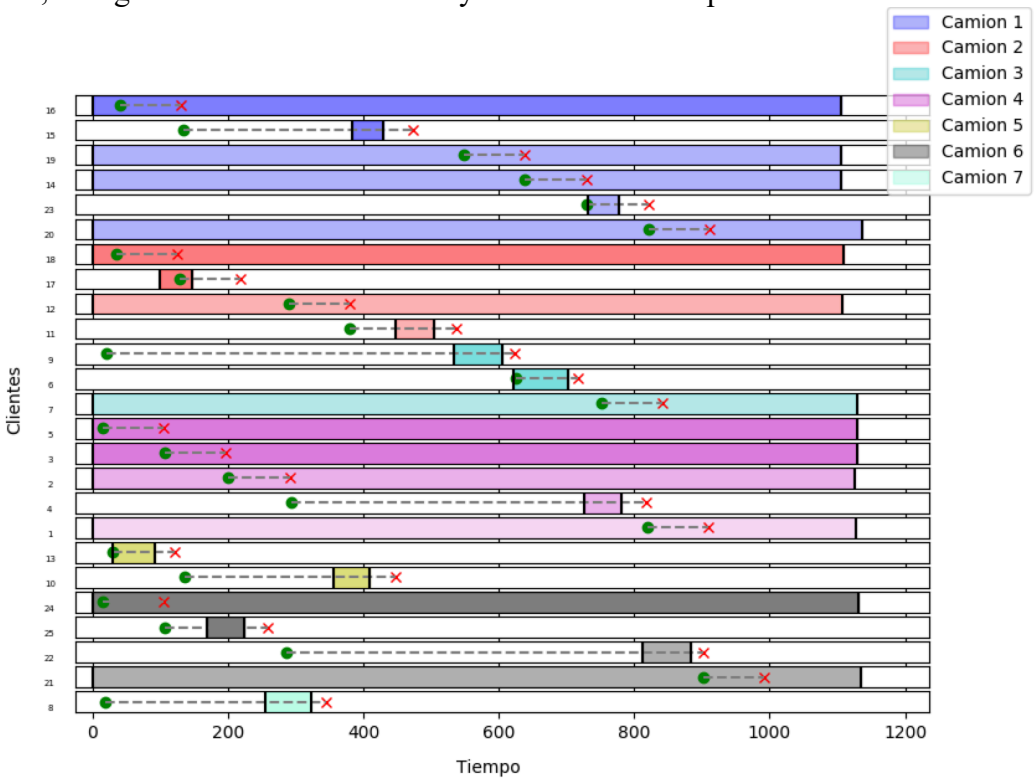


Figura 12. Representación el tiempo de llegada y salida a los clientes según las ventanas temporales para un MTVRPTW con 25 clientes. Algoritmo original.

Para la representación de los tiempos de llegada y salida en la figura 14 se ha hecho de forma análoga al problema VRPTW. En concreto, en este caso, para indicar los clientes que forman parte de un multiviaje, se ha reducido la opacidad del color de la franja temporal de disponibilidad del cliente. Así, el mismo color más oscuro representará el primer viaje y los consecutivos tendrán una tonalidad más suave.

5.2.2.2 MTVRP. Ejemplo de resultado con algoritmo mejorado.

Si se ejecuta el algoritmo con los mismos parámetros de entrada, pero esta vez usando el cálculo de ahorros mejorados se obtendría los siguientes resultados, que se mostrarán a través de las imágenes utilizadas en el resto de los ejemplos:

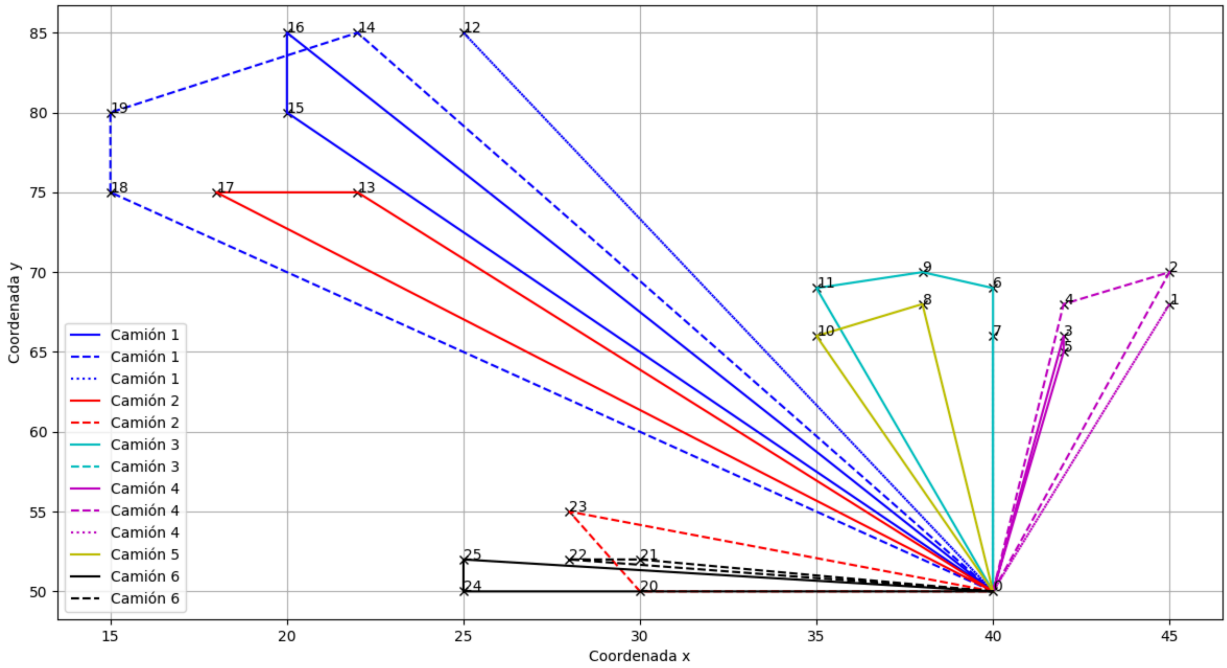


Figura 13. Representación en un mapa de puntos de la ruta obtenida de un MTVRPTW para 25 clientes. Algoritmo modificado.

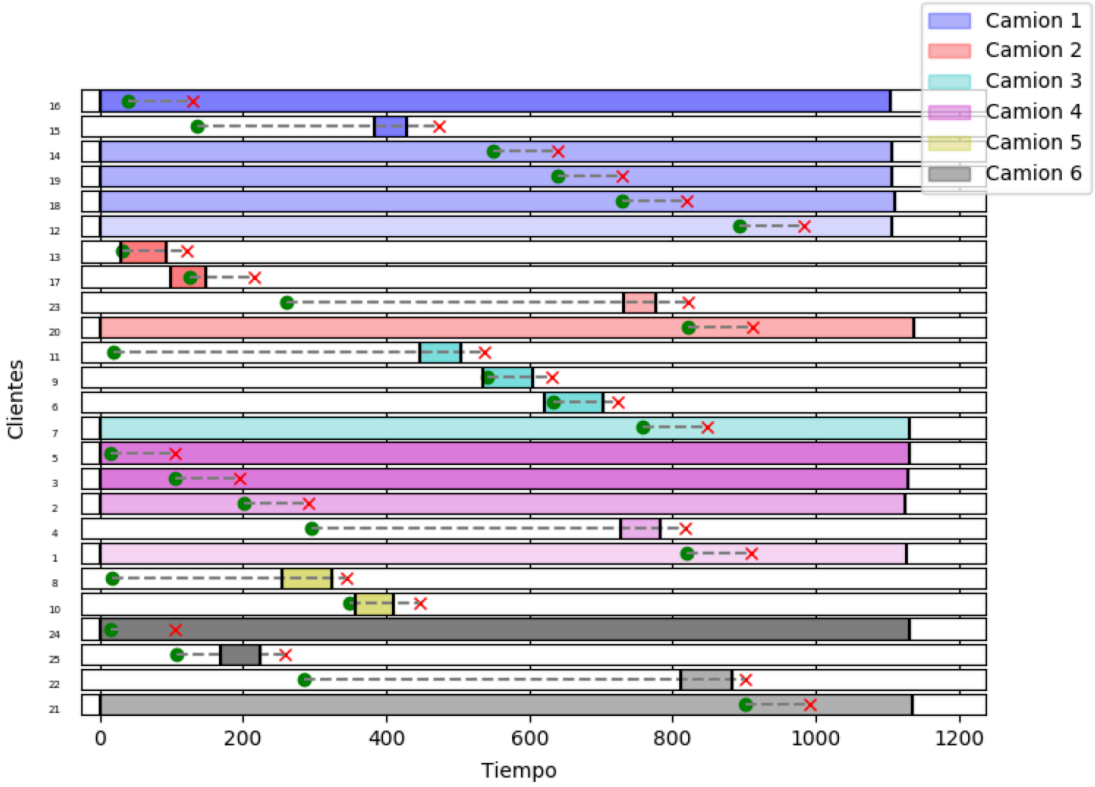


Figura 14. Representación el tiempo de llegada y salida a los clientes según las ventanas temporales para un MTVRPTW con 25 clientes. Algoritmo modificado.

Para este caso no solo no se observa que los resultados a nivel de ordenación en el mapa tienen mas sentido, sino que el número de vehículos se ha reducido. En este caso en concreto, la distancia total también se ha visto reducida debido a la mejor ordenación del reparto, pero esto no debería pasar siempre que se reduzca los vehículos de uso. Al utilizar un vehículo menos y, por tanto, más viajes múltiples, el tiempo de reparto ha de aumentar de manera prácticamente forzosa, siendo esto lo que ha ocurrido.

5.2.2.3 VRPTW. Ejemplo comparativo.

A continuación, se recreará el mismo escenario para un VRPTW con el fin de ver las diferencias más evidentes entre los dos tipos de solución, con multiviaje y sin ellos, siendo así los parámetros de entrada esta vez:

Tabla 17. Parámetros entrada. Ejemplo comparativo VRPTW.

Capacidad	40
Coste nuevo vehículo	0
Número máx. de vehículos	15
Param	900

Los resultados obtenidos tras ejecutar la heurística con esta configuración elegida se muestran en las figuras 18 y 19.

Figura 15.Valores obtenidos por pantalla tras la ejecución del VRPTW con 25 clientes.

