Programación de rutas de entrega para empresas distribuidoras con múltiples agentes y costos asimétricos (mATSP)

Luis Fernando Jalal Contreras
Juan Camilo Arrieta Berrocal

Universidad Pontificia Bolivariana
Escuela de Ingenierías y Arquitectura
Facultad de Ingeniería Industrial
Montería
2023

Programación de rutas de entrega para empresas distribuidoras con múltiples agentes y costos asimétricos (mATSP)

Luis Fernando Jalal Contreras
Juan Camilo Arrieta Berrocal

Trabajo de grado para optar al título de Ingeniero Industrial

Asesor:
Ader Luis Villar Ortega
M.Sc. en Ingeniería Industrial

Universidad Pontificia Bolivariana
Escuela de Ingenierías y Arquitectura
Facultad de Ingeniería Industrial
Montería
2023

Tabla de contenidos

1 Introducción **5**

2 Fundamentos **7**

3 Formulación Matemática **10**

4 Métodos de solución **13**

4.1 Métodos directos 13

4.2 Métodos indirectos 16

5 Metodología propuesta **22**

6 Análisis y conclusiones **30**

Referencias **33**

Anexos **36**

Anexo A. Código para el pre-procesamiento de los datos 36

Anexo B. Construcción de rutas a partir de la geolocalización de los clientes 39

Listado de Figuras

1 Ilustración de un problema con 3 vendedores 9
2 Transformación mATSP a ATSP 17
3 *Clustering* 20
4 Estrategia de Ruta "Espina de Pescado" 23
5 Localización en mapa de un subcojunto de 210 clientes 25
6 *Clustering* para 210 clientes usando PAM 27
7 *Clustering* para 210 clientes usando k-means balanceado 27
8 Primera ruta para el método PAM 28
9 Segunda ruta para el método PAM 29
10 Tercera ruta para el método PAM 29
11 Primera ruta para el método *k-means* balanceado 30
12 Segunda ruta para el método *k-means* balanceado 31
13 Tercera ruta para el método *k-means* balanceado 31

Listado de Tablas

1 Matriz de distancias original para un mATSP 18
2 Matriz de distancias ajustada considerando los valores mínimos en las filas (MF) y columnas (MC) 18
3 Distancias reducidas 19
4 Información de geolocalización para un subconjunto de clientes 24
5 Matriz de distancias para un subcojunto de clientes 25

1 Introducción

La programación eficiente de las rutas de entrega de productos es una preocupación crítica para las empresas en el ámbito de la distribución directa. Esta eficiencia puede medirse, entre otras, en términos de la distancia o el tiempo total empleado por todos los agentes que parten de un depósito y deben visitar un conjunto de clientes antes de volver a dicho depósito. Es común que esta planificación de rutas vaya acompañada de una serie de restricciones que varían según el contexto y la complejidad de la situación particular de cada empresa. Algunas de las restricciones más frecuentes se refieren a la capacidad limitada de cada agente o vehículo para transportar mercancías y a la disposición en tiempo de los clientes (conocido como ventanas de tiempo).

Los retrasos en las entregas y los errores en la planificación de rutas pueden tener un impacto negativo importante tanto en la satisfacción del cliente como en los costos logísticos. Para abordar este problema se pueden formular y resolver modelos matemáticos que, acompañados de un uso adecuado de herramientas de cómputo, ofrezcan soluciones rápidas y de calidad al problema de programación de rutas de distribución.

Junto con la relativa facilidad de entender y formular este tipo de problemas de forma exacta, viene la notoria dificultad para resolverlos. Estos problemas son combinatorios y el espacio de soluciones factibles crece muy rápidamente. Para el caso de un solo vendedor que debe visitar $n-1$[1] clientes y volver al depósito, el conocido problema del agente viajero o TSP, con distancias simétricas, el número de soluciones factibles es de $\frac{(n-1)!}{2}$, lo que alcanza una cifra astronómica de 1.52e64 con apenas 50 clientes. La gran mayoría de estas soluciones se pueden descartar rápidamente al encontrar cotas con métodos heurísticos sencillos como el de vecino más cercano, pero, encontrar la solución óptima es bastante retador y el problema está clasificado como *NP-Hard* [Wigderson, 2019].

Además de las formidables dificultades matemáticas que se pueden encontrar al tratar de resolver estos modelos, también se encuentran algunos aspectos técnicos adicionales que limitan la aplicabilidad de los resultados. Uno de estos es la falta de distancias reales y precisas entre clientes. En una ciudad, 2 puntos cercanos geográficamente en distancia euclidiana, pueden estar muy alejados en un sentido práctico ya que pueden estar separados por barreras como autopistas o ríos, o simplemente porque la ruta

[1] Se excluye el punto de partida o depósito.

de acceso no va en línea recta. De esta manera, aunque sea posible resolver algunas instancias del problema desde el punto de vista matemático, los resultados prácticos son poco satisfactorios al tener que usar distancias aproximadas con fórmulas como la del semiverseno o *haversine* (*"half-versed-sine"*) o algunas más sofisticadas que tienen en cuenta latitud y longitud de los puntos y la forma elipsoidal de la tierra para calcular distancias geodésicas [Hahsler and Hornik, 2023, Karney, 2013]. Este enfoque puede ser útil para trabajos donde el objetivo principal es desarrollar o mejorar algoritmos para la solución de los problemas de ruteo, pero se ve bastante limitado a la hora de resolver instancias reales.

Cuando se trata de implementar soluciones reales de estos modelos, hay otros aspectos que pueden afectar la operación. Algunas empresas cuentan con una flota propia de vehículos y/o repartidores por lo que son directamente responsables de la planificación de las rutas de entrega y la disponibilidad y disposición de estos recursos. En otros casos, el servicio logístico es subcontratado y esto trae a escena otras situaciones que deben considerarse. Un contratista puede estar a cargo de toda la operación logística de entrega, o puede ocurrir que la empresa cuente con varios contratistas pequeños a los que debe asignarles un subcojunto de los clientes para ser atendidos. En este último caso, que es común para distribuidoras pequeñas y medianas, cada contratista puede programar las rutas según su experticia, pero claramente la partición del conjunto de clientes será importante para la operación total.

Cuando las empresas cuentan con varios contratistas pequeños para la distribución de sus productos, a veces, por diversos motivos uno o varios de los contratistas, sin una debida anticipación, puede no estar disponible para operar durante un tiempo. Esto pone a las empresas distribuidoras en situaciones críticas en las que deben suplir rápidamente la falta de personal para entrega, normalmente a través de contratistas alternos que no conocen formas confiables de programar sus rutas, lo que incrementa la necesidad de estas distribuidoras y contratistas de contar con herramientas que les ayuden a programar las rutas de entrega de forma rápida.

En este trabajo, se propone un enfoque para la solución del problema anteriormente descrito, basado en modelos del agente viajero con múltiples vendedores y distancias asimétricas (mATSP), considerando la distribución real de la red de vías de un espacio geográfico particular y que ofrezca soluciones viables en un tiempo razonable para pequeñas y medianas empresas que trabajen con cientos de clientes. Los métodos propuestos se aplican al caso particular de una distribuidora tienda a tienda en la ciudad

de Montería con cerca de 1,200 clientes, pero que entrega o debe rutear al rededor de 200 clientes por día con 3 vehículos repartidores.

Los objetivos general y específicos planteados para este trabajo son:

1. **Objetivo general:**

   Desarrollar una herramienta que permita a las empresas distribuidoras programar de forma eficiente y en tiempo real sus rutas de entrega, basadas en modelos de ruteo de múltiples agentes con costos asimétricos (mATSP).

2. **Objetivos específicos:**

- Establecer un método de geolocalización de clientes y obtención de distancias de viaje reales teniendo en cuenta la red de vías
- Aplicar métodos de agrupamiento para dividir el problema de mATSP en múltiples modelos de ATSP
- Investigar y aplicar herramientas que permitan resolver de forma heurística el problema ATSP en cada grupo

El resto del documento se organiza de la siguiente manera: En la sección 2 se explica el problema general y algunas de sus aplicaciones en diferentes áreas; la sección 3 presenta algunas formulaciones matemáticas de los problemas tratados; en la sección 4 se encuentra una descripción de algunos enfoques comunes para resolver estos modelos matemáticos; la sección 5 contiene la metodología propuesta en este trabajo para abordar el problema descrito; por último la sección 6 presenta una discusión de los resultados y posibles extensiones del trabajo.

2 Fundamentos

El Problema del Agente Viajero (TSP, por sus siglas en inglés) se centra en encontrar la ruta óptima que pasa por un conjunto de ciudades, minimizando el costo total o la distancia recorrida. Se trata de encontrar una secuencia de visitas que cubra todas las ciudades exactamente una vez y regrese al punto de inicio. La complejidad del problema se relaciona con el número de posibles rutas, que se calcula mediante $\frac{(n-1)!}{2}$ para $n-1$ ciudades o clientes con distancias simétricas.

En temas de costos y distancias el TSP se puede presentar de forma simétrica y asimétrica: El TSP simétrico implica distancias y costos iguales en ambas direcciones entre ciudades, mientras que el TSP asimétrico permite que las distancias y costos varíen en cada dirección. En términos matemáticos, las distancias entre todas las posibles parejas de las n ciudades se registran en una matriz de distancias D, con elementos d_{ij}, donde i y j pueden tomar valores entre 1 y n, y los elementos de la diagonal principal (d_{ii}) son siempre cero. Si se consideran distancias simétricas, entonces $d_{ij} = d_{ji}$, de lo contrario $d_{ij} \neq d_{ji}$.

El TSP asimétrico (ATSP) se puede convertir en un TSP simétrico. Esto se logra duplicando el número de ciudades y agregando una ciudad ficticia para cada ciudad existente. Se establece un valor muy pequeño ($-\infty$) entre cada ciudad y su correspondiente ciudad ficticia, garantizando que cada ciudad siempre se incluya en la solución junto con su ciudad ficticia. Las distancias originales se utilizan entre las ciudades y las ciudades ficticias.

Entre las aplicaciones del TSP podemos encontrar [Hernandez-Saldana, 2010]:

- Conexión de cableado de computadoras: Se refiere a la conexión de componentes en placas de computadoras y la necesidad de encontrar un camino óptimo que conecte pines específicos en módulos sin permitir más de dos cables en un mismo pin. Esto se asemeja a la búsqueda de un camino que visite todos los puntos sin puntos de inicio o finalización predefinidos, en esencia, el problema consiste en encontrar un camino que pase por todos los puntos requeridos, similar a un camino Hamiltoniano, pero con distancias asimétricas y ubicaciones de inicio y fin predeterminadas.
- Selección de Pedidos en Almacenes: Este alberga la logística en un almacén para recoger una selección específica de productos y prepararlos para su envío. Se asemeja al TSP, donde los lugares de almacenamiento son puntos en un grafo, y las distancias representan el tiempo de desplazamiento entre ellos. El objetivo es encontrar la mejor ruta que recoja los productos minimizando el tiempo total requerido.

El Problema del Agente Viajero Múltiple (mTSP) representa una generalización del TSP en la cual se enfrenta al problema de optimizar múltiples rutas de viaje. En el mTSP, se involucra a varios vendedores o agentes, cada uno con la tarea de visitar un conjunto de ciudades exactamente una vez y luego regresar a su punto de partida,

con el objetivo de minimizar el costo total de los viajes realizados. [Cheikhrouhou and Khoufi, 2021]. La figura 1 muestra un bosquejo de este problema. Al igual que en el caso especial de un vendedor, el problema con múltiples vendedores puede considerar costos simétricos o asimétricos, en el último caso se puede denotar como mATSP.

