Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Resolución de un problema de gran escala de reparto de última milla con opciones de entrega y ventanas de tiempo mediante algoritmo k-means.

Autor: D. José Ballestero de Juan

Tutor: Dr. Alejandro Escudero Santana

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

Sevilla, 2025

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Resolución de un problema de gran escala de reparto de última milla con opciones de entrega y ventanas de tiempo mediante algoritmo k-means.

Autor:
D. José Ballestero de Juan

Tutor:
Dr. Alejandro Escudero Santana
Catedrático de Universidad

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

Sevilla, 2025

Trabajo Fin de Grado: Resolución de un problema de gran escala de reparto de última milla con opciones de entrega y ventanas de tiempo mediante algoritmo k-means.

Autor: D. José Ballestero de Juan
Tutor: Dr. Alejandro Escudero Santana

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

A mi familia y amigos, por su apoyo incondicional.

Jose Ballestero de Juan
Sevilla, 2025

Resumen

Resumen rápido crecimiento del comercio electrónico ha acarreado un aumento en la complejidad de la logística que lo acompaña. Siendo “la última milla”, el tramo más importante en los envíos. Debido a que los clientes prefieren la entrega a domicilio, el mayor reto de la logística del comercio electrónico es reducir el número de incidencias a la hora de entregar los productos a los clientes. Estas incidencias disminuyen la calidad percibida del servicio y un aumento en los costes de distribución y emisiones.

Este proyecto aborda la problemática actual de los envíos a domicilio relacionados con el comercio electrónico. Se pretende que la solución creada sirva de ayuda para intentar reducir el número de entregas fallidas por falta de comunicación y coordinación entre las empresas repartidoras y el cliente.

Este estudio aborda el problema utilizando técnicas de clusterización y planificación de rutas, con el objetivo de minimizar la distancia recorrida, reducir los tiempos de espera y mejorar la asignación de recursos en escenarios con diferentes volúmenes de demanda.

Abstract

The rapid growth of e-commerce has led to an increase in the complexity of the logistics involved. The "last mile" is the most important part of shipping. As customers prefer home delivery, the biggest challenge for e-commerce logistics is to reduce the number of incidents in delivering products to customers. These incidents decrease the perceived quality of service and increase distribution costs and emissions.

This project addresses the current problem of home deliveries related to e-commerce. It is intended that the created solution will help to reduce the number of failed deliveries due to lack of communication and coordination between the delivery companies and the customer.

This study addresses the problem using clustering techniques and route planning, aiming to minimize the distance traveled, reduce waiting times, and improve resource allocation in scenarios with different demand volumes.

Índice

Resumen III
Abstract V

1 Introducción 1
1.1 Motivación 2
1.2 Objetivos 2
1.3 Estructura del trabajo 3

2 Marco teórico 5
2.1 Estado del arte 5
2.2 Técnicas utilizadas 6
2.2.1 Algoritmos exactos 6
2.2.2 Algoritmos heurísticos y metahurísticos 7
2.2.2.1 Algoritmos heurísticos 7
2.2.2.2 Algoritmos metaheurísticos 7
2.3 Algoritmo K-Means 8
2.3.1 Definición matemática 8
2.3.2 Consideraciones 9
2.3.3 Aplicaciones del algoritmo 9

3 Formulación del problema 11
3.1 Definición del problema 11
3.1.1 Resumen de parametros y variables 11
3.1.1.1 Relación entre los parámetros 12
3.2 Modelo matemático 13
3.2.1 Restricciones del problema 13

4 Metodología 15
4.1 Generación de instancias del problema y restricciones aplicadas 15
4.1.1 Restricciones en la generación de la base de datos 15
4.1.1.1 Restricciones geográficas 15
4.1.1.2 Restricciones en la cantidad y tipo de puntos de entrega 16
4.1.2 Restricciones temporales 16
4.1.3 Restricciones de los vehículos 17
4.1.4 Impacto de la Generación de Instancias 17
4.2 Aplicación del algoritmo K-Means 17

4.2.1 Objetivo 17
4.2.2 Proceso de aplicación 18
4.2.3 Beneficios 18
4.3 Asignación de rutas 18
4.3.1 Objetivo 18
4.3.2 Proceso 19
4.3.3 Beneficios 20

5 Resultados 21
5.1 Tablas 21
5.1.1 Simulación de 50 pedidos 21
5.1.2 Simulación de 200 pedidos 22
5.1.3 Simulación de 500 pedidos 22
5.1.4 Simulación de 1000 pedidos 23
5.2 Gráficos 23
5.2.1 Distribución de entregas (Clusters) 23
5.2.2 Rutas generadas por los vehículos 25
5.2.3 Análisis de horarios de entrega 27
5.3 Análisis de resultados 29
5.3.1 Impacto del número de entregas 29
5.3.2 Efecto de los parámetros 29
5.3.3 Impacto del número de vehículos 29

6 Conclusiones 31
6.1 Conclusiones personales 31
6.2 Conclusiones técnicas 31
6.3 Líneas futuras 32
6.4 Conclusión final 33

Apéndice A Código 35
A.1 Clases principales del código 35
A.2 Funciones para generar los datos del problema 38
A.3 Funciones utilitarias 45

Índice de Figuras 57
Índice de Códigos 59
Bibliografía 61

1 Introducción

En la actualidad, el desarrollo de la tecnología se aprecia en todos los ámbitos de la vida. En los últimos años nuestro día a día depende cada vez más de las máquinas. A pesar de esto, todavía queda mucho camino por recorrer. Un aspecto que ha crecido mucho, especialmente desde el COVID, ha sido el comercio electrónico.

Se ha creado una tendencia a querer todo de la mejor manera y lo antes posible. Esto ha planteado serios problemas para los distribuidores. La eficiencia en la logística y la distribución se ha convertido en un factor crucial para el éxito de las empresas.

El auge del comercio electrónico y la demanda creciente de servicios de entrega rápida han planteado desafíos significativos en la gestión y optimización de las cadenas de suministro, especialmente en el contexto de la "última milla". En muchos casos, en los envíos, desde que el cliente realiza una compra online hasta que la empresa de transporte lo lleva a su ciudad pueden pasar menos de 24 horas. Sin embargo, que un repartidor se lo entregue al cliente puede demorarse en días.

La última milla se refiere al tramo final del proceso de entrega, donde los productos son transportados desde un centro de distribución hasta el destino final del cliente. Este tramo, aunque corto en distancia, representa una de las etapas más costosas y complejas de la cadena de suministro debido a la necesidad de precisión, puntualidad y flexibilidad. Optimizar esta fase implica superar desafíos relacionados con la ubicación de las entregas, las ventanas de entrega y la capacidad de reparto. Los algoritmos de búsqueda de ruta juegan un papel fundamental para encontrar una solución eficiente.

Según estudios recientes, el coste de la última milla lo convierte en una de las etapas más costosas del proceso de distribución. Este problema es ocasionado principalmente por la sobre población en los centros urbanos. Las entregas en zonas rurales presentan desafíos contrarios, menor densidad de entregas y mayores distancias, lo que dificulta la optimización de rutas.

La eficiencia en la última milla no solo afecta los costos, sino también la satisfacción del cliente. En un mercado tan competitivo como el actual es fundamental complacer al cliente. La saturación del mercado con infinidad de productos le da más valor a la parte de entrega. Este auge ha impulsado la necesidad de soluciones eficientes y escalables [1].

Este problema implica diversos desafíos debido a la alta variabilidad de los destinos de entrega, las restricciones de tiempo y las expectativas crecientes de los consumidores. Las empresas buscan continuamente métodos innovadores para optimizar esta fase crítica, minimizando costos y mejorando la satisfacción del cliente.

La investigación y el desarrollo en este campo son fundamentales para el futuro de la logística, ya que la demanda de entregas rápidas y eficientes seguirá creciendo. Este trabajo busca contribuir a este esfuerzo global, proporcionando una base sólida para la implementación de soluciones prácticas y efectivas en el reparto de última milla.

1.1 Motivación

Existen diferentes motivaciones para realizar este trabajo. Debido al fuerte impacto que tiene este problema en la sociedad, el interés académico, y los beneficios que se pueden obtener, hacen de este trabajo un objeto de gran atractivo.

Las empresas de logística son fundamentales en la economía global, y su éxito depende en gran medida de su capacidad para realizar entregas de manera eficiente y puntual. La última milla representa hasta un 30 por ciento del costo total del envío. Para mejorar los márgenes de beneficio, optimizar esta etapa puede convertirse en una reducción significativa de costos operativos.

La experiencia para el consumidor es algo prioritario hoy en día, las mejoras en la puntualidad y fiabilidad de las entregas contribuye a ello. Una logística eficiente y sostenible también se traduce en una reducción del tráfico y la contaminación, lo que mejora la calidad de vida de los ciudadanos.

La aplicación de la tecnología al negocio y a nuestras vidas es objeto de estudio académico. Es un desafío intelectual y técnico que puede impulsar avances en diversas disciplinas. Este trabajo puede contribuir a los avances existentes, proporcionando nuevos enfoques y soluciones que pueden ser de ayuda para otros investigadores. Esta colaboración entre la industria y la academia fomenta la transferencia de herramientas y conocimiento.

Mejorar la eficiencia de la última milla contribuye a reducir la congestión del tráfico y las emisiones de gases de efecto invernadero, alineándose con los objetivos globales de sostenibilidad. Al desarrollar soluciones que optimicen el uso de los recursos y reduzcan el impacto ambiental, este trabajo también contribuye a la creación de ciudades más sostenibles.

Ofrecer múltiples opciones de entrega no solo mejora la satisfacción del cliente, sino que también permite a las empresas distribuir mejor sus recursos y planificar las entregas de manera más eficiente. Sin embargo, gestionar estas opciones de manera óptima requiere el desarrollo de modelos avanzados de optimización y el uso de algoritmos robustos que puedan manejar la complejidad del problema.

1.2 Objetivos

El objetivo de este Trabajo Fin de Grado es desarrollar un algoritmo de búsqueda para optimizar el reparto en última milla. Para cumplir con este objetivo principal se han planteado diferentes objetivos secundarios:

- Realizar un estudio de diferentes modelos de búsqueda.
- Desarrollar una metodología que permita conocer con mayor fiabilidad la posibilidad de que un usuario esté disponible para la entrega.
- Desarrollar una metodología que permita encontrar buenas soluciones de forma eficiente.

- Analizar la eficacia de dicha metodología a través de la experimentación.

1.3 Estructura del trabajo

- Capítulo 1. Introducción: descripción breve sobre el origen de la problemática junto con la motivación y los objetivos del proyecto, además de incluirse la estructura del trabajo.
- Capítulo 2. Marco Teórico: revisión de la literatura existente, y descripción de diferentes técnicas.
- Capítulo 3. Problema: explicación formal del problema a resolver.
- Capítulo 3. Metodología: esquema lógico, visión de usuario y casos de uso.
- Capítulo 4. Resultados: se presentan y analizan los resultados obtenidos a partir de la implementación del programa.
- Capítulo 5. Conclusiones y líneas futuras: se evalúa en qué medida se han alcanzado los objetivos propuestos. Además, se proponen posibles líneas futuras de investigación y mejoras para el trabajo realizado.
- Anexo A. Código: se adjunta parte del código elaborado para la creación del algoritmo.

2 Marco teórico

El presente capítulo tiene como objetivo proporcionar el fundamento teórico y conceptual en el que se basa esta investigación, estableciendo las bases necesarias para comprender el problema de optimización de rutas en la última milla y las estrategias empleadas para su resolución. Para ello, se revisan los principales conceptos, modelos matemáticos y técnicas utilizadas en la literatura, permitiendo contextualizar la relevancia del estudio dentro del ámbito de la logística y la optimización.

2.1 Estado del arte

Desde los primeros estudios sobre optimización de rutas, como los planteados por Dantzig y Ramser (1959) [2], el Vehicle Routing Problem (VRP) ha sido un problema ampliamente estudiado dentro de la investigación operativa. Estos autores fueron pioneros en la formulación de un modelo matemático que permitía minimizar los costos de distribución a través de una asignación óptima de vehículos a rutas de entrega. Posteriormente, Clarke y Wright (1964) [3] introdujeron un algoritmo heurístico basado en un método de ahorro para encontrar soluciones eficientes en problemas de gran escala, lo que marcó un punto de inflexión en la aplicabilidad de estos modelos en la industria.

La evolución de los modelos de optimización de rutas ha llevado al desarrollo de enfoques híbridos que combinan métodos heurísticos y metaheurísticos con algoritmos de aprendizaje automático.

La importancia de considerar restricciones operativas, como las ventanas de tiempo y la capacidad de los vehículos, ha sido destacada por autores como Golden, Raghavan y Wasil (2008) [4], quienes señalan que una modelización adecuada de estas restricciones permite mejorar significativamente la eficiencia del sistema de transporte.

El Vehicle Routing Problem (VRP) es uno de los problemas clásicos más estudiados en el ámbito de la optimización logística y computacional. La versión básica de este problema es el Capaciteted VRP (CVRP). En este problema, las entregas se asocian a los clientes y son conocidas con anterioridad. Los vehículos son idénticos y empiezan su recorrido desde el almacén. Consiste en determinar las rutas óptimas que un conjunto de vehículos debe seguir para realizar las entregas o recogidas desde un depósito central a una serie de clientes. Cada vehículo tiene una capacidad limitada y debe minimizar una función objetivo, como la distancia total recorrida, el tiempo, o los costes asociados al transporte. Sin embargo, este problema básico ha evolucionado a lo largo del tiempo para incorporar restricciones adicionales que reflejan escenarios más complejos en la vida real. Una de sus variantes más relevantes es el Vehicle Routing Problem with Time Windows (VRPTW), donde se deben respetar las ventanas de tiempo asignadas a cada cliente para la entrega

o recogida de productos.

En el presente trabajo nos centramos en una variante más avanzada del VRPTW: el Vehicle Routing Problem with Time Windows and Delivery Options (VRPTWDO). Este problema incorpora no solo las restricciones de capacidad y tiempo que se encuentran en el VRPTW, sino también múltiples opciones de entrega por cliente, lo que añade complejidad tanto a la planificación como a la optimización de las rutas.[5]

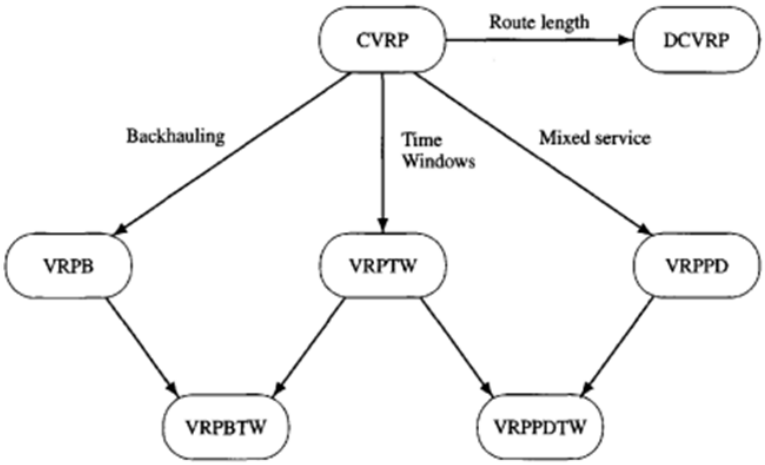

Figura 2.1 Los problemas básicos de la clase VRP y sus interconexiones[6].

