Sección de Matemáticas
Universidad de La Laguna

Carlos A. Pavillard Pérez

Problema de ruta y localización con recogida y entrega simultánea de producto

Location-routing problem with simultaneous pickup and delivery

Trabajo Fin de Grado
Grado en Matemáticas
La Laguna, julio de 2023

Dirigido por
Hipólito Hernández Pérez

Hipólito Hernández Pérez
Matemáticas, Estadística e
Investigación Operativa
Universidad de La Laguna
38200 La Laguna, Tenerife

Agradecimientos

En primer lugar, me gustaría agradecer a todos los profesores, que me han enseñado durante estos años, de los cuales he podido aprender mucho.

Especialmente, me gustaría hacer mención al profesor Hipólito Hernández Pérez, que ha tutorizado este proyecto todos estos meses, ya que su experiencia y sus enseñanzas han sido fundamentales para el desarrollo de este.

Quiero agradecerles a mis padres su dedicación. Sin ellos y su apoyo incondicional no lo hubiese conseguido. También, me gustaría mencionar a toda mi familia y amigos, que siempre han estado ahí, en las buenas y en la malas, apoyándome y ayudando en todo lo que podían. Sin todos ellos esto no habría sido posible.

Muchas gracias a todos.

Carlos A. Pavillard Pérez
La Laguna, 10 de julio de 2023

Resumen · Abstract

Resumen

En esta memoria se presentan dos formulaciones matemáticas para el problema de localización y rutas con entrega y recogida simultánea de producto. El objetivo del problema consiste en determinar la utilización de lo depósitos disponibles, la asignación de los clientes a los depósitos que serán utilizados y las correspondientes rutas de vehículos, minimizando los costos totales. Se realizan varios experimentos para ver con qué modelo se obtiene mejores resultados, y además, estos se comparan con los ya existentes en la literatura. Por último, se muestra en imágenes una representación de las soluciones.

Palabras clave: *Optimización – Pyhton – Gurobi – Rutas – Localización.*

Abstract

This paper presents two mathematical formulations for the location-routing problem with simultaneous pickup and delivery. The objective of the problem is to determine the utilization of available depots, the assignment of customers to the depots that will be used, and the corresponding vehicle routes, with the goal of minimizing total costs. Several experiments are conducted to compare the performance of the models and to compare them with existing literature. Finally, visual representations of the solutions are shown in images.

Keywords: *Optimization – Pyhton – Gurobi – Routing – Location.*

Contenido

Introducción

Hoy en día, dada la competitividad en el mundo empresarial, es de vital importancia cuando hablamos de cadenas de suministros, la decisión estratégica y la optimización de los recursos. Una de las decisiones estratégicas más relevantes es el diseño de redes de distribución, ya que ofrece oportunidades para reducir costos y mejorar la calidad del servicio. En este contexto, las decisiones clave se centran en la ubicación de las instalaciones y las rutas entre estas y los clientes.

Este trabajo trata sobre la optimización de ubicaciones de instalaciones y costos de rutas, a la hora de entregar y recoger productos de los clientes. Este tipo de cuestiones, deriva de uno de los problemas más trabajados en la optimización combinatoria, el problema de localización y rutas.

En el capítulo 1 se proporcionarán las definiciones y notaciones necesarias para comprender el contenido de este trabajo. Además, también se comenta el software utilizado para llevar a cabo una serie de experimentos.

En el capítulo 2 se comenzará abordando el principal objetivo de este trabajo, el problema de ruta y localización con recogida y entrega simultánea de producto. Luego se presentarán dos modelos matemáticos para resolverlo.

En el capítulo 3 se mostrarán los resultados al ejecutar un solucionador de modelos lineales mixtos con estos dos modelos, según el código fuente que encontramos en el Apéndice A. Los resultados serán comparados en tablas con los resultados obtenidos por Karaoglan et al. [1]. Además, se mostrarán las imágenes que representan las soluciones de las instancias utilizadas y el proceso iterativo que sigue el segundo modelo.

1

Herramientas matemáticas

En este capítulo se introducen algunas ideas y conceptos fundamentales del estudio de grafos y el desarrollo de modelos para la programación combinatoria, en especial los problemas de localización y rutas. Por último, también se habla del lenguaje y el solucionador matemático utilizado durante este proyecto.

1.1. Grafos

Un grafo $G = (N, A)$ es una estructura matemática que consiste en un conjunto N de nodos (también conocidos como vértices) y un conjunto A de arcos (también conocidas como aristas) que conectan los nodos. Los grafos se utilizan para representar relaciones entre elementos, personas, ciudades, empresas, entre otros. Los arcos, por la otra parte, representan las relaciones entre los nodos, como rutas, conexiones físicas, relaciones de parentesco, etc.

Un grafo no dirigido es aquel en el que las aristas no tienen una dirección específica. Esto significa que la relación entre dos nodos es simétrica y se puede recorrer en ambas direcciones. En un grafo no dirigido, la conexión entre dos nodos es bidireccional.

Por otro lado, un grafo dirigido es aquel en el que las aristas tienen una dirección definida. Esto significa que, las conexiones entre los nodos son unidireccionales, lo que implica que el flujo de información o relación va en una sola dirección.

Generalmente en un grafo no dirigido a la arista que une dos nodos i y j se la denota por $[i, j]$. Mientras que en un grafo dirigido, al arco con origen en i y destino en j se le denota por (i, j). En esta memoria utilizaremos fundamen-

talmente los arcos dirigidos, de forma que el conjunto de arcos del grafo sería $A \subseteq \{(i,j) : i,j \in V, i \neq j\}$.

Un grafo se dice completo cuando, tiene todos los posibles arcos $A = \{(i,j) : i,j \in V, i \neq j\}$, si $|V| = n$, entonces el número de arcos de un grafo dirigido completo es $n(n-1)$. Si se tuviera un grafo no dirigido completo el número de arcos sería $n(n-1)/2$.

Un camino es una secuencia de arcos de A, donde el nodo destino de un arco coincide con el nodo origen del siguiente. Por lo tanto, dos nodos consecutivos de un camino, comparten un mismo nodo incidente. Un camino simple es aquel, que conectando dos nodos en un grafo deba cumplir que todos los vértices intermedios son distintos. Por otra parte tenemos un círculo o circuito que es un camino simple, donde el primer y último vértice coinciden.

1.2. Introducción a la optimización combinatoria

La optimización combinatoria es un área de la investigación operativa que consiste en plantear y resolver problemas optimización donde el conjunto de posibles soluciones es finito (pero generalmente muy grande). Estos problemas se suelen plantear con modelos de programación matemática donde existen variables de decisión discretas. Así, la programación lineal entera o programación lineal entera mixta son dos técnicas para resolver este tipo de problemas, pero existen otras como la búsqueda exhaustiva, la programación cuadrática entera, metaheurísticas, etc. A lo largo de esta memoria se trabaja con los métodos de programación lineal entera mixta para resolver el problema planteado.

A menudo, la optimización combinatoria se enfrenta a una explosión combinatoria. Esto consiste en un crecimiento exponencial del número de posibles soluciones a medida que el problema se vuelve es más grande. Para abordar esta cuestión, es necesario desarrollar métodos y enfoques eficientes que permitan reducir el espacio de búsqueda y encontrar soluciones de alta calidad en un tiempo razonable.

Para obtener una comprensión más profunda sobre este tema, es recomendable consultar algún libro de programación matemática y optimización combinatoria como puede ser Salazar [2].

1.3. Problemas de localización

El problema de localización de instalaciones (*Facility Location Problem* (FLP) en inglés) es un problema de optimización cuyo objetivo es tratar de encontrar la localización óptima de una o varias instalaciones, para un conjunto I de localizaciones posibles, por ejemplo, para plantas de producción, almacenes, aeropuertos u otros servicios. Este problema es ampliamente estudiado en áreas como la logística, la planificación urbana, el diseño de redes y la toma de decisiones empresariales. Además, existe un conjunto J de clientes ya establecidos, los cuales son una condición para hallar una localización óptima de las instalaciones. Se puede plantear el problema con una capacidad de producción de las instalaciones o sin ella, aunque en este informe trabajaremos teniendo en cuenta una capacidad para cada instalación.

Este tiene un costo f_i de abrir o mantener abierta una instalación $i \in I$. También existe un costo c_{ij} de satisfacer la demanda del cliente $j \in J$ desde la planta $i \in I$, una capacidad de producción q_i de la planta $i \in I$, y una demanda d_j del cliente j. Luego, el problema consiste en determinar qué plantas se abren y desde qué plantas se ha de satisfacer la demanda de cada cliente, minimizando, por lo general, el coste total. Además, cada cliente tiene que estar asignado a una única planta.

Variables de decisión

- $x_{ij} = \begin{cases} 1 & \text{si cliente } j \text{ está asignado a planta } i \quad (\forall i \in I, j \in J) \\ 0 & \text{otro caso} \end{cases}$
- $y_i = \begin{cases} 1 & \text{si se abre o utiliza la planta } i \quad (\forall i \in I) \\ 0 & \text{otro caso} \end{cases}$

Modelo matemático

$$\text{mín} \sum_{i \in I} f_i y_i \; + \; \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} \tag{1.1}$$

sujeto a:

$$\sum_{i \in I} x_{ij} = 1 \qquad \forall j \in J \tag{1.2}$$

$$\sum_{j \in J} d_j x_{ij} \leqslant q_i y_i \qquad \forall i \in I \tag{1.3}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i \in I, j \in J \tag{1.4}$$

$$y_i \in \{0, 1\} \qquad \forall i \in I \tag{1.5}$$

En esta formulación, la función objetivo (1.1) busca minimizar el costo total del sistema, que incluye los costos de transporte y los de apertura de instalaciones. Las restricciones (1.2) aseguran que cada cliente es visitado por una única planta, mientras que las restricciones (1.3) garantizan que la demanda de los clientes asignados a una planta no supere su capacidad de producción. Por último, las restricciones (1.4) y (1.5) imponen que las variables sean binarias.

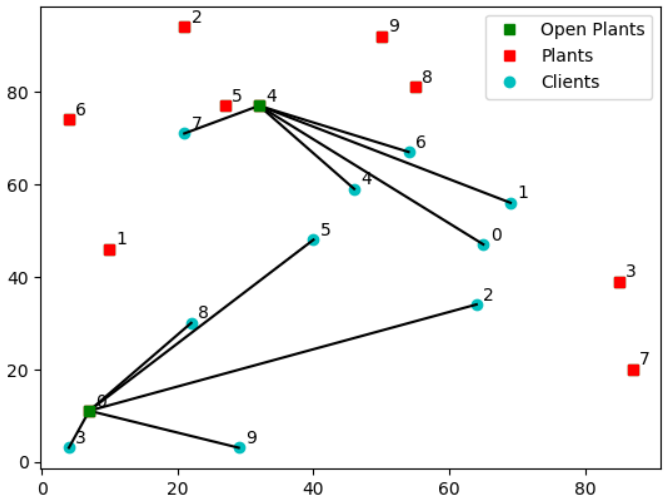

Figura 1.1: Solución óptima a un problema de localización

1.4. Problemas de rutas

Los problemas de rutas de vehículos [9] (*Vehicle Routing Problems* (VRPs) en inglés) son un tipo de problemas que buscan encontrar la mejor forma de recorrer un conjunto de ubicaciones, partiendo desde alguna ubicación fijada,

minimizando algún objetivo. Han sido problemas muy estudiados en la programación lineal entera y programación lineal entera mixta. Tienen muchas aplicaciones prácticas, como la planificación de rutas de distribución, la optimización del transporte de mercancías y el diseño de redes de comunicación entre otras.

En su forma más común, el problema de rutas tiene un único depósito (suele ser la localización 1), $n-1$ clientes, una flota de vehículos K y cada cliente tiene asociada una demanda d_i de producto que tiene que recibir. Además, hay que tener en cuenta restricciones como que cada cliente es visitado por un solo vehículo y una sola vez, la capacidad del vehículo Q, los costos de transporte $c_{ij} = c_a$ y las distancias entre ubicaciones. El objetivo principal suele ser minimizar la distancia total recorrida, el tiempo total empleado, el costo total o algún otro costo asociado, encontrando la secuencia óptima de visitas a los clientes.