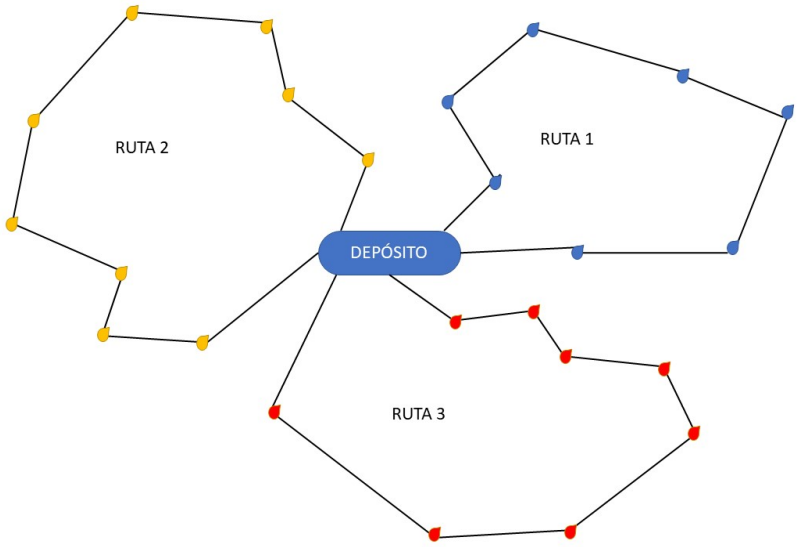

Figura 1: Ilustración de un problema con 3 vendedores

Este problema de optimización se puede encontrar en muchas situaciones prácticas. Por ejemplo, en el ámbito de la logística y el transporte, se utiliza para planificar rutas de entrega para flotas de camiones o vehículos de reparto, reduciendo los costos operativos asociados. En el campo de la robótica, el mTSP se aplica en la planificación de trayectorias para múltiples robots que deben llevar a cabo tareas de exploración, inspección o recopilación de datos [Bektas, 2006]. En gestión de emergencias se pueden aplicar modelos mTSP para coordinar los movimientos de equipos de rescate. En el contexto militar, se emplea para planificar misiones en las que varios agentes deben cumplir objetivos específicos en el menor tiempo posible. Otras aplicaciones incluyen [Hernandez-Saldana, 2010]:

- Diseño de Redes de Levantamiento para Sistemas Globales de Navegación por Satélite: Los sistemas de navegación global por satélite (GNSS) son sistemas satelitales que ofrecen cobertura a nivel mundial y tienen aplicaciones en monitoreo ambiental y agrícola. En el contexto del levantamiento, el objetivo es determinar las ubicaciones de puntos desconocidos tanto en la Tierra como en el espacio utilizando tecnología satelital. Con el fin de mejorar la eficiencia en

la coordinación de múltiples receptores y sesiones de observación, se aborda el problema como un mTSP, con la intención de encontrar la secuencia óptima de sesiones para los receptores.

- Programación de Entrevistas: Este Problema, basado en el mTSP, aborda la organización de encuentros entre agentes de viajes y proveedores de la industria turística en ferias y eventos. Cada agente debe visitar un conjunto específico de *stands* de proveedores, representados como ciudades. El objetivo es encontrar la programación que maximiza la interacción entre agentes y proveedores.

Los problemas de TSP y mTSP no incluyen restricciones sobre capacidad de los agentes o vehículos. El VRP (Vehicle Routing Problem) es una extensión del mTSP que incorpora estas restricciones adicionales. Desde sus primeros pasos en la década de 1950, el VRP se ha convertido en un pilar de la planificación logística en diversas industrias. Su objetivo principal es determinar las mejores rutas para una flota de vehículos, minimizando costos (como distancia o tiempo) y cumpliendo con restricciones específicas (como capacidad de carga o ventanas de tiempo). Las aplicaciones del VRP son variadas e incluyen la gestión de flotas de vehículos, la planificación de rutas de transporte público y la logística en general[Eksioglu et al., 2009].

3 Formulación Matemática

Las formulaciones matemáticas para el mATSP suelen expresarse mediante un modelo de programación entera de doble índice para un modelo de asignación con restricciones adicionales para la eliminación de subrutas [Bektas, 2006].

$$\mathcal{N} = \{1, 2, ..., n\} \text{ Conjunto de ciudades}$$

$$\mathcal{A} = \{(i, j) : i, j \in \mathcal{N}\} \text{ Conjunto de conexiones}$$

$$c_{ij} : \text{Costo de viajar desde el cliente i hasta el cliente j}, (i, j) \in \mathcal{A}$$

$$x_{ij} = \begin{cases} 1 \text{ si se utiliza la conexión } (i, j) \text{ en el recorrido.} (i, j) \in \mathcal{A} \\ 0 \text{ en caso contrario} \end{cases}$$

Teniendo en cuenta las definiciones anteriores de conjuntos, parámetros y variables, el modelo se puede expresar de la siguiente manera:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

sujeto a:

$$\sum_{j=2}^{n} x_{1j} = m \tag{1}$$

$$\sum_{j=2}^{n} x_{j1} = m \tag{2}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad \forall\ j = 2, ...n \tag{3}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \quad \forall\ i = 2, ...n \tag{4}$$

$$+ \text{ restricciones de eliminación de subrutas} \tag{5}$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in A \tag{6}$$

En esta formulación, las restricciones (3), (4) y (6) son las restricciones normales de un problema de asignación. Por otro lado, las restricciones (1) y (2) tienen como propósito asegurar que exactamente m vendedores inicien y finalicen en el nodo 1, que es el punto de partida (depósito). Las restricciones (1), (3) y (4) implican (2) pero suelen escribirse todas para una descripción completa del problema. Las restricciones (5) tienen la función de prevenir subrutas[2], que son rutas redundantes que se forman entre nodos intermedios y que no están conectados al punto de origen.

Existen varios criterios de eliminación de subrutas en la literatura. El primer grupo se refiere a las DFJ-SECs[3] de [Dantzig et al., 1954], inicialmente propuestas para el TSP, pero que también funcionan para un modelo con múltiples vendedores

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1, \quad \forall S \subseteq V \setminus \{1\}, \quad S \neq \emptyset \tag{7}$$

O de forma equivalente:

[2]Se conocen como SECs, por sus siglas en inglés: *Subtour Elimination Constraints*
[3]*Dantzig, Fulkerson and Johnson SECs*

$$\sum_{i \notin S} \sum_{j \in S} x_{ij} \geq 1, \quad \forall S \subseteq V \setminus \{1\}, S \neq \emptyset \tag{8}$$

Donde S es un subconjunto no vacío del conjunto de clientes, sin incluir el depósito. El funcionamiento de las restricciones (7) se basa en el hecho de que para todo conjunto de k nodos, cualquier red que contenga k o más arcos, necesariamente contendrá ciclos, por lo tanto, cada restricción garantiza que en cada subcojunto de clientes, existan a lo sumo k-1 conexiones. Las restricciones en (8) exigen que para cada subconjunto de clientes exista al menos un arco que viene desde afuera ($i \notin S$), previniendo así la formación de subrutas.

Aunque son sencillas, las restricciones (7) y (8) son imprácticas. Al tener que aplicarse a cada posible subconjunto, el número de restricciones en el modelo crece de forma exponencial con el número de clientes, lo que hace muy difícil poder resolver incluso la relajación lineal del problema. Una alternativa es agregar estas restricciones de forma dinámica, se resuelve el problema sin las SECs y luego se van añadiendo en la medida en que se necesiten, esto puede dar buenos resultados, teniendo en cuenta que en la mayoría de situaciones solo es necesario agregar una cantidad pequeña del total de las SECs.

Otros planteamientos para el conjunto de restricciones (5) terminan en una cantidad polinómica de SECs, como las conocidas MTZ-SECs[4], que agregan variables auxiliares u_i y tienen la siguiente forma

$$u_i - u_j + p x_{ij} \leq p - 1 \quad \forall\ 2 \leq i \neq j \leq n$$

Siendo p el número máximo de clientes que puede visitar cualquiera de los vendedores.

Se pueden encontrar en la literatura formulaciones alternativas para la eliminación de subrutas e incluso algunos planteamientos para el modelo que no se basan en el modelo general de asignación ([Gavish, 1976, Kara and Bektas, 2005]).

En [Bazrafshan et al., 2021] se comparan diferentes formulaciones de restricciones de eliminación de subrutas para el ATSP. Se examinan específicamente las formulaciones Danzig–Fulkerson–Johnson (DFJ), Miller–Tucker–Zemlin (MTZ) y Gavish–Graves (GG) para determinar cuál es la más adecuada en función de cinco criterios: el número

[4] *Miller, Tucker and Zemlin SECs*

de restricciones, el número de variables, el tipo de variables, el tiempo de resolución y las diferencias entre el valor óptimo y el valor relajado. La metodología utilizada incluye la aplicación del método de evaluación simultánea de criterios y alternativas (SECA) y el uso de software como CPLEX 12.8 y LINGO 11 para realizar los cálculos necesarios. Los resultados concluyen que la formulación de Gavish–Graves (GG) es la más apropiada para el ATSP.

4 Métodos de solución

4.1 Métodos directos

Los métodos directos son aquellos que intentan resolver el problema original sin ningún tipo de transformación o solución de algún problema equivalente. Aquí podemos encontrar métodos exactos y heurísticos. Los métodos exactos son enfoques computacionales utilizados en optimización de problemas que garantizan encontrar, si existe, la solución óptima. Estos métodos son precisos, pero pueden resultar costosos y llevar mucho tiempo su solución.

El método de Ramificacion y Acotamiento (B&B, por sus siglas en inglés) es un algoritmo de optimización y búsqueda que se utiliza para resolver problemas que requieren encontrar una solución exacta en un conjunto discreto de posibilidades. Su objetivo es encontrar las mejores soluciones a estos problemas dividiéndolos en grupos más pequeños. Cada uno de estos grupos tiene un valor mínimo predefinido para la mejor solución posible dentro de él. Luego, se elige el grupo con el valor mínimo nuevamente y se procede a dividirlo aún más. El proceso se detiene cuando se encuentra un subconjunto que contiene una solución cuyo valor es igual o menor que los límites inferiores de todos los nodos terminales. Asegura la convergencia cuando el número de soluciones es finito y el proceso de separación varía entre todas las soluciones, en un número finito de etapas. El B&B se utiliza en diversas aplicaciones de optimización, como la solución de problemas de enrutamiento, programación entera, diseño de redes y situaciones en las que es necesario encontrar la mejor solución dentro de un conjunto de opciones discretas. Además, este método se puede aplicar tanto a problemas deterministas como estocásticos. Por ejemplo, en [Gutjahr et al., 2000], el enfoque es optimizar actividades que dependen del tiempo en diversas aplicaciones, como la programación de proyectos y los procesos empresariales. En este caso, se

busca determinar la combinación óptima de medidas para cumplir con plazos, incluso cuando las duraciones de las actividades son inciertas. Se presenta un enfoque de optimización estocástica basado en el método B&B, que utiliza heurísticas para resolver subproblemas deterministas. A pesar del uso de heurísticas, este enfoque siempre converge hacia la solución exacta.

Con el objetivo de reducir el espacio de investigación y mejorar la búsqueda de soluciones enteras mediante el método B&B, la programación entera mixta utiliza cortes, lo que se conoce como ramificación y corte (B&C, por sus siglas en inglés), el cual combina el B&B con métodos de corte con la finalidad de eliminar soluciones inválidas en cada nodo del árbol de búsqueda. Estos métodos de corte identifican desigualdades válidas que restringen el espacio de búsqueda, lo que permite descartar soluciones que no pueden ser óptimas. Esto reduce significativamente la cantidad de nodos que deben explorarse y acelera la convergencia hacia la solución exacta. Dentro de las aplicaciones del B&C se encuentran los problemas de ruteo de vehículos, diseño de redes de comunicación, programación de horarios, optimización de cortes de material, diseño de circuitos electrónicos y problemas de empaquetamiento. En [Blasum and Hochstättler, 2000] se utiliza el método B&C para resolver un problema de ruteo de vehículos con restricción de capacidad, este presenta un algoritmo de optimización combinatoria basado en planos de corte y ramificación para resolver de forma exacta el problema CVRP. para esto es necesario la identificación de desigualdades válidas mediante heurísticas de separación.