El VRPTWDO [7] es especialmente relevante en el contexto actual. Las opciones de entrega incluyen la posibilidad de entregar a domicilio, en lockers (taquillas automáticas) o en puntos de recogida (tiendas o establecimientos asociados). Estas múltiples opciones permiten a las empresas optimizar el proceso de entrega, pero a la vez requieren una planificación cuidadosa para garantizar que se respeten las restricciones de capacidad y tiempo, minimizando a la vez los costes operacionales.

2.2 Técnicas utilizadas

La optimización de rutas de vehículos ha sido objeto de estudio durante décadas debido a su importancia en la logística y el transporte. Se han desarrollado diversas metodologías para abordar este problema, desde algoritmos exactos hasta enfoques heurísticos y metaheurísticos. A continuación, se presentan las principales técnicas utilizadas en la literatura para la optimización de rutas. [8]

2.2.1 Algoritmos exactos

Los métodos exactos [9] encuentran la solución óptima de un problema, pero suelen ser computacionalmente costosos para instancias grandes. Dentro de esta categoría se encuentran:

- Programación Lineal Entera (ILP): Se utilizan formulaciones como la de Dantzig-Fulkerson-Johnson (1954) [10] para resolver el problema del viajante de comercio (TSP) y variantes aplicadas a VRP.

- Branch & Bound: Explora todas las posibles soluciones de manera estructurada, descartando ramas no prometedoras para reducir el tiempo de cálculo.
- Branch & Cut: Variante del Branch & Bound que incorpora desigualdades válidas para mejorar la eficiencia del proceso de poda.
- Dynamic Programming: Método usado para problemas con estructura recursiva, aunque su alto consumo de memoria lo hace ineficiente en problemas de gran escala.

2.2.2 Algoritmos heurísticos y metahurísticos

Dado que el VRP y sus variantes son problemas NP-hard, los métodos exactos no son viables para instancias de gran tamaño debido al alto costo computacional. Por lo tanto, se han desarrollado algoritmos heurísticos y metaheurísticos que buscan soluciones cercanas a las óptimas en un tiempo razonable.

2.2.2.1 Algoritmos heurísticos

Estas estrategias suelen basarse en reglas empíricas que guían la búsqueda hacia soluciones factibles, sin garantizar que sean óptimas.

Características de los algoritmos heurísticos:

- Son específicos para un problema o tipo de problema.
- Proporcionan soluciones rápidas y razonablemente buenas.
- No garantizan encontrar la solución óptima global.
- No garantizan encontrar la solución óptima global.

Ejemplos de algoritmos heurísticos :

1. Greedy (Voraz): Selecciona la mejor opción local en cada paso con la esperanza de que conduzca a una solución global óptima. Se usa en problemas como el Cambio de Monedas y el Problema del Viajante de Comercio (TSP).
2. Búsqueda Local: Parte de una solución inicial y la mejora progresivamente explorando soluciones vecinas. Se utiliza en optimización de rutas y planificación de horarios.
3. Inserción y Eliminación: En problemas de ruteo, se emplean heurísticas que insertan clientes en posiciones viables de la ruta o eliminan aquellos que generan costos elevados.

2.2.2.2 Algoritmos metaheurísticos

Los algoritmos metaheurísticos son estrategias de optimización más avanzadas y generales que guían la búsqueda de soluciones en problemas complejos. A diferencia de las heurísticas, no dependen de reglas fijas, sino que emplean técnicas adaptativas para explorar el espacio de soluciones de manera eficiente.

Características de los algoritmos metaheurísticos [11]:

- Son aplicables a una gran variedad de problemas de optimización.
- Utilizan mecanismos de exploración global y explotación local.
- Suelen basarse en procesos iterativos y estocásticos
- No garantizan la óptima global, pero pueden encontrar soluciones cercanas a la mejor.

Ejemplos de algoritmos metaheurísticos [12]:

1. Algoritmos Genéticos (GA - Genetic Algorithms): Se inspiran en la evolución biológica. Utilizan operadores de selección, cruce y mutación para mejorar soluciones a lo largo de generaciones.
2. Recocido Simulado (Simulated Annealing - SA): Basado en la simulación del enfriamiento de metales, permite aceptar soluciones peores temporalmente para escapar de óptimos locales.
3. Optimización por Colonia de Hormigas (ACO - Ant Colony Optimization): Imita el comportamiento de las hormigas al buscar caminos óptimos entre la colonia y una fuente de alimento.
4. Búsqueda Tabú: Mantiene una lista de soluciones previamente visitadas para evitar ciclos y fomentar la exploración de nuevas áreas del espacio de búsqueda.
5. Enjambre de Partículas (Particle Swarm Optimization - PSO): Modela la inteligencia colectiva de grupos como bandadas de aves para buscar soluciones en espacios multidimensionales.

2.3 Algoritmo K-Means

El algoritmo K-Means es uno de los métodos de agrupamiento más utilizados en el ámbito del aprendizaje automático y la optimización. Su objetivo es dividir un conjunto de datos en K grupos (clusters) de manera que los elementos dentro de un mismo grupo sean lo más similares entre sí y lo más diferentes posible de los elementos en otros grupos. Se trata de un algoritmo no supervisado, lo que significa que no requiere etiquetas previas en los datos para realizar la clasificación.[13]

La idea principal de K-Means fue introducida por Lloyd (1957) y popularizada más adelante por MacQueen (1967) [14]. Se basa en un enfoque iterativo en el que se minimiza la varianza intra-cluster, lo que implica que cada punto se asigna al centroide más cercano y los centroides se recalculan en cada iteración. El proceso se repite hasta que los centroides dejan de cambiar significativamente entre iteraciones, indicando una convergencia estable.

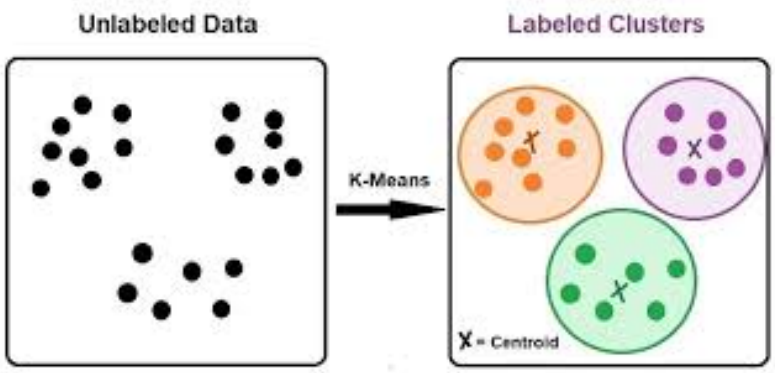

Figura 2.2 Algoritmo K-Means [15].

2.3.1 Definición matemática

Dado un conjunto de datos X=x1,x2,...,xn en un espacio métrico, el objetivo de K-Means es encontrar K centroides C=c1,c2,...,ck que minimicen la suma de las distancias cuadradas de cada punto a su centroide asignado:

$$\min_{C} \sum_{i=1}^{K} \sum_{x \in S_i} \|x - c_i\|^2 \tag{2.1}$$

Donde:

- K es el número de clusters predefinido.
- Si es el conjunto de puntos asignados al cluster i.
- Ci es el centroide del cluster i, calculado como el promedio de todos los puntos asignados a él.

Este problema se resuelve mediante un procedimiento iterativo que sigue los siguientes pasos:

1. Inicialización: se seleccionan K centroides iniciales de manera aleatoria o utilizando métodos más avanzados como K-Means++ (Arthur & Vassilvitskii, 2007).
2. Asignación de Clusters: cada punto de datos se asigna al centroide más cercano en términos de distancia euclidiana.
3. Reajuste de Centroides: se recalculan los centroides como el promedio de los puntos asignados a cada cluster.
4. Convergencia: se repiten los pasos 2 y 3 hasta que los centroides no cambian significativamente o se alcanza un número máximo de iteraciones.

2.3.2 Consideraciones

Si bien K-Means es un método eficiente y ampliamente utilizado, presenta algunas limitaciones que deben considerarse en su aplicación.

No siempre es fácil determinar el número óptimo de clusters. Métodos como la técnica del codo (Elbow Method) o el criterio de Silhouette pueden ayudar. La elección de los centroides iniciales puede afectar el resultado final. Para mitigar esto, se utilizan técnicas como K-Means++.

K-Means funciona mejor cuando los clusters tienen una forma aproximadamente esférica y tamaños similares. En datos con estructuras más complejas, otros métodos como DBSCAN o GMM (Gaussian Mixture Models) pueden ser más adecuados. Si las características tienen escalas muy diferentes, puede ser necesario normalizar los datos antes de aplicar el algoritmo.

2.3.3 Aplicaciones del algoritmo

El algoritmo K-Means se ha aplicado con éxito en múltiples disciplinas, incluyendo:

- Optimización de Rutas de Vehículos: Agrupamiento de entregas en zonas geográficas para reducir costos operativos (Gonzalez et al., 2019).
- Segmentación de Clientes: Identificación de patrones de compra en marketing (Wedel & Kamakura, 2000). [16]
- Procesamiento de Imágenes: Compresión y segmentación de imágenes basada en color (Sharma et al., 2013). [13]
- Análisis de Datos Biomédicos: Identificación de grupos en datos genéticos o clínicos (Xu & Wunsch, 2005). [17]

3 Formulación del problema

3.1 Definición del problema

El problema abordado en este estudio se centra en la planificación eficiente de rutas de entrega en la última milla, donde una flota de vehículos debe distribuir pedidos en diferentes ubicaciones bajo una serie de restricciones operativas. A medida que aumenta la demanda de entregas rápidas y flexibles, las empresas logísticas enfrentan el desafío de encontrar estrategias óptimas para minimizar costos, tiempos de espera y órdenes no servidas, sin comprometer la eficiencia del servicio.

El escenario de estudio contempla la distribución de paquetes a tres tipos de puntos de entrega:

- Viviendas: Entregas directas a domicilios particulares.
- Lockers: Puntos de recogida con espacio de almacenamiento limitado.
- Tiendas: Establecimientos comerciales donde los clientes pueden retirar sus paquetes.

Cada una de estas ubicaciones presenta restricciones específicas, como ventanas de tiempo, capacidades de almacenamiento y limitaciones operativas. En este contexto, el problema se define como la búsqueda de un conjunto de rutas óptimas que permitan asignar pedidos a vehículos y determinar su secuencia de entrega.

El problema adquiere mayor complejidad a medida que el número de pedidos aumenta, ya que las restricciones de tiempo y capacidad limitan la flexibilidad en la asignación de entregas. Para abordar esta situación, se emplean técnicas de clusterización para agrupar pedidos de forma estratégica y algoritmos de optimización de rutas, evaluando el impacto de distintos parámetros en la eficiencia de la solución propuesta.

Este trabajo busca desarrollar un enfoque que permita mejorar la gestión del reparto en la última milla, proporcionando un modelo adaptable a distintas escalas de operación y contribuyendo a la reducción de costos y tiempos de entrega en la logística urbana.

3.1.1 Resumen de parametros y variables

En este estudio, la optimización de rutas de entrega se basa en la definición y ajuste de varios parámetros clave, los cuales influyen directamente en la eficiencia y el rendimiento del sistema logístico. Estos parámetros determinan la prioridad de cada posible entrega en función de distintos criterios, permitiendo encontrar soluciones equilibradas.

A continuación, se presentan las principales variables y parámetros utilizados en el modelo:

- **Término de distancia**
  - Impacto:
    * Favorece la selección de puntos de entrega cercanos, reduciendo el consumo de combustible y los costos operativos.
    * Reduce el tiempo total de viaje, permitiendo completar más entregas dentro del tiempo máximo de operación.
    * Evita rutas ineficientes con desplazamientos largos innecesarios.
  - Ajuste del parámetro α: Un valor alto de α penaliza los trayectos largos, forzando al algoritmo a seleccionar entregas cercanas. Sin embargo, si es demasiado alto, puede priorizar entregas de baja prioridad o fuera de su ventana de tiempo solo por estar cerca.
- **Término de tiempo de espera**
  - Impacto:
    * Reduce los tiempos de espera, maximizando el uso del tiempo de trabajo de cada vehículo.
    * Prioriza entregas cuya ventana de tiempo esté abierta en el momento de llegada.
    * Evita situaciones donde un vehículo queda detenido sin poder avanzar en su ruta.
  - Ajuste del parámetro β: Si β es demasiado alto, el algoritmo podría ignorar entregas cercanas con espera corta y optar por puntos lejanos con ventana abierta, generando rutas ineficientes. Si es demasiado bajo, los vehículos pueden perder mucho tiempo esperando.
- **Término de prioridad**
  - Impacto:
    * Favorece la selección de entregas con mayor urgencia.
    * Evita que paquetes con prioridad alta sean pospuestos indefinidamente.
    * Balancea la asignación de rutas para cumplir con pedidos importantes dentro del tiempo de trabajo del vehículo.
  - Ajuste del parámetro γ: Si γ es demasiado alto, el algoritmo puede ignorar distancias y tiempos de espera, favoreciendo siempre los pedidos prioritarios, aunque se generen rutas ineficientes. Si es demasiado bajo, los pedidos urgentes pueden quedar sin atender.

3.1.1.1 Relación entre los parámetros

Los valores de α, β y γ deben ajustarse cuidadosamente para encontrar un equilibrio entre eficiencia de la ruta, cumplimiento de ventanas de tiempo y prioridad de pedidos. Dependiendo del escenario, se pueden definir distintas estrategias de optimización:

1. Minimizar distancias $\rightarrow$ Aumentar α, reducir β y γ.
2. Priorizar entregas urgentes $\rightarrow$ Aumentar γ, equilibrando α y β.
3. Evitar tiempos de espera largos $\rightarrow$ Aumentar β, manteniendo α y γ moderados.

El ajuste óptimo de estos coeficientes puede determinarse a través de experimentación y pruebas en distintos escenarios.

3.2 Modelo matemático

En la optimización del Problema de Rutas de Vehículos con Ventanas de Tiempo, es fundamental contar con un criterio que guíe la selección de las entregas en cada momento. Para ello, se define una función de coste que evalúa el impacto de cada posible movimiento y permite elegir la opción más conveniente.

La función de coste utilizada en este trabajo tiene el siguiente objetivo principal:

Minimizar la distancia total recorrida, el tiempo de espera y dar prioridad a las entregas más urgentes, maximizando así la eficiencia del sistema de distribución.

Para lograr este objetivo, la función de coste se expresa como:

$$C = \alpha \sum_{(i,j)\in R} d_{ij} + \beta \sum_{i\in P} w_i + \gamma \sum_{i\in P} p_i \tag{3.1}$$

Donde:

- C es el coste total de la solución.
- d_{ij} representa la distancia entre los puntos i y j.
- w_j es el tiempo de espera en la entrega j.
- p_j es la prioridad del pedido j.
- α, β y γ son los coeficientes de ponderación para cada término.

3.2.1 Restricciones del problema

Existen diversas restricciones que deben ser cumplidas para garantizar la viabilidad de las rutas y optimizar la distribución. A continuación, se describen detalladamente:

1. **Restricción de capacidad**

   Cada vehículo tiene una capacidad máxima, por lo que la suma de los volúmenes de los paquetes entregados en una ruta no debe exceder esta capacidad. Matemáticamente:

   $$\sum_{i\in R} \text{peso}_i \leq \text{capacidad}_{\text{vehículo}}, \quad \forall R \in \text{rutas} \tag{3.2}$$

   Donde:

   - R es la ruta asignada a un camión.
   - Peso_i es el peso del pedido i.
   - Capacidad del vehículo es el límite máximo de carga permitido.