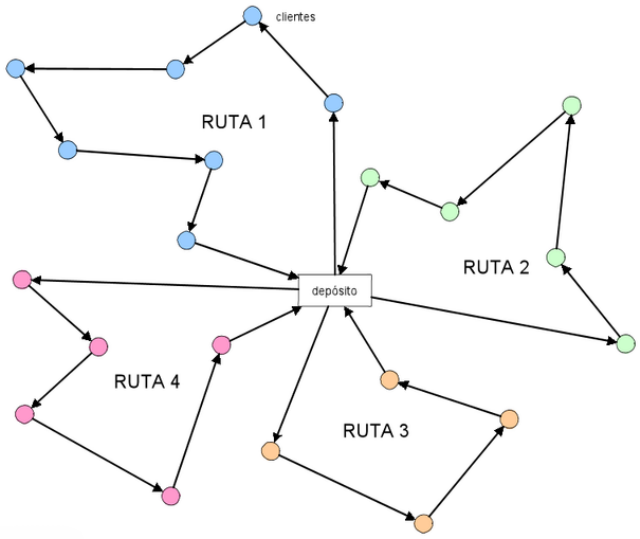

Figura 1.2: Esquema básico de un problema de rutas

Variables de decisión

- $x_{ij} = x_a = \begin{cases} 1 & \text{si el arco } a = (i,j) \text{ es utilizado por algún vehículo} \quad (\forall a \in A) \\ 0 & \text{otro caso} \end{cases}$

- $f_{ij} = f_a =$ carga del vehículo por el arco a=(i,j)

Modelo matemático

$$\text{mín} \sum_{a \in A} \sum_{k \in K} c_a x_a \tag{1.6}$$

sujeto a:

$$\sum_{a \in \delta^+(i)} x_a = 1 \qquad \forall i \in I \setminus \{1\} \tag{1.7}$$

$$\sum_{a \in \delta^-(1)} x_a \leqslant |K| \tag{1.8}$$

$$\sum_{a \in \delta^+(i)} x_a = \sum_{a \in \delta^-(i)} x_a \qquad \forall i \in I \tag{1.9}$$

$$\sum_{a \in \delta^-(i)} f_a - \sum_{a \in \delta^+(i)} f_a = d_i \qquad \forall i \in I \setminus \{1\} \tag{1.10}$$

$$0 \leqslant f_a \leqslant Q x_a \qquad \forall a \in A \tag{1.11}$$

$$x_a \in \{0, 1\} \qquad \forall a \in A \tag{1.12}$$

En esta formulación, la función objetivo (1.6) busca minimizar el costo total del sistema, que está compuesto únicamente por los costos de transporte. Las restricciones (1.7) aseguran, que exceptuando el depósito, cada cliente es visitado por solo un vehículo, mientras que la restricción (1.9) asegura que desde el depósito solo pueda salir como mucho el número total de vehículos de la flota. Además, las restricciones (1.8) garantizan que de cada nodo salga el mismo número de vehículos que llega. Las restricciones (1.10) aseguran que cada cliente recibe la cantidad de producto que demanda, y evita los subciclos. Por otro lado, las restricciones (1.11) garantizan que la carga del vehículo no supere la capacidad máxima del mismo. Por último, las restricciones (1.12) imponen que estas variables sean binarias.

El problema se puede plantear igual, si en lugar de entregar producto se recoge producto.

1.5. Lenguaje y solucionador matemático

Para empezar, se describe el lenguaje y el solucionador matemático que se ha utilizado en el artículo de Karaoglan et al. [1], para el cual todos los

algoritmos fueron implementados en C++, y se utilizó el solucionador CPLEX. A continuación, se explica los utilizados para los algoritmos de este trabajo, que se implementaron en Python y se utilizó el solucionador Gurobi.

En primer lugar, C++ [11] ampliamente utilizado en el desarrollo de software. Fue desarrollado por Bjarne Stroustrup en los años 80. Con su combinación de programación procedural y orientada a objetos, brinda flexibilidad a los desarrolladores para abordar una variedad de problemas. La gestión manual de memoria en C++ permite un control preciso de los recursos del sistema, aunque requiere atención para evitar errores.

CPLEX [10] es un software de optimización matemática desarrollado por IBM. Es ampliamente utilizado para resolver problemas de optimización en diversas industrias. Proporciona algoritmos avanzados y una interfaz fácil de usar. Es una herramienta poderosa para formular, resolver y analizar problemas de optimización en áreas como logística, finanzas y gestión de cadena de suministro.

Por otro lado, Python [4][12] es un lenguaje de programación de alto nivel, interpretado y versátil. Fue creado a finales de la década de 1980 por Guido van Rossum en los Países Bajos. Destaca por su sintaxis clara y legible, facilitando su aprendizaje y uso. Python se utiliza ampliamente en el desarrollo de software, análisis de datos, inteligencia artificial, automatización y más.

Finalmente, Gurobi [3] es un software líder en optimización matemática utilizado en múltiples sectores e industrias. Proporciona una plataforma de programación avanzada para resolver eficientemente problemas de optimización. Destaca por su capacidad de resolver problemas de programación lineal, entera mixta y cuadrática, entre otros. Gurobi se aplica ampliamente en logística, cadena de suministro, finanzas y otras áreas donde se busca optimizar decisiones y asignar recursos de manera eficiente.

2

Problema de localización y rutas con recogida y entrega simultánea de producto

Una vez vistas las herramientas matemáticas y los problemas de localización y rutas, en este capítulo damos paso al objetivo principal de este trabajo. Hablamos de una variante del problema de localización y rutas (que abreviaremos como LRP al venir del inglés *Location-Routing Problem*), conocida como problema de localización y rutas con recogida y entrega simultánea de producto (que abreviaremos como LRPSPD al venir del inglés *Location-Routing Problem with Simultaneous Pickup and Delivery*). Aquí describiremos los dos modelos matemáticos que se han utilizado para la resolución de este problema.

2.1. Introducción

El LRP aborda la determinación de las ubicaciones de instalaciones y rutas de los vehículos para atender a los clientes, considerando restricciones como capacidades de las instalaciones y los vehículos, longitud de las rutas, entre otras. El objetivo es satisfacer las demandas de los clientes y minimizar el costo total, que incluye costos de rutas, costos fijos de los vehículos, costos fijos de las instalaciones y costos operativos de las mismas. En su forma general, el LRP asume que los clientes solo tienen demanda de entrega y se enfoca en la distribución de productos utilizando una flota de vehículos estacionados en los depósitos. Sin embargo, en la práctica, los clientes pueden tener demandas de recogida y entrega, y a menudo requieren que ambas demandas sean atendidas al mismo tiempo, así abordaremos el LRPSPD.

Para este problema consideremos un grafo dirigido $G = (N, A)$ con $n := |N|$ nodos, donde $N = N_0 \cup N_C$. Los primeros nodos representan el conjunto de depósitos disponibles N_0, mientras que el resto de nodos representa el conjunto de clientes N_C. El conjunto de arcos se define como $A = \{(i, j) : (i \in N_C \wedge j \in$

$N_C) \vee (i \in N_C \wedge j \in N_0) \vee (i \in N_0 \wedge j \in N_C)\}$, donde cada arco tiene un coste c_{ij}, relacionado con la distancia recorrida por los vehículos. Además, hay una capacidad CD_k y un coste fijo FD_k, asociados a cada depósito disponible $k \in N_0$. También, existe una ilimitada flota de vehículos homogéneos dispuesta a servir a los clientes, de capacidad CV y con un costo fijo de utilización FV, incluyendo el costo de adquirir los vehículos utilizados en la ruta. Cada cliente $i \in N_C$ tiene una demanda de recogida p_i y entrega d_i, con $0 < p_i, d_i \leqslant CV$.

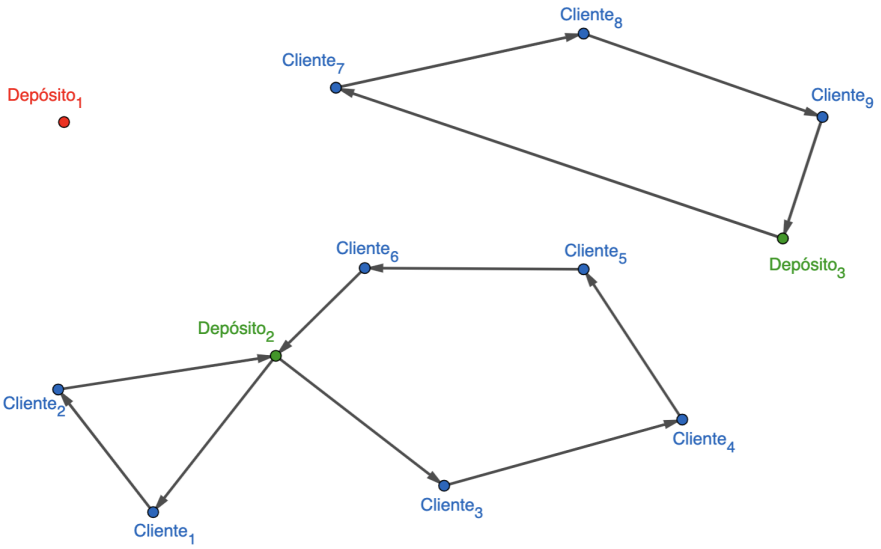

Figura 2.1: Esquema de un problema de localización y rutas con recogida y entrega simultánea de producto

El problema consiste en determinar la utilización óptima de los depósitos disponibles, asignar los clientes a los depósitos que se utilizan y encontrar las rutas de los vehículos correspondientes, minimizando el costo total, con las siguientes limitaciones:

- Cada vehículo es utilizado como mucho en una ruta.
- Cada cliente obtiene el servicio de un único depósito.
- Cada ruta comienza y finaliza en el mismo depósito.
- La carga total de un vehículo en cualquier momento de la ruta no debe sobrepasar la capacidad del vehículo.

- La carga total de recogida y entrega de los clientes asignados a un depósito no excede la capacidad del depósito.

2.2. Primer modelo matemático (Modelo 1)

En esta sección se reflejan dos formulaciones diferentes para el problema de localización y rutas con recogida y entrega simultánea de producto. La primera de ellas fue enunciada por Karaoglan et al. [1].

A continuación, se describen las cinco familias de variables de decisión, tres son variables de decisión binarias, x_{ij}, y_k y z_{ik}, y las dos restantes son variables de decisión continuas, U_{ij} y V_{ij}. Todas asociadas con los arcos y nodos de G, con el siguiente significado:

- $x_{ij} = \begin{cases} 1 & \text{si un vehículo viaja del nodo } i \text{ al nodo } j \quad \forall (i,j) \in A \\ 0 & \text{otro caso} \end{cases}$

- $y_k = \begin{cases} 1 & \text{si se abre el depósito } k \quad \forall k \in N_0 \\ 0 & \text{otro caso} \end{cases}$

- $z_{ik} = \begin{cases} 1 & \text{si el cliente } i \text{ está asignado al depósito } k \quad \forall i \in N_C,\ \forall k \in N_0 \\ 0 & \text{otro caso} \end{cases}$

- U_{ij}; demanda a ser entregada a los clientes en la ruta después del nodo i y transportada en el arco (i,j) si un vehículo viaja directamente desde el nodo i al nodo j, de lo contrario vale 0. Para todo $(i,j) \in A$.

- V_{ij}; demanda a ser recogida de los clientes en la ruta hasta el nodo i (incluyendo el nodo i) y transportada en el arco (i,j) si un vehículo viaja directamente desde el nodo i al nodo j, de lo contrario vale 0. Para todo $(i,j) \in A$.