En la teoría y en la práctica, se presentan problemas con gran número de variables y restricciones que exceden la capacidad del método B&B para resolverlos. El método de Ramificación y Generación de Variables (B&P, Por sus siglas en inglés) se presenta como una solución para resolver problemas con muchas variables de decisión. En lugar de trabajar con todas las variables al mismo tiempo, este método genera dinámicamente un conjunto inicial de variables. A medida que resuelve el problema, agrega o elimina variables según sea necesario. B&P se aplica en una variedad de contextos, como problemas de ruteo de vehículos, diseño de redes de telecomunicaciones, programación de horarios y más. Por ejemplo, en [Akca et al., 2009], se describe un algoritmo B&P que resuelve un problema combinado de localización y ruteo con restricciones de capacidad mediante generación de columnas e incorporación de ramificación para exploración exhaustiva.

Los métodos heurísticos son aquellos que presentan soluciones aproximadas de manera

más rápida, estos tienen enfoques de solución para problemas que se basan en reglas o estrategias prácticas, sin embargo, no garantizan una solución óptima.

El Método de vecino más cercano, es una heurística que comienza en un nodo y se mueve al más cercano no visitado hasta cubrir todos, se utiliza en [Rahma et al., 2020] para diseñar rutas de recolección de residuos sólidos en una ciudad de indonesia. Este método considera múltiples ubicaciones de puntos de transferencia de residuos y depósitos, generando varias rutas alternativas debido a algunas distancias iguales. Se establecen dos escenarios basados en el volumen de residuos y la capacidad restante del vehículo, obteniendo varias decisiones alternativas con diferentes tiempos y distancias totales. La mejor alternativa selecciona el punto de transferencia con el volumen restante más cercano al de la capacidad del vehículo.

Otro método heurístico usado para resolver el TSP es el de Inserción de menor costo (CI), presentado en [Hignasari and Mahira, 2018]. Este algoritmo comienza con una ciudad inicial y, en cada paso, selecciona la ciudad no visitada que requiere la menor distancia adicional para ser insertada en la ruta existente.

También se utiliza el método Búsqueda Tabú, un procedimiento de búsqueda local que examina el espacio de soluciones y se mueve desde una solución presente hacia la mejor solución adyacente dentro de un conjunto específico de soluciones permitidas. Este método, aplicado por [Kergosien et al., 2010], es utilizado para encontrar las rutas en el TSP, permitiendo la minimización de costos y cumpliendo con las restricciones de tiempo y prioridades. Este procedimiento se distingue por su habilidad para examinar el espacio de soluciones, moviéndose desde una solución presente hacia la mejor solución adyacente dentro de un conjunto específico de soluciones permitidas, llamado vecindario. Para prevenir ciclos y mejorar la exploración del espacio de búsqueda, la Búsqueda Tabú emplea una lista tabú que registra ciertos atributos de las soluciones previamente evaluadas, evitando que sean considerados nuevamente durante un número específico de iteraciones.

En [Necula et al., 2015], se presentan y evalúan algoritmos basados en colonias de hormigas para resolver el problema del viajante múltiple con un solo depósito (SD-mTSP) desde una perspectiva bi-objetivo. Los objetivos son minimizar el costo total de los subtours recorridos y la diferencia entre el costo del subtour más largo y el más corto (amplitud). Se proponen varios algoritmos: P-ACO (*Pareto Ant Colony Optimization*), una variante del sistema de colonia de hormigas (ACS) que usa múltiples matrices de feromona y una matriz heurística para optimización multi-objetivo; MACS

(*Multiple Ant Colony System*), que utiliza una matriz de feromona y múltiples funciones heurísticas, una por cada objetivo; MoACO/D-ACS (*Multi-objective Ant Colony Optimization based on Decomposition combined with ACS*), que se basa en la descomposición del problema original en subproblemas escalares que son optimizados simultáneamente por subcolonias de hormigas; y g-MinMaxACS, una variación del ACS estándar que usa el objetivo MinMax para abordar SD-mTSP de forma bi-objetivo, minimizando indirectamente el costo total y la amplitud de los subtours. Estos algoritmos se evaluaron experimentalmente en estándares de comparación de SD-mTSP.

Como resultado, [Necula et al., 2015] concluye que el algoritmo g-MinMaxACS presenta el mejor desempeño general, logrando soluciones de compromiso diversas y cercanas al frente de Pareto de referencia. El estudio resalta que un enfoque bi-objetivo es recomendado para SD-mTSP y que los algoritmos basados en colonias de hormigas propuestos constituyen alternativas prometedoras para obtener buenas aproximaciones del frente de Pareto.

En [Singh et al., 2018], se presentó un nuevo operador cruzado y un método de inicialización de población, diseñados para la solución efectiva mTSP mediante la aplicación de un algoritmo genético (GA). Haciendo especial mención a otras aplicaciones el mTSP tales como la optimización de procesos de laminación en caliente, la distribución equitativa de cargas de trabajo, la configuración de redes topográficas para sistemas de navegación global por satélite y la programación de rutas de autobuses escolares.

En [Li et al., 2015] se implementa un nuevo enfoque llamado problema del agente viajero coloreado (cTSP). Este modelo involucra dos tipos de grupos de ciudades: uno para ciudades exclusivas de un color asignado a cada vendedor y otro para ciudades compartidas con varios colores que pueden ser visitadas por todos los vendedores. Se demuestra que cTSP presenta una complejidad *NP-Hard* y que un mTSP y un TSP son casos especiales de cTSP. Para resolver cTSP, se propone un algoritmo genético (GA) que utiliza una codificación de doble cromosoma y se analiza el espacio de solución correspondiente. Luego, se mejoran las capacidades del GA mediante la incorporación secuencial de operaciones de búsqueda voraz, escalada de colina y recocido simulado, con el objetivo de lograr un mejor rendimiento. A través de experimentos, se revela la limitación de los métodos de solución exacta y se comparan las capacidades de los GA presentados. Los resultados sugieren que el GA con recocido simulado ofrece las

soluciones de mayor calidad, mientras que el GA de escalada de colina representa una opción que equilibra adecuadamente la calidad de la solución y el tiempo de cálculo.

4.2 Métodos indirectos

Los métodos indirectos son aquellos que primero hacen un cambio o transformación en el problema, con la finalidad de facilitar su solución. El mATSP es una extensión del clásico TSP, en el que se involucran múltiples vendedores y costos asimétricos. La característica de asimetría se refiere al hecho de que los costos o distancias entre las ciudades pueden variar en ambas direcciones, lo que significa que la distancia de la ciudad A a la ciudad B puede ser diferente a la distancia de la ciudad B a la ciudad A.

El mATSP se aplica en diversos contextos, como transporte, recopilación de datos, aplicaciones militares, misiones de emergencia y gestión de desastres, este se utiliza para planificar rutas en campos como la logística y distribución, el transporte público, la robótica móvil, redes de sensores inalámbricos, aplicaciones militares, misiones de rescate y emergencias, planificación de rutas de drones y vehículos aéreos no tripulados (UAVs) [Cheikhrouhou and Khoufi, 2021].

La complejidad del mATSP es No determinista Polinomial duro (*NP-Hard*). Esto significa que no se conoce un algoritmo eficiente que pueda resolver el mATSP en tiempo polinómico en función del tamaño del problema. En otras palabras, no existe una solución que pueda encontrar la respuesta óptima para todas las instancias del mATSP en tiempo razonable, por ende, el enfoque de solución involucra el uso de algoritmos de optimización y técnicas heurísticas como kNN, *Cluster-Based*, Búsqueda Tabú entre otras. Para simplificar la solución del problema, el mATSP puede convertirse en el ATSP al considerar solo a un vendedor como se menciona en [Sarin et al., 2014], Para lograr esta transformación, se agregan $m-1$ depósitos ficticios a la matriz de distancias del mATSP como se explican en [Ahmed and Al-Dayel, 2020], lo que resulta en el ATSP con un mayor número de ciudades. Cada depósito ficticio se asocia con distancias infinitas a otros depósitos. Esta conversión se ha propuesto para diversos casos de TSP simétricos y asimétricos, adaptándose al número de ciudades y vendedores.

Para ilustrar esta transformación del mATSP al ATSP utilizamos un ejemplo con $n = 10$ (representando 10 ciudades o clientes) y $m = 2$ (representando 2 vendedores). La

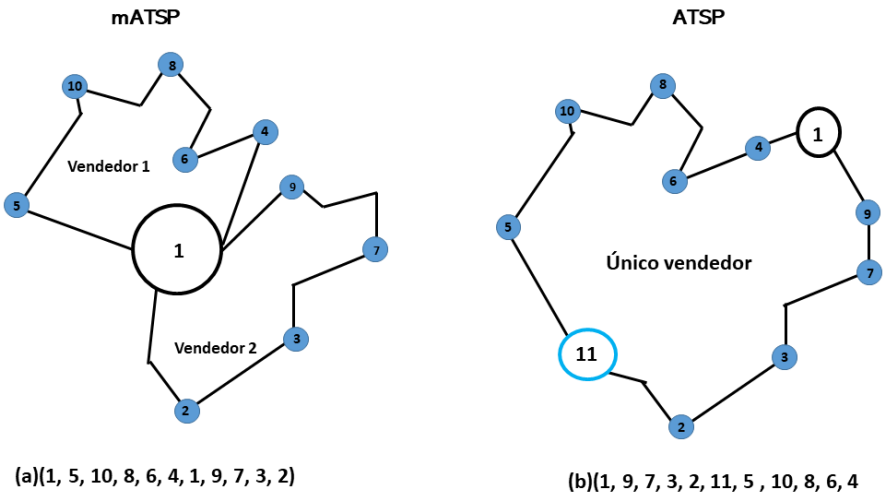

Figura 2: Transformación mATSP a ATSP

Figura 2 muestra la equivalencia entre estos dos problemas.

A partir de la matriz original, generada de forma aleatoria para un mATSP en la Tabla 1, obtenemos una matriz de distancias equivalente para el problema con un solo agente o vendedor, agregando $m - 1 = 2 - 1 = 1$ filas y columnas adicionales, como se muestra en la Tabla 2

Tabla 1: Matriz de distancias original para un mATSP

Ciudades	1	2	3	4	5	6	7	8	9	10
1	999	5	7	5	6	5	4	2	3	2
2	7	999	2	7	1	6	4	5	8	7
3	7	1	999	7	5	3	7	7	3	6
4	8	8	9	999	2	3	8	5	6	3
5	5	2	5	7	999	3	6	7	6	6
6	5	8	9	7	3	999	7	6	6	5
7	1	8	7	8	9	2	999	8	8	4
8	1	9	9	9	1	6	7	999	6	7
9	5	1	5	6	7	5	7	6	999	9
10	9	9	4	8	4	1	2	5	7	999

También se modifican los datos restando el valor mínimo de cada fila a sus elementos asociados. Luego, se aplica el mismo procedimiento en la matriz resultante pero ahora por columnas. La suma de los valores mínimos de fila y los valores mínimos de columna se conoce como la tendencia de la matriz.