2. **Restricción de ventanas de tiempo**

   Cada punto de entrega tiene una ventana de tiempo asignada, durante la cual el vehículo puede realizar la entrega. Si un vehículo llega antes del inicio de la ventana, debe esperar hasta que se abra. Si llega después del final de la ventana, la entrega no puede realizarse. Esta restricción se expresa como:

$$T_{\text{we}} \leq T_i \leq T_{\text{wl}} \tag{3.3}$$

Donde:

- T_{we} es la hora que se abre la ventana de tiempo.
- T_{wl} es la hora que se cierra la ventana de tiempo.

3. Restricción de tiempo máximo de trabajo

Cada vehículo tiene un límite de tiempo total en el que puede operar antes de regresar al depósito.

Cada camión puede operar un máximo de 480 minutos (8 horas) antes de finalizar su jornada. Si un camión alcanza su tiempo máximo, debe regresar al depósito antes de excederlo.

$$\sum_{i \in R} \text{tiempo de viaje}_i + \text{tiempo de entrega}_i \leq T_{\text{máx}}, \quad \forall R \in \text{rutas} \tag{3.4}$$

4. Restricción de distancia

Las entregas deben realizarse en ubicaciones específicas, generadas de forma aleatoria en un plano determinado. La distancia entre cada punto se calcula de forma euclídea.

5. Restricciones de tipos de puntos de entrega

Existen tres tipos de puntos de entrega con características y restricciones específicas:

- **Domicilios (H - Home)**:
  - Pertenecen al cliente y están sujetos a ventanas de tiempo estrictas.
- **Taquillas (L - Locker)**:
  - Tienen un límite de capacidad, lo que significa que solo pueden aceptar una cantidad determinada de paquetes.
  - Si una taquilla está llena, el pedido no puede ser entregado en ese punto.
- **Tiendas (S - Shop)**:
  - Funcionan de manera similar a las taquillas, pero tienen ventanas de tiempo restringidas.

6. Restricción de retorno al depósito

Todos los camiones deben regresar al depósito para terminar su jornada de trabajo, independientemente del número de entregas realizadas, asegurando que los camiones estén disponibles para la siguiente jornada.

4 Metodología

El capítulo de Metodología describe el enfoque utilizado para abordar el problema de optimización, detallando los procedimientos seguidos, los algoritmos empleados y los criterios de evaluación. El objetivo principal es garantizar que la solución propuesta sea eficiente y adaptable a distintos escenarios logísticos.[18]

4.1 Generación de instancias del problema y restricciones aplicadas

La fase de generación de instancias es un paso fundamental. Para evaluar distintas estrategias de optimización, se crean conjuntos de datos que representan escenarios logísticos realistas con diferentes condiciones de entrega, vehículos y restricciones operativas.

El proceso de generación de estas instancias sigue una estructura definida, asegurando que las características del problema reflejen las limitaciones del mundo real.

4.1.1 Restricciones en la generación de la base de datos

Se imponen diversas restricciones que aseguran que las instancias creadas sean realistas y viables para la optimización. Estas restricciones se aplican en distintos niveles, desde la distribución geográfica de los puntos de entrega hasta la disponibilidad de recursos y horarios de operación.

A continuación, se detallan las principales restricciones aplicadas en la creación de la base de datos.

4.1.1.1 Restricciones geográficas

Las ubicaciones de los puntos de entrega se generan aleatoriamente dentro de un área definida, respetando criterios espaciales que aseguran una distribución coherente:

- Área de operación: Se define un espacio de trabajo con límites en las coordenadas (x, y), dentro del cual se ubican el depósito, los lockers, las tiendas y los hogares.
- Distancia mínima entre puntos: Para evitar ubicaciones superpuestas o demasiado cercanas, se establece un mínimo de distancia entre los puntos de entrega. Esto previene situaciones donde múltiples destinos estén prácticamente en la misma ubicación, lo que distorsionaría la optimización de rutas.

- Ubicación del depósito: El depósito se coloca en una posición estratégica, usualmente en el centro del área de operación, para minimizar las distancias promedio entre el depósito y los puntos de entrega.

4.1.1.2 Restricciones en la cantidad y tipo de puntos de entrega

El número y la proporción de los distintos tipos de puntos de entrega varían según la configuración del problema:

- Número total de puntos de entrega: la cantidad de ubicaciones generadas depende del número de pedidos, ajustando la cantidad de hogares, lockers y tiendas en función del tipo de problema.
- Distribución: se equilibra la distribución de hogares, lockers y tiendas para representar un sistema de distribución mixto.
- Capacidad máxima de lockers y tiendas: para evitar que un solo punto de entrega reciba demasiadas órdenes, se establece un límite máximo de lockers y tiendas. Esto impone restricciones en la asignación de pedidos, asegurando que las entregas se repartan de manera más equitativa.
- Número de opciones de entrega por pedido: cada orden tiene tres opciones de entrega:
  - Una dirección de hogar fija.
  - Un locker, si hay disponibilidad.
  - Una tienda, si hay disponibilidad.

4.1.2 Restricciones temporales

Las ventanas de tiempo se generan para asegurar que las órdenes respeten horarios realistas y permitan evaluar restricciones operativas en la optimización de rutas.

Ventanas de Tiempo del Depósito
El depósito opera dentro de un horario predefinido, con una hora de apertura y una hora de cierre, que determinan los tiempos disponibles para realizar entregas.

1. Ventanas de tiempo del depósito El depósito opera dentro de un horario predefinido, con una hora de apertura y una hora de cierre, que determinan los tiempos disponibles para realizar entregas.

2. Ventanas de tiempo en lockers Están disponibles sin restricciones horarias, por lo que cualquier pedido puede entregarse en cualquier momento dentro de la jornada operativa del depósito.

3. Ventanas de tiempo en tiendas Operan con dos ventanas de tiempo, en turnos de mañana y tarde:
   - Mañana: 10:00 - 13:00
   - Tarde: 15:00 - 18:00 Se generan horarios de apertura y cierre específicos para cada tienda.

4. Ventanas de tiempo en hogares Los hogares tienen una ventana de tiempo, que se determina de forma aleatoria en una franja de mañana.

4.1.3 Restricciones de los vehículos

Los vehículos utilizados en la distribución también tienen limitaciones en su capacidad y operación:

- Número de vehículos: se define un número fijo de camiones disponibles para realizar las entregas.
- Capacidad máxima: cada camión tiene una capacidad máxima de carga en términos de peso y volumen, lo que restringe la cantidad de paquetes que puede transportar en una sola ruta.
- Velocidad de los vehículos: se asigna una velocidad de conducción para calcular los tiempos de viaje entre puntos de entrega.
- Costo de operación: se definen dos costos asociados al uso de los vehículos:
  - Costo por camión: representa el costo fijo de operar un camión en la jornada.
  - Costo por kilómetro: refleja el gasto por la distancia recorrida.

4.1.4 Impacto de la Generación de Instancias

El proceso de generación de datos influye directamente en la calidad y dificultad del problema de optimización. Algunas configuraciones pueden hacer que el problema sea más complejo:

- Mayor cantidad de órdenes → Más combinaciones de rutas posibles.
- Restricciones de ventanas de tiempo estrictas → Menos flexibilidad en la asignación de rutas.
- Menor cantidad de vehículos → Mayor carga por camión, aumentando los tiempos de entrega.

Estos factores hacen que cada instancia generada tenga características únicas, permitiendo evaluar la efectividad de los métodos de optimización en distintos escenarios.
[19]

4.2 Aplicación del algoritmo K-Means

La implementación del algoritmo K-Means en este contexto tiene como objetivo agrupar los puntos de entrega en función de su ubicación geográfica para luego asignarlos a diferentes vehículos de reparto. Este enfoque busca minimizar la distancia total recorrida y optimizar el uso de los recursos disponibles, asegurando que cada camión se encargue de un conjunto de entregas cercanas entre sí.

4.2.1 Objetivo

En el contexto del problema de rutas de vehículos (VRP, Vehicle Routing Problem), la aplicación de K-Means tiene los siguientes objetivos:

- Dividir eficientemente las entregas entre los vehículos disponibles de manera que cada camión se encargue de un subconjunto de pedidos geográficamente cercanos.
- Reducir el tiempo de recorrido y los costos de transporte, agrupando los puntos de entrega en clusters compactos y minimizando la distancia entre ellos.
- Facilitar la asignación inicial de rutas, proporcionando un esquema inicial de entregas que luego puede ser optimizado con otras técnicas.

4.2.2 Proceso de aplicación

El procedimiento seguido en la función de aplicación de K-Means consta de las siguientes fases:

Fase 1: Extracción de datos de los puntos de entrega

Antes de aplicar K-Means, se recopilan las coordenadas geográficas de todos los puntos de entrega disponibles en el problema. Cada punto de entrega se asocia a un pedido y representa una ubicación específica donde un camión podría realizar una entrega.

Fase 2: Aplicación de K-Means para la generación de clusters

Una vez obtenidas las coordenadas de los puntos de entrega, se aplica el algoritmo con un número de clusters igual al número de vehículos disponibles. Esto significa que K-Means dividirá los puntos de entrega en grupos, donde cada grupo representará las entregas asignadas a un camión.

Cada cluster generado define una zona de entrega en la que los pedidos serán atendidos por un solo vehículo. La asignación de clusters se basa exclusivamente en la proximidad geográfica, sin considerar todavía factores como el tiempo de servicio o las ventanas de tiempo.

Fase 3: Asignación de entregas a los vehículos

Una vez que se han determinado los clusters, los pedidos se asignan a los vehículos correspondientes. Cada camión recibe las entregas que pertenecen al cluster que le ha sido asignado. El resultado de esta asignación es una estructura de datos donde cada vehículo tiene asociado un conjunto de entregas que deberá gestionar.

4.2.3 Beneficios

El uso de K-Means en la asignación de rutas proporciona varias ventajas:

- Optimización de la logística de transporte: la asignación basada en la proximidad geográfica permite reducir la distancia total recorrida y, por tanto, los costos de operación.
- Estrategia escalable: puede aplicarse a un gran número de pedidos y camiones sin que el tiempo de procesamiento aumente exponencialmente.
- Simplicidad y facilidad de implementación: a diferencia de algoritmos de optimización más complejos, K-Means se basa en un enfoque intuitivo y de rápida convergencia.

4.3 Asignación de rutas

Este proceso es clave para optimizar los recorridos, minimizar tiempos y cumplir con las restricciones impuestas por el problema. Para ello, se utiliza una función de coste que guía la selección del siguiente punto de entrega en función de diferentes criterios de optimización.

4.3.1 Objetivo

El objetivo de la asignación de rutas es garantizar que cada vehículo complete su conjunto de entregas de manera eficiente y dentro del tiempo máximo permitido, respetando las restricciones establecidas. Para ello, se deben optimizar los siguientes aspectos:

- Cumplimiento de ventanas de tiempo: cada entrega tiene horarios predefinidos dentro de los cuales debe completarse.
- Optimización del tiempo total de recorrido: asegurando que cada camión complete el mayor número posible de entregas dentro de su jornada.
- Reducción de esperas innecesarias: minimizando tiempos de inactividad y maximizando la productividad de los vehículos.
- Distribución equitativa de la carga de trabajo: garantizando que los camiones operen de manera balanceada.

4.3.2 Proceso

El proceso de asignación sigue un enfoque iterativo, en el que cada vehículo avanza en su ruta seleccionando de manera óptima el siguiente punto de entrega según la función de coste.

Fase 1: Inicialización del vehículo

Cada vehículo inicia su jornada desde un depósito central, con una hora de salida predefinida. En este punto, se establecen las condiciones iniciales, como el tiempo disponible para operar y la carga asignada.

Fase 2: Selección del siguiente punto de entrega

Para determinar el próximo destino dentro del grupo de entregas asignadas, el vehículo evalúa las opciones disponibles mediante una función de coste. Se priorizan los destinos que minimicen el coste.

Fase 3: Comprobación de la ventana de tiempo

Cada punto de entrega cuenta con una o más ventanas de tiempo dentro de las cuales se permite la recepción del pedido. Antes de confirmar la asignación del destino, se verifica lo siguiente:

1. Llegada anticipada: si el vehículo arriba antes del inicio de la ventana de tiempo, se requiere una espera antes de completar la entrega.
2. Llegada fuera del horario permitido: si la entrega no puede realizarse dentro del intervalo especificado, el punto de entrega es descartado y se busca una alternativa.
3. Entrega viable: en caso de que la entrega pueda completarse dentro del rango permitido, se actualiza el tiempo estimado de llegada y se registra el tiempo de espera en caso de ser necesario.

Este control es esencial para evitar intentos de entrega fallidos y optimizar el tiempo de operación del vehículo.

Realización de la entrega

Si la entrega puede completarse, se siguen estos pasos:

1. Se registra la distancia recorrida.
2. Se calcula el tiempo de viaje hasta el punto de entrega.
3. Se suma el tiempo de entrega, el cual varía según el tipo de destino.

El pedido se marca como completado y el camión actualiza su ubicación y tiempo operativo.

Control del tiempo máximo de operación

Cada vehículo tiene un tiempo máximo de operación, que no puede superarse. Antes de aceptar una nueva entrega, se verifica que haya tiempo suficiente. Si el tiempo restante no permite completar la entrega y volver al depósito, el vehículo finaliza su jornada.

Cuando un camión completa todas sus entregas o alcanza su límite de tiempo, debe regresar al depósito. Se calcula el tiempo necesario para este regreso y se suma al total de operación del vehículo.

4.3.3 Beneficios

Este enfoque ofrece varias ventajas:

- Cumplimiento de Ventanas de Tiempo: asegura que todas las entregas se realicen dentro del horario permitido.
- Optimización del uso de los vehículos: cada camión recibe una carga de trabajo adecuada a su capacidad.
- Reducción de la distancia recorrida: minimiza los desplazamientos innecesarios.
- Evita tiempos de inactividad excesivos: se planifican las entregas para minimizar los tiempos de espera.

5 Resultados

En esta sección se presenta un análisis comparativo de los resultados obtenidos en la simulación del problema de optimización de rutas de entrega, considerando escenarios con 50, 200, 500 y 1000 pedidos. Se han evaluado distintos conjuntos de parámetros de optimización (alpha, beta, gamma), así como diferentes cantidades de vehículos, con el objetivo de determinar la configuración que ofrece el mejor equilibrio entre distancia recorrida, tiempo total de operación, tiempo de espera, coste total y órdenes no servidas.

Para cada caso de estudio, se han generado diversos gráficos que permiten visualizar la distribución de entregas, las rutas generadas y la planificación horaria de los vehículos, facilitando una comparación exhaustiva de las estrategias utilizadas.

El análisis se enfoca en determinar patrones de eficiencia, identificar limitaciones operativas y proponer estrategias óptimas para la toma de decisiones en logística de última milla.

5.1 Tablas

A continuación, se presentan las tablas de resultados para cada escenario de entrega, permitiendo una visión global del comportamiento de los diferentes parámetros de optimización y su impacto en las métricas clave.

5.1.1 Simulación de 50 pedidos

Para esta prueba solo he simulado con 1 vehículo debido al bajo número de entregas. Esta situación intenta emular un entorno rural o poco poblado. Se presenta la tabla de resultados para 50 pedidos:

Tabla 5.1 Resultados para 50 pedidos..

Índice	α	β	γ	Distancia Total	Tiempo Total	Prioridad Normalizada	Coste Total	Tiempo total de espera (min)	Órdenes No Servidas
1	1	0.1	1	21.96	365.02	1	91.96	15.81	0
2	10	0.1	1	7.41	200.29	0.48	77.41	18.10	0
3	1	1	1	24.17	341.57	0.94	94.17	0.03	0
4	1	0.1	5	21.96	365.02	1	91.96	15.81	0

Selección de la mejor prueba: La prueba 1 (α=1, β=0.1, γ=1) y la prueba 4 (α=1, β=0.1, γ=5), tienen los mismos resultados y han mostrado el mejor equilibrio. Obtienen la máxima priori-

dad normalizada y un bajo tiempo total de espera (15.81 min), sin órdenes no servidas y coste medio.