Modelo matemático

$$\text{mín} \sum_{i\in N}\sum_{j\in N} c_{ij}x_{ij} \; + \sum_{k\in N_0} FD_k y_k \; + \sum_{k\in N_0}\sum_{i\in N_C} FV x_{ki} \tag{2.1}$$

sujeto a:

$$\sum_{j\in N} x_{ij} = 1 \qquad \forall i \in N_C \tag{2.2}$$

$$\sum_{j\in N} x_{ji} = \sum_{j\in N} x_{ij} \qquad \forall i \in N \tag{2.3}$$

$$\sum_{j\in N} U_{ji} - \sum_{j\in N} U_{ij} = d_i \qquad \forall i \in N_C \tag{2.4}$$

$$\sum_{j\in N} V_{ij} - \sum_{j\in N} V_{ij} = p_i \qquad \forall i \in N_C \tag{2.5}$$

$$U_{ij} + V_{ij} \leqslant CV x_{ij} \qquad \forall i,j \in N, i \neq j \tag{2.6}$$

$$\sum_{j\in N_C} U_{kj} = \sum_{j\in N_C} z_{jk} d_j \qquad \forall k \in N_0 \tag{2.7}$$

$$\sum_{j\in N_C} U_{jk} = 0 \qquad \forall k \in N_0 \tag{2.8}$$

$$\sum_{j\in N_C} V_{jk} = \sum_{j\in N_C} z_{jk} p_j \qquad \forall k \in N_0 \tag{2.9}$$

$$\sum_{j\in N_C} V_{kj} = 0 \qquad \forall k \in N_0 \tag{2.10}$$

$$U_{ij} \leqslant (CV - d_i)x_{ij} \qquad \forall i \in N_C, \forall j \in N \tag{2.11}$$

$$V_{ij} \leqslant (CV - p_j)x_{ij} \qquad \forall i \in N, \forall j \in N_C \tag{2.12}$$

$$U_{ij} \geqslant d_j x_{ij} \qquad \forall i \in N, \forall j \in N_C \tag{2.13}$$

$$V_{ij} \geqslant p_i x_{ij} \qquad \forall i \in N_C, \forall j \in N \tag{2.14}$$

$$\sum_{k\in N_0} z_{ik} = 1 \qquad \forall i \in N_C \tag{2.15}$$

$$\sum_{i\in N_C} d_i z_{ik} \leqslant CD_k y_k \qquad \forall k \in N_0 \tag{2.16}$$

$$\sum_{i\in N_C} p_i z_{ik} \leqslant CD_k y_k \qquad \forall k \in N_0 \tag{2.17}$$

$$x_{ik} \leqslant z_{ik} \qquad \forall i \in N_C, \forall k \in N_0 \tag{2.18}$$

$$x_{ki} \leqslant z_{ik} \qquad \forall i \in N_C, \forall k \in N_0 \tag{2.19}$$

$$x_{ij} + z_{ik} + \sum_{m\in N_0, m\neq k} z_{jm} \leqslant 2 \qquad \forall i,j \in N_C, i \neq j, \forall k \in N_0 \tag{2.20}$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j \in N \tag{2.21}$$

$$z_{ik} \in \{0,1\} \qquad \forall i \in N_C, \forall k \in N_0 \tag{2.22}$$

$$y_i \in \{0,1\} \qquad \forall k \in N_0 \tag{2.23}$$
$$U_{ij}, V_{ij} \geqslant 0 \qquad \forall i,j \in N \tag{2.24}$$

En esta formulación, la función objetivo (2.1) busca minimizar el costo total del sistema, que incluye los costos de transporte, depósito y vehículo. Las restricciones (2.2) aseguran que cada cliente sea visitado exactamente una vez, mientras que las restricciones (2.3) garantizan que el número de arcos de entrada y el número de arcos de salida a cada nodo sean iguales. Las restricciones (2.4) y (2.5) son restricciones de conservación de flujo para las demandas de entrega y recogida, respectivamente. Estas restricciones eliminan los subciclos y garantizan que se satisfagan las demandas de recolección y entrega de cada cliente.

Las restricciones (2.6) establecen que la carga total en cualquier arco no debe exceder la capacidad del vehículo. Por otro lado, las restricciones (2.7) aseguran que la carga total despachada desde cada depósito coincida con la demanda total de entrega de los clientes asignados a ese depósito, mientras que las restricciones (2.8) garantizan que la cantidad total de carga de entrega que regresa a los depósitos sea cero. De manera similar, las restricciones (2.9) aseguran que la carga total de recolección que ingresa a cada depósito coincida con la demanda total de recolección de los clientes asignados a ese depósito, y las restricciones (2.10) garantizan que la cantidad total de carga de recolección despachada desde los depósitos sea cero.

Las restricciones (2.11)–(2.14) establecen cotas sobre las variables continuas en función de los parámetros y las variables binarias. Las restricciones (2.15) aseguran que cada cliente sea asignado a solo un depósito. Por su parte, las restricciones (2.16) y (2.17) garantizan que las cargas totales de entrega y recogida en cualquier depósito no superen la capacidad correspondiente de ese depósito.

Las restricciones (2.18)-(2.20) evitan las rutas ilegales, es decir, aquellas rutas que no comienzan y terminan en el mismo depósito. Por último, las restricciones (2.21)–(2.24) imponen que las variables sean binarias.

2.3. Segundo modelo matemático (Modelo 2)

Para el segundo modelo matemático se decide eliminar las variables de flujo U_{ij} y V_{ij} y todas las restricciones que contienen estas variables, respecto

del Modelo 1. Añadiendo otras restricciones en su lugar, con la idea de simplificar el problema.

A continuación, se describen las tres variables de decisión, las tres son variables binarias, x_{ij}, y_k y z_{ik}. Sin embargo, se añaden familias con un número exponencial de restricciones. Como notación adicional a este problema, denotamos por $r(S)$ a una cota inferior del número de vehículos que se necesita para visitar $S \subseteq N_C$.

- $x_a = x_{ij} = \begin{cases} 1 & \text{si un vehículo viaja del nodo i al nodo j} \quad \forall i, j \in N \\ 0 & \text{otro caso} \end{cases}$
- $y_k = \begin{cases} 1 & \text{si se abre el depósito k} \quad \forall k \in N_0 \\ 0 & \text{otro caso} \end{cases}$
- $z_{ik} = \begin{cases} 1 & \text{si el cliente i está asignado al depósito k} \quad \forall i \in N_C,\ \forall k \in N_0 \\ 0 & \text{otro caso} \end{cases}$
- $r(S) = \left\lceil \frac{max(\sum_{i \in S} d_i\ ,\ \sum_{i \in S} p_i)}{CV} \right\rceil \quad \forall S \subseteq N_C$

Modelo matemático

$$\text{mín} \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ij} \ + \sum_{k \in N_0} FD_k y_k \ + \sum_{k \in N_0} \sum_{i \in N_C} FV x_{ki} \tag{2.25}$$

sujeto a:

$$\sum_{j \in N} x_{ij} = 1 \qquad \forall i \in N_C \tag{2.26}$$

$$\sum_{j \in N} x_{ji} = \sum_{j \in N} x_{ij} \qquad \forall i \in N \tag{2.27}$$

$$\sum_{j \in N_C} x_{kj} \geq y_k \qquad \forall k \in N_0 \tag{2.28}$$

$$\sum_{k \in N_0} z_{ik} = 1 \qquad \forall i \in N_C \tag{2.29}$$

$$\sum_{i \in N_C} d_i z_{ik} \leqslant C D_k y_k \qquad \forall k \in N_0 \tag{2.30}$$

$$\sum_{i \in N_C} p_i z_{ik} \leqslant C D_k y_k \qquad \forall k \in N_0 \tag{2.31}$$

$$x_{ik} \leqslant z_{ik} \qquad \forall i \in N_C, \forall k \in N_0 \tag{2.32}$$

$$x_{ki} \leqslant z_{ik} \qquad \forall i \in N_C, \forall k \in N_0 \tag{2.33}$$

$$x_{ij} + z_{ik} + \sum_{m \in N_0, m \neq k} z_{jm} \leqslant 2 \qquad \forall i, j \in N_C, i \neq j, \forall k \in N_0 \tag{2.34}$$

$$z_{ik} \leqslant y_k \qquad \forall i \in N_C, k \in N_0 \tag{2.35}$$

$$\sum_{a \in \delta^+(S)} x_a \geq z_{jk} \qquad \forall k \in S \cap N_0, j \in N \setminus S \tag{2.36}$$

$$\sum_{i,j \in S} x_{ij} \leqslant |S| - r(S) \qquad \forall S \subseteq N_C \tag{2.37}$$

$$\sum_{a \in P} x_a \leqslant |P| - 1 \qquad \forall P \text{ camino no factible} \tag{2.38}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in N \tag{2.39}$$

$$z_{ik} \in \{0, 1\} \qquad \forall i \in N_C, \forall k \in N_0 \tag{2.40}$$

$$y_i \in \{0, 1\} \qquad \forall k \in N_0 \tag{2.41}$$

La función objetivo (2.25) y las restricciones (2.26)–(2.35) se describen en el Modelo 1.

Luego, las restricciones (2.36) nos aseguran que todo cliente asignado a un depósito deba tener estar conectado al mismo. Las restricciones (2.37) evitan los subciclos y que se violen las capacidades de los vehículos imponiendo que el número mínimo de vehículos para visitar un subconjunto de clientes S es $r(S)$. Nótese que estas restricciones (2.37), gracias a las ecuaciones (2.26) y (2.27), se pueden reescribir como:

$$\sum_{i \in S, j \in N \setminus S} x_{ij} \geq r(S) \qquad \forall S \subseteq N_C \tag{2.42}$$

Por otro lado, las restricciones (2.38) tienen en cuenta las cantidades d_i y p_i de forma combinada. La figura 2.2 ejemplifica esta restricciones. Se observa que en le primer dibujo el vehículo tiene que salir con 10 unidades de producto (igual a su capacidad máxima) ya que es lo que demandan los 3 clientes que se tienen que visitar. Sin embargo, al llegar al Cliente 1 se entrega 3 unidades del producto demandado y se recoge 4 unidades del producto ofrecido, cosa que no es posible por la capacidad límite del vehículo. En este ejemplo también se aprecia como si recorremos la ruta en sentido contrario esta sí sería factible.

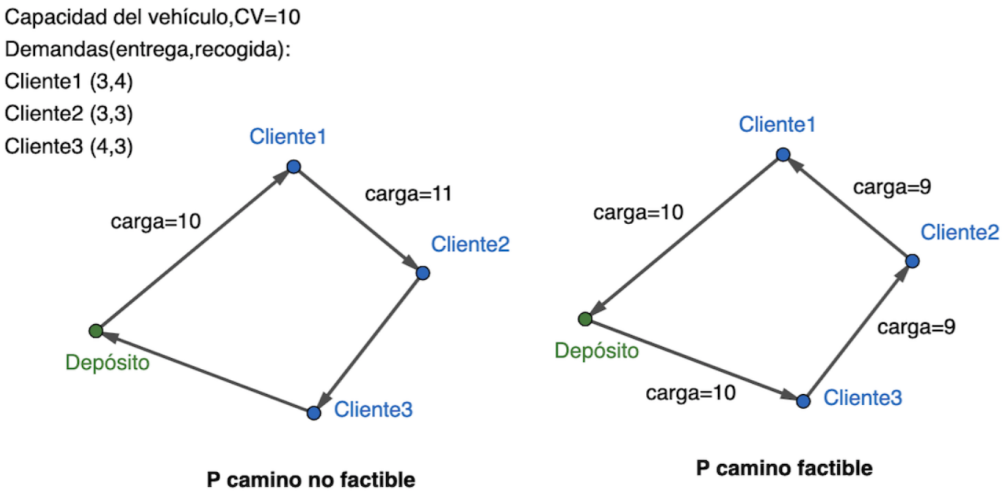

Figura 2.2: Ejemplo de P camino únicamente factible en un sentido

Por último, las restricciones (2.39)–(2.41) imponen que las variables sean binarias.

Como se verá más adelante, se utiliza un método iterativo para resolver el modelo, debido a que hay un número exponencial de restricciones (2.36)-(2.38), de forma que estas se añaden a medida que se necesiten. No obstante, ya que no son muchas y ayuda a acelerar la resolución del modelo, se añaden desde el principio las restricciones (2.37) que verifican que $|S| = 2$, $|S| = 3$ o $S = N_C$.

3

Resultados computacionales

En este último capítulo, utilizando los dos modelos comentados en el capítulo anterior, se presentan los resultados computacionales obtenidos a partir de las instancias de Prodhon [5]. Se muestran en forma de tablas y representaciones gráficas, buscando la solución óptima y minimizar el tiempo de ejecución del programa. Todos los tiempos se expresan en segundos y todos los resultados se han redondeado a dos decimales.

3.1. Instancias de referencia

En primer lugar, generamos instancias a partir de las instancias de Prodhon con 20 clientes y 5 depósitos disponibles. Dado que estas instancias fueron diseñadas para un problema de localización y rutas (LRP), tuvimos que modificarlas siguiendo las especificaciones del artículo mencionado.

Para generar las demandas de entrega y recogida de los clientes en cada instancia de prueba, utilizamos enfoques de separación de demanda propuestos por Salhi y Nagy [6] y Angelelli y Mansini [7]. Estos enfoques se definen de la siguiente manera: en el enfoque de Salhi y Nagy [6], se calcula una proporción $r_i = min(x_i/y_i\ ,\ y_i/x_i)$, donde x_i e y_i son las coordenadas del cliente i, y luego se generan las demandas de entrega y recogida como $d_i = r_i q_i$ y $p_i = q_i d_i$, donde q_i es la demanda original del cliente i para el LRP. Nos referimos a este tipo de problemas como tipo X. De manera similar, se genera otro tipo de problema llamado tipo Y, intercambiando las demandas de entrega y recogida de cada cliente.