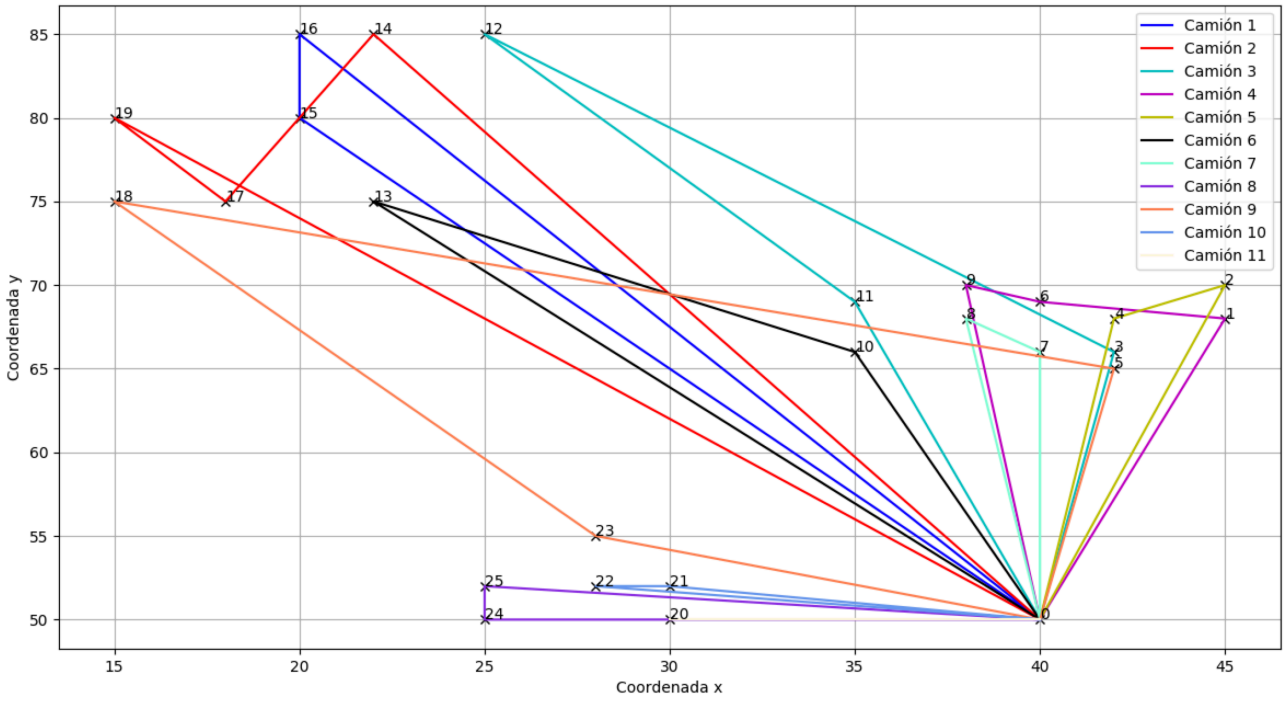


Figura 16. Representación en un mapa de puntos de la ruta obtenida de un VRPTW para 25 clientes.

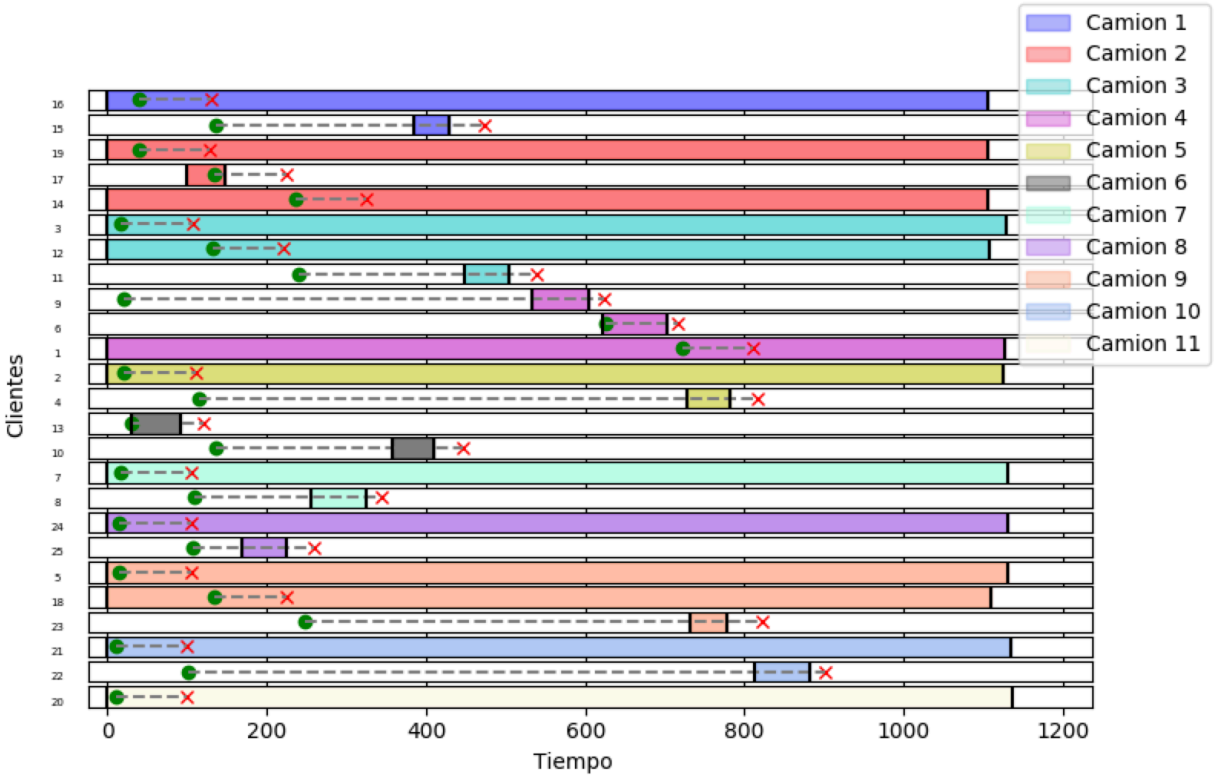


Figura 17.Representación el tiempo de llegada y salida a los clientes según las ventanas temporales para un VRPTW con 25 clientes.

5.2.2.4 Conclusiones tras los ejemplos

A través de este ejemplo donde se han utilizado los dos métodos existentes en este documento, MTVRPTW y VRPTW, se puede concluir que, generalmente, el multiviaje propiciará una reducción asegurada del número de vehículos con respecto al VRP tradicional cuando se traten de capacidades reducidas.

Debido a la estructura de estas instancias, con su distribución de coordenadas y de ventanas temporales, tanto la distancia total recorrida como el tiempo usado para el reparto completo por todos los camiones será mayor en el caso donde existan multiviajes, dado que el viaje múltiple exige la vuelta al depósito desde el último cliente al que se repartió y el viaje de ida al nuevo cliente, perdiéndose así gran parte de la distancia y tiempo ahorrados en un principio.

Por tanto, se puede concluir con los resultados del ejemplo que, por lo general, el multiviaje reducirá siempre el número de vehículos utilizados respecto al VRP estándar.

5.2.2.5 Ejecución de la instancia tipo C1 para MTVRPTW

Una vez explicado el comportamiento que toma el multi-trip VRP en su ejecución y los resultados que ofrece, se procede a ejecutar el algoritmo para la batería completa de problemas de Solomon tipo C1, como se hizo en el apartado anterior para el VRPTW, para el estándar establecido de 100 clientes.

El objetivo es recrear el escenario indicado por la instancia para VRPTW y obtener una comparación de resultados donde se espera, al menos, reducir el número de vehículos totales utilizados.

Los parámetros utilizados y los resultados obtenidos se muestran en las siguientes tablas:

Tabla 18. Parámetros de entrada para instancia tipo C1. Problema MTVRPTW.

Capacidad	200
Coste nuevo vehículo	1000
Número máx. de vehículos	50
Param	900

Tabla 19. MTVRPTW. Resultado para un total de 100 clientes. Algoritmo original.

Algoritmo Original. N=100.				
Problema	Distancia	N. Vehículos	Tiempo Reparto	Tiempo Ejecución (s)
c101	1049,70	15	13409,39	0,596
c102	1144,38	15	14302,69	0,329
c103	1200,22	15	14161,98	0,606
c104	1220,87	12	15335,61	0,267
c105	1049,70	15	13108,39	0,611
c106	1059,08	15	13227,62	0,590
c107	1056,29	14	12981,22	0,281
c108	1053,25	14	12196,06	0,329
c109	1019,14	13	11308,29	0,287

A continuación, se aplica la mejora en el algoritmo de cálculo de ahorros, obteniendo la siguiente tabla:

Tabla 20. Resultado para un total de 100 clientes. Algoritmo modificado.

Algoritmo Mejorado. N=100. Coeficiente = 0,0001							
Problema	Distancia	N. Vehículos	T. Reparto	T. Ejecución (s)	Dif. Distancia	Dif. Vehíc.	Dif. Reparto
c101	1049,70	15	13409,39	0,593	0,00	0	0,00
c102	1144,38	15	14302,69	0,470	0,00	0	0,00
c103	1137,71	14	13868,14	0,713	62,50	1	293,84
c104	1186,21	12	14484,20	0,244	34,66	0	851,41
c105	1049,70	15	13108,39	0,673	0,00	0	0,00
c106	1059,08	15	13227,62	0,698	0,00	0	0,00
c107	1056,29	14	12981,22	0,292	0,00	0	0,00
c108	1053,25	14	12196,06	0,279	0,00	0	0,00
c109	1019,14	13	11308,29	0,270	0,00	0	0,00
Mejora total					97,16	1	1145,25

Se puede comprobar como, al igual que ocurría en el ejemplo anterior, el algoritmo de mejora funciona correctamente aún incluyendo la posibilidad de multitrip. En este caso, no solo se mejora o mantiene la distancia recorrida, sino también los vehículos utilizados y el tiempo total de reparto.

En el siguiente apartado se comparará los resultados obtenidos a nivel global para la instancia tipo C1 del VRPTW y del MTVRTW.

5.2.3 Comparación de resultados y conclusiones

Una vez obtenidos los resultados para la instancia de problemas tipo C1 tanto para el VRPTW como para el MTVRPTW, con el algoritmo original y el mejorado, se puede afirmar con total seguridad como ya se anticipaba con el ejemplo que, para todos los casos, el MTVRPTW disminuye o mantiene el número de vehículos utilizados previamente por el VRPTW, teniendo en cuenta la correspondencia entre algoritmo original y algoritmo mejorado.