5.1.2 Simulación de 200 pedidos

Se presenta la tabla de resultados para 200 pedidos:

Tabla 5.2 Resultados para 200 pedidos.

Índice	α	β	γ	Nº Vehículos	Distancia Total	Tiempo Total	Prioridad Normalizada	Coste Total	Tiempo total de espera (min)	Órdenes No Servidas
1	1	0.1	1	2	34.59	959.79	0.83	174.59	1.46	2
2	1	0.1	1	3	42.59	997.14	0.83	252.59	6.75	8
3	1	0.1	1	4	49.49	1088.48	0.78	329.49	86.84	14
4	10	0.1	1	2	25.95	865.91	0.64	165.95	32.35	4
5	10	0.1	1	3	29.90	940.33	0.64	239.90	89.21	0
6	10	0.1	1	4	31.60	953.27	0.66	311.60	126.42	16
7	1	1	1	2	35.20	958.25	0.82	175.20	0.00	3
8	1	1	1	3	43.20	991.94	0.83	253.20	0.00	8
9	1	1	1	4	52.14	995.91	0.68	332.14	32.17	14
10	1	0.1	5	2	34.59	959.79	0.83	174.59	1.46	2
11	1	0.1	5	3	42.61	999.45	0.81	252.61	65.50	24
12	1	0.1	5	4	49.49	1088.48	0.78	329.49	86.84	14

Selección de la mejor prueba: La prueba 10 (α=1, β=0.1, γ=5) a pesar de dejar 2 entregas sin realizar, presenta muy buenos resultados. Estos son, un tiempo de espera casi cero, un coste total muy bajo y una prioridad normalizada alta, lo que la hace la mejor opción en este caso.

5.1.3 Simulación de 500 pedidos

Se presenta la tabla de resultados para 500 pedidos:

Tabla 5.3 Resultados de 500 pedidos..

Índice	α	β	γ	Nº Vehículos	Distancia Total	Tiempo Total	Prioridad Normalizada	Coste Total	Tiempo total de espera (min)	Órdenes No Servidas
1	1	0.1	1	5	77.56	2300.66	0.89	427.56	1.53	60
2	1	0.1	1	6	92.80	2461.98	0.80	512.80	14.41	9
3	1	0.1	1	7	97.62	2481.04	0.75	587.62	61.32	5
4	10	0.1	1	5	58.16	2075.91	0.60	408.16	131.83	35
5	10	0.1	1	6	71.15	2240.41	0.61	491.15	165.39	7
6	10	0.1	1	7	68.34	2158.13	0.58	558.34	125.19	4
7	1	1	1	5	72.70	2284.45	0.91	422.70	0.00	63
8	1	1	1	6	91.74	2417.98	0.78	511.74	0.00	6
9	1	1	1	7	97.71	2440.28	0.74	587.71	37.00	6
10	1	0.1	5	5	77.56	2300.66	0.89	427.56	1.53	60
11	1	0.1	5	6	91.73	2473.05	0.80	511.73	28.11	9
12	1	0.1	5	7	102.21	2568.23	0.76	592.21	90.69	5

Selección de la mejor prueba: en este caso, considero que hay dos buenas opciones según el objetivo buscado.

- La prueba 2 (α=1, β=0.1, γ=1) destaca por mantener una alta prioridad normalizada (0.80), junto con una menor cantidad de órdenes no servidas (9), optimizando el balance entre eficiencia y cobertura de pedidos. El coste es algo superior (512.80), lo que encarece el plan.
- La prueba 4 (α=10, β=0.1, γ=1) presenta el coste más bajo (408.16). Teniendo en cuenta este bajo coste, los resultados son bastantes buenos, una prioridad normalizada media (0.60) y un tiempo de espera de 131.83. El principal inconveniente serían las 35 órdenes sin servir.

5.1.4 Simulación de 1000 pedidos

Se presenta la tabla de resultados para 1000 pedidos:

Tabla 5.4 Resultados de 1000 pedidos..

Índice	α	β	γ	Nº Vehículos	Distancia Total	Tiempo Total	Prioridad Normalizada	Coste Total	Tiempo total de espera (min)	Órdenes No Servidas
1	1	0.1	1	11	167.23	4580.94	0.70	937.23	58.91	19
2	1	0.1	1	12	178.58	4615.55	0.66	1018.58	99.17	8
3	1	0.1	1	13	173.78	4415.91	0.60	1083.78	145.11	26
4	10	0.1	1	11	131.48	4329.68	0.60	901.48	290.93	32
5	10	0.1	1	12	136.85	4406.98	0.60	976.85	314.29	17
6	10	0.1	1	13	137.14	4343.10	0.56	1047.14	311.99	17
7	1	1	1	11	172.08	4518.16	0.69	942.08	12.54	24
8	1	1	1	12	177.19	4550.51	0.66	1017.19	43.67	9
9	1	1	1	13	171.08	4034.63	0.65	1081.08	39.00	129
10	1	0.1	5	11	171.63	4612.09	0.70	941.63	59.85	21
11	1	0.1	5	12	183.57	4651.09	0.67	1023.57	101.49	10
12	1	0.1	5	13	179.05	4507.36	0.61	1089.05	175.02	21

Selección de la mejor prueba: La prueba 1 (α=1, β=0.1, γ=1) destaca por un coste considerablemente bajo. Los resultados obtenidos mantienen una buena indicación, prioridad normalizada media-alta (0.70), tiempo de espera bajo para 11 vehículos (58.91) y solo 19 órdenes no servidas.

5.2 Gráficos

Una vez seleccionada la mejor prueba en cada caso, se presentan los gráficos de los resultados:

5.2.1 Distribución de entregas (Clusters)

Aplicando el algoritmo de Kmeans, se dividen las entregas en los centroides de acuerdo al número de vehículos. Los gráficos muestran la distribución geográfica de viviendas, lockers y tiendas, permitiendo identificar patrones en la agrupación de entregas.

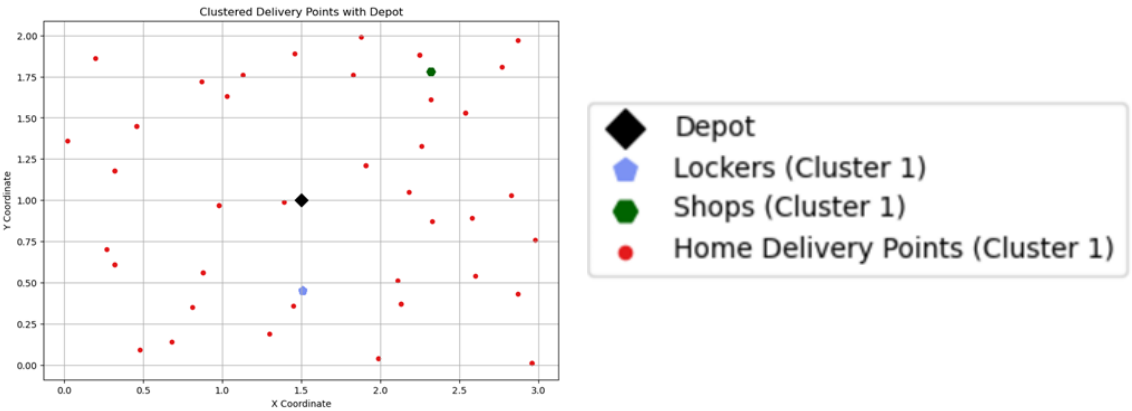

Figura 5.1 Cluster para 50 pedidos.

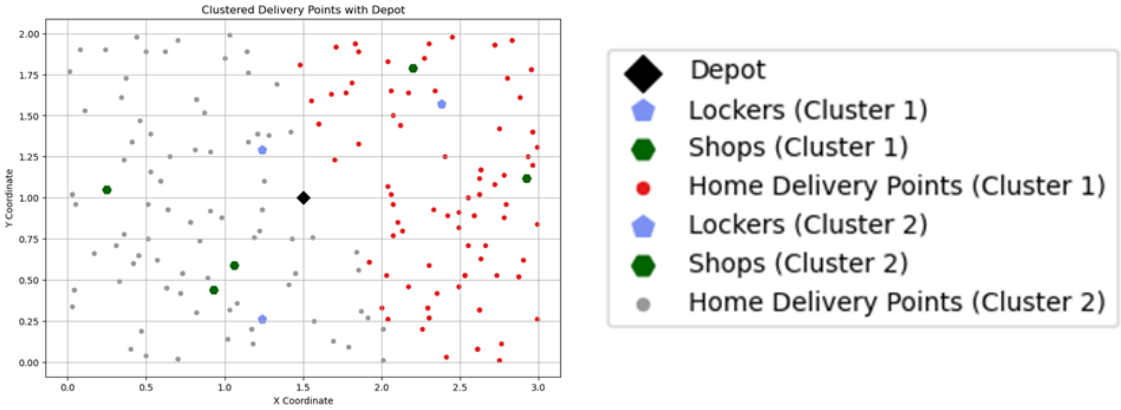

Figura 5.2 Clusters para 200 pedidos.

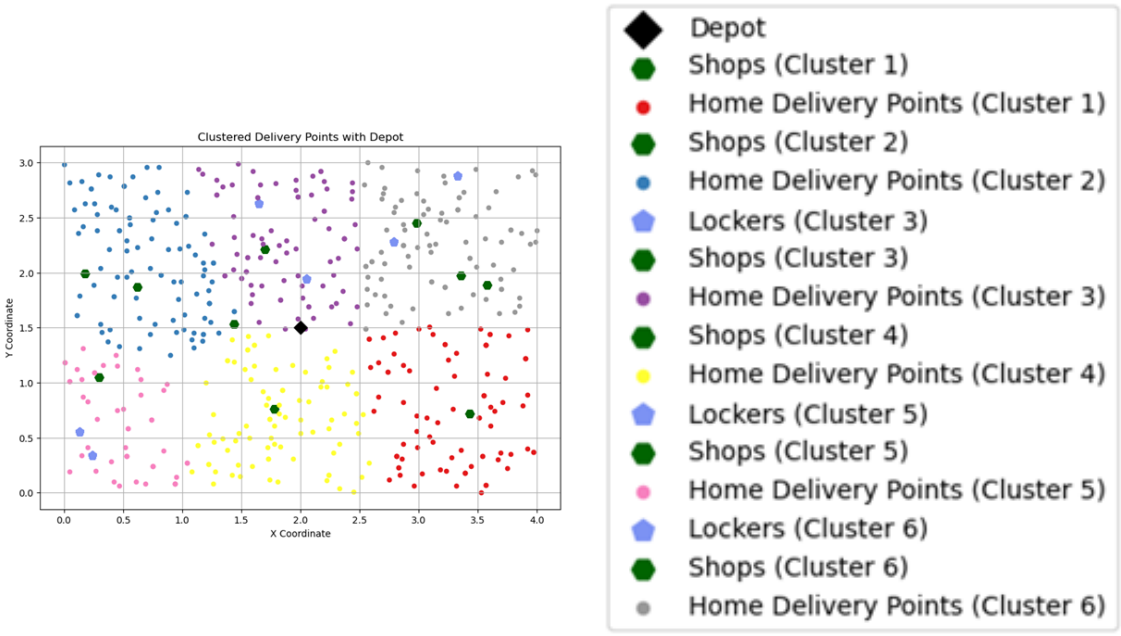

Figura 5.3 Clusters para 500 pedidos.

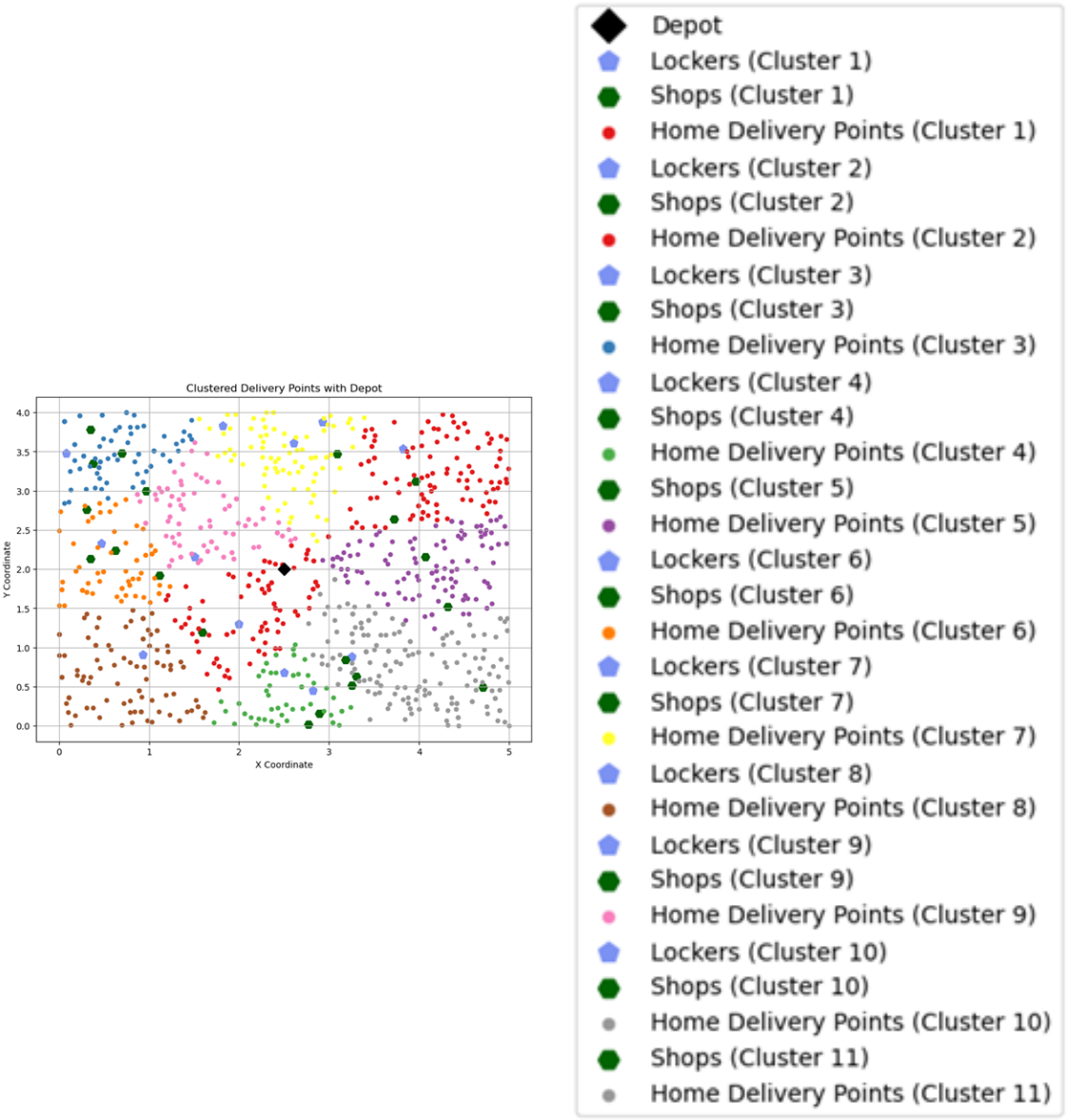

Figura 5.4 Clusters para 1000 pedidos.