Tabla 2: Matriz de distancias ajustada considerando los valores mínimos en las filas (MF) y columnas (MC)

Ciudades	1	2	3	4	5	6	7	8	9	10	11	MF
1	997	3	5	3	4	3	2	0	1	0	997	2
2	6	998	1	6	0	5	3	4	7	6	6	1
3	6	0	998	6	4	2	6	6	2	5	6	1
4	6	6	7	997	0	1	6	3	4	1	6	2
5	3	0	3	5	997	1	4	5	4	4	3	2
6	2	5	6	4	0	996	4	3	3	2	2	3
7	0	7	6	7	8	1	998	7	7	3	0	1
8	0	8	8	8	0	5	6	998	5	6	0	1
9	4	0	4	5	6	4	6	5	998	8	4	1
10	8	8	3	7	3	0	1	4	6	998	8	1
11	997	3	5	3	4	3	2	0	1	0	997	2
MC	0	0	1	3	0	0	1	0	1	0	0	**23**

En este caso, la tendencia de la matriz es igual a 23 (17 + 6). Como resultado de este proceso, la matriz de distancias resultante es no negativa y contiene al menos un cero en cada fila y columna. En problemas de asignación, al igual que en el TSP, si se suma o resta una constante a todos los elementos de una columna (o fila) en la matriz, una asignación que minimiza la distancia total en una matriz también reducirá la distancia total en la otra [Ahmed and Al-Dayel, 2020]. Dado que el mATSP es una generalización del ATSP, es suficiente abordar el problema utilizando la matriz de distancias reducidas, como se muestra en la Tabla 3

Tabla 3: Distancias reducidas

Ciudades	1	2	3	4	5	6	7	8	9	10	11
1	997	3	4	0	4	3	1	0	0	0	997
2	6	998	1	3	0	5	2	4	6	6	6

Ciudades	1	2	3	4	5	6	7	8	9	10	11
3	6	0	998	3	4	2	5	6	1	5	6
4	6	6	7	994	0	1	5	3	3	1	6
5	3	0	3	2	997	1	3	5	3	4	3
6	2	5	6	1	0	996	3	3	2	2	2
7	0	7	6	4	8	1	997	7	6	3	0
8	0	8	8	5	0	5	5	998	4	6	0
9	4	0	4	2	6	4	5	5	997	8	4
10	8	8	3	4	3	0	0	4	5	998	8
11	997	3	5	0	4	3	1	0	0	0	997

Otro enfoque que se basa en cierta transformación del problema, es el que reduce el problema con múltiples agentes a múltiples problemas con 1 solo agente (ATSP). En términos de datos, se encuentran patrones intrínsecos de similitud o proximidad entre los clientes o puntos. Estos presentan características parecidas y tienden a agruparse debido a que se parecen más entre sí que con otros datos en el conjunto.

El *clustering* es un proceso que consiste en dividir un conjunto de datos en subconjuntos más pequeños o grupos basándose en la similitud entre los elementos (Figura 6). La premisa es que los elementos dentro de un mismo grupo presentan mayor similitud entre sí que con los elementos pertenecientes a otros grupos. En [Mariescu-Istodor et al., 2021] se abordan problemas de VRP considerando las condiciones del mundo real, como la red de carreteras, las restricciones de velocidad y la congestión del tráfico. Los programas actuales de VRP solo son efectivos en escenarios pequeños que involucran unos pocos cientos de destinos. En el trabajo antes citado, se propone una solución llamada VRPDiv, que mejora la capacidad de cualquier programa de VRP, permitiéndole lidiar con conjuntos de destinos mucho más grandes, incluso hasta diez mil destinos, al dividirlos en grupos más manejables. VRPDiv es versátil y puede adaptarse a diferentes situaciones de VRP, utilizando una variedad de algoritmos de agrupación seleccionados en función de las características específicas de cada instancia.

En [Mariescu-Istodor et al., 2021] se comienza el proceso adquiriendo información sobre cuánto tiempo se tarda en viajar entre los diferentes destinos, lo que resulta útil en todas las etapas posteriores del proceso. Luego, selecciona un enfoque específico de agrupación en función de las características particulares de la situación que se está analizando. A

continuación, se divide la situación en grupos separados, cada uno de los cuales es un problema de VRP más pequeño y manejable. Estos grupos son de un tamaño reducido y se pueden resolver de manera eficiente utilizando métodos más tradicionales.

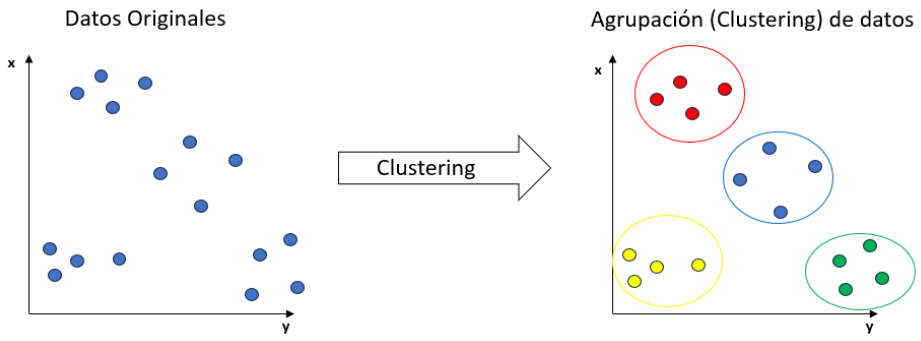

Figura 3: *Clustering*

Para resolver un mATSP con esta metodología, se busca crear grupos basados en los tiempos o distancias estimados de viaje entre los destinos, y se asignan agentes a estos grupos para resolver subproblemas de ATSP de manera independiente. Los algoritmos de agrupamiento generalmente necesitan saber de antemano cuántos grupos se desean (K) y este valor no debe exceder el número de agentes disponibles para la operación.

En [Mariescu-Istodor et al., 2021] se presentan algunos algoritmos para *clustering*

- *K-Means*: Este es un algoritmo que agrupa datos en grupos basándose en su similitud. Comienza eligiendo K puntos como centroides iniciales y asigna cada dato al centroide más cercano. Luego, recalcula los centroides y repite el proceso hasta que las asignaciones dejen de cambiar significativamente.
- *Balanced k-Means*: Es una variante del algoritmo k-Means que se utiliza para agrupar datos en grupos equilibrados, asegurando que tengan tamaños similares en lugar de asignar puntos al centroide más cercano, se asignan a centroides que no han alcanzado un tamaño máximo previamente definido.
- *X-Means*: Es una extensión del algoritmo *k-Means* que automáticamente determina el número óptimo de grupos. Comienza con un valor bajo de K, aplica *k-Means* y utiliza medidas de bondad para ajustar el número de grupos.
- *X-Means + Balancing*: Es una extensión mejorada del algoritmo *X-Means*, que automatiza la determinación del número óptimo de grupos en problemas de agrupamiento de datos. En esta mejora, se introduce un tercer paso conocido

como “intercambio de centroides”. El enfoque principal de esta modificación es reducir el tamaño máximo de los grupos mediante la reubicación planificada de elementos dentro de grupos con pocos elementos hacia aquellos con una alta densidad de elementos. Esto conlleva a la división de los grupos más grandes y la fusión de aquellos más pequeños con sus grupos vecinos.

- *SLINK* (*Single-Linkage Clustering* o agrupamiento de enlace simple): Es un método utilizado para agrupar los datos en función de la distancia entre ellos. En *SLINK*, se considera la distancia entre dos grupos como la distancia mínima entre cualquier par de puntos de datos, uno en cada grupo. Esto significa que los grupos se fusionan si tienen al menos un par de puntos cercanos entre sí, tiene un enfoque de “agrupamiento jerárquico aglomerativo”, lo que significa que comienza con cada punto como su propio grupo y luego fusiona gradualmente los grupos más cercanos hasta que se forme una jerarquía de grupos
- *PAM* (*Partition Around Medoids*): A diferencia de otros algoritmos de agrupamiento, como el *k-Means*, PAM se centra en encontrar medoides en lugar de centroides. Un medoide es un objeto del grupo para el cual la suma de sus disimilitudes a todos los demás miembros del grupo, es mínima [Pandya and Saket, 2020]. PAM busca mejorar la calidad de los grupos mediante intercambios de medoides y es más resistente a la presencia de valores atípicos o ruido en los datos. El proceso de K-Medoid implica la selección inicial de K medoides, la asignación de objetos de datos a los medoides más similares y, si es beneficioso, se intercambian medoides por objetos no medoides para mejorar la formación de los grupos.

5 Metodología propuesta

En esta sección se detalla la metodología propuesta en el trabajo para abordar los problemas de planificación de rutas. Se establece un marco general de fases que pueden aplicarse a una gran variedad de empresas en diferentes entornos, siempre que dichos problemas de entrega puedan representarse como un mATSP. A medida que se describe el proceso, se irá aplicando a un caso particular para poder validar los resultados del método y las herramientas utilizadas.

El estudio de caso se refiere a la empresa ANCOR, una empresa distribuidora con sede en la ciudad de Montería, Córdoba, especializada en la comercialización y distribución de forma directa de productos de hogar a pequeños comercios (tiendas). Cuenta con una red de aproximadamente 4,000 clientes, abarcando no solo el departamento de Córdoba, sino también Sucre y una parte de Antioquia. La aplicación se limitará a los clientes en la ciudad de Montería que son cerca de 1,200.

La empresa no posee una flota de vehículos de reparto propia. En su lugar, depende de contratistas para llevar a cabo las entregas de sus productos a los diversos puntos de venta en Montería y sus alrededores. Este enfoque ha resultado en problemas recurrentes de entregas tardías, lo que ha generado insatisfacción entre los clientes, así mismo, uno de los problemas clave es que los conductores de los vehículos de reparto no están familiarizados con las rutas más eficientes ni con las ubicaciones exactas de algunos clientes. Por diversos motivos, la rotación de los contratistas suele ser alta, por lo que la empresa requiere de un método rápido y confiable para programar sus rutas.

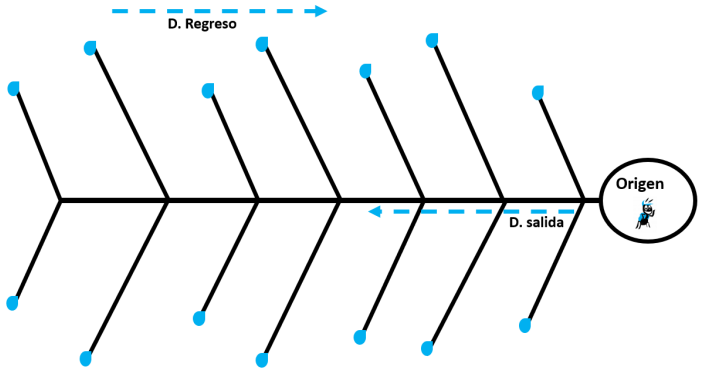

Figura 4: Estrategia de Ruta “Espina de Pescado”

Para abordar la complejidad de las entregas y diseñar un enfoque efectivo, la empresa ha desarrollado su propia estrategia de programación de rutas, conocida como la estrategia de “espina de pescado”. Esta estrategia se basa en la idea de que la ruta de reparto se asemeja a un pez con una cabeza (el punto de partida) y una cola (el punto final) (Figura 4). La ruta se traza siguiendo la mitad del pez y se realizan entregas desde el cliente más lejano al más cercano. Este enfoque ha sido efectivo en cierta medida, y la empresa ha acumulado datos a lo largo de los años, lo que les ha permitido construir algunas rutas empíricas.

Para resolver los problemas logísticos relacionados a la planificación de rutas en empresas distribuidoras pequeñas y medianas, tal como se ha establecido en los

Tabla 4: Información de geolocalización para un subconjunto de clientes

nombre	barrio	dir	lat	lng
ANCOR	CENTRO	CLL 40 #1B-67	8.764384	-75.88121
KIOSCO VERSALLES	VERSALLES	CLL 43 # 14 - 48	8.759219	-75.87219
FRUTAS Y VERDURAS EL PAISA	LOS ANGELES	CRA 14D #44-09	8.760844	-75.87130
MINIMERCADO EL PAISA	VILLA CAMPESTRE	CRA 15 #43-38	8.757949	-75.86891
TIENDA PORTAL DE LA CANDELARIA	MONTEVERDE	CRA 15C CLL 42A	8.756945	-75.86821
MERCATIENDA LA FORTUNA	CASTELLANA	CRA 8 #57-60	8.768415	-75.86666
DOMI YA MONTEVERDE	MONTEVERDE	CLL 44 #16C-05	8.754579	-75.86615

objetivos de este trabajo, es esencial contar con un conjunto de herramientas computacionales que faciliten la labor. Un lenguaje de programación es imprescindible y aquí usamos `R` [R Core Team, 2023]. Otra de esas herramientas debe permitir gestionar datos geoespaciales que proporcionen información sobre la ubicación precisa de clientes en términos de latitud y longitud. Esta información geoespacial es importante ya que permite calcular distancias y tiempos de viajes más ajustados a la realidad al considerar aspectos como la red de vías, características del tráfico vehicular y sentido de circulación de las vías. Todo esto, unido a la capacidad de diseñar rutas en tiempo real, aumenta la aplicabilidad de las soluciones ofrecidas.