En el enfoque de Angelelli y Mansini [7], la demanda original de cada cliente i se considera como la demanda de entrega ($d_i = q_i$), y se genera la demanda

de recogida del cliente correspondiente como $p_i = \lfloor (1-\gamma)q_i \rfloor$ si i es par y $p_i = \lfloor (1+\gamma)q_i \rfloor$ si i es impar. En este artículo, consideramos dos valores de γ, 0.2 y 0.8, para generar dos tipos diferentes de problemas llamados tipo Z y tipo W, respectivamente.

Como resultado, se generaron un total de 16 instancias de LRPSPD a partir de cuatro instacias de Prodhon [5] con 20 clientes y 5 depósitos disponibles, utilizando 4 estrategias de separación diferentes (tipos X, Y, Z y W).

A pesar de que la construcción de estas instancias parece clara, no se especifica si se redondean las demandas u otros parámetros para alguno de estos enfoques. Así en Praxedesa et al. [8] observan que en algunas instancias tienen una valores óptimos ligeramente diferentes a los obtenidos por Karaoglan et al. [1].

3.2. Resultados computacionales

Para empezar, se comentará el software que se ha utilizado en el artículo de Karaoglan et al. [1], con el que se comparan los resultados obtenidos. Y a continuación, se explicará el utilizado en los experimentos de este trabajo.

Todos los experimentos del artículo fueron realizados con un procesador Intel Xeon 3.16 GHz con 1 GB RAM y con un tiempo máximo de ejecución del programa de 4 horas, para cada instancia. Los algoritmos fueron implementados en C++, y se utilizó el solucionador CPLEX.

Por otro lado, para los experimentos se realizados en este trabajo se utilizó un procesador 11th Gen Intel Core i5 2.5 GHz con 16 GB RAM, y también con un tiempo máximo de ejecución del programa de 4 horas, para cada instancia. Los algoritmos fueron implementados en Python, versión 3.9 (64-bit), y se utilizó el solucionador Gurobi 10.0.

En la tabla 3.1, se comparan directamente los resultados y el tiempo de ejecución de los resultados obtenidos en los experimentos de este trabajo y el artículo de Karaoglan et al. [1], utilizando en ambos casos el modelo propuesto por el artículo (Modelo 1), pero con diferente solucionador matemático (y máquinas diferentes).

Cuando examinamos la tabla 3.1, observamos que, de las 16 instancias consideradas, las 16 fueron óptimamente resueltas por los algoritmos con Gurobi.

		Gurobi			CPLEX en [1]		
Fichero	Tipo	Objetivo	Gap	Tiempo	Objetivo	Gap	Tiempo
coord20-5-1	W	26464.67	0.00	**339.97**	26457.70	0.94	14400.00
	Z	26456.87	0.00	**3282.99**	26461.30	1.41	14400.00
	X	16820.34	0.00	**150.18**	16816.50	4.45	14400.00
	Y	16820.34	0.00	**113.20**	16816.00	4.41	14400.00
coord20-5-1b	W	18704.44	0.00	**140.02**	18718.80	5.02	14400.00
	Z	18702.96	0.00	**16.50**	18703.00	4.87	14400.00
	X	9167.15	0.00	1.10	9167.14	0.00	**0.65**
	Y	9167.15	0.00	1.11	9167.14	0.00	**0.46**
coord20-5-2	W	27988.31	0.00	**495.60**	27988.70	1.90	14400.00
	Z	27980.64	0.00	**612.43**	27980.60	1.69	14400.00
	X	17808.17	0.00	**140.42**	17814.70	1.95	14400.00
	Y	17808.17	0.00	**323.10**	17814.70	1.94	14400.00
coord20-5-2b	W	17122.20	0.00	**35.60**	17125.50	5.21	14400.00
	Z	17117.18	0.00	**26.90**	17120.50	5.53	14400.00
	X	10257.34	0.00	**3.98**	10257.30	0.00	22.70
	Y	10257.34	0.00	**5.45**	10257.30	0.00	29.25
Promedio			0.00	355.53		2.46	10803.32

Tabla 3.1: Soluciones al Modelo 1

Por otro lado, vemos que los únicamente 4 instancias fueron resueltas óptimamente por los algoritmos con CPLEX, por lo que 12 instancias quedan sin resolver después de 4 horas de ejecución. Para estos casos, la brecha porcentual final (*Gap*) es en promedio $2,46\,\%$. El *Gap* se refiere a la diferencia relativa entre la mejor cota superior y la mejor cota inferior dadas por el solucionador en cada caso (CPLEX o Gurobi). Se utiliza para evaluar y medir la calidad de las soluciones obtenidas en algoritmos de optimización y se calcula:

$$Gap = \frac{(\text{Cota superior} - \text{Cota inferior})}{\text{Cota inferior}} \times 100\,\% \tag{3.1}$$

También, se debe comentar que en el artículo [1], se obtienen resultados en un menor periodo de tiempo, gracias a la utilización un algoritmo de ramificación y corte (*branch and cut*) en el que se insertan numerosas desigualdades válidas. Este es un enfoque utilizado en la optimización combinatoria para resolver problemas de programación lineal entera (PLE) de manera eficiente.

En la tabla 3.2, se comparan directamente los resultados y el tiempo de ejecución de los resultados obtenidos en los experimentos de este trabajo, utilizando en ambos casos el solucionador Gurobi, pero comparando los dos modelos mostrados en este trabajo.

		Modelo 1		Modelo 2		
Fichero	Tipo	Objetivo	Tiempo	Objetivo	Tiempo	Iter
coord20-5-1	W	26464.67	**339.97**	26419.54	14400.17	37
	Z	26456.87	**3282.99**	26413.88	14400.18	40
	X	16820.34	**150.18**	16820.34	11602.04	213
	Y	16820.34	**113.20**	16820.34	12433.26	209
coord20-5-1b	W	18704.44	140.02	18704.44	**28.92**	8
	Z	18702.96	16.50	18702.96	**13.84**	4
	X	9167.15	**1.10**	9167.15	4.63	1
	Y	9167.15	**1.11**	9167.15	7.99	2
coord20-5-2	W	27988.31	**495.60**	27988.31	5032.57	46
	Z	27980.64	**612.43**	27980.64	2644.49	21
	X	17808.17	**140.42**	17808.17	620.95	30
	Y	17808.17	**323.10**	17808.17	531.73	28
coord20-5-2b	W	17122.20	35.60	17122.20	**32.58**	6
	Z	17117.18	26.90	17117.18	**15.29**	3
	X	10257.34	**3.98**	10257.34	30.08	6
	Y	10257.34	**5.45**	10257.34	32.11	6
Promedio			355.53		3987.35	41.25

Tabla 3.2: Comparación de tiempos del Modelo 1 y del Modelo 2

Analizando la tabla 3.2, como hemos visto anteriormente, de las 16 instancias consideradas, las 16 fueron óptimamente resueltas por Gurobi utilizando el Modelo 1 (propuesto propuesto por Karaoglan et al. [1]). Mientras que, al utilizar el Modelo 2 de forma iterativa, se resuelven 14 de las 16 instancias consideradas. En promedio, el Modelo 1 es más rápido, sobre todo debido a que en dos instacias el Modelo 2 llega al tiempo límite de 4 horas. Sin embargo, en cuatro de las instancias, los ficheros coord20-5-1b y coord20-5-2b con las construcciones para las demandas W y Z, se resuelven más rápido con esta segunda opción.

Otra observación que vemos es que las instancias construidas utilizando el enfoque de de Angelelli y Mansini [7] (instancias X e Y) son más díficiles de resolver que las construidas por el enfoque de Salhi y Nagy [6] (instancias W y Z).

3.3. Ilustraciones de las soluciones

En esta sección, se representaran algunas de las soluciones de las instancias generadas. Se mostrará una solución de uno de los enfoques de Angelelli y Mansini [7] y uno de los enfoques de Salhi y Nagy [6], para cada uno de los ficheros utilizados. Por último, se presentará alguna iteración que sigue el algoritmo de ejecución del Modelo 2.

En las Figuras 3.1–3.8 se observan las rutas de vehículos a seguir, que depósitos han de ser abiertos y sumando el número de rutas, se tiene el número de vehículos necesarios para la solución de los problemas de tipo W y X.