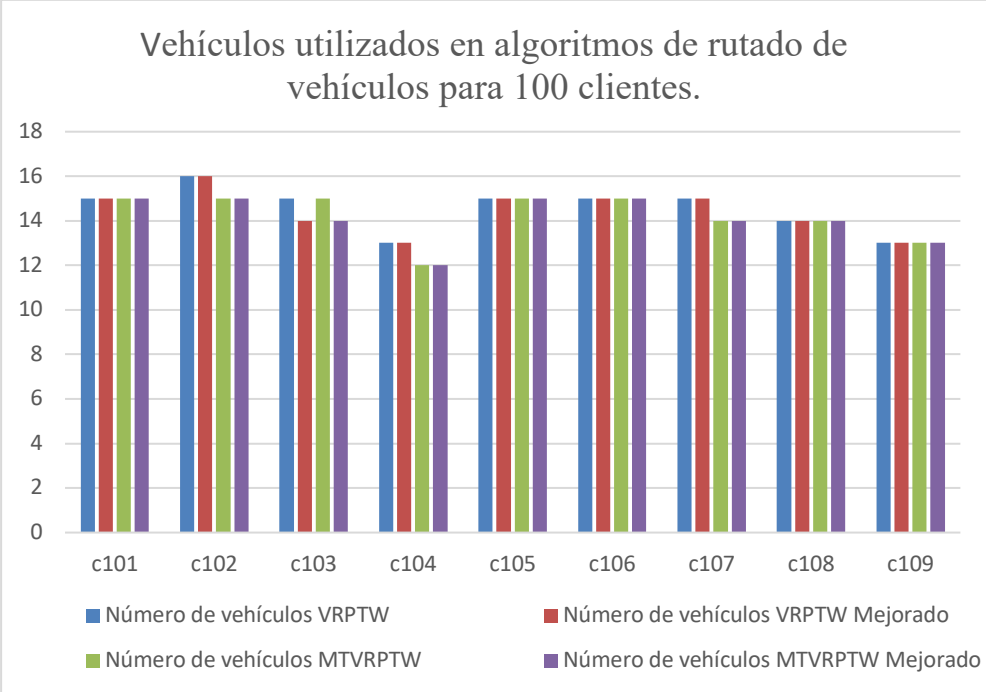


Figura 18. Comparación entre los vehículos utilizados de diferentes algoritmos.

Puede observarse con total claridad que la mejor opción en términos de ahorro de número de vehículos para todos los casos será el MTVRPTW con la mejora del cálculo de ahorros. Si bien no se aprecia una disminución del número de vehículos utilizados tan acusada como podría producirse en el ejemplo de 25 clientes, cabe destacar que la capacidad utilizada ha sido la indicada en la misma instancia de problemas, destacando que esta instancia no está diseñada exclusivamente para encontrar soluciones óptimas con multiviajes. Teniendo, además, que la disposición de las ventanas temporales del total de clientes no permite en la mayoría de los casos el multiviaje como una opción factible debido a lo ajustadas que están muchas franjas temporales. No obstante, la mejora en términos de la cantidad de vehículos utilizados sea notable o no, está en todos los casos presente.

Además, para el tipo de problema propuesto al inicio del documento donde se planteaba la posibilidad de usar vehículos con tamaños reducidos y características relacionadas con las emisiones y consumo muy concretas, la compra de un solo vehículo más podría suponer un gasto mayor que la posible eficiencia que pudiese tener en términos de distancia o tiempo total recorrido para el proveedor de transporte.

5.3 Justificación de la elección de esta heurística

Una vez expuestos cada uno de los resultados que ofrece la heurística, cabe concluir que es evidente que tanto para el caso del VRPTW como para el MTVRPTW existe un amplio margen de mejora. Si bien esto podría llevarse a cabo por medio de metaheurísticas más complejas que

permitiesen, mediante permutaciones, lograr una opción factible con mejores resultados, el coste computacional que esto supondría sería altísimo. Es decir, si se observan los resultados del tiempo de ejecución total de la heurística en cualquiera de sus variantes, se puede apreciar que ninguno de estos casos supera el segundo de ejecución, reduciéndose en muchos casos a menos de una décima de segundo.

Por ello, si bien la solución que se obtiene en el problema podría ser mejorada como ya se preveía, el coste computacional es tan bajo que, siendo una solución perfectamente factible y operativa como lo es, la diferencia a nivel de software entre obtener un resultado en menos de un segundo y obtener un resultado en un alto número de minutos o incluso horas, convierte la solución ofrecida en este documento como una solución atractiva si se pretende obtener una solución rápida, efectiva y factible a un problema tan complejo como es el Multi-trip VRP con ventanas temporales.

6 CONCLUSIONES

Una vez finalizada la muestra de resultados y volviendo al problema original donde se planteaba la situación del multiviaje en la logística urbana, se puede concluir que la heurística desarrollada da una solución factible al problema planteando llegando a mejorarla en algunos casos con pequeñas modificaciones en el algoritmo.

Si bien los resultados para las instancias elegidas no son tan disruptivos como cabría esperar, toca destacar lo señalado anteriormente: la instancia no está diseñada para un algoritmo de rutado de vehículos con multiviaje, por tanto, la pequeña franja temporal que ofrecen determinados clientes hace que el MTVRPTW no explote su máximo potencial para determinados conjuntos de datos. Sin embargo, se ha podido evidenciar que para capacidades muy bajas y un número de clientes reducido como se expuso en el ejemplo del punto 5, el comportamiento términos de rendimiento en cuanto a la reducción de vehículos totales es más que notable. Por ello, para situaciones reales de logística urbana con esta tipología, si un proveedor de transporte quisiera utilizar este algoritmo, obtendría resultados más que satisfactorios con relación a los que podría obtener con un VRPTW estándar.

En concreto, se puede retomar la idea expuesta acerca de los viajes de última milla destinados al centro de una ciudad con una serie de restricciones a nivel de tráfico que solo permite vehículos de muy bajo consumo o vehículos eléctricos. Si fuesen entregas a pequeños comercios, situados en zonas de acceso limitado, por parte de sus proveedores extranjeros solicitados por comercio electrónico y suponiendo que estos paquetes tendrían un volumen no despreciable, el funcionamiento de la heurística podría proporcionar notables mejoras en términos de coste a la hora de la adquisición de este tipo de vehículos con características especiales por parte de la compañía de transporte. Se podría simular la demanda futura en función de los datos de pedidos previos y así estimar el número máximo de vehículos eléctricos o de bajo consumo que podría necesitar para sus repartos efectivos con un algoritmo MTVRPTW. En caso de usar un VRPTW sin multiviaje, claramente el cliente necesitaría de un exceso de vehículos en comparación con el caso anterior.

6.1 Futuras vías de mejora

Si bien la solución obtenida es factible, según los resultados encontrados como mejores hasta la fecha por diferentes investigadores para la instancia elegida, se deduce que el margen de mejora de las soluciones es amplio.

Así, en futuras investigaciones se propone utilizar este proyecto como solución inicial o base de metaheurísticas mucho más complejas como podrían ser la búsqueda tabú o el algoritmo genético ya planteadas en el marco teórico de este documento. Dado que la solución obtenida es factible y plantea resultados más que razonables, el punto de partida al que se ha llegado en este documento proporcionaría a la metaheurística desarrollada una ventaja en términos de computación, pues al partir de una buena solución, las iteraciones para llegar a un posible óptimo serían más reducidas que si utilizara una solución aleatoria como solución inicial.

Además, también se propone continuar con el estudio del reparto multi-trip tratando de simular un entorno real a través de instancias que podrían ser generadas de forma “aleatoria” conforme a unos parámetros de entrada controlados. Bajo esta tesitura, se podría afinar el funcionamiento de la metaheurística a desarrollar tratando de minimizar el tiempo de ejecución de esta hasta convertirla en una herramienta que diese una solución óptima o cerca de esta en un periodo de tiempo lo más reducido posible.

REFERENCIAS

[1]. TANIGUCHI, E., THOMPSON R.G., y YAMADA, T. (1999). «Modelling city logistics». In: City Logistics I (E. Taniguchi and R.G. Thompson, eds.), Institute of Systems Science Research, Kyoto, pp. 3-37.

[2]. UN DESA, (2018). 68% of the world population projected to live in urban areas by 2050, says UN. (16 de mayo, 2018) United Nations. Department of Economic and Social Affairs. News.

[3]. CALVIÑO, A. (2011). Cooperación en los problemas del viajante (TSP) y de rutas de vehículos (VRP): una panorámica. Trabado de fin de Máster. Máster Interuniversitario en Técnicas Estadísticas. Universidad de Santiago de Compostela.

[4]. COOK, W.J. (2012) In Pursuit of the Traveling Salesman: Mathematics at the Limit of Computation. Ed: Princeton UP.

[5]. DANTZIG, G.B. y RAMSER, J.H. (1959) «The truck dispatching problem». Management Science, 6, pp. 80–91.

[6]. CLARKE, G. y WRIGHT, J.R. (1964) «Scheduling of Vehicle Routing Problem from a Central Depot to a Number of Delivery Points». Operations Research, 12, pp. 568-581.

[7]. LÜER-VILLAGRA, A., BENAVENTE, M., BUSTOS, J. y VENEGAS, B. (2009) El problema de rutas de vehículos: Extensiones y métodos de resolución, estado del arte.

[8]. PRINS, C. (2002). «Efficient heuristic for the heterogeneous fleet multitrip vrp with application to a large-scale real case». Journal of Mathematical Modelling and Algorithms, 1, pp. 135-150.

[9]. CANCA, J.D. y GONZÁLEZ, P.L. (2016). Técnicas de Optimización. Ed: Iris-copy S.L.

[10]. GLOVER, F. (1986) «Future paths for integer programming and links to artificial intelligence». Computers & Operations Research, 13, pp. 533-549.

[11]. http://web.cba.neu.edu/~msolomon/problems.htm

7 ANEXO. DESARROLLO DEL CÓDIGO

7.1 Generador de problemas

```
#Se importa numpy para obtener la función distancia
from numpy import *

#Se define la función generador que se importará en el archivo Clarke_wright
def generador(archivo):

    #Apertura el archivo con los datos necesarios del problema: Nº de
clientes, coordenadas x e y, demanda, ventana temporal
    # y tiempo de servicio
    handle = open(archivo)

    #Definición de las variables diccionario para el almacenamiento de datos
    custno = dict()
    x_dict = dict()
    y_dict = dict()
    demand_dict = dict()
    readytime_dict = dict()
    duedate_dict = dict()
    servicetime_dict = dict()

    #Definición de las variables auxiliares
    listaaux= list()
    cont = 0
    key = 0
    #Nº de clientes a usar para las diferentes pruebas de problema
    n = 25 + 1

    #Asignación de las claves según el número de clientes a cada variable
diccionario
    for i in range(n):
        x_dict[i] = None
        y_dict[i] = None
        demand_dict[i] = None
        readytime_dict[i] = None
        duedate_dict[i] = None
        servicetime_dict[i] = None

    #Lectura del documento y asígnación de valores
    for linea in handle:
        cont = cont + 1
        #Al pasar de la línea 9, comienzan los valores en el documento
        if cont > 9 and cont <= (n+9):
            #Se divide la línea en una lista con cadenas de caracteres en su
interior
            listaaux = linea.split()
            #Se asigna a cada variable diccionario su posición
correspondiente de la lista, obviando la posición 0, que
            #corresponde a los clientes
            x_dict[key]=(int(listaaux[1]))
            y_dict[key] = (int(listaaux[2]))
            demand_dict[key] = (int(listaaux[3]))
            readytime_dict[key] = (int(listaaux[4]))
            duedate_dict[key] = (int(listaaux[5]))
```

```
            servicetime_dict[key] = (int(listaaux[6]))
            #Actualización de la key para recorrerlas todas del 0 al n
            key = key + 1

    return x_dict, y_dict, demand_dict, readytime_dict, duedate_dict,
servicetime_dict, n
```