5.2.2 Rutas generadas por los vehículos

Estos gráficos muestran la asignación de rutas a cada vehículo, diferenciando los trayectos seguidos. En el caso de 50 y 200 pedidos, aparecen los destinos no visitados. En las siguiente situaciones, por cuestión de visibilidad, solo aparecen los destinos visitados.

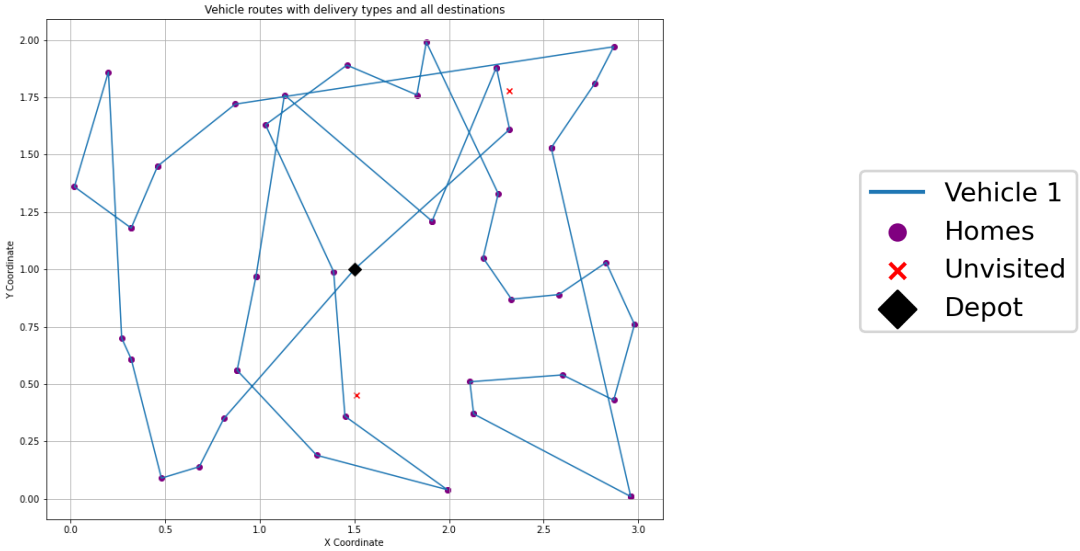

Figura 5.5 Rutas de la prueba seleccionada para 50 pedidos.

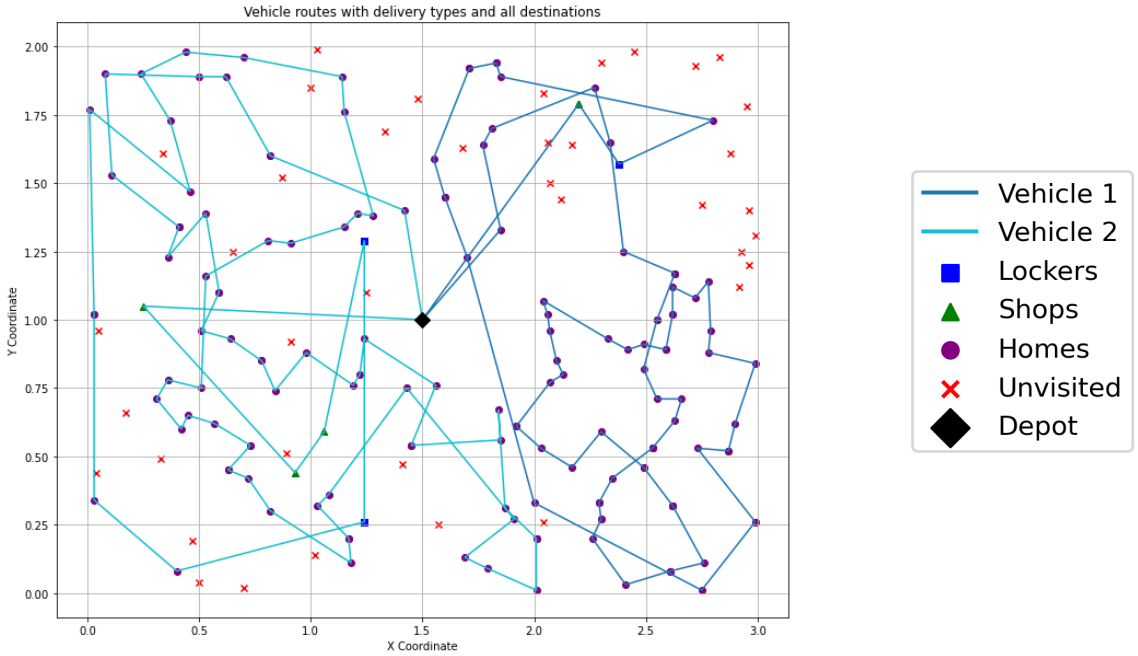

Figura 5.6 Rutas de la prueba seleccionada para 200 pedidos.

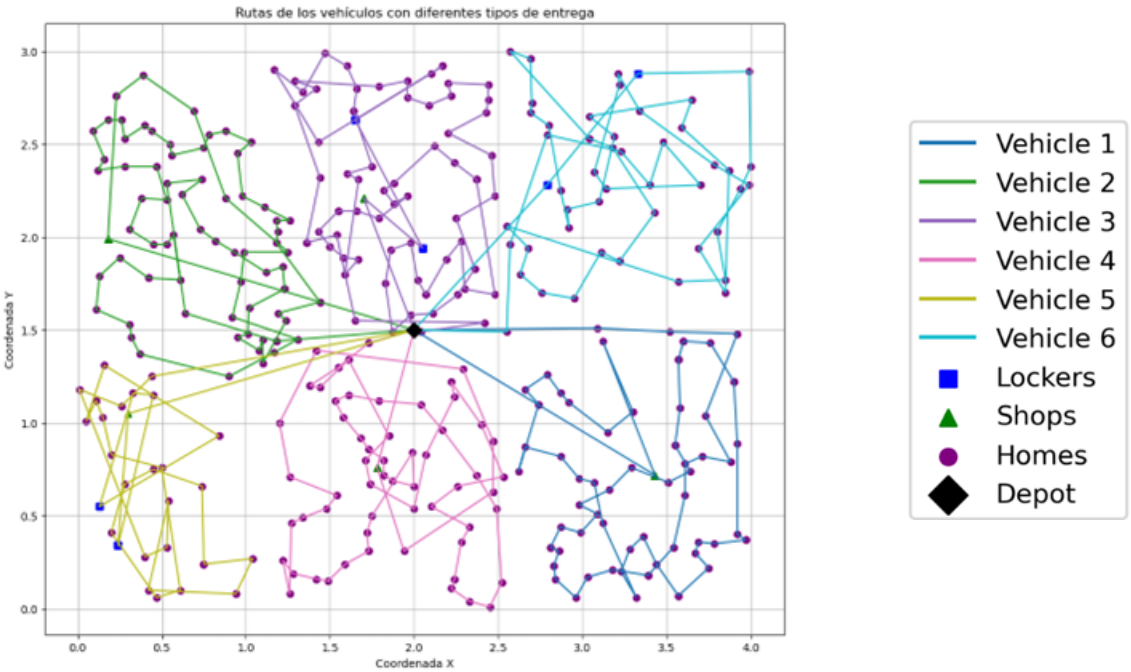

Figura 5.7 Rutas de la prueba seleccionada para 500 pedidos.

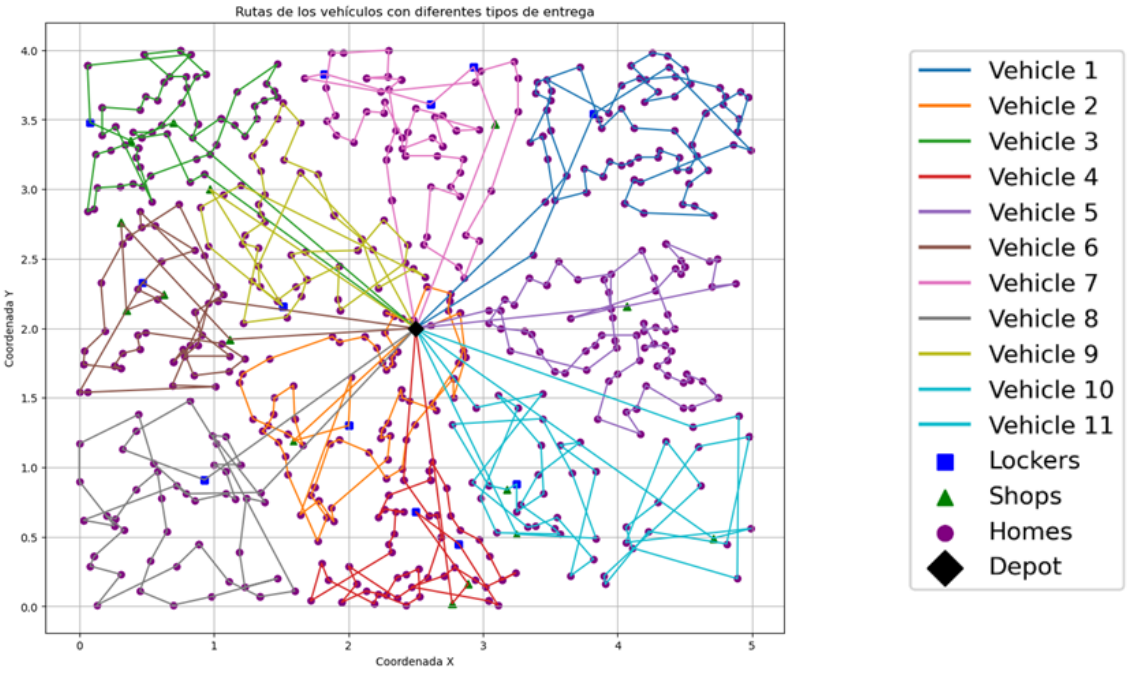

Figura 5.8 Rutas de la prueba seleccionada para 1000 pedidos.

5.2.3 Análisis de horarios de entrega

El gráfico de Gantt permite evaluar la planificación temporal de las entregas.

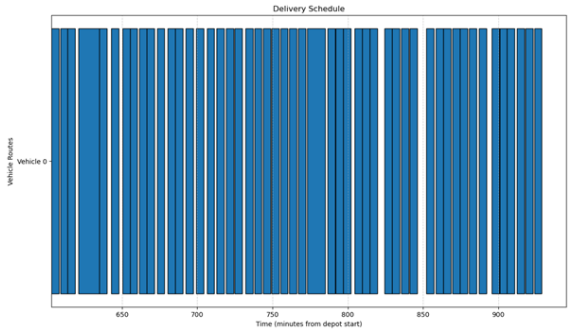

Figura 5.9 Horario de entregas para 50 pedidos.

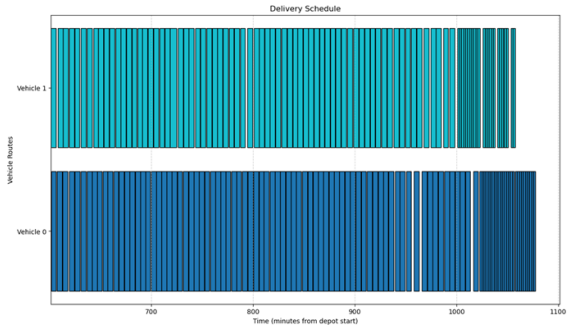

Figura 5.10 Horario de entregas para 200 pedidos.

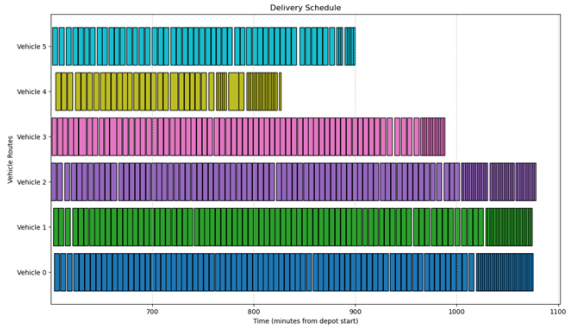

Figura 5.11 Horario de entregas para 500.

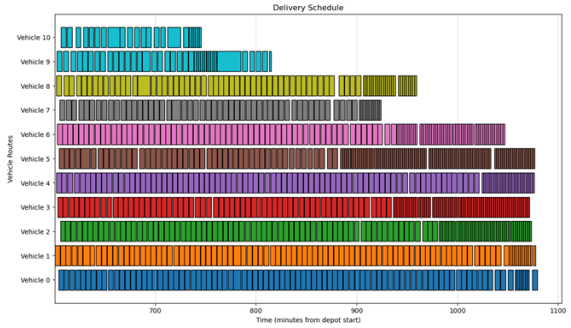

Figura 5.12 Horario de entrega para 1000.

5.3 Análisis de resultados

5.3.1 Impacto del número de entregas

Los resultados muestran que, a medida que aumenta la cantidad de entregas, la optimización de rutas se vuelve más compleja debido a:

- Aumento en la distancia total recorrida: Se observa una tendencia creciente en la distancia acumulada cuando el número de pedidos incrementa. Sin embargo, este crecimiento no es estrictamente lineal, ya que factores como la distribución geográfica y las restricciones de capacidad influyen en la eficiencia.
- Mayor número de órdenes no servidas: En escenarios con 500 y 1000 entregas, se identifican casos con hasta 32 pedidos no entregados debido a restricciones de tiempo o capacidad.
- Mayor tiempo total de operación: En los casos de 1000 entregas, el tiempo total de trabajo por vehículo alcanza valores elevados, lo que sugiere que un mayor número de vehículos podría mitigar este efecto.

5.3.2 Efecto de los parámetros

Se observa que la elección de los pesos en la función de coste tiene un impacto significativo en el rendimiento del sistema:

Minimización de distancia (alpha alto)

Reduce la distancia total recorrida, lo que optimiza el consumo de combustible y el tiempo de operación. Sin embargo, genera un aumento en las órdenes no servidas, lo que sugiere que minimizar distancia no debe ser la única métrica priorizada.

Maximización del tiempo de espera (beta alto)

Reduce el tiempo total de espera de los vehículos en puntos de entrega. Por otro lado, incrementa la distancia recorrida y el costo total, ya que puede obligar a los vehículos a tomar rutas más largas para cumplir con las restricciones de tiempo.

Maximización de prioridad (gamma alto)

En escenarios donde se da más importancia a los pedidos con alta prioridad, se observa que el tiempo total de espera se incrementa. Esto sugiere que la optimización debe balancear adecuadamente este parámetro para no perjudicar el rendimiento general.

5.3.3 Impacto del número de vehículos

La cantidad de vehículos utilizados en cada escenario tiene un papel fundamental en la eficiencia del sistema. Un número insuficiente de vehículos genera cuellos de botella, aumentando tiempos de espera y órdenes no servidas. Cuando se incrementa el número de vehículos, se reduce la distancia media recorrida por cada uno, lo que indica que una mejor distribución de la flota mejora el rendimiento. A partir de cierto punto, agregar más vehículos no genera mejoras significativas, lo que sugiere la existencia de un umbral óptimo en la cantidad de vehículos necesarios.

6 Conclusiones

6.1 Conclusiones personales

El desarrollo de este trabajo ha permitido explorar a fondo las estrategias de optimización para la distribución de última milla, abordando no solo los aspectos teóricos sino también las dificultades prácticas que surgen en la implementación de modelos computacionales aplicados a logística. A lo largo del estudio, se han probado distintos enfoques para la asignación de entregas y planificación de rutas, analizando cómo la interacción de múltiples restricciones impacta en el desempeño global del sistema.