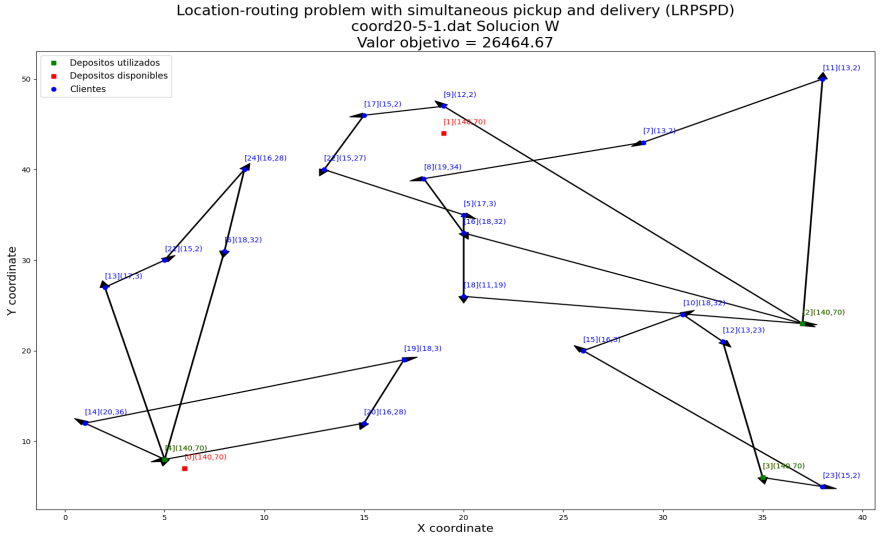

Figura 3.1: coord20-5-1

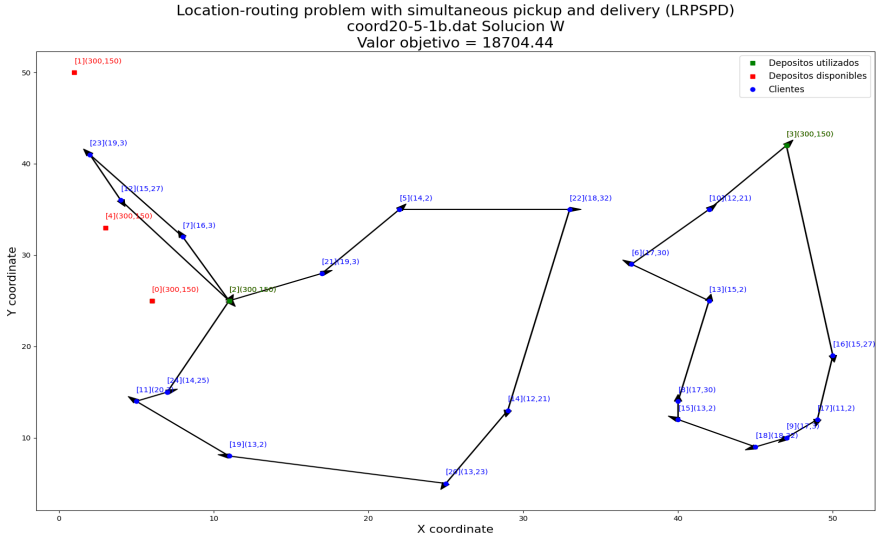

Figura 3.2: coord20-5-1b

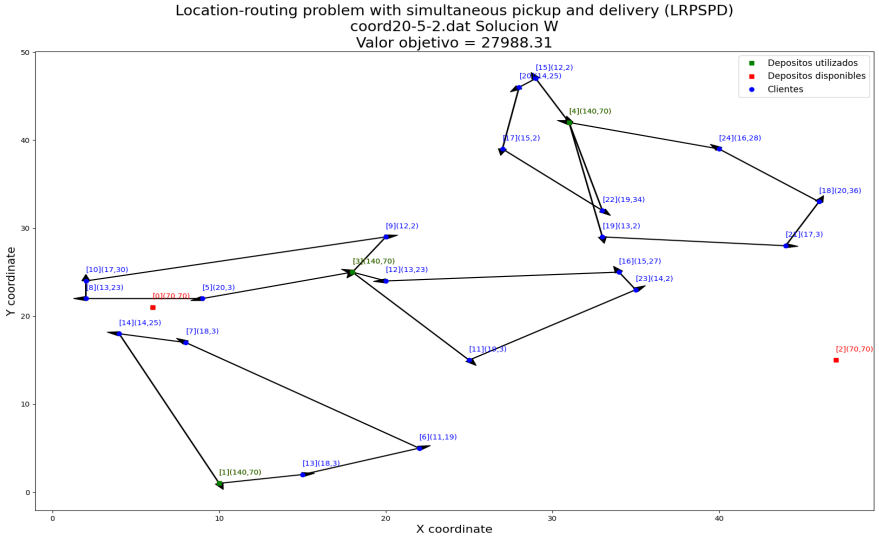

Figura 3.3: coord20-5-2

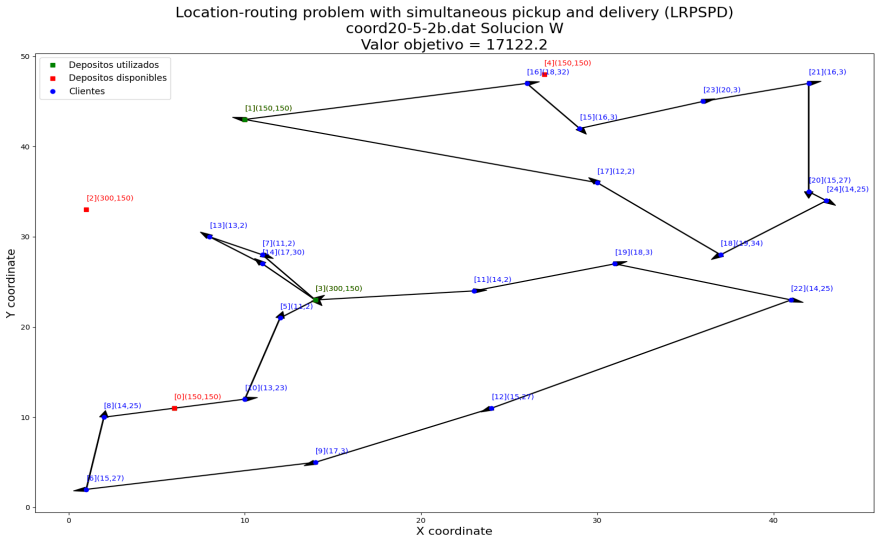

Figura 3.4: coord20-5-2b

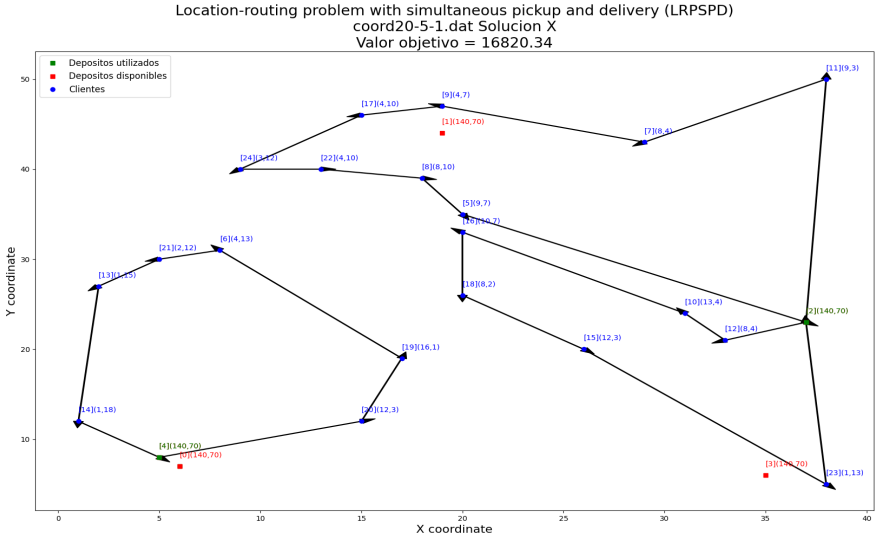

Figura 3.5: coord20-5-1

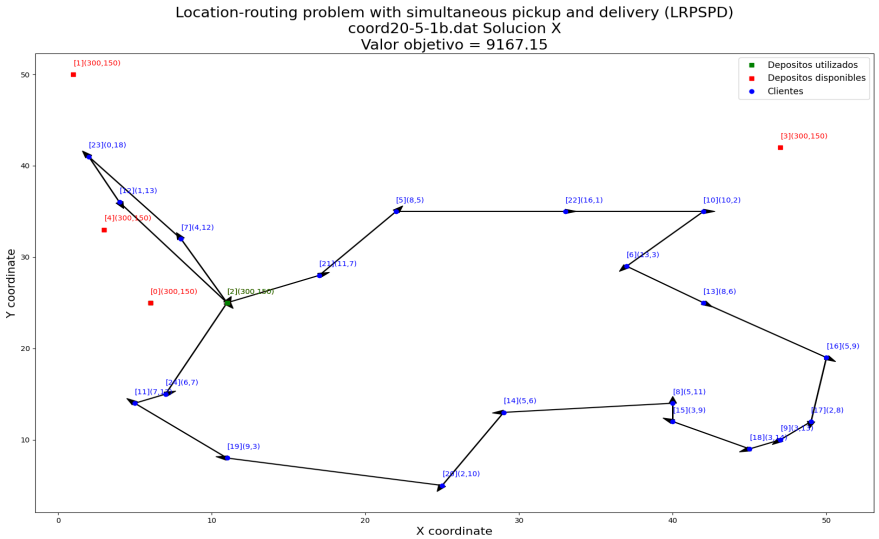

Figura 3.6: coord20-5-1b

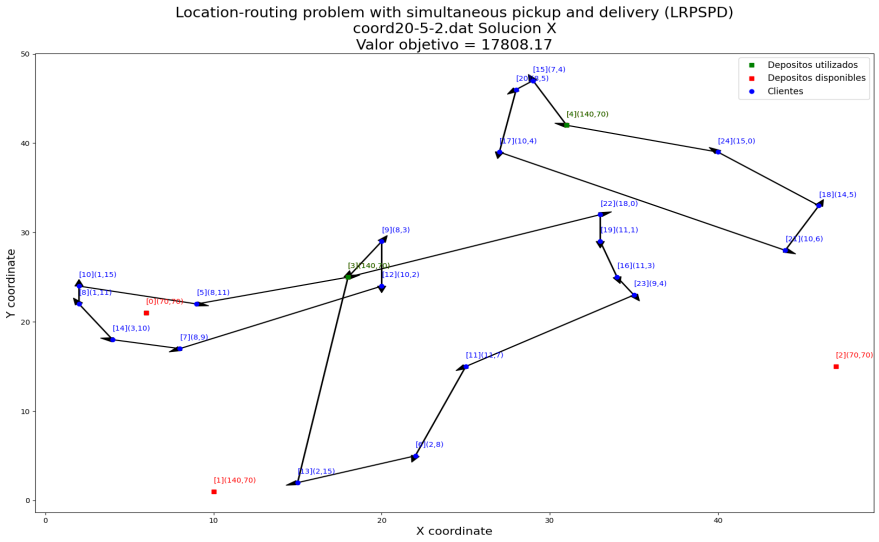

Figura 3.7: coord20-5-2

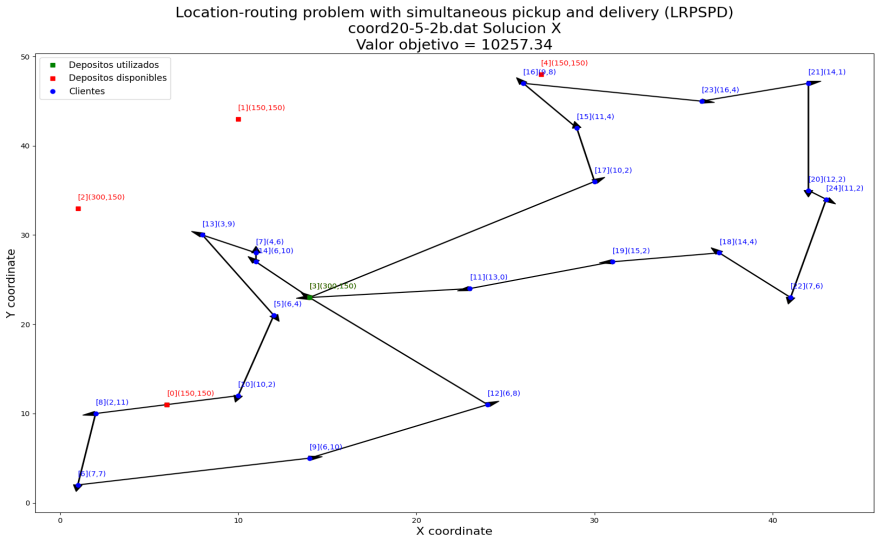

Figura 3.8: coord20-5-2b

En las figuras 3.9, 3.10, 3.11 y 3.12 se muestra el proceso de iteración seguido para la resolución de la instancia generada a partir de coord20-5-1b con el enfoque Z utilizando el Modelo 2.