La primera fase consta de un levantamiento de las coordenadas de latitud y longitud, junto con información complementaria tal como la dirección de cada cliente. Esto en general es un proceso que depende de cada empresa, que debe llevar registro de la posición geográfica exacta de sus clientes. Para el estudio de caso de este trabajo, se empezó con una base de datos de cerca de 1,200 clientes. Fue necesario un preprocesamiento de los datos facilitados inicialmente por ANCOR para obtener una base de datos *limpia* tal como se muestra en la tabla Tabla 4. La estrategia usada en esta fase puede variar mucho dependiendo de la información inicial con la que se cuente y del estado o calidad de los datos suministrados. En el Anexo A se encuentra el código usado para este caso específico.

Contando con una base de datos de clientes debidamente geolocalizados, se pueden construir mapas interactivos que facilitan la presentación de las rutas planificadas. Se usa Leaflet que es una librería de código abierto que sirve para construir mapas interactivos y que está escrita en `Javascript`, haciéndola muy adecuada para el desarrollo de aplicaciones web. Para aprovechar esta librería desde `R` se usa la API `leaflet` [Cheng et al., 2023]. La construcción y visualización con `leaflet` requiere de datos de mapas base, hay varias opciones disponibles y aquí se usa OSM (*Open Street Map*) que también es de código abierto. Con estas herramientas se puede lograr

visualizar un grupo de clientes, tal como se muestra en La figura 5.

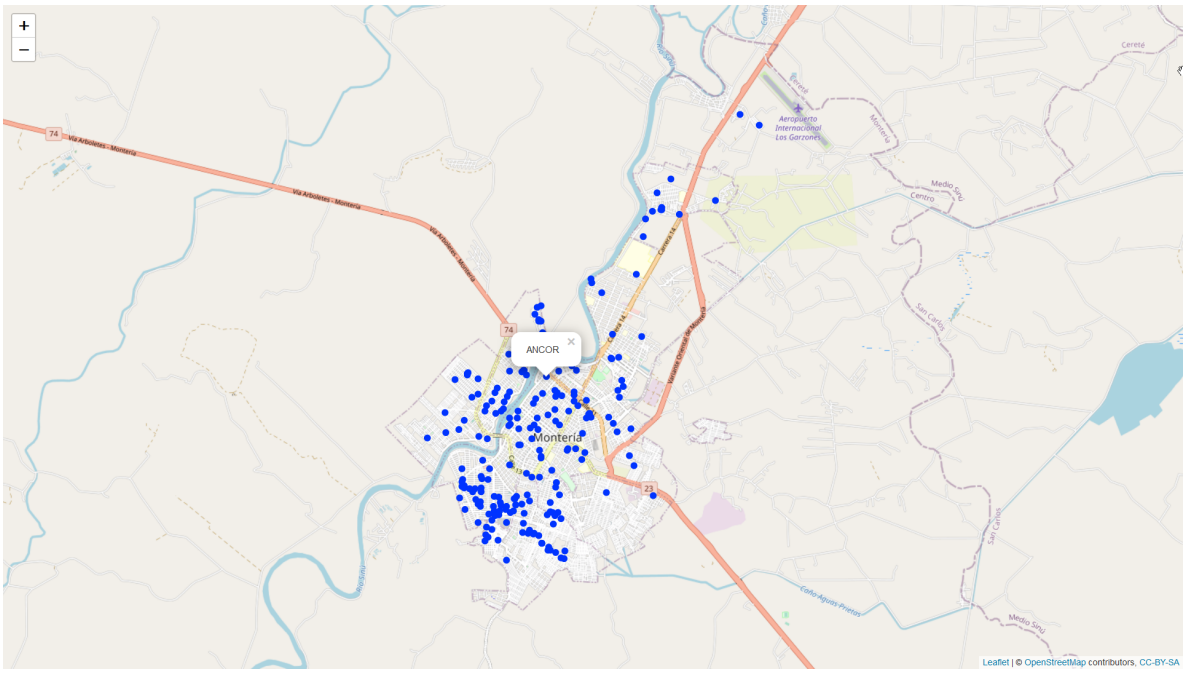

Figura 5: Localización en mapa de un subcojunto de 210 clientes

Cuando los clientes que deben rutearse son identificados, viene una parte clave del proceso que es estimar las distancias reales entre todo par de puntos. Para llevar a cabo esta tarea, se usa el servicio abierto[5] de OSRM (*Open Source Routing Machine*) que implementa modernos algoritmos en `C++` para el cálculo de rutas más cortas en redes de carreteras. Como ya se ha mencionado, la ventaja de esto es que se obtienen distancias reales que tienen en cuenta la configuración de la red de carreteras. Nuevamente, es necesaria una API o paquete para usar esta herramienta desde `R`, la librería `osrm` [Giraud, 2022] cumple con este propósito. El límite de la API es de $10,000$ requerimientos, por lo que se programa de tal forma que se puedan hacer varias solicitudes para no exceder el límite y poder completar la matriz de distancias que se necesita. La Tabla 5 muestra un subconjunto de estas distancias calculadas usando el servicio de OSRM.

Al completar el cálculo de las distancias entre todos los clientes, se cuenta con la información de entrada para resolver el modelo mATSP. Tal como se expuso en las secciones anteriores, aunque el volumen de clientes no es demasiado grande, resolver el modelo con cientos de clientes puede tomar un tiempo considerable. Teniendo esto en

[5]Con ciertos límites

Tabla 5: Matriz de distancias para un subcojunto de clientes

	1	2	6	10	15	16	22	24	28	36
1	0	1790	1984	2422	2367	2576	3831	3143	2741	4430
2	1662	0	1638	999	944	1152	3485	2797	1038	3722
6	2600	2660	0	1333	964	765	2456	1100	2239	5299
10	2818	1239	1394	0	289	562	3602	1732	920	4878
15	3369	1296	1139	282	0	306	3346	1713	1188	4935
16	3063	1781	833	556	529	0	3040	1517	1462	5763
22	4124	4184	2551	3983	3627	3428	0	1990	5135	6823
24	3440	3501	1212	1756	1759	1559	1881	0	2662	6140
28	2954	1561	2405	920	1189	1397	4252	2639	0	5014
36	4004	3788	5044	4847	4792	5432	6891	6203	5166	0

cuenta, se opta por resolver el modelo de forma heurística dando prioridad al objetivo de lograr soluciones rápidas en tiempo real.

Un enfoque de solución es convertir el mATSP a un ATPS y luego a un clásico TSP, hay formas de hacer esto, tal como se explicó en la sección 4.2 y la ventaja radica en que se pueden encontrar en la literatura más algoritmos eficientes para un TSP que algoritmos que resuelvan directamente un mATSP. Los resultados muestran, sin embargo, que este camino lleva a soluciones donde las rutas suelen estar muy desbalancedas, lo que es indeseable y limita la aplicación de los resultados.

De esta manera, aplicamos una metodología similar a [Mariescu-Istodor et al., 2021], que consiste en dividir el conjunto de clientes en subgrupos (*cluster*) en los que se debe resolver un problema de ruteo de vehículos de manera individual para cada subgrupo. Para lograrlo de forma eficiente, se usa la librería `cluster` [Maechler et al., 2022] que implementa algunos algoritmos modernos para *clustering*.

Se usa el algoritmo PAM[6] para realizar el *clustering* de los clientes a partir de una matriz de disimilitud. Una matriz de disimilitud contiene las "diferencias" entre cada par de puntos o clientes y debe ser una matriz simétrica. Para el conjunto de clientes, es claro que la matriz de disimilitud es la matriz de distancias, pero esta matriz no es simétrica, así que antes de aplicar PAM se convierte la matriz de distancias original D a una matriz simétrica D' mediante $d'_{ij} = d'_{ji} = \min\{d_{ij}, d_{ji}\}$. La razón de esto es por comprobación empírica, se comparó el resultado usando el máximo de cada distancia y para se obtenían mejores resultados al usar el mínimo.

Como se describió en la sección anterior, se ha demostrado en la literatura que PAM es

[6] *Partitioning Around Medoids*

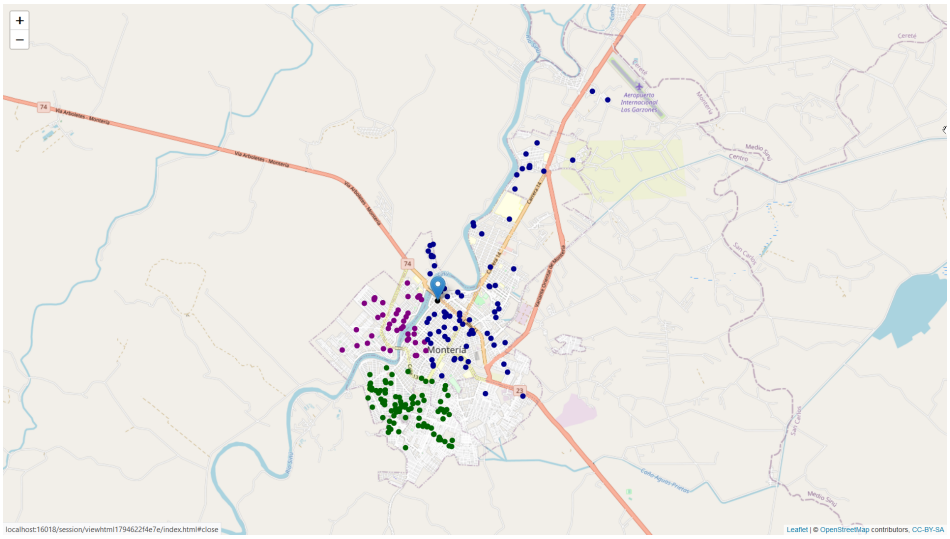

Figura 6: *Clustering* para 210 clientes usando PAM

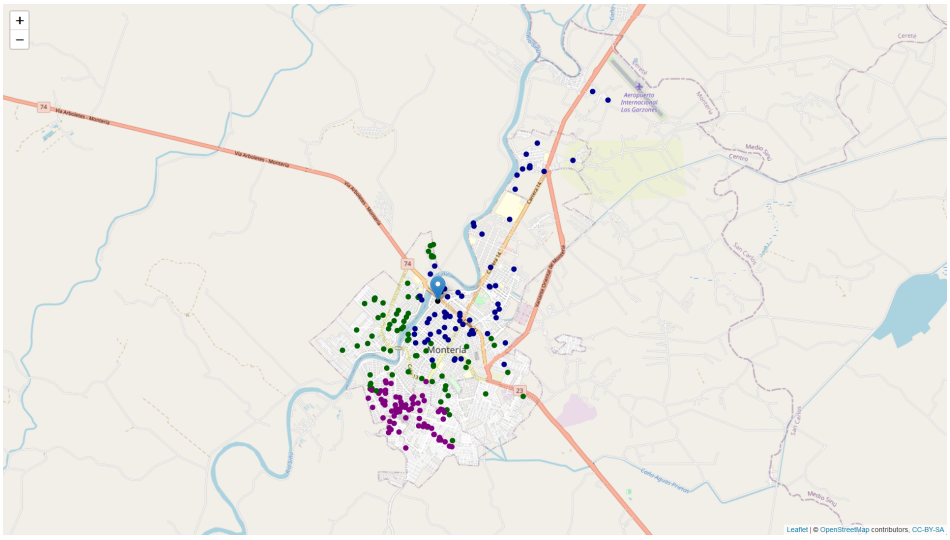

Figura 7: *Clustering* para 210 clientes usando k-means balanceado

un método más robusto a la presencia de valores atípicos en los datos, por lo que puede ser más adecuado que otros métodos de *clustering* en situaciones donde el área geográfica de interés tenga clientes en la periferia. Por otra parte, en ocasiones los grupos obtenidos muestran cierto desbalance en cuanto al número de clientes asignados por lo que se puede aplicar de manera alternativa el método de *Balanced k-means* implementado en la librería `anticlust` [Papenberg and Klau, 2021], en donde se obtienen grupos equilibrados. La Figura 6 muestra un conjunto de 210 clientes de ANCOR agrupados usando PAM y considerando un total de 3 vehículos, que debe ser el mismo número de grupos a crear. La Figura 7 muestra el resultado del agrupamiento usando *Balanced k-means*