Uno de los aprendizajes más significativos ha sido la identificación de la necesidad de un equilibrio entre distintas métricas clave: minimizar la distancia recorrida, reducir los tiempos de espera y maximizar la prioridad de las entregas sin generar ineficiencias. La optimización de uno de estos factores de manera aislada tiende a producir efectos adversos en otros aspectos del sistema, lo que refuerza la importancia de abordar el problema desde una perspectiva global.

Asimismo, la implementación del modelo ha puesto de manifiesto la complejidad inherente de la logística de última milla, donde la distribución de los pedidos, la cantidad de vehículos disponibles y las restricciones de tiempo juegan un papel crucial en la eficiencia operativa. A medida que el número de pedidos aumenta, las soluciones simples pierden efectividad y es necesario recurrir a enfoques más sofisticados que integren heurísticas avanzadas y métodos de optimización híbridos.

Además, este estudio ha permitido desarrollar una mayor comprensión sobre cómo la clusterización puede facilitar la asignación de entregas, pero también generar ciertas limitaciones cuando no se tienen en cuenta factores dinámicos, como la capacidad de almacenamiento en puntos de entrega o la variabilidad del tráfico. En este sentido, la aplicación de técnicas de optimización combinadas con modelos de predicción podría representar una mejora significativa en futuras investigaciones.

6.2 Conclusiones técnicas

Desde un punto de vista técnico, los resultados obtenidos han permitido extraer diversas conclusiones clave sobre la planificación de rutas en entornos con restricciones complejas. Entre los hallazgos más relevantes se destacan:

- **Interacción entre métricas de optimización:** No existe una única configuración de parámetros que optimice simultáneamente todas las métricas. Minimizar la distancia recorrida puede

aumentar los tiempos de espera, mientras que priorizar la reducción de tiempos de espera puede generar trayectorias más largas e ineficientes. El diseño de un sistema de optimización debe buscar un punto de equilibrio entre estos factores.

- **Impacto del número de vehículos:** Se ha comprobado que incrementar la flota de vehículos reduce el número de órdenes no servidas y los tiempos de espera, pero también aumenta el coste operativo. La relación entre el número de vehículos y la eficiencia del sistema no es lineal, y su optimización debe considerar tanto la capacidad logística disponible como los costos asociados a la operación.
- **Estrategias de asignación de entregas:** La utilización de algoritmos de clusterización como K-Means ha demostrado ser útil para segmentar zonas de entrega, permitiendo una mejor organización de las rutas. Sin embargo, cuando las restricciones de tiempo y capacidad son muy estrictas, la clusterización estática puede generar agrupaciones subóptimas, lo que sugiere que métodos más flexibles podrían mejorar el rendimiento.
- **Efecto de los coeficientes de optimización:**
  - Un valor alto de α (peso de la distancia) fuerza la selección de puntos de entrega cercanos, reduciendo el consumo de combustible y los costos operativos, pero sin valorar ventanas de tiempo y prioridades de entrega.
  - Un valor elevado de β (peso del tiempo de espera) puede reducir significativamente los tiempos de inactividad de los vehículos, aunque puede generar rutas poco eficientes al priorizar entregas inmediatas en lugar de optimizar la secuencia general de la jornada.
  - Un coeficiente γ alto (peso de la prioridad) mejora la calidad del servicio, lo que se traduce en más ventas. Por otro lado, puede aumentar los tiempos de espera de otros pedidos y reducir la eficiencia global del sistema.
- **Limitaciones del modelo:** A pesar de la capacidad de adaptación del modelo a diferentes volúmenes de demanda, existen ciertos aspectos que no han sido abordados en profundidad, como la influencia del tráfico, las restricciones dinámicas en la disponibilidad de lockers y tiendas, y la posibilidad de re programar entregas en tiempo real.

En términos generales, se ha confirmado que la eficiencia en la logística de última milla depende de una adecuada combinación de heurísticas de optimización, una planificación precisa de los recursos y una correcta evaluación de las prioridades operativas. La aplicación de modelos de optimización adaptativos podría representar una solución para mejorar aún más la planificación de rutas en entornos con restricciones dinámicas.

6.3 Líneas futuras

Dado el impacto de los factores analizados en este estudio, se identifican diversas áreas de mejora y expansión que podrían ser abordadas en trabajos futuros para optimizar la planificación de rutas en logística de última milla:

- **Optimización basada en datos en tiempo real:** La incorporación de modelos de predicción del tráfico y herramientas de toma de decisiones en tiempo real permitiría ajustar las rutas dinámicamente, reduciendo los efectos negativos de retrasos inesperados o cambios en la demanda.
- **Exploración de algoritmos híbridos:** Se sugiere la combinación de algoritmos de clusterización con enfoques de búsqueda avanzada como búsqueda tabú, optimización basada en

colonias de hormigas o algoritmos genéticos, con el fin de mejorar la asignación de rutas y la eficiencia de la distribución.

- **Uso de aprendizaje automático para la optimización de parámetros:** Un modelo que pueda ajustar automáticamente los valores de α, β y γ en función de datos históricos y operacionales podría mejorar la toma de decisiones y adaptar la estrategia de optimización a distintas condiciones de trabajo.
- **Modelado de logística multimodal:** Explorar la integración de distintos medios de transporte, como vehículos eléctricos, bicicletas o drones, permitiría evaluar la viabilidad de sistemas de distribución híbridos que optimicen la entrega en función de la infraestructura disponible y las restricciones urbanas.
- **Implementación en entornos reales:** La validación del modelo con datos operacionales de empresas de logística permitiría evaluar su efectividad en escenarios reales, identificando posibles mejoras en la estrategia de asignación de pedidos y en la configuración de restricciones dinámicas.
- **Incorporación de restricciones ambientales y de sostenibilidad:** Modelar estrategias que optimicen rutas no solo en términos de coste y eficiencia operativa, sino también considerando factores de sostenibilidad, como la reducción de emisiones de CO2 o el uso eficiente de recursos energéticos.

6.4 Conclusión final

El presente estudio ha demostrado que la optimización de rutas en la logística de última milla no es un problema con una única solución, sino un desafío dinámico donde la toma de decisiones estratégicas define el equilibrio entre eficiencia operativa, costos y satisfacción del cliente. La aplicación de modelos de clusterización y planificación avanzada ha permitido comprender cómo la interacción entre distancia, tiempos de espera y prioridades puede transformar la logística de distribución, abriendo nuevas posibilidades para su mejora y automatización.

Sin embargo, lo que hoy consideramos optimización puede ser solo el punto de partida. La creciente disponibilidad de datos en tiempo real, junto con el avance de la inteligencia artificial y los algoritmos de optimización adaptativa, sugieren que en los próximos años veremos sistemas logísticos mucho más inteligentes y autónomos. Tecnologías emergentes como la computación en la nube, los modelos de predicción avanzados y la integración de vehículos autónomos podrían revolucionar la forma en que se gestionan las entregas, permitiendo no solo una optimización estática, sino una adaptación continua a un entorno cambiante.

El futuro de la logística de última milla apunta hacia sistemas que no solo optimicen rutas, sino que aprendan y evolucionen con cada nueva entrega. Modelos capaces de anticipar patrones de demanda, ajustar dinámicamente las rutas en función del tráfico o incluso combinar diferentes métodos de transporte en tiempo real marcarán un antes y un después en la eficiencia de la distribución.

La optimización de la logística sigue siendo uno de los retos más apasionantes de la ingeniería moderna, con un impacto directo en la sostenibilidad, la economía y la calidad de vida de las personas. Este estudio ha demostrado que los avances en este campo no solo son posibles, sino necesarios, y que las herramientas del presente son la base sobre la cual se construirá la logística del mañana.

Apéndice A
Código

En este apéndice se incluyen las funciones más relevantes desarrolladas para la optimización de rutas de entrega. Por razones de extensión, no se muestra el código completo. Sin embargo, el repositorio completo está disponible en el siguiente enlace de GitHub:

A.1 Clases principales del código

En este apartado se presentan las clases principales utilizadas en la implementación del problema de optimización de rutas de entrega. Se proporciona una breve descripción de cada clase junto con su estructura de atributos y métodos más relevantes.

Código A.1 Clase Customer.

```
class Customer:
    """
    Representa un cliente en el problema.

    Atributos:
        id (str): Identificador único del cliente.
        mean_satisfaction (float): Nivel medio de satisfacción del
            cliente.
    """
    def __init__(self, id=None):
        self.id = id if id else str(uuid.uuid4())
        self.mean_satisfaction = None

    def __str__(self):
        return f"Customer(id={self.id})"

    def __eq__(self, other):
        return self.id == other.id

    def __hash__(self):
        return hash(self.id)
```

Código A.2 Clase Order.

```
class Order:
    """
    Representa un pedido realizado por un cliente.

    Atributos:
        customer (Customer): Cliente que genera el pedido.
        volume (int): Espacio que ocupa el pedido.
        weight (int): Peso del pedido.
        delivery_deadline (int): Plazo de entrega.
        delivery_options (list): Opciones de entrega posibles.
    """
    def __init__(self, customer, volume, weight, delivery_deadline,
        delivery_options, id=None):
        self.id = id if id else str(uuid.uuid4())
        self.customer = customer
        self.volume = volume
        self.weight = weight
        self.delivery_deadline = delivery_deadline
        self.delivery_options = delivery_options if delivery_options
            else []
        self.release_time = None
        self.delivery_point = None

    def add_delivery_option(self, delivery_option):
        if delivery_option not in self.delivery_options:
            self.delivery_options.append(delivery_option)
```

Código A.3 Clase Location.

```
class Location:
    """
    Representa una ubicación en el sistema de entregas.

    Atributos:
        x (float): Coordenada X.
        y (float): Coordenada Y.
    """
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def distance(self, other):
        return ((self.x - other.x)**2 + (self.y - other.y)**2)**0.5
```

Código A.4 Clase DeliveryPoint.

```
class DeliveryPoint:
    """
```

```
    Representa un punto de entrega en el problema.

    Atributos:
        delivery_type (str): Tipo de punto de entrega (home, locker,
            shop).
        loc (Location): Ubicación del punto de entrega.
        time_windows (list): Lista de ventanas de tiempo disponibles.
        capacity (int): Capacidad máxima del punto de entrega.
    """
    def __init__(self, delivery_type, loc, time_to_depot, time_windows
        =[], capacity=-1, id=None):
        self.id = id if id else str(uuid.uuid4())
        self.delivery_type = delivery_type
        self.loc = loc
        self.time_to_depot = time_to_depot
        self.time_windows = time_windows
        self.capacity = capacity if delivery_type != 'home' else -1
```

Código A.5 Clase Truck.

```
class Truck:
    """
    Representa un camión en la flota de entregas.

    Atributos:
        id (str): Identificador único del camión.
        capacity (int): Capacidad del camión.
        cost (int): Coste del camión.
    """
    def __init__(self, cost, capacity=20):
        self.id = str(uuid.uuid4())
        self.capacity = capacity
        self.cost = cost
```

Código A.6 Clase Depot.

```
class Depot:
    """
    Representa el depósito desde donde operan los camiones.

    Atributos:
        loc (Location): Ubicación del depósito.
        fleet (list): Lista de camiones disponibles.
        time_window (tuple): Ventana de tiempo operativa.
    """
    def __init__(self, loc, fleet=[], time_window=None):
        self.id = str(uuid.uuid4())
        self.loc = loc
        self.fleet = fleet
```

```
        self.time_window = time_window
```

Código A.7 Clase Route.

```
class Route:
    """
    Representa una ruta asignada a un camión.

    Atributos:
        truck (Truck): Camión asignado a la ruta.
        stops (list): Lista de paradas en la ruta.
        distance (float): Distancia total recorrida.
    """
    def __init__(self, truck, stops=[]):
        self.id = str(uuid.uuid4())
        self.truck = truck
        self.stops = stops
        self.distance = 0.0
```

Código A.8 Clase Problem.

```
class Problem:
    """
    Representa el problema de optimización de rutas de entrega.

    Atributos:
        depot (Depot): Depósito principal.
        customers (list): Lista de clientes.
        orders (list): Lista de pedidos.
        delivery_points (list): Lista de puntos de entrega.
    """
    def __init__(self, depot, truck_speed, truck_cost, km_cost,
        customers=[], delivery_points=[], orders=[]):
        self.id = str(uuid.uuid4())
        self.depot = depot
        self.customers = customers
        self.delivery_points = delivery_points
        self.orders = orders
        self.truck_speed = truck_speed
        self.truck_cost = truck_cost
        self.km_cost = km_cost
```

A.2 Funciones para generar los datos del problema

En este capítulo se presentan las funciones principales utilizadas para la generación de los datos del problema. Estas funciones permiten la creación de escenarios realistas con clientes, puntos de entrega, vehículos y restricciones operativas. Se incluyen funciones para generar instancias del

problema con diferentes configuraciones de entrega y restricciones.

Código A.9 Generación de instancias de problema.

```
def generate_benchmark(problem_type: str, n_orders: int, min_distance:
    int, tw_duration: int,
                       n_trucks: int, t_speed:int, x_range: tuple,
                           y_range: tuple,
                       truck_cost: int = None, km_cost: int = None,
                           file_name: str = 'benchmark',
                       max_lockers: int = 5, max_shops: int = 5, plot=
                           False):
    """
    Generates a benchmark problem instance with different delivery
        configurations and constraints.

    Parameters:
        problem_type (str): Type of problem ('HL', 'HLR', '3R', '3H').
        n_orders (int): Number of orders.
        min_distance (int): Minimum distance between points.
        tw_duration (int): Duration of time windows.
        n_trucks (int): Number of available trucks.
        t_speed (int): Speed of the trucks.
        x_range, y_range (tuple): Range of location coordinates.
        truck_cost (int, optional): Cost per truck.
        km_cost (int, optional): Cost per kilometer.
        file_name (str, optional): Name of the output file.
        max_lockers (int, optional): Maximum number of lockers.
        max_shops (int, optional): Maximum number of shops.
        plot (bool, optional): If True, generates a plot of the problem.

    Returns:
        Generates a file containing the problem instance data.
    """

    if problem_type == 'HL':
        n_homes = n_orders
        n_lockers = min(math.ceil(n_orders * 0.15), max_lockers)
        n_shops = 0
    elif problem_type == 'HLR':
        n_homes = n_orders
        n_lockers = min(math.ceil(n_orders * 0.15), max_lockers)
        n_shops = min(math.ceil(n_orders * 0.1), max_shops)
    elif problem_type == '3R':
        n_homes = int(n_orders * 2)
        n_lockers = min(math.ceil(n_orders * 0.2), max_lockers)
        n_shops = min(math.ceil(n_orders * 0.2), max_shops)
    elif problem_type == '3H':
        n_homes = 3 * n_orders
```

```
        n_lockers, n_shops = 0, 0

    if not os.path.exists(problem_type):
        os.makedirs(problem_type + '/problem_files')
        os.makedirs(problem_type + '/solution_files')
        os.makedirs(problem_type + '/results')

    file = open(str(problem_type) + '/' + 'problem_files/' + str(
        file_name) + '.txt', 'w')

    total_num_locations = n_homes + n_lockers + n_shops + 1
    depot_location, locker_locations, shop_locations, home_locations =
        __generate_random_locations(
        total_num_locations, x_range, y_range, min_distance, 10000,
            n_homes, n_lockers, n_shops)
    depot_location = ((x_range[0] + x_range[1]) / 2, (y_range[0] +
        y_range[1]) / 2)

    truck_capacity = int(10 * random.uniform(1.0, 1.5))
    depot_av_capac = truck_capacity * n_trucks
    truck_speed = t_speed
    if truck_cost is None:
        truck_cost = random.randint(50, 100)
    if km_cost is None:
        km_cost = random.randint(70, 100)
    tw_e = random.choice([600])
    tw_l = random.choice([1260, 1290, 1320])
    depot_tw = (tw_e, tw_l)

    file.write("NUM_VEHICLES\tTRUCK_COST\tTRUCK_SPEED\tKM_COST\tX_DEPOT\
        tY_DEPOT\tTW_E\tTW_L\n")
    file.write(f"{n_trucks}\t{truck_cost}\t{truck_speed}\t{km_cost}\t{
        depot_location[0]}\t{depot_location[1]}\t{tw_e}\t{tw_l}\n")

    if locker_locations:
        file.write("\nLOCKER_ID\tX\tY\tAV_CAPAC.\n")
        for i, loc in enumerate(locker_locations[:max_lockers]):
            locker_av_capac = random.randint(5, 20)
            file.write(f"L{i+1}\t{loc[0]}\t{loc[1]}\t{locker_av_capac}\n"
                )

    file.close()

    if plot:
        __plot_problem_from_tuples(depot_location, locker_locations,
            shop_locations, home_locations, problem_type + '/
            problem_files/' + file_name)
        print('Plot saved to ' + problem_type + '/problem_files/' +
            file_name + '.pdf')
```