7.2 Heurística y presentación de resultados

```
from Generador_1_1 import *
from scipy.spatial import distance
from numpy import *
import copy
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
from openpyxl import Workbook
import matplotlib.patches as mpatches
import time

# ---------------------------------
# ALGORITMO CLARKE Y WRIGHT ADAPTADO
# ---------------------------------

def algoritmo_entrega(archivo):

    # DEFINICION DE FUNCIONES #

    # Función que realiza el cálculo de distancias entre clientes
    def calculo_distancias(n, x_dict, y_dict):
        # Se recorre la matriz rellenándola con la distancia entre cada uno
de los clientes, obteniendo una matriz simétrica
        for i in range(n):
            for j in range(n):
                distancia[i, j] = distance.euclidean((x_dict[i], y_dict[i]),
(x_dict[j], y_dict[j]))

        return distancia

    # Función que realiza el cálculo de ahorros según Clarke & Wright
    def calculo_ahorros(n, distancia):
        contador = 0
        # Se rellena la matriz de ahorros simétrica diagonal superior con la
fórmula de ahorros
        for i in range(n):
            for j in range(n):
                if i != j and i > 0 and j > contador:
                    s[i, j] = distancia[i, 0] + distancia[j, 0] -
distancia[i, j]
            contador = contador + 1

        # Se elimina del cálculo la fila y columna del depósito fijándolo a
un valor negativo alto
        for k in range(n):
            s[0][k] = -10000
            s[k][0] = -10000
```

```
        return s

    # Función para una versión alternativa del cálculo de la matriz de
ahorros
    def calculo_ahorros_ponderada(coef1, n, distancia):

        contador = 0
        # Se rellena la matriz de ahorros simétrica diagonal superior con la
fórmula de ahorros modificada
        for i in range(n):
            for j in range(n):
                if i != j and i > 0 and j > contador:
                    # variable = distancia[i, 0] + distancia[0, j] -
distancia[i, j] + coef1 * abs(((duedate_dict[i] - readytime_dict[i]) -
(duedate_dict[j] - readytime_dict[j])))
                    variable = distancia[i, 0] + distancia[0, j] -
distancia[i, j] + coef1 * (
                                    (duedate_dict[i] - duedate_dict[j]) -
(readytime_dict[i] - readytime_dict[j]))
                    if variable >= 0:
                        s[i, j] = variable
                    else:
                        s[i, j] = 0
            contador = contador + 1

        # Se elimina del cálculo la fila y columna del depósito fijándolo a
un valor negativo alto
        for k in range(n):
            s[0][k] = -10000
            s[k][0] = -10000

        return s


    # Función que permite comprobar si los indices encontrados están al final
o al principio de una ruta ya establecida
    def comprob_inicio_fin(indice, ruta):

        # Se inicializan variables de comprobación y posición
        ii = 0
        jj = 0
        kk = 0
        nueva_ruta = False
        inicio = False
        fin = False

        # Se recorre el vector ruta en busca del índice al principio o al
final de alguna de las rutas establecidas
        for i in range(len(ruta)):
            # Si aún no se ha realizado ningún multiviaje
            if len(ruta[i]) == 1:
                for j in range(len(ruta[i])):
                    #indice en la última posición
                    if ruta[i][j][-1] == indice:
                        fin = True
                        # Se establece la posición de la ruta donde se
encuentra el índice
                        ii = i
                        jj = j
                        kk = len(ruta[i][j]) - 1
                    #indice en la primera posicion
```

```
                    elif ruta[i][j][0] == indice:
                        inicio = True
                        # Se establece la posición de la ruta donde se
encuentra el índice
                        ii = i
                        jj = j
                        kk = 0

            # Para el multitrip, en caso de que los índices encontrados estén
presentes en diferentes desplazamientos, se toma como prioritario
            # el último desplazamiento realizado por el mismo camión
            else:
                for j in range(len(ruta[i])):
                    if ruta[i][0][0] == indice:
                        inicio = True
                        # Se anula la variable fin en caso de que fuera
positiva en otros desplazamientos
                        fin = False
                        ii = i
                        jj = 0
                        kk = 0

                    elif ruta[i][-1][-1] == indice:
                        fin = True
                        # Se anula la variable inicio en caso de que fuera
positiva en otros desplazamientos
                        inicio = False
                        ii = i
                        jj = len(ruta[i]) - 1
                        kk = len(ruta[i][j]) - 1

        if not inicio and not fin:
            # Se usa comprob como variable booleana indicadora de que el
índice no está en ninguna ruta
            nueva_ruta = True

        return inicio, fin, ii, jj, kk, nueva_ruta


    # Funcion que indica, en función de las ventanas temporales, el orden que
han de tomar los índices obtenidos en el recorrido
    def orden_indices(hora_max, indice0, indice1, distancia, readytime_dict,
duedate_dict):

        # Definición del tipo de variable
        indice_aux0 = int()
        indice_aux1 = int()
        # Variable que indicará viabilidad del proceso
        orden = False

        # OPCION POR DEFECTO
        # Se comprueba que la opción según el orden natural de los índices es
factible en términos de ventanas temporales
        tiempo_aux00 = distancia[0, indice0]

        # En caso de que tenga que esperar al llegar al primer cliente para
servirle, el tiempo de llegada al segundo varía de la siguiente forma:
        if tiempo_aux00 < readytime_dict[indice0]:
            tiempo_aux01 = tiempo_aux00 + servicetime_dict[indice0] +
(readytime_dict[indice0] - distancia[0, indice0]) + distancia[
                indice0, indice1]
```

```
        # En caso de que no esperase:
        else:
            tiempo_aux01 = tiempo_aux00 + servicetime_dict[indice0] +
distancia[indice0, indice1]

        # Se calcula el hipotético tiempo de llegada al depósito
        if tiempo_aux01 < readytime_dict[indice1]:
            tiempo_deposito = tiempo_aux01 + servicetime_dict[indice1] +
(readytime_dict[indice1] - distancia[indice0, indice1]) + \
                              distancia[indice1, 0]
        else:
            tiempo_deposito = tiempo_aux01 + servicetime_dict[indice1] +
distancia[indice1, 0]

        # Se establece las condiciones que se necesita para cambiar el orden
        cond01 = readytime_dict[indice0] < readytime_dict[indice1]
        cond02 = duedate_dict[indice0] >= tiempo_aux00
        cond03 = duedate_dict[indice1] >= tiempo_aux01
        cond04 = hora_max > tiempo_deposito

        if cond01 and cond02 and cond03 and cond04:
            indice_aux0 = indice0
            indice_aux1 = indice1
            orden = True

        # OPCION CONTRARIA
        # Sse comprueba que los clientes en el orden contrario son factibles
        tiempo_aux11 = distancia[0, indice1]

        if tiempo_aux11 < readytime_dict[indice1]:
            tiempo_aux10 = tiempo_aux11 + servicetime_dict[indice1] +
(readytime_dict[indice1] - distancia[0, indice1]) + distancia[indice1,
indice0]

        else:
            tiempo_aux10 = tiempo_aux11 + servicetime_dict[indice1] +
distancia[indice1, indice0]

        if tiempo_aux10 < readytime_dict[indice0]:
            tiempo_deposito = tiempo_aux10 + servicetime_dict[indice0] +
(readytime_dict[indice0] - distancia[indice1, indice0]) + \
                              distancia[indice0, 0]
        else:
            tiempo_deposito = tiempo_aux10 + servicetime_dict[indice0] +
distancia[indice0, 0]

        cond11 = readytime_dict[indice1] < readytime_dict[indice0]
        cond12 = duedate_dict[indice1] >= tiempo_aux11
        cond13 = duedate_dict[indice0] >= tiempo_aux10
        cond14 = hora_max > tiempo_deposito

        if cond11 and cond12 and cond13 and cond14:
            indice_aux0 = indice1
            indice_aux1 = indice0
            orden = True

        return indice_aux0, indice_aux1, orden


    # Funcion para comprobar la limitación de capacidad establecida
    def limite_capacidad(capacidad_max, capacidad, i0, i1, j0, j1, indice0,
```

```
indice1, inicio0, inicio1, fin0, fin1):

        comprob_capacidad = None

        # Opcion para el indice0
        if inicio0 or fin0:
            comprob_capacidad = (capacidad_max >= capacidad[i0][j0] +
demand_dict[indice1])
        # Opcion para el indice1
        elif inicio1 or fin1:
            comprob_capacidad = (capacidad_max >= capacidad[i1][j1] +
demand_dict[indice0])

        return comprob_capacidad


    # Funcion utilizada para la comprobación de la posible nueva ruta según
las limitaciones temporales
    def ventanas_temporales(ruta_aux, hora_max, ii, jj, kk, distancia,
readytime_dict, servicetime_dict, inicio, fin, indice, tiempo_llegada_aux,
tiempo_salida_aux):
        cont = 0
        comprob_llegada = False
        comprob_salida = False

        # Opción para una ruta con un solo viaje
        if len(ruta_aux[ii]) == 1:
            # Índice al principio de una ruta establecida
            if inicio:
                # Se actualiza vector ruta aux
                ruta_aux[ii][jj].insert(kk, indice)
                # Condición de que sobrepase o no el tiempo de llegada en el
viaje desde el depósito
                if readytime_dict[indice] <= distancia[0, indice]:
                    variable_aux = distancia[0, indice] +
servicetime_dict[indice]
                else:
                    variable_aux = readytime_dict[indice] +
servicetime_dict[indice]

                # Se añaden las variables del nuevo índice
                tiempo_llegada_aux[ii][jj].insert(kk, distancia[0, indice])
                tiempo_salida_aux[ii][jj].insert(kk, variable_aux)

                # Actualizacion del vector tiempo de llegada y vector tiempo
de salida
                for l in range(1, len(tiempo_llegada_aux[ii][jj])):
                    # Definición de las nuevos valores del tiempo de llegada
                    tiempo_llegada_aux[ii][jj][l] =
tiempo_salida_aux[ii][jj][l - 1] + distancia[
                        ruta_aux[ii][jj][l - 1], ruta_aux[ii][jj][l]]

                    # El vector de tiempos de salida estará condicionado al
valor de llegada al cliente de la siguiente forma:
                    if tiempo_llegada_aux[ii][jj][l] >=
readytime_dict[ruta_aux[ii][jj][l]]:
                        tiempo_salida_aux[ii][jj][l] =
tiempo_llegada_aux[ii][jj][l] + servicetime_dict[ruta_aux[ii][jj][l]]
                    else:
                        tiempo_salida_aux[ii][jj][l] =
readytime_dict[ruta_aux[ii][jj][l]] + servicetime_dict[ruta_aux[ii][jj][l]]
```

```
            # Índice al final de una ruta establecida
            if fin:
                # Se establece el valor del tiempo de llegada en el vector
del nuevo cliente
                tiempo_llegada_aux[ii][jj].append(tiempo_salida_aux[ii][jj][-
1] + distancia[ruta_aux[ii][jj][-1], indice])

                # Se establece el valor del tiempo de salida en el vector del
nuevo cliente
                if readytime_dict[indice] <= tiempo_llegada_aux[ii][jj][-1]:

tiempo_salida_aux[ii][jj].append(tiempo_llegada_aux[ii][jj][-1] +
servicetime_dict[indice])
                else:
                    tiempo_salida_aux[ii][jj].append(readytime_dict[indice] +
servicetime_dict[indice])

                # Se actualiza el vector ruta auxiliar
                ruta_aux[ii][jj].append(indice)

        # Opcion multitrip. Solo se actualizarán los tiempos a partir de la
posición jj.
        else:
            if inicio:
                # Se actualiza vector ruta
                ruta_aux[ii][jj].insert(kk, indice)

                if jj == 0:
                    if readytime_dict[indice] <= distancia[0, indice]:
                        variable_aux = distancia[0, indice] +
servicetime_dict[indice]
                    else:
                        variable_aux = readytime_dict[indice] +
servicetime_dict[indice]

                else:
                    # Condición de que sobrepase o no el tiempo de llegada en
el viaje desde el depósito
                    if readytime_dict[indice] <= tiempo_salida_aux[ii][jj -
1][-1] + distancia[ruta_aux[ii][jj - 1][-1], indice]:
                        variable_aux = tiempo_salida_aux[ii][jj - 1][-1] +
distancia[ruta_aux[ii][jj - 1][-1], indice] + servicetime_dict[
                            indice]
                    else:
                        variable_aux = readytime_dict[indice] +
servicetime_dict[indice]

                # Se añade las variables del nuevo índice
                tiempo_llegada_aux[ii][jj].insert(kk,
tiempo_salida_aux[ii][jj - 1][-1] + distancia[ruta_aux[ii][jj - 1][-1],
indice])
                tiempo_salida_aux[ii][jj].insert(kk, variable_aux)

                # Actualizacion del vector tiempo de llegada y vector tiempo
de salida
                for h in range(jj, len(tiempo_llegada_aux[ii])):
                    for l in range(len(tiempo_llegada_aux[ii][h])):
                        # Condición para las primeros clientes de cada
trayecto
                        if l == 0:
```

```
                            # La primera componente ya se ha incluído
                            if h == jj:
                                continue
                            # La primera componente de las siguientes rutas
han de ser igual al tiempo de salida anterior más la distancia hasta el
cliente
                            else:
                                tiempo_llegada_aux[ii][h][l] =
tiempo_salida_aux[ii][h - 1][-1] + distancia[
                                        ruta_aux[ii][h - 1][-1],
ruta_aux[ii][h][l]]
                        else:
                            tiempo_llegada_aux[ii][h][l] =
tiempo_salida_aux[ii][h][l - 1] + distancia[
                                ruta_aux[ii][h][l - 1], ruta_aux[ii][h][l]]

                        # El vector de tiempos de salida estará condicionado
al valor de llegada al cliente de la siguiente forma:
                        if tiempo_llegada_aux[ii][h][l] >=
readytime_dict[ruta_aux[ii][h][l]]:
                            tiempo_salida_aux[ii][h][l] =
tiempo_llegada_aux[ii][h][l] + servicetime_dict[ruta_aux[ii][h][l]]
                        else:
                            tiempo_salida_aux[ii][h][l] =
readytime_dict[ruta_aux[ii][h][l]] + servicetime_dict[ruta_aux[ii][h][l]]

            if fin:
                # Se actualiza vector ruta
                ruta_aux[ii][jj].append(indice)

                # Se actualiza vectores de tiempo de llegada y salida
                tiempo_llegada_aux[ii][jj].append(tiempo_salida_aux[ii][jj][-
1] + distancia[ruta_aux[ii][jj][-1], indice])

                if readytime_dict[indice] <= tiempo_llegada_aux[ii][jj][-1]:

tiempo_salida_aux[ii][jj].append(tiempo_llegada_aux[ii][jj][-1] +
servicetime_dict[indice])
                else:
                    tiempo_salida_aux[ii][jj].append(readytime_dict[indice] +
servicetime_dict[indice])

        #Bucle de comprobación ded que las variables axuiliares de tiempos
cumplen las restricciones establecidas de límite dde llegada
        for m in range(len(tiempo_llegada_aux[ii])):
            for r in range(len(tiempo_llegada_aux[ii][m])):
                if duedate_dict[ruta_aux[ii][m][r]] <
tiempo_llegada_aux[ii][m][r]:
                    cont = cont + 1

        comprob_llegada = cont == 0
        comprob_salida = (tiempo_salida_aux[ii][jj][-1] +
distancia[ruta_aux[ii][jj][-1], 0]) < hora_max

        return comprob_llegada, comprob_salida


    # Función que comprueba la viabilidad de establecer el nuevo viaje
multitrip antes de la ruta ya establecida
    def comprob_multitrip_inicio(ii, jj, indice, readytime_dict,
servicetime_dict, duedate_dict, distancia, tiempo_llegada_aux,
```

```
tiempo_salida_aux, ruta_aux, hora_max, coste_nuevo_vehiculo):
        # Variables de comprobación
        cont = 0
        comprob_salida = None
        comprob_llegada = None

        # Se actualiza el vector ruta
        ruta_aux[ii].insert(jj, [indice])

        # Si el índice encontrado está localizado en el primer desplazamiento
del vehículo
        if jj == 0:
            if readytime_dict[indice] <= distancia[0, indice]:
                variable_aux = distancia[0, indice] +
servicetime_dict[indice]
            else:
                variable_aux = readytime_dict[indice] +
servicetime_dict[indice]
            tiempo_llegada_aux[ii].insert(jj, [distancia[0, indice]])

            coste_tiempo = tiempo_llegada_aux[ii][jj + 1][0]
        #Si el indice encontrado está localizado en desplazamaientos
posteriores
        else:
            # Condicional que indica si sobrepasa o no el tiempo de llegada
en el viaje desde el depósito
            if readytime_dict[indice] <= tiempo_salida_aux[ii][jj - 1][-1] +
distancia[ruta_aux[ii][jj - 1][-1], 0] + distancia[0, indice]:
                variable_aux = tiempo_salida_aux[ii][jj - 1][-1] +
distancia[ruta_aux[ii][jj - 1][-1], 0] + distancia[0, indice] +
servicetime_dict[indice]
            else:
                variable_aux = readytime_dict[indice] +
servicetime_dict[indice]
            tiempo_llegada_aux[ii].insert(jj, [tiempo_salida_aux[ii][jj -
1][-1] + distancia[ruta_aux[ii][jj - 1][-1], 0] + distancia[0, indice]])

            coste_tiempo = distancia[ruta_aux[ii][jj - 1][-1], 0] +
distancia[0, indice]

        tiempo_salida_aux[ii].insert(jj, [variable_aux])

        # Actualizacion del vector tiempo de llegada y vector tiempo de
salida
        for h in range(jj + 1, len(tiempo_llegada_aux[ii])):
            for l in range(len(tiempo_llegada_aux[ii][h])):
                # Condición para las primeros clientes de cada trayecto
                if l == 0:
                    # La primera componente de las siguientes rutas han de
ser igual al tiempo de salida anterior más la distancia hasta el cliente
                    tiempo_llegada_aux[ii][h][l] = tiempo_salida_aux[ii][h -
1][-1] + distancia[ruta_aux[ii][h - 1][-1], 0] + distancia[0,
ruta_aux[ii][h][l]]
                else:
                    tiempo_llegada_aux[ii][h][l] = tiempo_salida_aux[ii][h][l
- 1] + distancia[ruta_aux[ii][h][l - 1], ruta_aux[ii][h][l]]
                # El vector de tiempos de salida estará condicionado al valor
de llegada al cliente de la siguiente forma:
                if tiempo_llegada_aux[ii][h][l] >=
readytime_dict[ruta_aux[ii][h][l]]:
                    tiempo_salida_aux[ii][h][l] =
```

```
tiempo_llegada_aux[ii][h][l] + servicetime_dict[ruta_aux[ii][h][l]]
                else:
                    tiempo_salida_aux[ii][h][l] =
readytime_dict[ruta_aux[ii][h][l]] + servicetime_dict[ruta_aux[ii][h][l]]

        #Bucle de comprobación de la viabilidad temporal de la posible nueva
ruta
        for m in range(len(ruta_aux[ii])):
            for r in range(len(ruta_aux[ii][m])):
                if duedate_dict[ruta_aux[ii][m][r]] <
tiempo_llegada_aux[ii][m][r]:
                    cont = cont + 1

        #No se sobrepasa la hora máxima
        comprob_salida = (tiempo_salida_aux[ii][jj][-1] +
distancia[ruta[ii][jj][-1], 0]) < hora_max
        #Se cumplen las ventanas temporales
        comprob_llegada = cont == 0
        #El coste del multitrip es menor que el del nuevo vehículo
        comprob_coste = coste_tiempo < coste_nuevo_vehiculo

        return comprob_llegada, comprob_salida, comprob_coste


    # Función para definir una nueva ruta cuando ninguno de los índices se
encuentra en rutas ya establecidas
    def ruta_nueva_doble(clientes_ptes, ruta, indice0, indice1, distancia,
demand_dict, readytime_dict, servicetime_dict, tiempo_llegada,
tiempo_salida):

        # Se actualiza vector ruta
        ruta.append([[indice0, indice1]])
        # Se actualiza vector capacidad
        capacidad.append([demand_dict[indice0] + demand_dict[indice1]])
        # Condicion de que el vehículo llega al primer cliente sobrepasada su
horario de apertura
        if distancia[0, indice0] >= readytime_dict[indice0]:

            # Se definen variables auxiliares para calcular los tiempos de
llegada
            aux1 = distancia[0, indice0]
            aux2 = distancia[0, indice0] + servicetime_dict[indice0] +
distancia[indice0, indice1]

            # Se actualiza vector tiempo de llegada
            tiempo_llegada.append([[aux1, aux2]])
            # Condicion si se llega al cliente indice1 despues de apertura
            if aux2 >= readytime_dict[indice1]:
                tiempo_salida.append([[aux1 + servicetime_dict[indice0], aux2
+ servicetime_dict[indice1]]])
            else:
                tiempo_salida.append([[aux1 + servicetime_dict[indice0],
readytime_dict[indice1] + servicetime_dict[indice1]]])

        # Si llega sin sobrepasar la hora de entrega
        else:
            aux1 = distancia[0, indice0]
            aux2 = readytime_dict[indice0] + servicetime_dict[indice0] +
distancia[indice0, indice1]

            tiempo_llegada.append([[aux1, aux2]])
```

```
            # Condicion si se llega al cliente indice1 despues de apertura
            if aux2 >= readytime_dict[indice1]:
                tiempo_salida.append([[readytime_dict[indice0] +
servicetime_dict[indice0], aux2 + servicetime_dict[indice1]]])
            else:
                tiempo_salida.append(
                    [[readytime_dict[indice0] + servicetime_dict[indice0],
readytime_dict[indice1] + servicetime_dict[indice1]]])

        #Borrar clientes de la lista de pendientes
        if indice0 in clientes_ptes:
            clientes_ptes.remove(indice0)

        if indice1 in clientes_ptes:
            clientes_ptes.remove(indice1)

        return ruta, capacidad, tiempo_llegada, tiempo_salida


    # Función que establece un nuevo viaje con un mismo vehículo tras
realizar la descarga en el depósito
    def ruta_multitrip(inicio, fin, ruta, ii, jj, capacidad, distancia,
demand_dict, readytime_dict, servicetime_dict, indice, indiceaux,
                       tiempo_salida, tiempo_llegada):

        # Ruta multitrip al inicio de la ruta ya establecida
        if inicio:
            # Se actualiza vector ruta
            ruta[ii].insert(jj, [indice])

            if jj == 0:
                if readytime_dict[indice] <= distancia[0, indice]:
                    variable_aux = distancia[0, indice] +
servicetime_dict[indice]
                else:
                    variable_aux = readytime_dict[indice] +
servicetime_dict[indice]
                tiempo_llegada[ii].insert(jj, [distancia[0, indice]])
            else:
                # Condición de que sobrepase o no el tiempo de llegada en el
viaje desde el depósito
                if readytime_dict[indice] <= tiempo_salida[ii][jj - 1][-1] +
distancia[ruta[ii][jj - 1][-1], 0] + distancia[0, indice]:
                    variable_aux = tiempo_salida[ii][jj - 1][-1] +
distancia[ruta[ii][jj - 1][-1], 0] + distancia[0, indice] + \
                                   servicetime_dict[indice]
                else:
                    variable_aux = readytime_dict[indice] +
servicetime_dict[indice]
                tiempo_llegada[ii].insert(jj, [tiempo_salida[ii][jj - 1][-1]
+ distancia[ruta[ii][jj - 1][-1], 0] + distancia[0, indice]])

            tiempo_salida[ii].insert(jj, [variable_aux])

            # Actualizacion del vector tiempo de llegada y vector tiempo de
salida
            for h in range(jj + 1, len(tiempo_llegada[ii])):
                for l in range(len(tiempo_llegada[ii][h])):
                    # Condición para las primeros clientes de cada trayecto
                    if l == 0:
```

```
                        # La primera componente de las siguientes rutas han
de ser igual al tiempo de salida anterior más la distancia hasta el depósito
y vuelta
                        tiempo_llegada[ii][h][l] = tiempo_salida[ii][h - 1][-
1] + distancia[ruta[ii][h - 1][-1], 0] + distancia[
                            0, ruta[ii][h][l]]
                    else:
                        tiempo_llegada[ii][h][l] = tiempo_salida[ii][h][l -
1] + distancia[ruta[ii][h][l - 1], ruta[ii][h][l]]
                        # El vector de tiempos de salida estará condicionado
al valor de llegada al cliente de la siguiente forma:
                    if tiempo_llegada[ii][h][l] >=
readytime_dict[ruta[ii][h][l]]:
                        tiempo_salida[ii][h][l] = tiempo_llegada[ii][h][l] +
servicetime_dict[ruta[ii][h][l]]
                    else:
                        tiempo_salida[ii][h][l] =
readytime_dict[ruta[ii][h][l]] + servicetime_dict[ruta[ii][h][l]]

            # Se actualiza capacidad
            capacidad[ii].insert(jj, demand_dict[indice])

        if fin:
            # Se actualiza vector ruta
            ruta[ii].append([indice])

            # Se actualizaa vector capacidad
            capacidad[ii].append(demand_dict[indice])

            # Variable auxiliar con el tiempo de vuelta del camión al