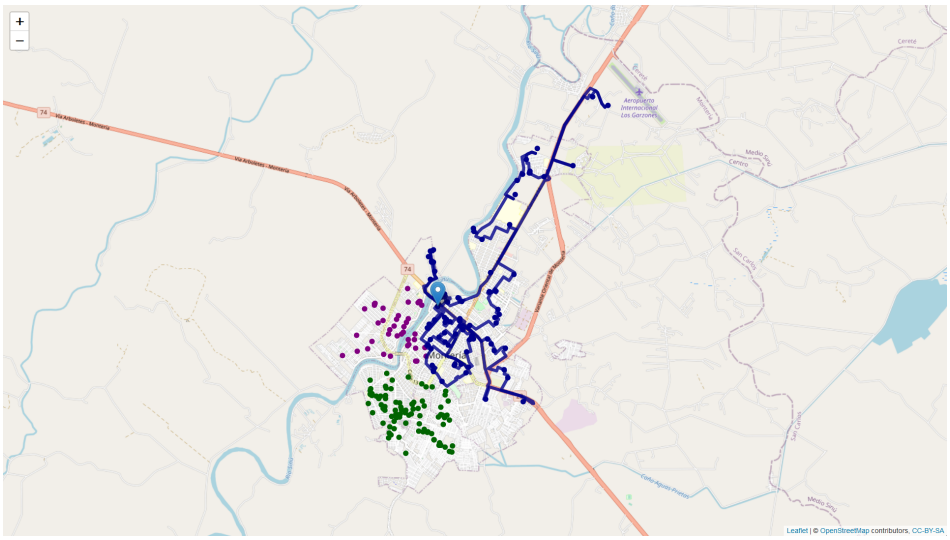

Figura 8: Primera ruta para el método PAM

Una vez realizado el agrupamiento de los clientes, se procede a resolver un modelo ATSP asignando un vehículo o vendedor a cada grupo obtenido. En `R` se pueden usar algunos algoritmos ya implementados en la librería `TSP` [Hahsler and Hornik, 2007, 2023] que resuelven de forma eficiente instancias grandes del TSP y ATSP a través de heurísticas de inserción y otros métodos aproximados. Además, incluye una interfaz que permite usar el solver *Concorde*[7] que contiene algunos de los algoritmos más sofisticados y rápidos para resolver el TSP de forma exacta. La ventaja de usar directamente la librería `TSP` es que se tiene control sobre los algoritmos y sus parámetros, lo que permite calibrar dichos algoritmos para obtener mejores resultados.

Cuando se ha resuelto el problema de optimización, parte importante del trabajo es

[7]Previa instalación del solver

poder representar la ruta para cada grupo en el mapa, lo que se puede lograr usando el servicio de OSRM para obtener la geometría y los datos geoespaciales que ayuden a visualizar la trayectoria en un mapa interactivo, sin embargo, este enfoque requiere que se hagan múltiples solicitudes a OSRM. Por ejemplo, para 3 rutas (permutaciones) de 100 clientes cada una, son 300 solicitudes desde la API que incluyen el cálculo de la distancia[8] más corta entre 2 puntos consecutivos y los datos geoespaciales que permitirán trazar la ruta en el mapa. Estas múltiples solicitudes, ralentizan en demasía la obtención de la solución, pero ya no debido a la complejidad matemática del problema en cuestión sino por motivos de conexión a internet y límites en la API para la cantidad de solicitudes[9].

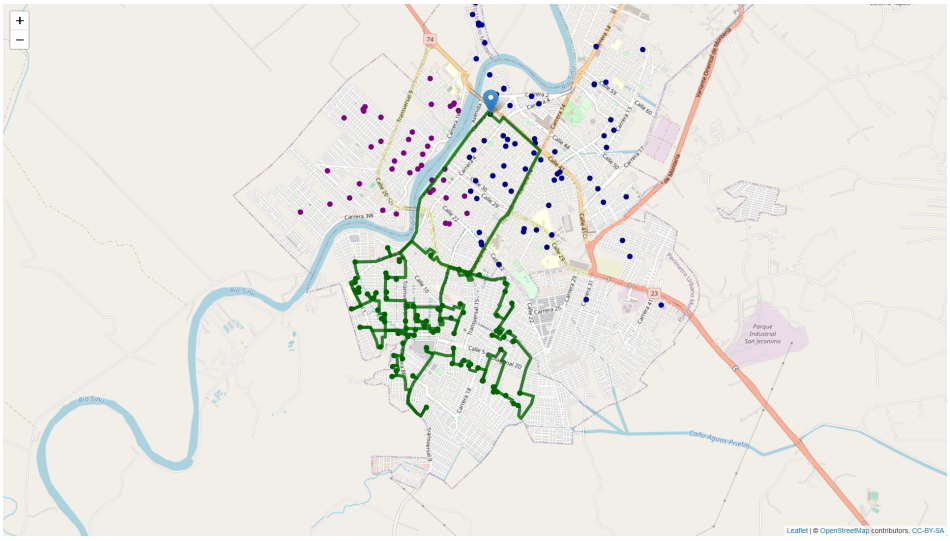

Figura 9: Segunda ruta para el método PAM

Debido a lo anterior, se hace necesario resolver el problema de optimización directamente con OSRM, que implementa algoritmos que resuelven el modelo de forma exacta para instancias de menos de 10 puntos, y con un algoritmo de *farthest-insertion* para instancias más grandes, tal como se describe en la documentación. La desventaja inmediata de este enfoque es que no permite calibrar los algoritmos usados, y se pudo verificar a través de varias pruebas, que las soluciones dadas por los algoritmos de la librería `TSP` suele ser mejor, en términos del valor de la función objetivo (distancia total), aunque no es una diferencia significativa. Las figuras 8, 9, 10 muestran las rutas

[8] Aunque las distancias se hayan calculado previamente en la matriz de distancias, no hay forma de incluir esta información como entrada en una nueva solicitud al servicio de OSRM, por lo que la distancia se volverá a calcular resolviendo un problema de ruta más corta

[9] El servidor limita la cantidad de solicitudes para evitar el colapso del servicio debido a un tráfico elevado de datos en la red.

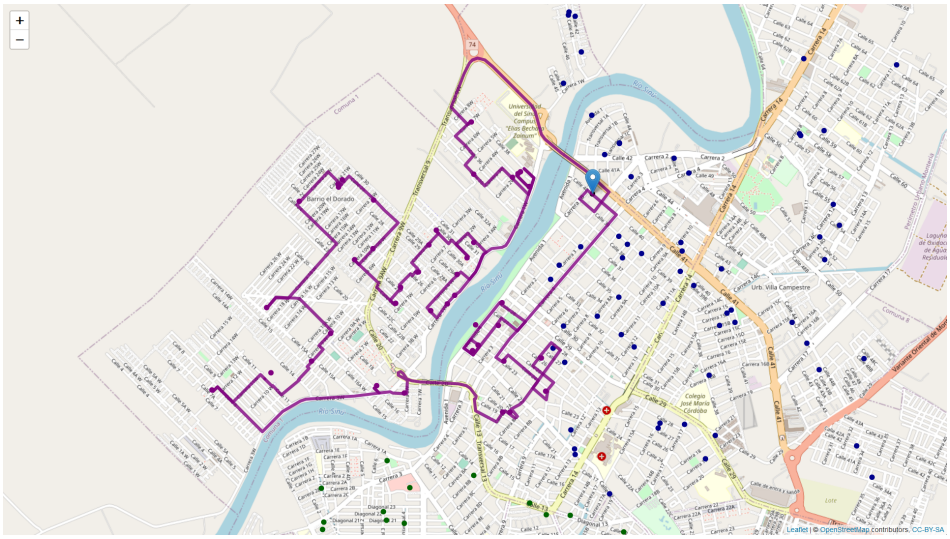

Figura 10: Tercera ruta para el método PAM

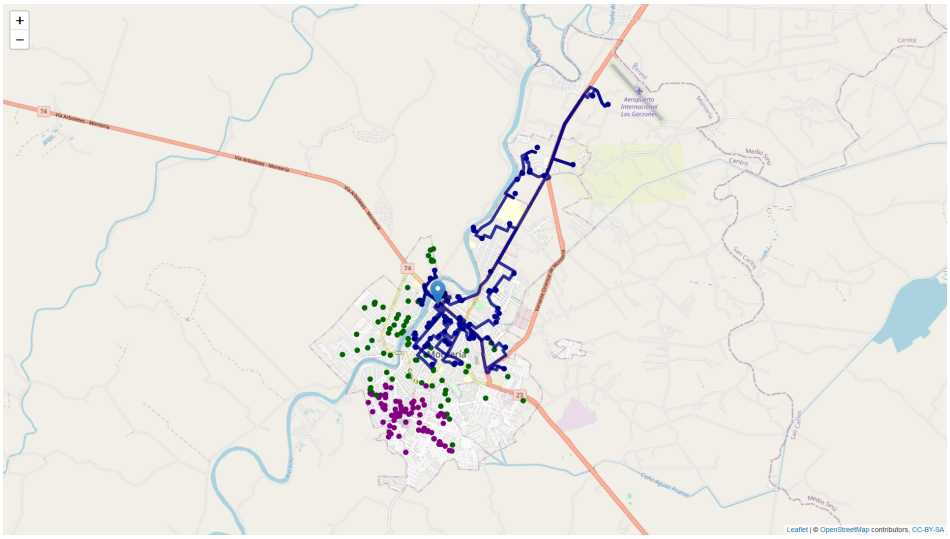

Figura 11: Primera ruta para el método *k-means* balanceado

obtenidas luego de aplicar los métodos descritos con el algoritmo PAM para hacer el *clustering*. La figuras 11, 12, 13 muestran lo propio para el caso en que el *clustering* se hace con *k-means* balanceado.

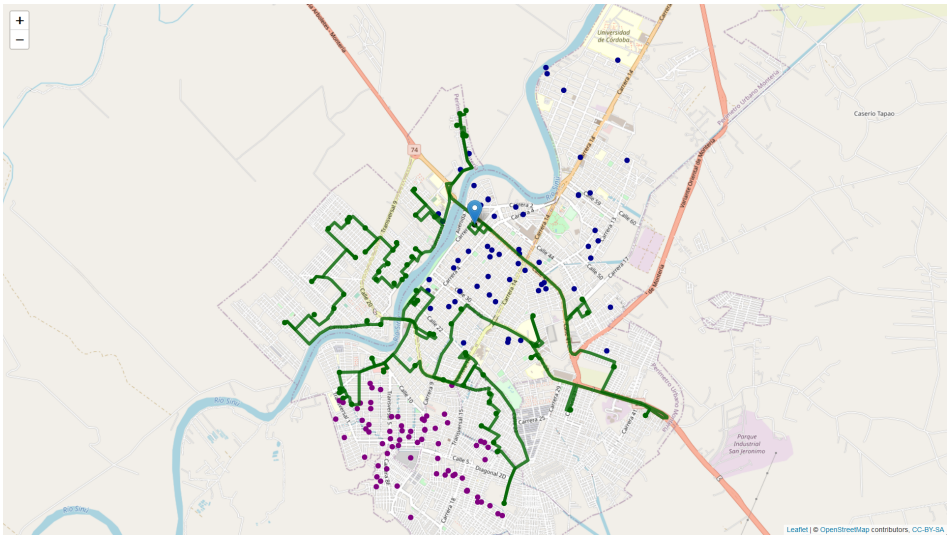

Figura 12: Segunda ruta para el método *k-means* balanceado

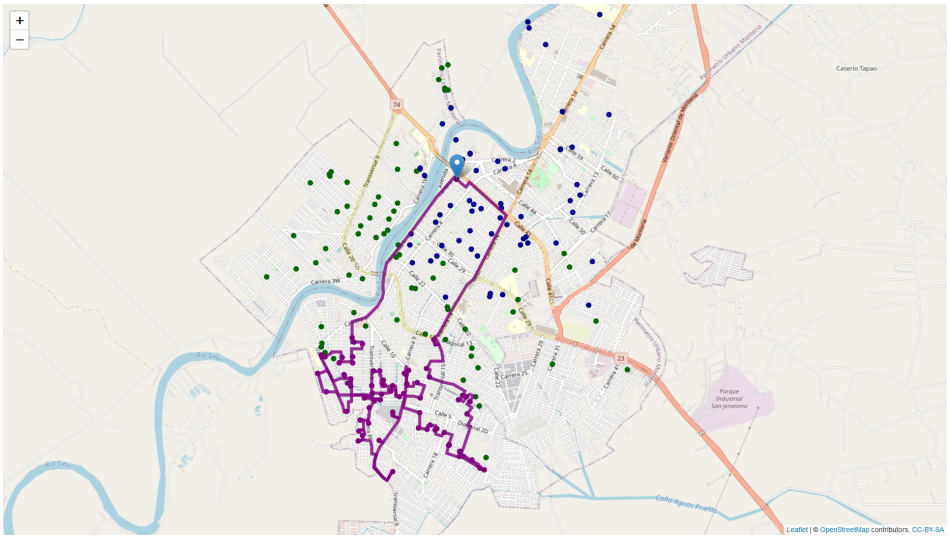

Figura 13: Tercera ruta para el método *k-means* balanceado

El Anexo B muestra el código requerido para construir las rutas. Esta herramienta puede programar las rutas de entrega para cientos de clientes de forma rápida, tomando como información inicial, una base de datos que contenga como mínimo las coordenadas de latitud y longitud de todos los clientes, asumiendo que el primer dato corresponde al depósito.