Código A.10 Cálculo de distancia Euclidiana.

```
def __euclidean_distance(loc1, loc2):
    """
    Computes the Euclidean distance between two points in a 2D space.

    Parameters:
        loc1 (tuple): Coordinates (x, y) of the first point.
        loc2 (tuple): Coordinates (x, y) of the second point.

    Returns:
        float: The Euclidean distance between the two points.
    """

    return math.sqrt((loc1[0] - loc2[0]) ** 2 + (loc1[1] - loc2[1]) **
        2)
```

Código A.11 Determinación del punto central.

```
def __find_central_location(locations):
    """
    Computes the central location (centroid) of a set of 2D points.

    Parameters:
        locations (list of tuples): A list of coordinates, where each
            element is a tuple (x, y).

    Returns:
        tuple: The centroid (mean x, mean y) of the given locations.
    """

    x_coords, y_coords = zip(*locations)
    return (sum(x_coords) / len(x_coords), sum(y_coords) / len(y_coords)
        )
```

Código A.12 Generador de los pedidos.

```
def __generate_orders(n_orders: int, lockers: list, shops: list,
    home_delivery_points: list, max_total_lockers: int = 10,
    max_total_shops: int = 10, problem_type: str = None):
    """
    Generates orders with exactly 3 delivery options for each order.
    Each order has one home delivery point, one locker (if available),
        and one shop (if available).

    Parameters:
        n_orders : int
            Number of orders.
```

```
        lockers : list
            List of lockers.
        shops : list
            List of shops.
        home_delivery_points : list
            List of home delivery points.
        max_total_lockers : int
            Maximum number of unique locker locations.
        max_total_shops : int
            Maximum number of unique shop locations.
        problem_type : str
            Problem type ('HL', 'HLR', '3R', '3H').

    Returns:
        orders : list
            List of orders, each with 3 delivery options (Home, Locker,
                Shop).
    """

    orders = []
    used_hd_points = set()

    # Limita lockers y tiendas a la cantidad máxima especificada
    if len(lockers) > max_total_lockers:
        lockers = random.sample(lockers, max_total_lockers)
    if len(shops) > max_total_shops:
        shops = random.sample(shops, max_total_shops)

    # Verifica que haya suficientes viviendas
    if len(home_delivery_points) < n_orders:
        raise ValueError("No hay suficientes puntos de viviendas
            disponibles para generar las órdenes.")

    for i in range(n_orders):
        if i % 10 == 0:
            print(f"Generando orden {i+1}/{n_orders}")

        order_id = f"O{i + 1}"
        weight = random.randint(1, 10)
        volume = random.randint(1, 10)
        priority = 1
        options = []

        # Asigna siempre una vivienda
        dp_id, x, y, tw_probabilities = random.choice(
            home_delivery_points)

        # Eliminar "HH" usando f"H{dp_id[1:]}" si dp_id ya contiene una
            "H"
        if dp_id.startswith("H"):
```

```
            dp_id = dp_id[1:]
        home_option = (order_id, weight, volume, priority, f"H{dp_id}", x
            , y, tw_probabilities)

        options.append(home_option)

        # Asigna un locker si hay disponible
        if lockers:
            locker_choice = random.choice(lockers)
            dp_id = f"L{lockers.index(locker_choice) + 1}" # Locker ID
                basado en la posición
            locker_option = (order_id, weight, volume, priority + 1,
                dp_id)
            options.append(locker_option)

        # Asigna una tienda si hay disponible
        if shops:
            shop_choice = random.choice(shops)
            dp_id = f"S{shops.index(shop_choice) + 1}" # Shop ID basado
                en la posición
            shop_option = (order_id, weight, volume, priority + 2, dp_id)
            options.append(shop_option)

        # Asegura que haya exactamente 3 opciones por orden
        if len(options) < 3:
            # Añade una segunda vivienda si no hay lockers o tiendas
                suficientes
            dp_id, x, y, tw_probabilities = random.choice(
                home_delivery_points)
            while len(options) < 3:
                if f"H{dp_id}" not in [opt[4] for opt in options]: #
                    Verifica si la vivienda no está ya en las opciones
                    home_option = (order_id, weight, volume, priority +
                        len(options), f"H{dp_id}", x, y, tw_probabilities)
                    options.append(home_option)

        orders.extend(options)

    print(f"Órdenes generadas correctamente")
    return orders
```

Código A.13 Generador delos destinos en tiendas.

```
def __generate_shops(shop_locations: list, depot_location: tuple,
    depot_tw: tuple, truck_speed: int = 20, tw_duration: int = None):
    """
    Generates shops.

    Parameters:
```

```
        shop_locations : list
            List of shop locations.

    Returns:
        shops : list
            List of shops.
    """
    shops = []

    # some random time windows for shops
    shop_tws = []

    morning_start_times = [600, 630, 660] # 10:00, 10:30, 11:00
    morning_end_times = [840, 870, 900] # 14:00, 14:30, 15:00
    afternoon_start_times = [900, 930, 960] # 15:00, 15:30, 16:00
    afternoon_end_times = [1020, 1050, 1080] # 17:00, 17:30, 18:00

    for i in range(5):
        shop_tws.append(
            f'{random.choice(morning_start_times)}\t{random.choice(
                morning_end_times)}\t-\t-')
        shop_tws.append(
            f'{random.choice(afternoon_start_times)}\t{random.choice(
                afternoon_end_times)}\t-\t-')
        shop_tws.append(
            f'{random.choice(morning_start_times)}\t{random.choice(
                morning_end_times)}\t{random.choice(afternoon_start_times)
                }\t{random.choice(afternoon_end_times)}')
        shop_tws.append(
            f'{random.choice(morning_start_times)}\t{random.choice(
                afternoon_end_times)}\t-\t-')

    for loc in shop_locations:
        time_to_depot = __euclidean_distance(loc, depot_location) /
            truck_speed * 60
        tws = random.choice(shop_tws)

        #check if time windows are valid
        valid_tw = False
        while not valid_tw:
            separated_tws = tws.split('\t')
            if int(separated_tws[1]) + time_to_depot <= depot_tw[1]:
                valid_tw = True

            if separated_tws[2] != '-' and separated_tws[3] != '-':
                if int(separated_tws[3]) + time_to_depot <= depot_tw[1]:
                    valid_tw = True

            if not valid_tw:
                tws = random.choice(shop_tws)
```

```
        av_capac = random.randint(1, 30)
        shops.append((f"S{len(shops) + 1}", loc[0], loc[1], tws,
            av_capac))

    return shops
```

A.3 Funciones utilitarias

En esta sección se presentan las funciones auxiliares utilizadas en la generación, evaluación y visualización del problema.

Código A.14 Lectura de archivo de problema.

```
def read_problem_file(file_path: str):
    """
    Reads a problem file and returns a Problem instance.

    Parameters:
        file_path (str): File path of the problem file.

    Returns:
        Problem: Instance of the problem containing vehicles, depot,
            delivery points, and orders.
    """

    with open(file_path, 'r') as f:
        lines = f.readlines()

    num_vehicles = int(lines[1].split()[0])
    truck_cost = int(lines[1].split()[1])
    truck_speed = int(lines[1].split()[2])
    km_cost = int(lines[1].split()[3])
    depot_location = Location(float(lines[1].split()[4]), float(lines[1].
        split()[5]))
    depot_tw = tuple(map(int, lines[1].split()[6:8]))

    fleet = [Truck(truck_cost) for _ in range(num_vehicles)]
    depot = Depot(depot_location, fleet, depot_tw)

    delivery_points = []
    customers = []
    orders = []

    for line in lines[3:]:
        strings = line.split()
        strings = [string.strip() for string in strings]
```

```
        if line[0] == 'L' and 'LOCKER_ID' not in line:
            loc = Location(float(strings[1]), float(strings[2]))
            time_to_depot = loc.distance(depot_location) / truck_speed *
                60
            new_locker = DeliveryPoint(DeliveryType.LOCKER, loc,
                time_to_depot, [], int(strings[3]), strings[0])
            delivery_points.append(new_locker)

        elif line[0] == 'S' and 'SHOP_ID' not in line:
            tws = [tuple(map(int, strings[3:5]))]
            loc = Location(float(strings[1]), float(strings[2]))
            time_to_depot = loc.distance(depot_location) / truck_speed *
                60
            new_shop = DeliveryPoint(DeliveryType.SHOP, loc,
                time_to_depot, tws, int(strings[7]), strings[0])
            delivery_points.append(new_shop)

    problem = Problem(depot, truck_speed, truck_cost, km_cost, customers,
         delivery_points, orders)

    return problem
```

Código A.15 Aplicación de K-Means para la asignación de puntos de entrega.

```
def apply_kmeans(problem, num_vehicles, alpha, beta, gamma,
    max_time_per_vehicle=480, plot_name="kmeans_clusters"):
    """
    Applies K-Means clustering to assign delivery points to different
        vehicles based on location.

    Parameters:
        problem (Problem): The problem instance containing delivery
            points.
        num_vehicles (int): Number of available vehicles (trucks).
        alpha (float): Weight for distance in the cost function.
        beta (float): Weight for waiting time in the cost function.
        gamma (float): Weight for priority in the cost function.
        max_time_per_vehicle (int): Maximum operational time per vehicle
             in minutes (default: 480 minutes).
        plot_name (str): Name for the cluster visualization plot.

    Returns:
        clustered_solution (dict): Dictionary mapping vehicle IDs to
            lists of delivery points.
    """

    delivery_coords = []
    delivery_points = []
```

```
    for order in problem.orders:
        for option in order.delivery_options:
            dp = option[0]
            delivery_coords.append([dp.loc.x, dp.loc.y])
            delivery_points.append((order.id, dp.id))

    delivery_coords = np.array(delivery_coords)

    kmeans = KMeans(n_clusters=num_vehicles, random_state=42).fit(
        delivery_coords)
    labels = kmeans.labels_

    clustered_solution = {i: [] for i in range(num_vehicles)}
    for i, label in enumerate(labels):
        clustered_solution[label].append(delivery_points[i])

    plot_clusters_with_depot(problem, clustered_solution, plot_name)

    return clustered_solution
```

Código A.16 Asignación de entregas a vehículos considerando restricciones de capacidad y tiempo.

```
def assign_deliveries_to_vehicles(clustered_solution, problem,
    num_vehicles, max_time_per_vehicle, alpha, beta, gamma):
    """
    Assigns deliveries to vehicles while respecting capacity constraints
         and time windows.

    Parameters:
        clustered_solution (dict): Dictionary mapping vehicle IDs to
            lists of delivery points.
        problem (Problem): The problem instance.
        num_vehicles (int): Number of vehicles available.
        max_time_per_vehicle (int): Maximum time per vehicle in minutes.
        alpha (float): Weight for distance in the cost function.
        beta (float): Weight for waiting time in the cost function.
        gamma (float): Weight for priority in the cost function.

    Returns:
        final_routes (list): List of routes, each containing tuples of (
            order_id, delivery_point).
        total_waiting_time (float): Total waiting time in minutes.
        current_storage (dict): Dictionary tracking the current storage
            usage of Lockers and Shops.
    """

    MAX_WAITING_TIME = 20

    final_routes = []
```

```
    served_orders = set()
    total_waiting_time = 0
    current_storage = {dp_id: 0 for dp_id in problem.dict_capacity.keys
        ()}

    delivery_time_mapping = {
        'H': problem.dict_delivery_time['HOME'],
        'L': problem.dict_delivery_time['LOCKER'],
        'S': problem.dict_delivery_time['SHOP']
    }

    for vehicle_id in range(num_vehicles):
        vehicle_route = []
        current_location = 'DEPOT'
        current_time = problem.dict_twe['DEPOT']
        elapsed_time = 0
        delivery_points = clustered_solution[vehicle_id]

        while delivery_points and elapsed_time < max_time_per_vehicle:
            next_option = None
            min_cost = float('inf')

            for dp_order_id, dp_id in delivery_points:
                if dp_order_id in served_orders:
                    continue

                if dp_id.startswith(('L', 'S')) and dp_id in problem.
                    dict_capacity:
                    if current_storage[dp_id] >= problem.dict_capacity[
                        dp_id]:
                        continue

                distance = problem.dict_distance[(current_location, dp_id
                    )]
                travel_time = distance * 60 / problem.truck_speed
                arrival_time = current_time + travel_time

                waiting_time = 0
                if dp_id.startswith('S'):
                    valid_window = False
                    for twe, twl in zip(problem.dict_twe[dp_id], problem.
                        dict_twl[dp_id]):
                        if twe <= arrival_time <= twl:
                            valid_window = True
                            waiting_time = max(0, twe - arrival_time)
                            break
                    if not valid_window:
                        continue
                else:
```

```
                twe, twl = problem.dict_twe[dp_id], problem.dict_twl[
                    dp_id]
                waiting_time = max(0, twe - arrival_time)
                if arrival_time > twl:
                    continue

            if waiting_time > MAX_WAITING_TIME:
                continue

            delivery_time = delivery_time_mapping.get(dp_id[0],
                problem.dict_delivery_time['DEFAULT'])
            if arrival_time + waiting_time + delivery_time > twl:
                continue

            delivery_priority = problem.dict_priority[(dp_order_id,
                dp_id)]
            cost = (alpha * distance) + (beta * waiting_time) + (
                gamma * delivery_priority)

            if cost < min_cost:
                min_cost = cost
                next_option = (dp_order_id, dp_id, waiting_time)

        if next_option:
            order_id, dp_id, waiting_time = next_option
            served_orders.add(order_id)
            vehicle_route.append((order_id, dp_id))

            if dp_id.startswith(('L', 'S')):
                current_storage[dp_id] += 1

            distance_to_next = problem.dict_distance[(
                current_location, dp_id)]
            travel_time = distance_to_next * 60 / problem.truck_speed
            delivery_time = delivery_time_mapping.get(dp_id[0],
                problem.dict_delivery_time['DEFAULT'])

            current_time += travel_time + waiting_time +
                delivery_time
            elapsed_time += travel_time + waiting_time +
                delivery_time
            total_waiting_time += waiting_time
            current_location = dp_id

            return_time = problem.dict_distance[(current_location, '
                DEPOT')] * 60 / problem.truck_speed
            if elapsed_time + return_time > max_time_per_vehicle:
                break
        else:
            break
```

```
        if current_location != 'DEPOT':
            return_time = problem.dict_distance[(current_location, 'DEPOT
                ')] * 60 / problem.truck_speed
            current_time += return_time
            elapsed_time += return_time

        final_routes.append(vehicle_route)

    return final_routes, total_waiting_time, current_storage
```