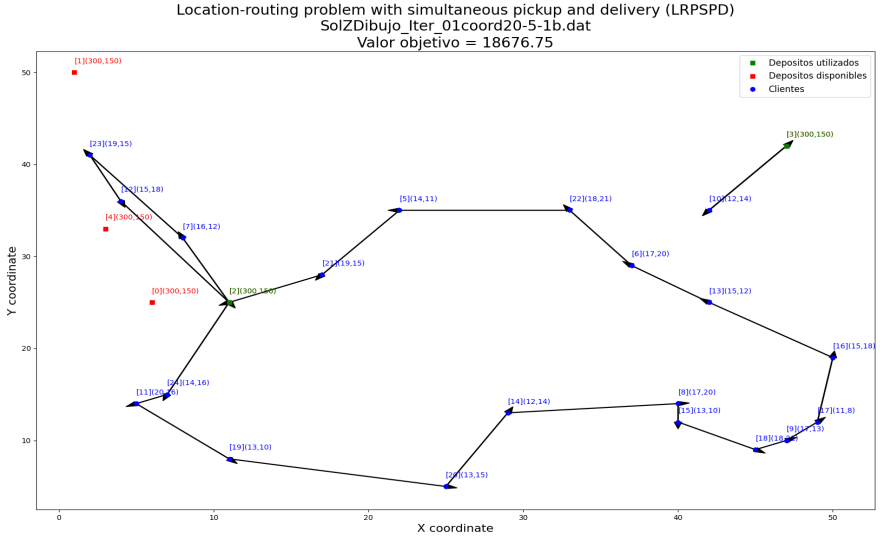

Figura 3.9: Iteración 1

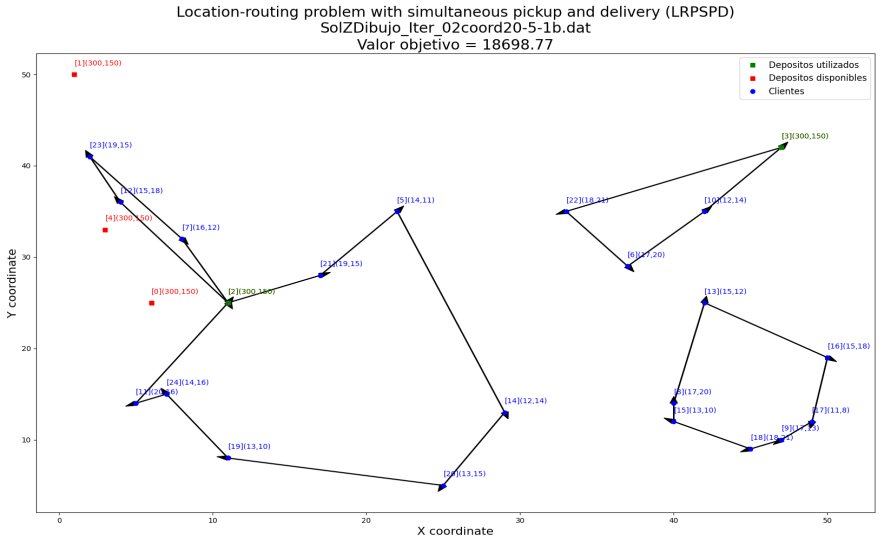

Figura 3.10: Iteración 2

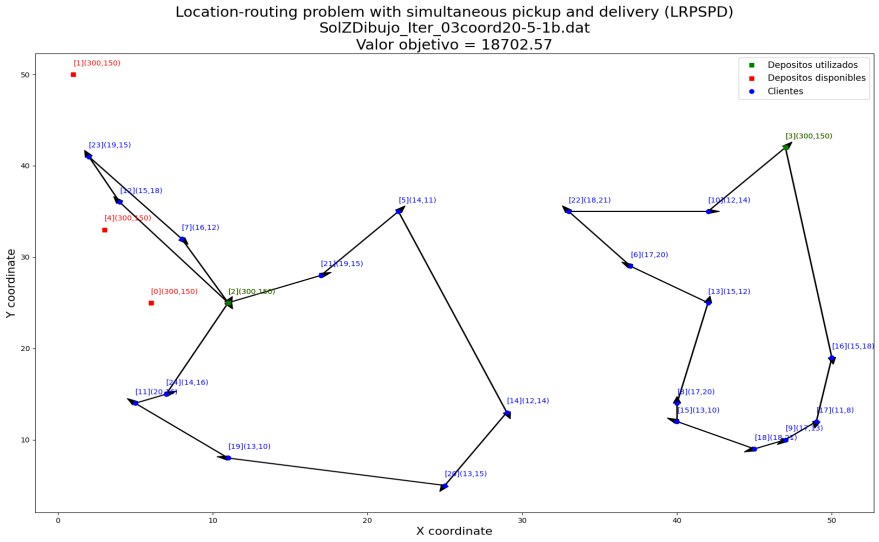

Figura 3.11: Iteración 3

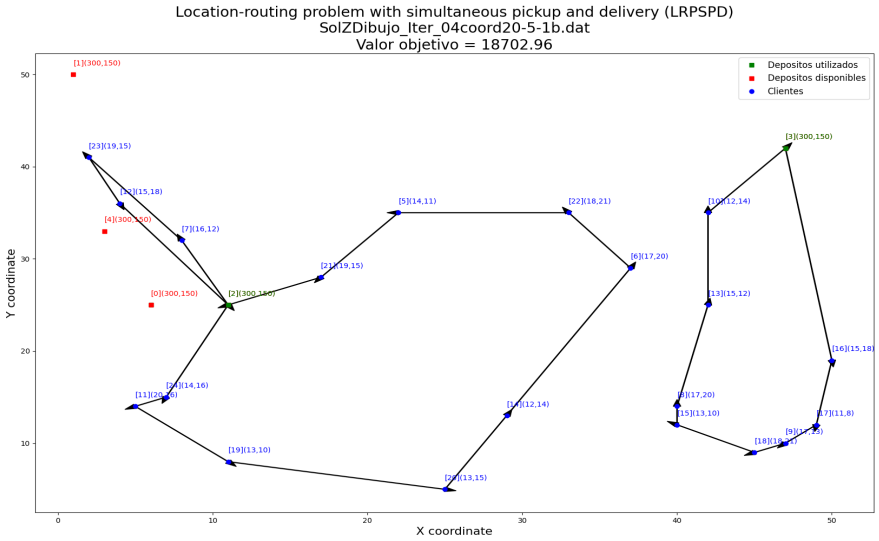

Figura 3.12: Iteración 4

La primera iteración 3.9 muestra una solución no factible, ya que para poder suministrarle la demanda de entrega a todos los clientes, del camino que sale del nodo 2 hacia el nodo 24, se necesitaría 235 unidades de producto, pero los vehículos únicamente tienen una capacidad de 150 unidades. Así, pasamos a la iteración 3.10, la cuál se puede ver que tiene un subciclo, por lo que tampoco es factible. Esta nos lleva a la iteración 3.11, donde volvemos a observar un camino, que sale del nodo 3, cuya suma de la demanda de entrega de todos los clientes es de 153 unidades de producto, así que sobrepasa la capacidad del vehículo. Por último, la iteración 3.12 nos muestra la solución óptima del problema, cuyo valor objetivo es $18702,96$ unidades de costo total. También, podemos observar que a medida que pasamos de iteración el valor objetivo aumenta, hasta llegar a la solución óptima.

Conclusiones

Este trabajo describe el problema que consiste en determinar la utilización óptima de unos depósitos disponibles, asignar unos clientes a los depósitos que se utilizan y encontrar las rutas de los vehículos correspondientes, de forma que se satisfagan las restricciones de capacidad de los depósitos y los vehículos y se minimice el costo total. Además describe dos modelos matemáticos para este problema denominado problema de localización y ruta con recogida y entrega simultánea de producto. También se muestran resultados computacionales.

De los dos modelos, uno de ellos se ha propuesto como novedad en este trabajo, llamado Modelo 2. A través de diversos experimentos realizados se han comparado estos dos modelos y, adicionalmente, se han comparado los resultados del Modelo 1, con los ya existente en la literatura de Karaoglan et al [1].

Los resultados computacionales reflejan que en promedio el Modelo 1 es mas eficiente. Aunque el Modelo 2 resuelve más rápido cuatro instancias.

Como trabajos de ampliación a esta memoria, sería interesante, ver algún otro modelo o algunas restricciones que mejoren los tiempos de ejecución. También se podrían aprovechar las funcionalidad de algunos solucionadores (como CPLEX o Gurobi) para insertar restricciones de forma dinámica sin necesidad de que la solución sea entera.

Creo que es un problema relativamente joven, para el cual todavía hay mejoría y tiene una gran aplicación en el mundo real.

A

Apéndice

En este último apartado, se muestra la implementación de código que se ha utilizado para obtener los resultados de los experimentos de este trabajo.

A.1. Código

A.1.1. Paquetes y función lectura de ficheros

```
import math
import time
import numpy as np
import matplotlib.pyplot as plt

import gurobipy as gp
from gurobipy import GRB
```

Listing A.1: Librerías utilizadas

```
def lectura_fichero(nombre_fichero):
  lista=[]
  lista_limpia=[]
  f = open(nombre_fichero, 'r')
  lines = f.readlines() # Lee linea a linea
  for i in lines:
    lista.append(i)
  for k in range(0,len(lista)):
    limpiar=lista[k].strip('\n').split()
    if limpiar != []:
      lista_limpia.append(limpiar)

  # Imprimimos por pantalla nuestra lista con todos los datos del
    fichero
  if(DISPLAY_LEVEL > 1) :
    print('Lista sin espacios:')
```

```
16      print(lista_limpia)
17
18    # Declaramos las variables que vamos a usar en el resto del problema
        como globales
19    global nC  # Numero de clientes
20    global nDD # Numero de depositos disponibles
21    global coord_DD # Coordenadas de los nodos de los depositos
        disponibles
22    global coord_C # Coordenadas de los nodos de los clientes
23    global coord_T # Coordenadas globales
24    global N # Conjunto de todos los nodos
25    global NDD # Conjunto de todos los nodos de los depositos
26    global NC # Conjunto de todos los nodos de los clientes
27    global CV # Capacidad del vehiculo
28    global CD # Capacidad de cada deposito
29    global DC # Demanda de entrega de cada cliente
30    global CA  # Costo de apertura de cada deposito
31    global CR   # Costo de apertura de una ruta
32    global cost # costo de recorrer cada arco
33    global DP # Demanda de recogida de cada cliente
34
35    # De las dos primeras lineas, tenemos:
36    nC = int(lista_limpia[0][0])
37    nDD = int(lista_limpia[1][0])
38    # Imprimimos por pantalla para comprobar si los datos se han guardado
         correctamente
39    # print('\n')
40    #print('El numero de clientes es', nC)
41    #print('El numero de depositos disponibles es', nDD)
42
43    # Guardamos las coordenadas de los nodos de los depositos disponibles
         en una matrix de tamano (nDD x 2)
44    coord_DD = np.zeros((nDD,2))
45    first_line = 2
46    for i in range(nDD) :
47      coord_DD[i][0] = float(lista_limpia[first_line + i][0])
48      coord_DD[i][1] = float(lista_limpia[first_line + i][1])
49    # print('\n')
50    if(DISPLAY_LEVEL > 0) :
51      print('Las coordenadas de los', nDD, 'nodos para los depositos
        disponibles son: \n', coord_DD)
52
53    # Guardamos las coordenadas de los nodos de los clientes en una
        matrix de tamano (nC x 2)
54    coord_C = np.zeros((nC,2))
55    first_line = 2+nDD
56    for i in range(nC) :
57      coord_C[i][0] = float(lista_limpia[first_line + i][0])
58      coord_C[i][1] = float(lista_limpia[first_line + i][1])
59    # print('\n')
60    if(DISPLAY_LEVEL > 0) :
61      print('Las coordenadas de los', nC, 'nodos para los clientes son: \
        n', coord_C)
62
63    # print('Conjunto total de nodos')
```

```
64  N=range(nDD+nC)
65  NDD=range(nDD)
66  NC=range(nDD,nDD+nC)
67  coord_T = np.append(coord_DD ,coord_C, axis=0)
68  if(DISPLAY_LEVEL > 1) :
69    print("Cordenadas de todos los puntos \n",coord_T)
70    print("NC", NC)
71    print("NDD", NDD)
72
73  # De las lineas posteriores a las coordenadas, tenemos:
74  # Capacidad de los vehiculos
75  first_line = 2+nDD+nC
76  CV = int(lista_limpia[first_line][0])
77  if(DISPLAY_LEVEL > 0) :
78    print('La capacidad de los vehiculos es:')
79    print(CV)
80
81  # Capacidad de cada deposito
82  CD = []
83  first_line = 2+nDD+nC+1
84  for i in range(nDD):
85    CD.append(int(lista_limpia[first_line + i][0]))
86  if(DISPLAY_LEVEL > 0) :
87    print(CD)
88
89  # Costo de aperturas de cada deposito
90  CA = []
91  first_line = 2+nDD+nC+1+nDD+nC
92  for i in range(nDD):
93    CA.append(int(lista_limpia[first_line + i][0]))
94  if(DISPLAY_LEVEL > 0) :
95    print(CA)
96
97  # Costo de apertura de una ruta
98  first_line = 2+nDD+nC+1+nDD+nC+nDD
99  CR = int(lista_limpia[first_line][0])
100 if(DISPLAY_LEVEL > 0) :
101   print(CR)
102
103 cost =  np.zeros((nDD+nC,nDD+nC))
104 # Matriz de costos
105 for i in N :
106   for j in N:
107     if j != i:
108       dx = coord_T[i][0] - coord_T[j][0]
109       dy = coord_T[i][1] - coord_T[j][1]
110       # The cost is equal to the rounded Euclidean distance
111       cost[i,j] = math.sqrt(dx*dx + dy*dy)
112 if(DISPLAY_LEVEL > 1) :
113   print('Costos: ', cost, '\n')
114
115 if eleccion == 0:
116     # Demanda de cada cliente
117     DC = []
118     first_line = 2+nDD+nC+1+nDD
```

```
      for i in range(nC):
        DC.append(int(lista_limpia[first_line + i][0]))
      if(DISPLAY_LEVEL > 1) :
        print('La demanda de entrega de cada cliente es:')
        print(DC)

      # Demanda de recogida
      if tipo == 1:
          # Para solucion Z
          gamma = 0.2
          DP = np.zeros(nC, int)
          for i in range(nC):
            if i%2==0:
              DP[i]=math.floor((1-gamma)*DC[i])
            else:
              DP[i]=math.floor((1+gamma)*DC[i])
          #print("Demanda de recogida:",DP)
          #print("CV=",CV)

      # Para solucion W
      else:
          gamma = 0.8
          DP = np.zeros(nC, int)
          for i in range(nC):
            if i%2==0:
              DP[i]=math.floor((1-gamma)*DC[i])
            else:
              DP[i]=math.floor((1+gamma)*DC[i])
          #print("Demanda de recogida:",DP)
          #print("CV=",CV)

  if eleccion == 1:
  # Para solucion X
    r = []
    for i in range(nC):
      r.append(min(coord_C[i][0]/coord_C[i][1], coord_C[i][1]/coord_C[i
    ][0]))
    # Demanda de cada cliente
    qi = []
    DC = []
    first_line = 2+nDD+nC+1+nDD
    for i in range(nC):
      qi.append(int(lista_limpia[first_line + i][0]))
    for i in range(nC):
      DC.append(round(r[i]*qi[i],2))
    if(DISPLAY_LEVEL > 0) :
      print('La demanda de entrega de cada cliente es:')
      print(DC, qi)
    # Demanda de recogida
    DP = np.zeros(nC)
    for i in range(nC):
      DP[i] = qi[i]-DC[i]
    #print("Demanda de recogida:",DP)