depósito y tiempo de llegada al nuevo cliente
            aux1 = tiempo_salida[ii][jj][-1] + distancia[ruta[ii][jj][-1], 0]
+ distancia[0, indice]

            # Condicion de que el vehículo llega al primer cliente
sobrepasada su horario de apertura
            if aux1 >= readytime_dict[indice]:

                # Se actualiza vector tiempo de llegada y salida
                tiempo_llegada[ii].append([aux1])
                tiempo_salida[ii].append([aux1 + servicetime_dict[indice]])

            # Si llega sin sobrepasar la hora de entrega
            else:
                # Se actualiza vector tiempo de llegada y salida
                tiempo_llegada[ii].append([aux1])
                tiempo_salida[ii].append([readytime_dict[indice] +
servicetime_dict[indice]])

        if indice in clientes_ptes:
            clientes_ptes.remove(indice)

        # Borrar de la matriz de ahorros las combinaciones de la ruta ya
incluida
        for t in range(len(ruta[ii])):
            for r in range(len(ruta[ii][t])):
                s[ruta[ii][t][r]][indice] = -1000
                s[indice][ruta[ii][t][r]] = -1000
        for m in range(n):
            s[m][indiceaux] = -10000
            s[indiceaux][m] = -10000
```

```
        return ruta, capacidad, tiempo_llegada, tiempo_salida


    # Función para actualizar una ruta cuando uno de los índices está al
principio de una ruta ya establecida
    def ruta_inicio(clientes_ptes, s, ruta, indice, indiceaux, ii, jj, kk,
distancia, demand_dict, servicetime_dict, tiempo_llegada,
                    tiempo_salida):

        # Se actualiza vector ruta
        ruta[ii][jj].insert(kk, indice)
        # Se actualiza vector capacidad
        capacidad[ii][jj] = capacidad[ii][jj] + demand_dict[indice]

        # Para ruta sin multitrip:
        if len(ruta[ii]) == 1 and jj == 0:

            # Condición de que sobrepase o no el tiempo de llegada en el
viaje desde el depósito
            if readytime_dict[indice] <= distancia[0, indice]:
                variable_aux = distancia[0, indice] +
servicetime_dict[indice]
            else:
                variable_aux = readytime_dict[indice] +
servicetime_dict[indice]

            # Se añaaden las variables del nuevo índice
            tiempo_llegada[ii][jj].insert(kk, distancia[0, indice])
            tiempo_salida[ii][jj].insert(kk, variable_aux)

            # Actualizacion del vector tiempo de llegada y vector tiempo de
salida
            for l in range(1, len(tiempo_llegada[ii][jj])):
                # Condición inicial para tiempo de llegada
                tiempo_llegada[ii][jj][l] = tiempo_salida[ii][jj][l - 1] +
distancia[ruta[ii][jj][l - 1], ruta[ii][jj][l]]

                # El vector de tiempos de salida estará condicionado al valor
de llegada al cliente de la siguiente forma:
                if tiempo_llegada[ii][jj][l] >=
readytime_dict[ruta[ii][jj][l]]:
                    tiempo_salida[ii][jj][l] = tiempo_llegada[ii][jj][l] +
servicetime_dict[ruta[ii][jj][l]]
                else:
                    tiempo_salida[ii][jj][l] =
readytime_dict[ruta[ii][jj][l]] + servicetime_dict[ruta[ii][jj][l]]

        # Para ruta con multitrip
        else:
            # Se actualiza el vector ruta
            ruta_aux[ii][jj].insert(kk, indice)

            if jj == 0:
                if readytime_dict[indice] <= distancia[0, indice]:
                    variable_aux = distancia[0, indice] +
servicetime_dict[indice]
                else:
                    variable_aux = readytime_dict[indice] +
servicetime_dict[indice]
                tiempo_llegada[ii][jj].insert(kk, distancia[0, indice])
```

```
            else:
                # Condición de que sobrepase o no el tiempo de llegada en el
viaje desde el depósito
                if readytime_dict[indice] <= tiempo_salida[ii][jj - 1][-1] +
distancia[ruta[ii][jj - 1][-1], indice]:
                    variable_aux = tiempo_salida[ii][jj - 1][-1] +
distancia[ruta[ii][jj - 1][-1], indice] + servicetime_dict[indice]
                else:
                    variable_aux = readytime_dict[indice] +
servicetime_dict[indice]
                tiempo_llegada[ii][jj].insert(kk, tiempo_salida[ii][jj - 1][-
1] + distancia[ruta[ii][jj - 1][-1], indice])

            # Se añaden las variables del nuevo índice

            tiempo_salida[ii][jj].insert(kk, variable_aux)

            # Actualizacion del vector tiempo de llegada y vector tiempo de
salida
            for h in range(jj, len(tiempo_llegada[ii])):
                for l in range(len(tiempo_llegada[ii][h])):
                    # Condición para las primeros clientes de cada trayecto
                    if l == 0:
                        # La primera componente ya se ha incluído
                        if h == jj:
                            continue
                        # La primera componente de las siguientes rutas han
de ser igual al tiempo de salida anterior más la distancia hasta el cliente
                        else:
                            tiempo_llegada[ii][h][l] = tiempo_salida[ii][h -
1][-1] + distancia[ruta[ii][h - 1][-1], ruta[ii][h][l]]
                    else:
                        tiempo_llegada[ii][h][l] = tiempo_salida[ii][h][l -
1] + distancia[ruta[ii][h][l - 1], ruta[ii][h][l]]
                    # El vector de tiempos de salida estará condicionado al
valor de llegada al cliente de la siguiente forma:
                    if tiempo_llegada[ii][h][l] >=
readytime_dict[ruta[ii][h][l]]:
                        tiempo_salida[ii][h][l] = tiempo_llegada[ii][h][l] +
servicetime_dict[ruta[ii][h][l]]
                    else:
                        tiempo_salida[ii][h][l] =
readytime_dict[ruta[ii][h][l]] + servicetime_dict[ruta[ii][h][l]]

        # Se eliminan los índices de la matriz de ahorros
        for m in range(n):
            s[m][indiceaux] = -10000
            s[indiceaux][m] = -10000

        # Se elimina el nuevo cliente de la lista de clientes pendientes
        if indice in clientes_ptes:
            clientes_ptes.remove(indice)

        return ruta, capacidad, tiempo_llegada, tiempo_salida


    # Función para actualizar una ruta cuando uno de los índices está al
final de una ruta ya establecida
    def ruta_final(clientes_ptes, s, ruta, indice, indiceaux, ii, jj,
distancia, demand_dict, readytime_dict, servicetime_dict,
                   tiempo_llegada, tiempo_salida):
```

```
        # Se actualiza vector capacidad
        capacidad[ii][jj] = capacidad[ii][jj] + demand_dict[indice]

        # Sin multitrip
        if len(ruta[ii]) == 1:
            # Se establece el valor del tiempo de llegada en el vector del
nuevo cliente
            tiempo_llegada[ii][jj].append(tiempo_salida[ii][jj][-1] +
distancia[ruta[ii][jj][-1], indice])

            # Se establece el valor del tiempo de salida en el vector del
nuevo cliente
            if readytime_dict[indice] <= tiempo_llegada[ii][jj][-1]:
                tiempo_salida[ii][jj].append(tiempo_llegada[ii][jj][-1] +
servicetime_dict[indice])
            else:
                tiempo_salida[ii][jj].append(readytime_dict[indice] +
servicetime_dict[indice])

            # Se actualiza vector ruta auxiliar
            ruta[ii][jj].append(indice)

        # Con multitrip
        else:
            # Se actualiza vector ruta
            ruta[ii][jj].append(indice)

            # Se actualiza vectores de tiempo de llegada y salida
            tiempo_llegada[ii][jj].append(tiempo_salida[ii][jj][-1] +
distancia[ruta[ii][jj][-1], indice])

            if readytime_dict[indice] <= tiempo_llegada[ii][jj][-1]:
                tiempo_salida[ii][jj].append(tiempo_llegada[ii][jj][-1] +
servicetime_dict[indice])
            else:
                tiempo_salida[ii][jj].append(readytime_dict[indice] +
servicetime_dict[indice])

        # Se eliminan los índices de la matriz de ahorros
        for m in range(n):
            s[m][indiceaux] = -10000
            s[indiceaux][m] = -10000

        # Se eliminan el nuevo cliente de la lista de clientes
        if indice in clientes_ptes:
            clientes_ptes.remove(indice)

        return ruta, capacidad, tiempo_llegada, tiempo_salida


    # ----------------
    # CÓDIGO ALGORITMO
    # ----------------

    # Se comienza a contar el tiempo de ejecución
    tic = time.clock()

    # Se define el tipo de variables que se tomará del generador
    x_dict = dict()
    y_dict = dict()
```

```
    demand_dict = dict()
    readytime_dict = dict()
    duedate_dict = dict()
    n = int()

    # Se importan las variables desde el generador
    x_dict, y_dict, demand_dict, readytime_dict, duedate_dict,
servicetime_dict, n = generador(archivo)

    # Parametros
    hora_max = duedate_dict[0]
    capacidad_max = 40
    coste_nuevo_vehiculo = 1000
    numero_vehiculos = 50

    #Iterador
    it = 0

    #Parametro de salida del bucle
    param = 900

    # Se definen las variables necesarias antes de entrar en el bucle
    # Matriz de distancias
    distancia = zeros((n, n))
    distancia = calculo_distancias(n, x_dict, y_dict)

    # Matriz de ahorros
    s = zeros((n, n))
    s = calculo_ahorros(n, distancia)
    #s = calculo_ahorros_ponderada(0.0001, n, distancia)

    # Vector de clientes
    clientes_ptes = list(range(1, n))

    # Tipos de las variables que se usarán en el bucle
    ruta = list()
    tiempo_llegada = list()
    tiempo_salida = list()
    capacidad = list()

    # BUCLE DE EJECUCIÓN DEL ALGORITMO

    while len(clientes_ptes) != 0 and it < 5000:

        #Variable de iteración
        it = it + 1

        # Se busca el máximo ahorro en la matriz y sus índices
correspondientes
        maximo = s.max()
        indices = where(s == maximo)
        indice0 = int(indices[0][0])
        indice1 = int(indices[1][0])

        # Se eliminan de la matriz los índices usados
        s[indice0, indice1] = -10000

        # Valores inciales variables de posicion referente al indice0 y al
indice1
        inicio0 = False
        inicio1 = False
```

```
        fin0 = False
        fin1 = False
        nueva_ruta0 = True
        nueva_ruta1 = True
        comprob_llegada = False
        comprob_salida = False

        # Se comprueba si los indices obtenidos están al principio o al final
de una ruta ya establecida
        inicio0, fin0, i0, j0, k0, nueva_ruta0 = comprob_inicio_fin(indice0,
ruta)
        inicio1, fin1, i1, j1, k1, nueva_ruta1 = comprob_inicio_fin(indice1,
ruta)

        # Copia de los vectores ruta, tiempo llegada y tiempo salida para que
la referencia no modifique los vectores originales en las funciones
        ruta_aux = copy.deepcopy(ruta)
        tiempo_llegada_aux = copy.deepcopy(tiempo_llegada)
        tiempo_salida_aux = copy.deepcopy(tiempo_salida)

        #Salida del bucle cuando no encuentra soluciones
        if it > param:
            if len(clientes_ptes) != 0 and len(ruta) < numero_vehiculos:
                eliminar = int(clientes_ptes[0])
                ruta.append([[int(clientes_ptes[0])]])
                clientes_ptes.remove(int(clientes_ptes[0]))
                tiempo_llegada.append([[distancia[0, eliminar]]])
                capacidad.append([demand_dict[eliminar]])
                if readytime_dict[eliminar] <= distancia[0, eliminar]:
                    tiempo_salida.append([[distancia[0, eliminar] +
servicetime_dict[eliminar]]])
                else:
                    tiempo_salida.append([[readytime_dict[eliminar] +
servicetime_dict[eliminar]]])

        # 6 Casuísticas para formar ruta: Nueva ruta, ruta inicio con un
índice y otro, ruta final con un índice y otro y ruta para cerrar bucle

        elif nueva_ruta0 and nueva_ruta1:
            # Se calcula el orden de los índices y la viabilidad de la ruta
            indice0, indice1, orden = orden_indices(hora_max, indice0,
indice1, distancia, readytime_dict, duedate_dict)

            # Si el proceso es viable en términos de ventanas temporales
            if orden and numero_vehiculos > len(ruta):
                ruta, capacidad, tiempo_llegada, tiempo_salida =
ruta_nueva_doble(clientes_ptes, ruta, indice0, indice1,
distancia,demand_dict,

readytime_dict, servicetime_dict, tiempo_llegada, tiempo_salida)
            else:
                continue

        elif inicio0 and not (inicio1 or fin1):

            comprob_capacidad = limite_capacidad(capacidad_max, capacidad,
i0, i1, j0, j1, indice0, indice1, inicio0, inicio1, fin0, fin1)
            comprob_llegada, comprob_salida = ventanas_temporales(ruta_aux,
hora_max, i0, j0, k0, distancia, readytime_dict, servicetime_dict,
                                                                  inicio0,