Código A.17 Evaluación de las rutas generadas.

```
def evaluate_routes(problem, final_routes):
    """
    Evaluates each route and aggregates total results.

    Parameters:
        problem (Problem): The problem instance.
        final_routes (list): List of routes, each containing tuples (
            order_id, delivery_point).

    Returns:
        total_priority (float): Total priority score of completed
            deliveries.
        total_distance (float): Total distance covered by all routes.
        total_time (float): Total time spent on all deliveries.
        not_served_count (int): Number of orders not served.
        delivery_times (list): List of delivery times for each route.
    """

    total_priority = []
    total_distance = 0
    total_time = 0
    served_orders = set()
    delivery_times = []

    delivery_time_mapping = {
        'H': problem.dict_delivery_time['HOME'],
        'L': problem.dict_delivery_time['LOCKER'],
        'S': problem.dict_delivery_time['SHOP']
    }

    for vehicle_id, route in enumerate(final_routes):
        if not route:
            continue

        route_priority = 0
        route_distance = 0
```

```
        route_time = 0
        route_times = []
        last_dp = 'DEPOT'
        current_time = problem.dict_twe['DEPOT']

        for order_id, dp in route:
            if order_id in served_orders:
                continue

            travel_time = problem.dict_distance[(last_dp, dp)] * 60 /
                problem.truck_speed
            arrival_time = current_time + travel_time
            waiting_time = max(0, problem.dict_twe[dp] - arrival_time)

            delivery_time = delivery_time_mapping.get(dp[0], problem.
                dict_delivery_time['DEFAULT'])
            current_time += waiting_time + delivery_time

            route_distance += problem.dict_distance[(last_dp, dp)]
            route_time += travel_time + waiting_time + delivery_time
            total_priority.append(problem.dict_priority[(order_id, dp)])
            route_times.append(round(current_time, 2))

            served_orders.add(order_id)
            last_dp = dp

        total_distance += route_distance
        total_time += route_time
        delivery_times.append(route_times)

    not_served_count = len(problem.orders) - len(served_orders)

    return total_priority, total_distance, total_time, not_served_count,
         delivery_times
```

Código A.18 Método del codo para determinar el número óptimo de clusters.

```
def elbow_method(problem, max_clusters=10, random_state=42):
    """
    Applies the elbow method to determine the optimal number of clusters
        for K-Means.

    Parameters:
        problem (Problem): The problem instance containing delivery
            points.
        max_clusters (int): Maximum number of clusters to evaluate.
        random_state (int): Random state for K-Means initialization.

    Returns:
```

```
        None: Generates a plot displaying the elbow curve.
    """

    delivery_coords = np.array([coords for _, coords in problem.dict_xy.
        items() if _ != 'DEPOT'])

    wcss = []
    for n_clusters in range(1, max_clusters + 1):
        kmeans = KMeans(n_clusters=n_clusters, random_state=random_state)

        kmeans.fit(delivery_coords)
        wcss.append(kmeans.inertia_)

    plt.figure(figsize=(10, 6))
    plt.plot(range(1, max_clusters + 1), wcss, marker='o', linestyle='--
        ', color='b')
    plt.title('Elbow Method for Optimal K')
    plt.xlabel('Number of Clusters')
    plt.ylabel('WCSS')
    plt.grid(True)
    plt.show()
```

Código A.19 Visualización de rutas y clusters de vehículos.

```
def plot_vehicle_routes(problem, final_routes):
    """
    Plots the routes of the vehicles, differentiating between lockers,
        shops, and homes.

    Parameters:
        problem (Problem): The problem instance containing delivery
            points and coordinates.
        final_routes (list): List of routes assigned to each vehicle.

    Returns:
        None: Generates and saves a visualization of vehicle routes.
    """

    fig, ax = plt.subplots(figsize=(12, 10))
    depot_coords = problem.dict_xy['DEPOT']
    num_vehicles = len(final_routes)
    colors = cm.get_cmap('tab10', num_vehicles)

    locker_coords, shop_coords, home_coords = [], [], []
    handles_labels = {}

    for vehicle_idx, route in enumerate(final_routes):
        x_coords, y_coords = [depot_coords[0]], [depot_coords[1]]
```

```
        for order_id, dp_id in route:
            if dp_id in problem.dict_xy:
                dp_coords = problem.dict_xy[dp_id]
                x_coords.append(dp_coords[0])
                y_coords.append(dp_coords[1])

                if dp_id.startswith('L'):
                    locker_coords.append(dp_coords)
                elif dp_id.startswith('S'):
                    shop_coords.append(dp_coords)
                else:
                    home_coords.append(dp_coords)

        x_coords.append(depot_coords[0])
        y_coords.append(depot_coords[1])
        vehicle_plot, = ax.plot(x_coords, y_coords, label=f'Vehicle {
            vehicle_idx + 1}', color=colors(vehicle_idx))
        handles_labels[f'Vehicle {vehicle_idx + 1}'] = vehicle_plot

    if locker_coords:
        locker_coords = np.array(locker_coords)
        scatter_lockers = ax.scatter(locker_coords[:, 0], locker_coords
            [:, 1], marker='s', color='blue')
        handles_labels['Lockers'] = scatter_lockers

    if shop_coords:
        shop_coords = np.array(shop_coords)
        scatter_shops = ax.scatter(shop_coords[:, 0], shop_coords[:, 1],
             marker='^', color='green')
        handles_labels['Shops'] = scatter_shops

    if home_coords:
        home_coords = np.array(home_coords)
        scatter_homes = ax.scatter(home_coords[:, 0], home_coords[:, 1],
             marker='o', color='purple')
        handles_labels['Homes'] = scatter_homes

    depot_plot = ax.scatter(depot_coords[0], depot_coords[1], color='
        black', marker='D', s=100, zorder=5)
    handles_labels['Depot'] = depot_plot

    ax.set_title('Vehicle Routes with Different Delivery Types')
    ax.set_xlabel('X Coordinate')
    ax.set_ylabel('Y Coordinate')
    ax.grid(True)

    fig.savefig("./Images/routes.png")
    plt.show()
```

Código A.20 Evaluación de la solución final generada por los vehículos.

```
def evaluate_final_solution(final_routes, problem, alpha, beta, gamma):
    """
    Evaluates the final routes generated by the vehicles and calculates
        key performance metrics.

    Parameters:
        final_routes (list): List of vehicle routes, each containing
            tuples (order_id, delivery_point).
        problem (Problem): The problem instance containing relevant
            delivery data.
        alpha (float): Weight for distance in the cost function.
        beta (float): Weight for waiting time in the cost function.
        gamma (float): Weight for priority in the cost function.

    Returns:
        normalized_priority (float): Normalized priority value for
            fitness evaluation.
        total_cost (float): Total cost computed based on distance and
            number of vehicles used.
        total_priority (float): Sum of priority values across all
            deliveries.
        not_served_count (int): Number of orders that were not
            successfully delivered.
        total_time (float): Total time spent across all routes.
        total_distance (float): Total distance covered by all vehicles.
    """

    total_distance = 0
    total_time = 0
    not_served_count = 0
    delivery_times = []

    # Combine all routes into a single list for evaluation
    clustered_routes = [delivery for route in final_routes for delivery
        in route]

    # Evaluate routes using the existing evaluation function
    priority, distance, total_time, not_served_count, delivery_times =
        evaluate_routes(problem, final_routes)

    # Compute total cost based on distance and vehicle usage
    total_cost = distance * problem.km_cost + len(final_routes) *
        problem.truck_cost

    # Compute the fitness score based on cost, priority, and unserved
        orders
    normalized_priority = fitness(problem, alpha, beta, total_cost,
        priority, not_served_count)
```

```
    return normalized_priority, total_cost, np.sum(priority),
        not_served_count, total_time, distance
```

Código A.21 Cálculo de los tiempos de entrega para cada pedido.

```
def calculate_delivery_times(final_routes, problem, depot_start_time=
    None):
    """
    Calculates the exact delivery times for each order based on travel
        distance, time windows, and delivery constraints.

    Parameters:
        final_routes (list): List of routes, where each sublist
            represents a vehicle's route
                            with tuples (order_id, location_id).
        problem (Problem): The problem instance containing delivery
            points, time constraints, and distances.
        depot_start_time (float, optional): Start time at the depot.
            Defaults to the earliest possible departure time.

    Returns:
        delivery_times (list): List of lists, where each sublist
            contains tuples
                              (vehicle_id, order_id, location_id,
                                  arrival_time, end_time).
    """

    delivery_times = []
    delivery_time_mapping = {
        'H': problem.dict_delivery_time['HOME'],
        'L': problem.dict_delivery_time['LOCKER'],
        'S': problem.dict_delivery_time['SHOP']
    }

    for vehicle_id, route in enumerate(final_routes):
        route_times = []
        current_location = 'DEPOT'
        current_time = depot_start_time if depot_start_time else problem.
            dict_twe['DEPOT']

        for order_id, dp_id in route:
            # Compute travel time based on real distances
            distance = problem.dict_distance.get((current_location, dp_id
                ), 10) # Default to 10 if missing
            travel_time = (distance / problem.truck_speed) * 60 # Convert
                 to minutes

            # Compute arrival time
```

```
            arrival_time = current_time + travel_time

            # Adjust for time window constraints
            waiting_time = 0
            if isinstance(problem.dict_twe[dp_id], list): # Multiple time
                windows (e.g., Stores 'S')
                valid_window = False
                for twe, twl in zip(problem.dict_twe[dp_id], problem.
                    dict_twl[dp_id]):
                    if twe <= arrival_time <= twl:
                        valid_window = True
                        break # Found a valid window
                    elif arrival_time < twe:
                        waiting_time = twe - arrival_time # Wait until the
                            time window opens
                        valid_window = True
                        break

                if not valid_window:
                    continue # Skip the delivery if no valid window is
                        found
            else: # Single time window (e.g., Homes 'H' or Lockers 'L')
                twe, twl = problem.dict_twe[dp_id], problem.dict_twl[
                    dp_id]
                if arrival_time < twe:
                    waiting_time = twe - arrival_time # Wait until the
                        window opens
                elif arrival_time > twl:
                    continue # Skip delivery if outside the time window

            # Get delivery time based on location type
            delivery_time = delivery_time_mapping.get(dp_id[0], problem.
                dict_delivery_time['DEFAULT'])

            # Compute end time
            end_time = arrival_time + waiting_time + delivery_time

            # Store the result
            route_times.append((vehicle_id, order_id, dp_id, arrival_time
                , end_time))

            # Update current time and location
            current_time = end_time
            current_location = dp_id

        delivery_times.append(route_times)

    return delivery_times
```

Índice de Figuras

2.1	Los problemas básicos de la clase VRP y sus interconexiones[6]	6
2.2	Algoritmo K-Means [15]	8
5.1	Cluster para 50 pedidos	23
5.2	Clusters para 200 pedidos	24
5.3	Clusters para 500 pedidos	24
5.4	Clusters para 1000 pedidos	25
5.5	Rutas de la prueba seleccionada para 50 pedidos	26
5.6	Rutas de la prueba seleccionada para 200 pedidos	26
5.7	Rutas de la prueba seleccionada para 500 pedidos	27
5.8	Rutas de la prueba seleccionada para 1000 pedidos	27
5.9	Horario de entregas para 50 pedidos	28
5.10	Horario de entregas para 200 pedidos	28
5.11	Horario de entregas para 500	28
5.12	Horario de entrega para 1000	28

Índice de Códigos

A.1	Clase Customer	35
A.2	Clase Order	35
A.3	Clase Location	36
A.4	Clase DeliveryPoint	36
A.5	Clase Truck	37
A.6	Clase Depot	37
A.7	Clase Route	38
A.8	Clase Problem	38
A.9	Generación de instancias de problema	39
A.10	Cálculo de distancia Euclidiana	41
A.11	Determinación del punto central	41
A.12	Generador de los pedidos	41
A.13	Generador delos destinos en tiendas	43
A.14	Lectura de archivo de problema	45
A.15	Aplicación de K-Means para la asignación de puntos de entrega	46
A.16	Asignación de entregas a vehículos considerando restricciones de capacidad y tiempo	47
A.17	Evaluación de las rutas generadas	50
A.18	Método del codo para determinar el número óptimo de clusters	51
A.19	Visualización de rutas y clusters de vehículos	52
A.20	Evaluación de la solución final generada por los vehículos	54
A.21	Cálculo de los tiempos de entrega para cada pedido	55

Bibliografía

[1] Alejandro Escudero-Santana, Jesús Muñuzuri, Antonio Lorenzo-Espejo, and María-Luisa Muñoz-Díaz. Improving e-commerce distribution through last-mile logistics with multiple possibilities of deliveries based on time and location. *Journal of Theoretical and Applied Electronic Commerce Research*, 17(2):507–521, 2022. Submission received: 15 February 2022 / Revised: 28 March 2022 / Accepted: 12 April 2022 / Published: 15 April 2022.

[2] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959.

[3] G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, 1964.

[4] B. Golden, S. Raghavan, and E. Wasil. *The Vehicle Routing Problem: Latest Advances and New Challenges*. Springer, 2008.

[5] Santiago Muelas, Antonio LaTorre, and José-María Peña. A distributed vns algorithm for optimizing dial-a-ride problems in large-scale scenarios. *Transportation Research Part C: Emerging Technologies*, 2015. Received 8 August 2014, Revised 15 December 2014, Accepted 11 February 2015, Available online 27 March 2015.

[6] P. Toth and D. Vigo. *The Vehicle Routing Problem*. SIAM, 2002.

[7] J. F. Cordeau, M. Gendreau, G. Laporte, J. Y. Potvin, and F. Semet. A guide to vehicle routing heuristics. *Journal of the Operational Research Society*, 2002.

[8] G. Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 1992.

[9] G. Laporte, M. Gendreau, J.-Y. Potvin, and F. Semet. Classical and modern heuristics for the vehicle routing problem. *International Transactions in Operational Research*, 21(4):479–506, 2014.

[10] G. B. Dantzig, D. R. Fulkerson, and S. M. Johnson. Solution of a large-scale traveling salesman problem. *Operations Research*, 1954.

[11] M. Gendreau, G. Laporte, and J. Y. Potvin. Metaheuristics for the vehicle routing problem. *INFORMS Journal on Computing*, 2002.

[12] Santiago Muelas, Antonio LaTorre, and José-María Peña. A variable neighborhood search algorithm for the optimization of a dial-a-ride problem in a large city. *Transportation Research Part B: Methodological*, 147:1–21, 2021. Computer Architecture Department, Facultad de

Informática, Universidad Politécnica de Madrid, Spain; Instituto Cajal, Consejo Superior de Investigaciones Científicas, Madrid, Spain.

[13] N. Sharma, L. M. Aggarwal, and S. Srivastava. Color image segmentation using k-means clustering approach. *International Journal of Computer Applications*, 2013.

[14] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Berkeley Symposium on Mathematical Statistics and Probability*, 1967.

[15] Antonio Richaud. K-means: Concepto y aplicaciones, 2021. Última consulta: 1 de abril de 2025.

[16] M. Wedel and W. Kamakura. *Market Segmentation: Conceptual and Methodological Foundations*. Kluwer Academic Publishers, 2000.

[17] R. Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 2005.

[18] P. Toth and D. Vigo. *Vehicle Routing: Problems, Methods, and Applications*. SIAM, 2014.

[19] L. Moccia, J. F. Cordeau, and G. Laporte. Vehicle routing with cross-docking. *Journal of the Operational Research Society*, 2012.