6 Análisis y conclusiones

Este trabajo propone un enfoque para resolver el problema de programación de rutas en empresas de distribución. Se presenta una metodología para aplicar y resolver problemas en empresas con múltiples agentes y/o vehículos y que manejen una cantidad moderada de cientos de clientes. La parte inicial incluye un preprocesamiento de datos de clientes y la construcción de mapas interactivos utilizando la librería como `leaflet` de `R` para visualizar la distribución geográfica de los clientes. Esto depende en gran medida del estado inicial de la información con la que cuenta la empresa.

Para estimar las distancias reales entre pares de puntos, se utiliza el servicio OSRM que da acceso a la información real de carreteras y permite construir modelos más ajustados a la realidad. Sin embargo, debido a las restricciones en la cantidad de solicitudes, este proceso se divide en múltiples solicitudes para no exceder los límites de la API. La limitación, al momento de realizar este trabajo, es de 10,000 cálculos, es decir, una matriz de distancias de 100x100. Desde esta limitación, se propone una extensión del trabajo, donde se puedan estimar directamente estás distancias a través de la resolución de problemas de ruta más corta entre nodos, construyendo una red que represente la distribución de las vías en la ciudad o el espacio geográfico considerado.

La solución del modelo mATSP se aborda mediante un enfoque heurístico. Se aplican técnicas de *clustering*, como PAM y *k-means* balanceado, para dividir el problema total en subproblemas. Las pruebas mostraron que el algoritmo de PAM es adecuado para casos en que puedan existir clientes aislados, sin embargo, los grupos obtenidos pueden mostrar un desbalance en la cantidad de clientes que se asigna a cada vehículo. Se propone un método alternativo basado en el algoritmo de *k-means* balanceado para resolver esta dificultad en caso de que sea necesario. Otra de las oportunidades de extensión del trabajo radica en aumentar las opciones y algoritmos para hacer el agrupamiento, considerando incluso, programar procedimientos propios o investigando métodos del estado del arte pensados específicamente para datos o puntos geográficos.

A pesar de los avances logrados, resolver el problema de optimización en cada grupo (un ATSP) recae nuevamente sobre la API de OSRM a través de la librería `osrm` de `R`. Esto facilita el problema pero impide la manipulación y parametrización de las heurísticas que resuelven el problema, evitando así el uso de algoritmos que muestran un mejor desempeño si se usan directamente en el lenguaje de programación. La razón de hacer esto es reducir el tráfico de red con el servicio de OSRM, ya que deben hacerse

solicitudes individuales, lo que resulta ser un procedimiento bastante desfavorable. Una última recomendación es poder construir las geometrías de las rutas para visualizar en el mapa, directamente desde la interfaz de `R`, usando librerías de forma local, lo que puede mejorar bastante el tiempo de ejecución y las opciones de algorirmos para la solución del problema de optimización.

Referencias

Zakir Hussain Ahmed and Ibrahim Al-Dayel. An Exact Algorithm for the Single-Depot Multiple Travelling Salesman Problem. *International Journal of Computer Science and Network Security*, 20(9):65–75, September 2020. doi: 10.22937/IJCSNS.2020.20.09.9. URL https://doi.org/10.22937/IJCSNS.2020.20.09.9.

Z. Akca, R. T. Berger, and T. K. Ralphs. A Branch-and-Price Algorithm for Combined Location and Routing Problems Under Capacity Restrictions. In John W. Chinneck, Bjarni Kristjansson, and Matthew J. Saltzman, editors, *Operations Research and Cyber-Infrastructure*, pages 309–330. Springer US, Boston, MA, 2009. ISBN 978-0-387-88842-2 978-0-387-88843-9. doi: 10.1007/978-0-387-88843-9_16. URL http://link.springer.com/10.1007/978-0-387-88843-9_16.

Ramin Bazrafshan, Sarfaraz Hashemkhani Hashemkhani Zolfani, and S. Mohammad J. Mirzapour Al-e hashem. Comparison of the Sub-Tour Elimination Methods for the Asymmetric Traveling Salesman Problem Applying the SECA Method. *Axioms*, 10(1):19, February 2021. ISSN 2075-1680. doi: 10.3390/axioms10010019. URL https://www.mdpi.com/2075-1680/10/1/19.

Tolga Bektas. The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, 34(3):209–219, June 2006. ISSN 0305-0483. doi: 10.1016/j.omega.2004.10.004. URL https://www.sciencedirect.com/science/article/pii/S0305048304001550.

Ulrich Blasum and Winfried Hochstättler. Application of the branch and cut method to the vehicle routing problem. *Zentrum für Angewandte Informatik Köln Technical Report zpr2000-386*, 2000.

Omar Cheikhrouhou and Ines Khoufi. A comprehensive survey on the Multiple Traveling Salesman Problem: Applications, approaches and taxonomy.

Computer Science Review, 40:100369, May 2021. ISSN 1574-0137. doi: 10.1016/j.cosrev.2021.100369. URL https://www.sciencedirect.com/science/article/pii/S1574013721000095.

Joe Cheng, Bhaskar Karambelkar, and Yihui Xie. *leaflet: Create Interactive Web Maps with the JavaScript 'Leaflet' Library*, 2023. URL https://CRAN.R-project.org/package=leaflet. R package version 2.1.2.

George Dantzig, Ray Fulkerson, and Selmer Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2(4):393–410, 1954.

Burak Eksioglu, Arif Volkan Vural, and Arnold Reisman. The vehicle routing problem: A taxonomic review. *Computers & Industrial Engineering*, 57(4):1472–1483, November 2009. ISSN 03608352. doi: 10.1016/j.cie.2009.05.009. URL https://linkinghub.elsevier.com/retrieve/pii/S0360835209001405.

Bezalel Gavish. Note—a note on "the formulation of the m-salesman traveling salesman problem". *Management Science*, 22(6):704–705, 1976.

Timothée Giraud. osrm: Interface Between R and the OpenStreetMap-Based Routing Service OSRM. *Journal of Open Source Software*, 7(78):4574, aug 2022. doi: 10.21105/joss.04574. URL https://doi.org/10.21105/joss.04574.

W. J. Gutjahr, C. Strauss, and E. Wagner. A Stochastic Branch-and-Bound Approach to Activity Crashing in Project Management. *INFORMS Journal on Computing*, 12 (2):125–135, May 2000. ISSN 1091-9856, 1526-5528. doi: 10.1287/ijoc.12.2.125.11894. URL https://pubsonline.informs.org/doi/10.1287/ijoc.12.2.125.11894.

Michael Hahsler and Kurt Hornik. **TSP** - Infrastructure for the Traveling Salesperson Problem. *Journal of Statistical Software*, 23(2), 2007. ISSN 1548-7660. doi: 10.18637/jss.v023.i02. URL http://www.jstatsoft.org/v23/i02/.

Michael Hahsler and Kurt Hornik. *TSP: Traveling Salesperson Problem (TSP)*, 2023. URL https://CRAN.R-project.org/package=TSP. R package version 1.2-4.

Hugo Hernandez-Saldana. *A Sociophysical Application of Tsp: the Corporate Vote*. IntechOpen, Place of publication not identified, 2010. ISBN 978-953-307-426-9. OCLC: 1250419254.

L. Virginayoga Hignasari and Eka Diana Mahira. Optimization of goods distribution route assisted by google map with cheapest insertion heuristic algorithm (cih). *SINERGI*, 22(2):132, June 2018. ISSN 24601217, 14102331. doi: 10.22441/sinergi.2018.2.010. URL http://publikasi.mercubuana.ac.id/index.php/sinergi/article/view/2501.

I Kara and T Bektas. Integer programming formulations of multiple salesmen problems and its variations. *European Journal of Operational Research (available currently at http://www.crt.umontreal.ca/tolga/)*, 2005.

Charles FF Karney. Algorithms for geodesics. *Journal of Geodesy*, 87:43–55, 2013.

Yannick Kergosien, Christophe Lenté, D. Piton, and Jean-Charles Billaut. A tabu search heuristic for a dynamic transportation problem of patients between care units. May 2010. URL https://hal.science/hal-00488270.

Jun Li, MengChu Zhou, Qirui Sun, Xianzhong Dai, and Xiaolong Yu. Colored Traveling Salesman Problem. *IEEE Transactions on Cybernetics*, 45(11):2390–2401, November 2015. ISSN 2168-2267, 2168-2275. doi: 10.1109/TCYB.2014.2371918. URL http://ieeexplore.ieee.org/document/6975134/.

Martin Maechler, Peter Rousseeuw, Anja Struyf, Mia Hubert, and Kurt Hornik. *cluster: Cluster Analysis Basics and Extensions*, 2022. URL https://CRAN.R-project.org/package=cluster. R package version 2.1.4 — For new features, see the 'Changelog' file (in the package source).

Radu Mariescu-Istodor, Alexandru Cristian, Mihai Negrea, and Peiwei Cao. VRPDiv: A Divide and Conquer Framework for Large Vehicle Routing Problems. *ACM Transactions on Spatial Algorithms and Systems*, 7(4):1–41, December 2021. ISSN 2374-0353, 2374-0361. doi: 10.1145/3474832. URL https://dl.acm.org/doi/10.1145/3474832.

Raluca Necula, Mihaela Breaban, and Madalina Raschip. *Tackling the Bi-criteria Facet of Multiple Traveling Salesman Problem with Ant Colony Systems*. November 2015. doi: 10.1109/ICTAI.2015.127.

Sharnil Pandya and Swarndeep Saket. An overview of partitioning algorithms in clustering techniques. *International Journal of Electrical and Computer Engineering*, 5, September 2020.

Martin Papenberg and Gunnar W. Klau. Using anticlustering to partition data sets into equivalent parts. *Psychological Methods*, 26(2):161–174, 2021. doi: 10.1037/met0000301.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2023. URL https://www.R-project.org/.

Nafia Rahma, Annie Purwani, and Dwi Nita Febriyanto. The best route determination using nearest neighbor approach. *International Journal of Industrial Optimization*, 1(1):43, February 2020. ISSN 2714-6006. doi: 10.12928/ijio.v1i1.1423. URL http://journal2.uad.ac.id/index.php/ijio/article/view/1423.

Subhash C. Sarin, Hanif D. Sherali, Jason D. Judd, and Pei-Fang (Jennifer) Tsai. Multiple asymmetric traveling salesmen problem with and without precedence constraints: Performance comparison of alternative formulations. *Computers & Operations Research*, 51:64–89, November 2014. ISSN 03050548. doi: 10.1016/j.cor.2014.05.014. URL https://linkinghub.elsevier.com/retrieve/pii/S0305054814001439.