fin0, indice1, tiempo_llegada_aux, tiempo_salida_aux)
```

```
            if comprob_capacidad:
                if comprob_llegada and comprob_salida:
                    ruta, capacidad, tiempo_llegada, tiempo_salida =
ruta_inicio(clientes_ptes, s, ruta, indice1, indice0, i0, j0, k0, distancia,

demand_dict, servicetime_dict, tiempo_llegada, tiempo_salida)
                else:
                    continue
            else:
                # Comprobación que evalúa la rentabilidad de hacer multiviaje
o usar un nuevo vehículo y la viabilidad temporal del multitrip

                comprob_multitrip1, comprob_multitrip2, comprob_multitrip3 =
comprob_multitrip_inicio(i0, j0, indice1, readytime_dict, servicetime_dict,
duedate_dict,

distancia, tiempo_llegada_aux, tiempo_salida_aux,ruta_aux, hora_max,
coste_nuevo_vehiculo)
                comprob_multitrip = comprob_multitrip1 and comprob_multitrip2
and comprob_multitrip3

                if comprob_multitrip:
                    ruta, capacidad, tiempo_llegada, tiempo_salida =
ruta_multitrip(inicio0, fin0, ruta, i0, j0, capacidad, distancia,
demand_dict,

readytime_dict, servicetime_dict, indice1, indice0, tiempo_salida,
tiempo_llegada)
                else:
                    continue

        elif inicio1 and not (inicio0 or fin0):

            comprob_capacidad = limite_capacidad(capacidad_max, capacidad,
i0, i1, j0, j1, indice0, indice1, inicio0, inicio1, fin0, fin1)
            comprob_llegada, comprob_salida = ventanas_temporales(ruta_aux,
hora_max, i1, j1, k1, distancia, readytime_dict, servicetime_dict, inicio1,
                                                                  fin1,
indice0, tiempo_llegada_aux, tiempo_salida_aux)

            if comprob_capacidad:
                if comprob_llegada and comprob_salida:
                    ruta, capacidad, tiempo_llegada, tiempo_salida =
ruta_inicio(clientes_ptes, s, ruta, indice0, indice1, i1, j1, k1, distancia,

demand_dict, servicetime_dict, tiempo_llegada, tiempo_salida)
                else:
                    continue
            else:

                comprob_multitrip1, comprob_multitrip2, comprob_multitrip3 =
comprob_multitrip_inicio(i1, j1, indice0, readytime_dict, servicetime_dict,
duedate_dict,

distancia, tiempo_llegada_aux, tiempo_salida_aux, ruta_aux, hora_max,
coste_nuevo_vehiculo)
                comprob_multitrip = comprob_multitrip1 and comprob_multitrip2
and comprob_multitrip3

                if comprob_multitrip:
```

```
                    ruta, capacidad, tiempo_llegada, tiempo_salida =
ruta_multitrip(inicio1, fin1, ruta, i1, j1, capacidad, distancia,
demand_dict,

readytime_dict,servicetime_dict, indice0, indice1,
tiempo_salida,tiempo_llegada)
                else:
                    continue

        elif fin0 and not (inicio1 or fin1):

            comprob_capacidad = limite_capacidad(capacidad_max, capacidad,
i0, i1, j0, j1, indice0, indice1, inicio0, inicio1, fin0, fin1)
            comprob_llegada, comprob_salida = ventanas_temporales(ruta_aux,
hora_max, i0, j0, k0, distancia, readytime_dict, servicetime_dict, inicio0,
                                                                  fin0,
indice1, tiempo_llegada_aux, tiempo_salida_aux)

            if comprob_capacidad:
                if comprob_llegada and comprob_salida:
                    ruta, capacidad, tiempo_llegada, tiempo_salida =
ruta_final(clientes_ptes, s, ruta, indice1, indice0, i0, j0, distancia,
demand_dict,

readytime_dict, servicetime_dict, tiempo_llegada, tiempo_salida)
                else:
                    continue
            else:
                # Comprobación que evalúa la rentabilidad de hacer multiviaje
o usar un nuevo vehículo y viabilidad temporal del multitrip
                comprob_multitrip1 = tiempo_salida[i0][j0][-1] +
distancia[ruta[i0][j0][-1], 0] + distancia[
                    0, indice1] < coste_nuevo_vehiculo
                comprob_multitrip2 = tiempo_salida[i0][j0][-1] +
distancia[ruta[i0][j0][-1], 0] + distancia[0, indice1] < duedate_dict[
                    indice1]
                comprob_multitrip = comprob_multitrip1 and comprob_multitrip2

                if comprob_multitrip:
                    ruta, capacidad, tiempo_llegada, tiempo_salida =
ruta_multitrip(inicio0, fin0, ruta, i0, j0, capacidad, distancia,
demand_dict,

readytime_dict, servicetime_dict, indice1, indice0, tiempo_salida,
tiempo_llegada)
                else:
                    continue

        elif fin1 and not (inicio0 or fin0):

            comprob_capacidad = limite_capacidad(capacidad_max, capacidad,
i0, i1, j0, j1, indice0, indice1, inicio0, inicio1, fin0, fin1)
            comprob_llegada, comprob_salida = ventanas_temporales(ruta_aux,
hora_max, i1, j1, k1, distancia, readytime_dict, servicetime_dict, inicio1,
                                                                  fin1,
indice0, tiempo_llegada_aux, tiempo_salida_aux)
            if comprob_capacidad:
                if comprob_llegada and comprob_salida:
                    ruta, capacidad, tiempo_llegada, tiempo_salida =
ruta_final(clientes_ptes, s, ruta, indice0, indice1, i1, j1, distancia,
demand_dict,
```

```
readytime_dict, servicetime_dict, tiempo_llegada, tiempo_salida)
                else:
                    continue
            else:
                comprob_multitrip1 = tiempo_salida[i1][j1][-1] +
distancia[ruta[i1][j1][-1], 0] + distancia[0, indice0] < coste_nuevo_vehiculo
                comprob_multitrip2 = tiempo_salida[i1][j1][-1] +
distancia[ruta[i1][j1][-1], 0] + distancia[0, indice0] < duedate_dict[
                    indice0]
                comprob_multitrip = comprob_multitrip1 and comprob_multitrip2

                if comprob_multitrip:
                    ruta, capacidad, tiempo_llegada, tiempo_salida =
ruta_multitrip(inicio1, fin1, ruta, i1, j1, capacidad, distancia,
demand_dict,

readytime_dict, servicetime_dict, indice0, indice1, tiempo_salida,
tiempo_llegada)
                else:
                    continue

    print("La ruta solución del problema es:\n", ruta)
    print("El número de vehículos utilizado es %d \n" % len(ruta))
    print("Los clientes pendiente de reparto son: \n", clientes_ptes)

    # Cálculo Distancia recorrida

    distancia_calculo = 0

    for i in range(len(ruta)):
        for j in range(len(ruta[i])):
            for k in range(len(ruta[i][j])):
                if len(ruta[i][j]) == 1:
                    distancia_calculo = distancia_calculo + 2 * distancia[0,
ruta[i][j][k]]
                else:
                    if k == 0:
                        distancia_calculo = distancia_calculo + distancia[0,
ruta[i][j][k]] + distancia[ruta[i][j][k], ruta[i][j][k + 1]]
                    elif k != 0 and k < (len(ruta[i][j]) - 1):
                        distancia_calculo = distancia_calculo +
distancia[ruta[i][j][k], ruta[i][j][k + 1]]
                    elif k == (len(ruta[i][j]) - 1):
                        distancia_calculo = distancia_calculo +
distancia[ruta[i][j][k], 0]

    print("\nDistancia total recorrida: ", distancia_calculo)

    # Calculo tiempo usado

    tiempo_total = 0

    for i in range(len(ruta)):
        for j in range(len(ruta[i])):
            tiempo_total = tiempo_total + tiempo_salida[i][j][-1] +
distancia[ruta[i][j][-1], 0]

    print("\nTiempo total de reparto:", tiempo_total)
```

```
    dimension_ruta = len(ruta)

    toc = time.clock()

    tiempo_ejecucion = toc - tic

    print("\nEl tiempo de ejecución ha sido de %f segundos." %
tiempo_ejecucion)
    # ----------------------
    # VISUALIZACION DE DATOS
    # ----------------------

    # Los tiempos de llegada y salida se ordenan en nueva lista con la
posición igual al número de cliente

    numero_cliente = 1
    vector_tiempos_llegada = [0]
    vector_tiempos_salida = [0]
    while numero_cliente <= n:
        for i in range(len(ruta)):
            for j in range(len(ruta[i])):
                for k in range(len(ruta[i][j])):
                    if ruta[i][j][k] == numero_cliente:

vector_tiempos_llegada.append(tiempo_llegada[i][j][k])
                        vector_tiempos_salida.append(tiempo_salida[i][j][k])

        numero_cliente = numero_cliente + 1

    # Los datos de coordenadas se pasan a una lista para su tratamiento
    x = list()
    y = list()

    for key in x_dict:
        x.append(x_dict[key])

    for key in y_dict:
        y.append(y_dict[key])

    # Se procede a hacer un plot con las rutas y localizaciones elegidas en
un mapa de puntos

    plt.plot(x, y, 'x', color='black')
    plt.axis()
    plt.xlabel("Coordenada x")
    plt.ylabel("Coordenada y")
    plt.grid()

    #Se crea un vector con diferentes colores
    colors = dict(mcolors.BASE_COLORS, **mcolors.CSS4_COLORS)
    del colors['w']
    del colors['aqua']
    del colors['antiquewhite']
    del colors['paleturquoise']
    del colors['blue']
    del colors['black']
    del colors['chocolate']
    del colors['aliceblue']
    del colors['beige']
    del colors['blanchedalmond']
    del colors['bisque']
```

```
    del colors['burlywood']
    del colors['azure']
    del colors['cadetblue']
    del colors['chartreuse']
    del colors['brown']
    del colors['g']

    names = list()
    color = 0

    for cliente in range(len(x)):
        plt.annotate('{}'.format(cliente), xy=[x[cliente], y[cliente]])

    for key in colors:
        names.append(key)

    for i in range(len(ruta)):
        for j in range(len(ruta[i])):
           #En cada iteracion se crean los vectores que formarán la ruta
            vector_vaciox = list()
            vector_vacioy = list()
            #Salida del depósito
            vector_vaciox.append(float(x[0]))
            vector_vacioy.append(float(y[0]))

            for k in range(len(ruta[i][j])):
                vector_vaciox.append(x[ruta[i][j][k]])
                vector_vacioy.append(y[ruta[i][j][k]])

            #Vuelta al depósito
            vector_vaciox.append(float(x[0]))
            vector_vacioy.append(float(y[0]))
            color = color + 1
            if j == 0:
                plt.plot(vector_vaciox, vector_vacioy, c=names[i],
label="Camión %d" % (i + 1))
            elif j == 1:
                plt.plot(vector_vaciox, vector_vacioy, linestyle="--",
c=names[i], label="Camión %d" % (i + 1))
            elif j == 2:
                plt.plot(vector_vaciox, vector_vacioy, linestyle=":",
c=names[i], label="Camión %d" % (i + 1))
            elif j == 3:
                plt.plot(vector_vaciox, vector_vacioy, linestyle="-.",
c=names[i], label="Camión %d" % (i + 1))
    plt.legend(loc='best')

    plt.savefig("figura1", dpi=450)
    # Plot con las ventanas temporales

    handles = list(range(len(ruta)))

    fig, ax = plt.subplots(nrows=n-1, sharex=True)
    cont = 0

    intensidad_color = [0.5, 0.3, 0.15, 0.05]

    #Bucle para crear tantas imágenes de intervalos como clientes haya
    for i in range(len(ruta)):
        for j in range(len(ruta[i])):
            for k in range(len(ruta[i][j])):
```

```
                ax[cont].set(yticks=[])
                ax[cont].set_ylabel(ylabel = "%d" % int(ruta[i][j][k]),
ma='left', y=0)
                ax[cont].yaxis.set_label_coords(-0.03, 0)

                ax[cont].plot([tiempo_llegada[i][j][k]],[0],'go',)
                ax[cont].plot([tiempo_salida[i][j][k]],[0],'rx')
                ax[cont].plot([tiempo_llegada[i][j][k],
tiempo_salida[i][j][k]],[0,0],'grey',linestyle='--')
                ax[cont].yaxis.label.set_size(5)
                ax[cont].yaxis.label.set_rotation(0)
                ax[cont].axvline(readytime_dict[ruta[i][j][k]],
color='black', ymax=1, ymin=0)
                ax[cont].axvline(duedate_dict[ruta[i][j][k]], color='black',
ymax=1, ymin=0)

                ax[cont].axvspan(readytime_dict[ruta[i][j][k]],
duedate_dict[ruta[i][j][k]], alpha=intensidad_color[j], color=names[i],
label="Camión %d" % (i + 1))

                plt.xlim(-25, hora_max)
                plt.ylim(-0.1, 0.1)

                cont = cont + 1


        variable, labels = ax[i].get_legend_handles_labels()
        handles[i] = mpatches.Patch(color=names[i], alpha=0.3, label="Camion
%d" % (i + 1))

    # Se representan las leyendas

    fig.legend(handles=[i for i in handles])

    #Etiquetas globales
    fig.text(0.5, 0.04, 'Tiempo', ha='center', va='center')
    fig.text(0.07, 0.5, 'Clientes', ha='center', va='center',
rotation='vertical')


    plt.show()
    plt.savefig("figura2", figsize=[9, 16], dpi=450)
    '''
        return distancia_calculo, dimension_ruta, tiempo_ejecucion,
tiempo_total
```