```

```
173     # Para solucion Y
174     if tipo!=0:
175       s = DP
176       DP = DC
177       DC = s
```

Listing A.2: Función lectura de ficheros

A.1.2. Función para representar la solución

```
1  def dibuja(selected, abrir, nombre_fichero,obj):
2    # Se crea la figura
3    plt.figure(figsize=(20,12))
4    plt.xlabel("X coordinate", fontsize='16')
5    plt.ylabel("Y coordinate", fontsize='16')
6    plt.title('\n Location-routing problem with simultaneous pickup and
       delivery (LRPSPD) \n' + nombre_fichero + '\n' + 'Valor objetivo = '
       + str(obj), fontsize='20')
7
8    # Se dibujan los depositos en rojo y los clientes en azul
9    plt.plot([coord_DD[0][0]], [coord_DD[0][1]], 'gs', label='Depositos
       utilizados')
10   plt.plot([p[0] for p in coord_DD], [p[1] for p in coord_DD], 'rs',
       label='Depositos disponibles')
11   plt.plot([p[0] for p in coord_C], [p[1] for p in coord_C], 'bo',
       label='Clientes')
12
13   for i in NDD :
14     plt.annotate('[%i](%i,%i)'%(i,CD[i],CV), (coord_T[i][0], coord_T[i
       ][1] +1), fontsize = 10, color='red')
15   for i in NC :
16     plt.annotate('[%i](%i,%i)'%(i,DC[i-nDD], DP[i-nDD]), (coord_T[i
       ][0], coord_T[i][1] + 1), fontsize = 10, color='blue')
17   for (i,j) in selected:
18     plt.arrow(coord_T[i][0],coord_T[i][1], coord_T[j][0]-coord_T[i][0],
       coord_T[j][1]-coord_T[i][1], head_width=0.5,width=0.05, facecolor='k
       ')
19   for k in abrir:
20      plt.plot([coord_DD[k][0]], [coord_DD[k][1]], 'gs')
21      plt.annotate('[%i](%i,%i)'%(k,CD[k],CV), (coord_DD[k][0], coord_DD
       [k][1] +1), fontsize = 10, color='g')
22
23   plt.legend( loc='best', fontsize=12)
24   plt.savefig( nombre_fichero + '.png')
25
26 # dibuja({},{})  #Nos da la distribucion de puntos inicial
```

Listing A.3: Función imagen de la solución

A.1.3. Función Modelo 1

```
# Crear el modelo m
m = gp.Model('LRPSPD.lp')
# Se pone un tiempo limite de ejecucion
m.setParam('TimeLimit', TIME_LIMIT)


def modelo1():
  start_time = time.time()
  # Variables de decision
  # Variables binarias
  x = m.addVars(((i,j) for i in N for j in N if i != j), vtype=GRB.
    BINARY, name='x')
  y = m.addVars(((k) for k in NDD ), vtype=GRB.BINARY, name='y')
  z = m.addVars(((i,k) for i in NC for k in NDD) , vtype=GRB.BINARY,
    name='z')
  # Variables continuas no negativas
  u = m.addVars(((i,j) for i in N for j in N if i != j), vtype=GRB.
    CONTINUOUS, name='u')
  v = m.addVars(((i,j) for i in N for j in N if i != j), vtype=GRB.
    CONTINUOUS, name='v')

  # Funcion objetivo: minimizar los costes totales
  m.setObjective(gp.quicksum(cost[i,j]*x[i,j] for i in N for j in N if
    i!=j) + gp.quicksum( CA[k]*y[k] for k in NDD) + gp.quicksum(CR*x[k,i
    ] for k in NDD for i in NC), GRB.MINIMIZE)

  # Condiciones
  m.addConstrs((x.sum(i,'*') == 1 for i in NC ))
  m.addConstrs((x.sum('*',i) == x.sum(i,'*') for i in N ))
  m.addConstrs((u.sum('*',i) - u.sum(i,'*') == DC[i-nDD] for i in NC ))
  m.addConstrs((v.sum(i,'*') - v.sum('*',i) == DP[i-nDD] for i in NC ))
  m.addConstrs((u[i,j] + v[i,j] <= CV*x[i,j] for i in N for j in N if i
     != j))
  m.addConstrs((u.sum(k,'*') == gp.quicksum(z[j,k]*DC[j-nDD] for j in
    NC) for k in NDD ))
  m.addConstrs((u.sum('*',k) == 0 for k in NDD ))
  m.addConstrs((v.sum('*',k) == gp.quicksum(z[j,k]*DP[j-nDD] for j in
    NC) for k in NDD ))
  m.addConstrs((v.sum(k,'*') == 0 for k in NDD ))
  m.addConstrs((u[i,j] <= (CV-DC[i-nDD])*x[i,j] for i in NC for j in N
    if i != j)) #poner que son distintos lo veo necesario
  m.addConstrs((v[i,j] <= (CV-DP[j-nDD])*x[i,j] for i in N for j in NC
    if i != j))
  m.addConstrs((u[i,j] >= DC[j-nDD]*x[i,j] for i in N for j in NC if i
    != j))
  m.addConstrs((v[i,j] >= DP[i-nDD]*x[i,j] for i in NC for j in N if i
    != j))
  m.addConstrs((z.sum(i,'*') == 1 for i in NC ))
  m.addConstrs((gp.quicksum(DC[i-nDD]*z[i,k] for i in NC) <= CD[k]*y[k]
     for k in NDD ))
  m.addConstrs((gp.quicksum(DP[i-nDD]*z[i,k] for i in NC) <= CD[k]*y[k]
     for k in NDD ))
  m.addConstrs((x[i,k] <= z[i,k] for i in NC for k in NDD ))
```

```
  m.addConstrs((x[k,i] <= z[i,k] for i in NC for k in NDD  ))
  m.addConstrs((x[i,j] + z[i,k] + gp.quicksum(z[j,m] for m in NDD if m
    !=k) <= 2 for i in NC for j in NC if i!=j for k in NDD ))
  m.addConstrs((u[i,j] >= 0 for i in N for j in N if i != j))
  m.addConstrs((v[i,j] >= 0 for i in N for j in N if i != j))
  m.addConstrs((x[i,j] + x[j,i] <= 1 for i in NC for j in NC if i<j ))

  # Establecer el parametro OutputFlag en 0 para evitar que se impriman
     todos los calculos
  m.setParam("OutputFlag", 0)

  m.optimize()

  end_time = time.time()
  duration = round(end_time - start_time, 2)

  print('\nEl valor objetivo obtenido es: ', round(m.ObjVal,2))
  print('Duracion: ', duration, 'segundos. \n')

  global selected
  selected = [i for i, z in x.items() if z.X > Eps]
  if(DISPLAY_LEVEL > 1) :
    print('Los arcos utilizados para los proveedores son:')
    print(selected)
  global abrir
  abrir = [j for j, w in y.items() if w.X > Eps]
  if(DISPLAY_LEVEL > 1) :
    print('Los depositos abiertos son:')
    print(abrir)
```

Listing A.4: Código Modelo 1

A.1.4. Función Modelo 2

```
def modelo2():
  start_time = time.time()
  # variables de decision
  # Variables binarias
  x = m.addVars(((i,j) for i in N for j in N if i != j), vtype=GRB.
    BINARY, name='x')
  y = m.addVars(((k) for k in NDD ), vtype=GRB.BINARY, name='y')
  z = m.addVars(((i,k) for i in NC for k in NDD) , vtype=GRB.BINARY,
    name='z')

  # Funcion objetivo: minimizar los costes totales
  m.setObjective(gp.quicksum(cost[i,j]*x[i,j] for i in N for j in N if
    i!=j) + gp.quicksum(CA[k]*y[k] for k in NDD) + gp.quicksum(CR*x[k,i]
     for k in NDD for i in NC), GRB.MINIMIZE)

  # Condiciones
  m.addConstrs(x.sum(i,'*') == 1 for i in NC )
  m.addConstrs(x.sum('*',i) == x.sum(i,'*') for i in N )
```

```
m.addConstrs(gp.quicksum(x[k,j] for j in NC) >= y[k] for k in NDD)
m.addConstrs(z.sum(i,'*') == 1 for i in NC )
m.addConstrs(gp.quicksum(DC[i-nDD]*z[i,k] for i in NC) <= CD[k]*y[k]
  for k in NDD )
m.addConstrs(gp.quicksum(DP[i-nDD]*z[i,k] for i in NC) <= CD[k]*y[k]
  for k in NDD )
m.addConstrs(x[i,k] <= z[i,k] for i in NC for k in NDD )
m.addConstrs(x[k,i] <= z[i,k] for i in NC for k in NDD  )
m.addConstrs(x[i,j] + z[i,k] + gp.quicksum(z[j,m] for m in NDD if m!=
  k) <= 2 for i in NC for j in NC if i!=j for k in NDD )
m.addConstrs(z[i,k] <= y[k] for i in NC for k in NDD )
# Eliminamos los subciclos de tamano 2
m.addConstrs(x[i,j] + x[j,i] <= 1 for i in NC for j in NC if i<j)
# Eliminamos los subclos de tamano 3 de nodos que estan proximos
cost_medio = sum(sum(cost)) / (nC+nDD) / (nC+nDD-1)
m.addConstrs(x[i,j] + x[j,i] + x[i,k] + x[k,i] + x[j,k] + x[k,j] <= 2
          for i in NC for j in NC for k in NC if i<j if j<k )
          # if (cost[i,j] + cost[j,i] + cost[i,k] + cost[k,i] + cost[j
  ,k] + cost[k,j]) < 2 *costo_medio )
# Insertamos una restriccion sobre el num minimo de vehiculos
# o equivalente el minimo numero de arcos que salen de los depositos
sum_DC = sum(DC)
sum_DP = sum(DP)
min_num_veh = math.ceil(max(sum_DC,sum_DP)/CV)
m.addConstr(gp.quicksum(x[k,j] for j in NC for k in NDD) >=
  min_num_veh)
# Insertamos unas cotas inferiores al numero de arcos que salen de un
   deposito, pero va mas lento
#m.addConstrs(gp.quicksum(x[k,j] for j in NC)
#             >= gp.quicksum( (DC[i-nDD]/CV) * z[i,k] for i in NC)
  for k in NDD)
#m.addConstrs(gp.quicksum(x[k,j] for j in NC)
#             >= gp.quicksum( (DP[i-nDD]/CV) * z[i,k] for i in NC)
  for k in NDD)

encontrado = True
iteraciones=0
while encontrado:
  # Establecer el parametro OutputFlag en 0 para evitar que se
  impriman todos los calculos
  m.setParam("OutputFlag", 0)
  iteraciones+=1
  medio = time.time()
  tiempo = round((medio-start_time), 2)
  if (TIME_LIMIT - tiempo < 1.0):
    break
  m.setParam('TimeLimit', TIME_LIMIT - tiempo)

  m.optimize()

  if(DISPLAY_LEVEL > 0) :
    print('Iter: ', iteraciones, 'Time: ', tiempo, 'Obj. vale = ',
  round(m.ObjVal,2))
    selected2 = [i for i, z in x.items() if z.X > Eps]
    print('Los arcos utilizados para los proveedores son:')
```

```
60        print(selected2)
61        abrir2 = [j for j, w in y.items() if w.X > Eps]
62        print('Los depositos abiertos son:')
63        print(abrir2)
64        if iteraciones < 10 :
65          nombre_figura = 'Dibujo_Iter_0' + str(iteraciones)
66        else :
67          nombre_figura = 'Dibujo_Iter_' + str(iteraciones)
68        dibuja(selected2, abrir2, 'Sol' + letra + nombre_figura +
      nombre_fichero,round(m.ObjVal,2))
69
70      observados = np.zeros(nC+nDD, int)
71      next = np.zeros(nC+nDD, int)
72      for i in N:
73        for j in N:
74          if i!=j:
75            if x[i,j].X > Eps:
76              next[i]=j
77      component = np.zeros(nC+nDD, int)
78      for i in NC :
79        component[i] = -1
80      for k in NDD :
81        component[k] = k
82        if y[k].X > Eps :
83          for j in NC :
84            if x[k,j].X > Eps:
85              siguiente = j
86              while siguiente != k :
87                component[siguiente] = k
88                siguiente = next[siguiente]
89      encontrado = False
90
91      # Busca restricciones del tipo x(delta(S) >= z[j,k])
92      for k in NDD :
93        if y[k].X > Eps :
94          for j in NC :
95            if z[j,k].X > Eps :
96              if component[j] != k :
97                m.addConstrs((gp.quicksum(x[i,j] for i in N for j in NC
      if component[i]==k if component[j] != k)
98                              >= z[jp,k] for jp in NC if component[jp] !=
       k   ))
99                encontrado = True
100               break
101     for i in NC:
102       if observados[i]==1:
103           continue
104       tour = np.zeros(nC+nDD, int)
105       tour[0]=i
106       seguir = True
107       pos = 1
108       anterior = i
109       dem1 = DC[i-nDD]
110       dem2 = DP[i-nDD]
111       while seguir:
```

```
112         siguiente = next[anterior]
113         if siguiente < nDD:
114           break
115         tour[pos] = siguiente
116         dem1 += DC[siguiente-nDD]
117         dem2 += DP[siguiente-nDD]
118         # Restricciones de subciclos
119         if siguiente == i:
120           m.addConstr((gp.quicksum(x[tour[p1],tour[p2]] for p1 in range
    (0,pos) for p2 in range(0,pos) if p1!=p2)  <= pos-1   ))
121           for j in range(pos):
122             observados[tour[j]]=1
123           encontrado = True
124           break
125         # Restricciones de capacidad
126         elif dem1 > CV or dem2 > CV :
127           m.addConstr((gp.quicksum(x[tour[p1],tour[p2]] for p1 in range
    (0,pos+1) for p2 in range(0,pos+1) if p1!=p2)  <= pos-1   ))
128           encontrado = True
129           break
130         pos+=1
131         anterior=siguiente
132
133     if encontrado==False:
134       for i in NC:
135         if observados[i]==1:
136             continue
137         tour = np.zeros(nC+nDD, int)
138         tour[0]=i
139         seguir = True
140         pos = 1
141         anterior = i
142         dem1 = DC[i-nDD]
143         dem2 = DP[i-nDD]
144         load = dem1 - dem2
145         minload=min(0,load)
146         camino_encontrado=False
147         while seguir:
148           siguiente = next[anterior]
149           if siguiente < nDD:
150               break
151           tour[pos] = siguiente
152           dem1 += DC[siguiente-nDD]
153           dem2 += DP[siguiente-nDD]
154           load = dem1 - dem2
155           if load < minload:
156             minload=load
157           # Restriciones de capacidad combinadas DC y DP
158           if dem1 - minload > CV:
159             m.addConstr((gp.quicksum(x[tour[p1],tour[p1+1]] for p1 in
    range(0,pos))  <= pos-1   ))
160             encontrado = True
161             break
162           pos+=1
163           anterior=siguiente
```

```
164
165   end_time = time.time()
166   duration = round(end_time - start_time, 2)
167
168   print('\nEl valor objetivo obtenido es: ', m.ObjVal)
169   print('Duracion: ', duration, 'segundos. \n')
170   print('Numero de iteraciones:', iteraciones, '\n')
171
172   global selected
173   selected = [i for i, z in x.items() if z.X > Eps]
174   if(DISPLAY_LEVEL > 0) :
175     print('Los arcos utilizados para los proveedores son:')
176     print(selected)
177   global abrir
178   abrir = [j for j, w in y.items() if w.X > Eps]
179   if(DISPLAY_LEVEL > 0) :
180     print('Los depositos abiertos son:')
181     print(abrir)
182
183   return iteraciones
```