Dharm Raj Singh, Manoj Kumar Singh, Tarkeshwar Singh, and Rajkishore Prasad. Genetic Algorithm for Solving Multiple Traveling Salesmen Problem using a New Crossover and Population Generation. *Computación y Sistemas*, 22(2), July 2018. ISSN 2007-9737, 1405-5546. doi: 10.13053/cys-22-2-2956. URL http://www.cys.cic.ipn.mx/ojs/index.php/CyS/article/view/2764.

Avi Wigderson. *Mathematics and computation: A theory revolutionizing technology and science*. Princeton University Press, 2019.

Anexos

Anexo A. Código para el pre-procesamiento de los datos

```
## Preparación de los datos

# cargar librerías
library(tidyverse)
library(leaflet)

# cargar la base de datos de clientes original
```

```r
data_0 <- readxl::read_xlsx("coordenadas.xlsx")
glimpse(data_0)

# renombrar las variables para llamarlas de forma más simple
data <- data_0 |> rename(nombre = `Razon Social`,
                         municipio = Municipio,
                         barrio = Barrio,
                         dir = `Direccion`,
                         lng = lon)

# agregar el punto decimal a las coordenadas
data <- data |> mutate(lat = str_replace(as.character(lat),
                                  pattern = "(.{1})(.*)",
                                  replacement = "\\1.\\2"),
                lng = str_replace(as.character(lng),
                                  pattern = "(.{3})(.*)",
                                  replacement = "\\1.\\2"),) |>
  mutate(lat = as.numeric(lat), lng = as.numeric(lng))

# Observamos la información en la variable municipio
table(data$municipio)
# Seleccionar solo Montería (hay algunos de Cereté) y eliminar
# la variable municipio (no aporta nada)
data <- data |> filter(municipio == "MONTERIA") |>
  select(-municipio)

# agregar las coordenadas de ANCOR (depósito)
data_1 <- data |> add_row(nombre = "ANCOR", barrio = "CENTRO",
                          dir = "CLL 40 #1B-67",
                          lat = 8.764383767330743, # de google maps
                          lng = -75.8812053434209, # de google maps
                          .before = 1)

# Construir y revisar mapa
mapa <- data_1 |> leaflet() |>
  addTiles() |>
  addMarkers(lng = ~lng, lat = ~lat, popup = ~nombre)
# mapa
# Después de una Inspección visual de los puntos agregados al mapa,
# algunas direcciones aparecen en Caquetá (una sobre un río). Estas son:
# TIENDA CARLUZ
# PRODUCTOS MI FINCA
```

```r
# Revisar información de cada uno
glimpse(data |> filter(nombre == "TIENDA CARLUZ"))
glimpse(data |> filter(nombre == "PRODUCTOS MI FINCA"))
# Se recomienda a la empresa revisar esta información en el futuro
# para ser corregida de ser necesario. Por ahora se eliminan de
# la base de datos ya que afectarían la construcción de las rutas
data_2 <- data_1 |> filter(
  nombre != "TIENDA CARLUZ" & nombre != "PRODUCTOS MI FINCA"
)
# revisar cuántas entradas fueron eliminadas
nrow(data_1) - nrow(data_2)  # se eliminaron 3 entradas

# mapa con clientes filtrados
mapa2 <- data_2 |> leaflet() |>
  addTiles() |>
  addMarkers(lng = ~lng, lat = ~lat, popup = ~nombre)
# mapa2

# verificar NAs
sum(is.na(data_2)) # hay 13 NAs
data_na <- data_2 |> filter(if_any(names(data_2), ~is.na(.)))
data_na
# Los NA están sobre las variables: barrio y dir, que no se
# usarán para el cálculo de las distancias (esto se hará a
# partir de las coordenadas de lat y lng) por lo que no será
# necesario omitir estas entradas y se puede actualizar
# esta información más adelante.

# Verificación de duplicados
# verificar entradas con todos los valores iguales
nrow(data_2) - nrow(data_2 |> distinct()) # 0, no hay duplicados exactos

# verificar solo por nombre
nrow(data_2) - nrow(data_2 |> distinct(nombre)) # 425 nombres repetidos
# esto puede deberse a que hay varios puntos o sedes de una misma
# razón social.
# Construir tabla de nombres repetidos y frecuencia
data_nombres_rep <- data_2 |> group_by(nombre) |> summarise(n = n()) |>
  arrange(desc(n))
data_nombres_rep

# Verificamos algunos de los nombres más repetidos
```

```r
data_2 |> filter(nombre == "TIENDA LA ECONOMIA")
data_2 |> filter(nombre == "TIENDA LA ESPERANZA")
data_2 |> filter(nombre == "TIENDA LA NUEVA")
data_2 |> filter(nombre == "TIENDA EL PROGRESO")
data_2 |> filter(nombre == "TIENDA LA 11")
# En algunos casos, es la misma razón social o tienda en diferentes lugares
# pero también hay casos del mismo nombre con el mismo barrio y
# dirección. Las diferencias en las coordenadas pueden deberse a
# errores de precisión, la calidad de estos datos debe ser revisada
# por la empresa y no hace parte de nuestro trabajo

# Verificamos duplicados de latitud y longitud
nrow(data_2) - nrow(data_2 |> distinct(lat, lng))
# hay 6 puntos cuyo valor de lat y lng no son únicos. Revisamos cuáles son
data_lat_lng_rep <- data_2 |> group_by(lat, lng) |>
  mutate(n = n()) |> filter(n > 1) |> arrange(desc(n))
data_lat_lng_rep # 9 clientes con la misma ubicación.
# eliminamos estos clientes ya que no podemos tener
# varios clientes en la misma ubicación
data_3 <- data_2 |> group_by(lat, lng) |>
  mutate(n = n()) |> filter(n==1) |> select(-n) |> ungroup()

# Guardamos la base de datos "limpia" para usar en la programación
# de las rutas
writexl::write_xlsx(x = data_3, path = "clientes-ancor.xlsx")
```

Anexo B. Construcción de rutas a partir de la geolocalización de los clientes

```r
# Cluster y Programación de rutas

# cargar las librerías necesarias
library(tidyverse)
library(leaflet)
library(osrm)
library(cluster)
library(anticlust)
library(readxl)

# cargar la base de datos "limpia"
```

```r
clientes <- read_xlsx(path = "clientes-ancor.xlsx", col_names = TRUE)
clientes <- clientes |> mutate(id = 1:(nrow(clientes)), .before = 1)
glimpse(clientes)
# seleccionar una muestra aleatoria de clientes
# para programar las rutas
m_salesman <- 3 # número de vendedores
n_clientes <- 210 # número de clientes
set.seed(1711)
muestra <- c(1, sample(x = 2:(nrow(clientes)), size = n_clientes,
                       replace = F)) # 1: ANCOR
clientes_sub <- clientes |> filter(id %in% muestra)

# Mostrar en el mapa, los puntos (clientes) a rutear
mapa_sub <- clientes_sub |> leaflet() |>
  addTiles() |> addCircleMarkers(lng = ~lng,
                                 lat = ~lat,
                                 popup = ~nombre,
                                 radius = 2,
                                 opacity = 1)
# mapa_sub

# Calcular la matriz de distancias
# extrae los datos de latitud y longitud en el formato adecuado:
# un data frame con las variables "lon" y "lat"
puntos_ruteo <- clientes_sub |> select(lon = lng, lat) |> as.data.frame()
rownames(puntos_ruteo) <- clientes_sub$id

# dividir el cálculo de la matriz para no superar el límite
# de la API de osrm (10000 solicitudes, matriz de 100x100)
# calcular las distancias entre cada par de clientes
# incluyendo también el depósito
costo_cij <- matrix(nrow = n_clientes + 1, ncol = n_clientes + 1)
bloques <- ceiling((n_clientes+1)/100)
for(i in 1:bloques){
  for(j in 1:bloques){
    costo_cij[((i-1)*100 + 1):min(i*100, ncol(costo_cij)),
              ((j-1)*100 + 1):min(j*100, nrow(costo_cij))] <-
      osrmTable(src = puntos_ruteo[((i-1)*100 + 1):min(i*100, ncol(costo_cij)),],
                dst = puntos_ruteo[((j-1)*100 + 1):min(j*100, nrow(costo_cij)),],
                measure = "distance")$distances
  }
}
```

```r
rownames(costo_cij) <- colnames(costo_cij) <- clientes_sub$id

# Hacer clustering
# se retira el origen o depósito
dist_matrix_cl <- costo_cij[-1, -1]
# convertimos en una matriz simétrica, conservando
# la distancia más pequeña entre cada par de puntos
# Esto se hace para facilitar el clustering
dist_matrix_cl <- pmin(dist_matrix_cl, t(dist_matrix_cl))
isSymmetric(dist_matrix_cl) # TRUE
# hacer cluster con Partitioning Around Medoids
# con la función pam de la librería cluster
particion <- pam(x = dist_matrix_cl, k = m_salesman,
    diss = T)
distribucion <- table(particion$clustering)
# Haciendo clustering con balanced k-means
particion2 <- balanced_clustering(x = dist_matrix_cl,
                                  K = m_salesman)
names(particion2) <- rownames(dist_matrix_cl)
distribucion2 <- table(particion2)


calculo_rutas <- function(x){
  # x es el vector que contiene el grupo al que pertenece
  # cada cliente.
  # Se separan los grupos encontrados para resolver un problema
  # de ATSP en cada grupo, usando osrm
  grupos <- list()
  rutas <- list()
  for(i in 1:bloques){
    grupos[[i]] <-  c(1,as.integer(names(which(x == i))))
    sub_ruteo <- clientes_sub |> filter(id %in% grupos[[i]]) |>
      select(lon = lng, lat) |> as.data.frame()
    rownames(sub_ruteo) <- grupos[[i]]
    rutas[[i]] <- osrmTrip(loc = sub_ruteo,
                           overview = "full")
  }

  # Construir el mapa con los clientes en cada grupo
  colores <- c("darkblue", "darkgreen", "purple", "darkred")
  ruta_mapa <- clientes_sub |>
    mutate(color = c("black",
```

```
                    colores[x])) |>
    leaflet() |>
    addTiles() |>
    addCircleMarkers(lng = ~lng,
                     lat = ~lat,
                     radius = 2,
                     color = ~color,
                     opacity = 1) |>
    addMarkers(lng = clientes_sub$lng[1],
               lat = clientes_sub$lat[1],
               popup = clientes_sub$nombre[1])

  # Construir las rutas resolviendo un problema de
  # atsp en cada grupo, usando osrm
  ruta_mapa_t <- ruta_mapa
  for(i in 1:m_salesman){
    # construir cada ruta de forma individual
    assign(paste0("ruta_mapa",i),
           {ruta_mapa |>
               addPolylines(data = rutas[[i]][[1]]$trip$geometry,
                            color = colores[i], opacity = 0.8)})
    # agregar todas las rutas en un solo mapa
    ruta_mapa_t <- ruta_mapa_t |>
      addPolylines(data = rutas[[i]][[1]]$trip$geometry,
                   color = colores[i], opacity = 0.5)
  }
  return(list(
    ruta_mapa = ruta_mapa, # mapa que contiene los clientes agrupados
    ruta_mapa1 = ruta_mapa1, # mapa con la ruta para el vendedor 1
    ruta_mapa2 = ruta_mapa2, # mapa con la ruta para el vendedor 2
    ruta_mapa3 = ruta_mapa3, # mapa con la ruta para el vendedor 3
    ruta_mapa_t = ruta_mapa_t # mapa con todas las rutas
  ))
}

matsp_pam <- calculo_rutas(particion$clustering)
matsp_bkmeans <- calculo_rutas(particion2)

matsp_pam$ruta_mapa
matsp_pam$ruta_mapa1
matsp_pam$ruta_mapa2
matsp_pam$ruta_mapa3
```

```
matsp_pam$ruta_mapa_t
matsp_bkmeans$ruta_mapa
matsp_bkmeans$ruta_mapa1
matsp_bkmeans$ruta_mapa2
matsp_bkmeans$ruta_mapa3
matsp_bkmeans$ruta_mapa_t
```