7.3 Script de ejecución principal

```
from Algoritmo_Entrega import *
from openpyxl import Workbook

#Problemas tipo C

calculo_distancia_c = list()
numero_vehiculos_c = list()
tiempo_reparto_c = list()
tiempo_ejecucion_c =list()

numeracion_c = ["c101", "c102", "c103", "c104", "c105", "c106", "c107",
```

```
"c108", "c109", "c201", "c202", "c203", "c204", "c205", "c206", "c207",
"c208"]
#numeracion_c = ["c201", "c202", "c203", "c204", "c205", "c206", "c207",
"c208"]


for i in range(len(numeracion_c)):
    archivo = "/Users/carlossanchezgomez/PycharmProjects/master/Archivos
Solomon/%s.txt" % numeracion_c[i]
    print(archivo)
    distancia, numero_vehiculos, tiempo_ejecucion, tiempo_reparto =
algoritmo_entrega(archivo)

    calculo_distancia_c.append(distancia)
    numero_vehiculos_c.append(numero_vehiculos)
    tiempo_reparto_c.append(tiempo_reparto)
    tiempo_ejecucion_c.append(tiempo_ejecucion)


print(calculo_distancia_c, numero_vehiculos_c, tiempo_reparto_c)
print("\n")


# EXPORTACION A EXCEL
wb = Workbook()
problemas_vrp_mejora = 'problemas_vrp_c_100_COMPROBAR.xlsx'

hoja = wb.active
hoja.title = "Tiempos"

fila = 1 #Fila donde se empieza
col_tipo_problema = 1
col_distancia = 2
col_vehiculos = 3
col_tiempo_reparto = 4
col_tiempo_ejecucion = 5

for i in range(len(numeracion_c)):
        hoja.cell(column=col_tipo_problema, row=fila, value=numeracion_c[i])
        hoja.cell(column=col_distancia, row=fila,
value=calculo_distancia_c[i])
        hoja.cell(column=col_vehiculos, row=fila,
value=numero_vehiculos_c[i])
        hoja.cell(column=col_tiempo_reparto, row=fila,
value=tiempo_reparto_c[i])
        hoja.cell(column=col_tiempo_ejecucion, row=fila,
value=tiempo_ejecucion_c[i])
        fila = fila + 1

wb.save(filename = problemas_vrp_mejora)
```