Listing A.5: Código Modelo 2

A.1.5. Resultados en fichero .txt

```
1  # Tabla
2  modelos = [1,2]
3  tamanos = ['20-5']
4  semillas = ['1','1b','2','2b']
5  elecciones = [0, 1]  # 0 la opcion de Angelelli y Mansini y para 1 la
       opcion de Salhi y Nagi
6  tipos = [0,1]
7
8  for mo in modelos:
9    for tam in tamanos :
10     for s in semillas :
11       nombre_fichero = 'coord' + tam + '-' + s + '.dat'
12       #print('\n')
13       #print('---------',nombre_fichero, '--------- ')
14       for opt in elecciones:
15         eleccion=opt
16         for o in tipos:
17           tipo=o
18           if eleccion==0 and tipo==0:
19             letra='W'
20           if eleccion==0 and tipo==1:
21             letra='Z'
22           if eleccion==1 and tipo==0:
23             letra='X'
24           if eleccion==1 and tipo==1:
25             letra='Y'
26           print('RESOLVIENDO modelo ' + str(mo) + '  Fichero: ' +
       nombre_fichero
```

```
                + '  Solucion= ' + letra)
        lectura_fichero(nombre_fichero)
        start_time = time.time()

        # Establecer el parametro OutputFlag en 0 para evitar que se
    impriman todos los calculos
        m.setParam("OutputFlag", 0)

        if (mo == 1):
          modelo1()
        else:
          iteraciones=modelo2()

        end_time = time.time()
        duration = round(end_time - start_time, 2)
        dibuja(selected, abrir, nombre_fichero + ' Solucion ' + letra
    ,round(m.ObjVal,2))
        f = open('fichero_resultados.txt', 'a')
        f.write(nombre_fichero)
        f.write("\t")
        f.write(str(mo))
        f.write("\t")
        f.write(letra)
        f.write("\t")
        f.write(str(round(m.ObjVal,2)))
        f.write("\t")
        f.write(str(duration))
        if mo == 2:
          f.write("\t")
          f.write(str(iteraciones))
        f.write("\n")
        f.close()
```

Listing A.6: Código generador de tabla de resultados

Bibliografía

[1] Ismail Karaoglan, Fulya Altiparmak, Imdat Kara, Berna Dengiz. *A branch and cut algorithm for the location-routing problem with simultaneous pick-up and delivery*, European Journal of Operational Research 211 (2011) 318–332.

[2] Ed. Diaz de Santos, J.J. Salazar González *Programación Matemática* (2001).

[3] *Gurobi Optimization*. [Fecha de consulta: 25-10-2022]. Disponible en: https://www.gurobi.com/.

[4] Python [Fecha de consulta: 15-10-2022] https://www.python.org/.

[5] Classical instances for LRP, Prodhon (2008) http://prodhonc.free.fr/Instances/instances_us.htm

[6] Salhi, S., Nagy, G., (1999). *A cluster insertion heuristic for single and multiple depot vehicle routing problems with backhauling.* Journal of the Operational Research Society 50, 1034–1042.

[7] Angelelli, E., Mansini, R., (2002). *The vehicle routing problem with time windows and simultaneous pick-up and delivery.* In: Lecture Notes in Economics and Mathematical Systems. Springer, Germany, pp. 249–267.

[8] R. Praxedesa, T. Bulhoes, A. Subramaniand, E. Uchoa (2023) *A unified exact approach for a broad class of vehicle routing problems with simultaneous pickup and delivery*, Technical Report.

[9] *Problema de rutas de vehículos* [Fecha de consulta: 20-06-2023]. Disponible en: https://es.wikipedia.org/wiki/Problema_de_rutas_de_

vehculos.

[10] CPLEX [Fecha de consulta: 20-06-2023] https://www.ibm.com/es-es/products/ilog-cplex-optimization-studio

[11] C++ [Fecha de consulta: 20-06-2023] https://es.wikipedia.org/wiki/C%2B%2B

[12] Python [Fecha de consulta: 20-06-2023] https://es.wikipedia.org/wiki/Python

Lista de símbolos y abreviaciones

$\boldsymbol{LRPSPD}$	Problema de localización y rutas con recogida y entrega simultánea de producto
$\boldsymbol{LRP}$	Problema de localización y rutas
$\boldsymbol{VRP}$	Problema de rutas de vehículos
$\boldsymbol{FLP}$	Problema de localización de instalaciones
$\boldsymbol{S}$	Conjunto de nodos
$\boldsymbol{P}$	Camino de un grafo
$(\boldsymbol{i}, \boldsymbol{j})$	arco que uno los nodos $\boldsymbol{i}$ y $\boldsymbol{j}$
$\boldsymbol{G} = (\boldsymbol{N}, \boldsymbol{A})$	Grafo cuyo conjunto de puntos es $\boldsymbol{N}$ y tiene a $\boldsymbol{A}$ como conjunto de aristas

Location and Routing Problem with Simultaneous Pickup and Delivery

Carlos A. Pavillard Pérez
Facultad de Ciencias • Sección de Matemáticas
Universidad de La Laguna
alu0101266619@ull.edu.es

Abstract

This paper presents two models in order to approach the problem of location and routing with simultaneous delivery and pickup of products (LRPSPD). The objective is to determine the optimal use of available depots, assign customers to the depots to be used, and find the corresponding vehicle routes, with minimum total cost. The obtained results are compared with those already existing in the literature.

1. Introduction

A directed graph $G = (N, A)$ is considered, where $N = N_0 \cup N_C$ represents the set of nodes, with N_0 denoting the set of available depots and N_C representing the set of customers. The set A consists of arcs defined as $A = \{(i, j) : i, j \in N\}$. On the other hand, we must take into account the following problem constraints:

- Each vehicle can be used in only one route.
- Each customer is serviced by a single vehicle.
- Each route begins and ends at the same depot.
- The total load of a vehicle at any point along the route must not exceed its capacity.
- The combined pickup and delivery load of customers assigned to a depot must not exceed the depot's capacity.

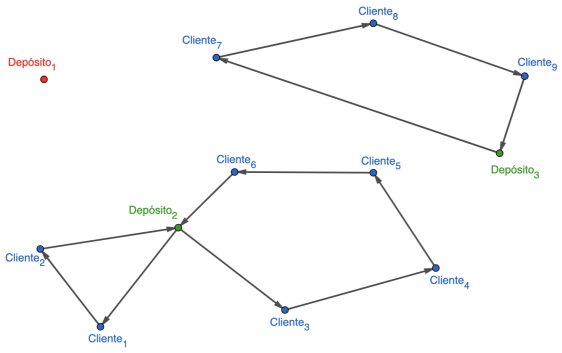

Figure 1: *Scheme of a location and routing problem with simultaneous pickup and delivery*

2. Models

The initial model (Model 1) was proposed by Karaoglan et al. [1], which is based on location and routing problems and make use of flow variables.

For Model 2, the decision was made to remove the flow variables and all the constraints associated with these variables, as compared to Model 1. And to add the following constraints instead:

$$\sum_{a \in \delta^+(S)} x_a \geq z_{jk} \qquad \forall k \in S \cap N_0, j \in N \setminus S \qquad (1)$$

$$\sum_{i,j \in S} x_{ij} \leqslant |S| - r(S) \qquad \forall S \subseteq N_C \qquad (2)$$

$$\sum_{a \in P} x_a \leqslant |P| - 1 \qquad \forall P \text{ Infeasible path} \qquad (3)$$

3. Computational Results

Several experiments were conducted to determine which model was more efficient. To solve Model 2, an iterative method is used due to the exponential number of constraints (1)-(3). In this way, these constraints are added as they are needed.

		Model 1		Model 2		
File	Type	Objective	Time	Objective	Time	Iter
coord20-5-1	W	26464.67	**339.97**	26419.54	14400.17	37
	Z	26456.87	**3282.99**	26413.88	14400.18	40
	X	16820.34	**150.18**	16820.34	11602.04	213
	Y	16820.34	**113.20**	16,20.22	14400.09	209
coord20-5-1b	W	18704.44	140.02	18704.44	**28.92**	8
	Z	18702.96	16.50	18702.96	**13.84**	4
	X	9167.15	**1.10**	9167.15	4.63	1
	Y	9167.15	**1.11**	9167.15	7.99	2
coord20-5-2	W	27988.31	**495.60**	27988.31	5032.57	46
	Z	27980.64	**612.43**	27980.64	2644.49	21
	X	17808.17	**140.42**	17808.17	620.95	30
	Y	17808.17	**323.10**	17808.17	531.73	28
coord20-5-2b	W	17122.20	35.60	17122.20	**32.58**	6
	Z	17117.18	26.90	17117.18	**15.29**	3
	X	10257.34	**3.98**	10257.34	30.08	6
	Y	10257.34	**5.45**	10257.34	32.11	6
Promedio			355.53		3987.35	41.25

In conclusion, it was determined that, on average, Model 1 is more efficient, although Model 2 performs faster on the simpler instances of types W and Z. Furthermore, the results obtained using the same model as in the article [1], but with a different mathematical solver, were better.

References

[1] Ismail Karaoglan, Fulya Altiparmak, Imdat Kara, Berna Dengiz. *A branch and cut algorithm for the location-routing problem with simultaneous pickup and delivery*, European Journal of Operational Research 211 (2011) 318–332.