Sección de Matemáticas
Universidad de La Laguna

Patricia Sánchez Vega

El Problema Periódico de Rutas de Vehículos

The Periodic Vehicle Routing Problem

Trabajo Fin de Grado
Grado en Matemáticas
La Laguna, Julio de 2024

Dirigido por
Inmaculada Rodríguez Martín

Inmaculada Rodríguez Martín
Departamento de Matemáticas,
Estadística e Investigación
Operativa
Universidad de La Laguna
38200 La Laguna, Tenerife

Agradecimientos

A mi tutora, Macu, por brindarme toda la ayuda y apoyo necesarios, y por impulsarme en esta recta final.

A mis padres y mi hermana, por confiar siempre en mi, apoyarme incondicionalmente y mostrar todo su interés en escucharme hablar sobre cada asignatura que cursaba y cada examen que tenía.

A mis amigas y amigos de la carrera, por entender mejor que nadie esto, por apoyarnos en cada paso y animarnos en cada bajón, por estar siempre ahí para levantarnos. Gracias.

A mis amigas de toda la vida, por sentirlas siempre cerca a pesar de la distancia. Gracias por ser y estar, por todo lo vivido y todo lo que nos espera. Siempre serán mi suerte y mi lugar favorito.

Patricia Sánchez Vega
La Laguna, 7 de julio de 2024

Resumen · Abstract

Resumen

Esta memoria está dedicada al Problema Periódico de Rutas de Vehículos (PVRP). Este problema es una generalización del clásico Problema de Rutas Vehículos (VRP) en el que se dispone de una flota de vehículos, ubicados en un depósito, con los que se tiene que dar servicio a una serie de clientes. En el PVRP cada cliente tiene una demanda y requiere una o varias visitas a lo largo de un horizonte temporal de varios periodos. Las fechas de las visitas no están fijadas de antemano, y en su lugar cada cliente tiene asociados unos posibles calendarios de visita, de los que se debe elegir uno. El problema consiste en elegir un calendario de visita para cada cliente y diseñar las rutas de los vehículos para cada periodo de manera que el coste total de las rutas sea mínimo. En este trabajo presentamos dos modelos matemáticos distintos para el PVRP, junto con algunas familias de restricciones válidas que sirven para fortalecerlos. Además, codificamos ambos modelos en Python y presentamos sus resultados computacionales.

Palabras clave: *Problemas periódicos de rutas – Formulaciones.*

Abstract

This memory is dedicated to the Periodic Vehicle Routing Problem (PVRP). This problem is a generalization of the classic Vehicle Routing Problem (VRP) in which there is a fleet of vehicles, located at a depot, with which a set of customers must be served. In the PVRP, each customer has a demand and requires one or several visits over a time horizon of several periods. The dates of the visits are not set in advance, and instead each customer has associated a list of allowable visit schedules, from which one must be chosen. The problem therefore consists of choosing a visit schedule for each client, and designing the vehicle routes for each period, so that the total cost of the routes is minimal. In this work we present two different mathematical models for the PVRP, along with some families of valid inequalities that serve to strengthen them. Furthermore, we code both models in Python and present computational results.

Keywords: *Periodic routing problems – Formulations.*

Contenido

Introducción

El Problema Periódico de Rutas de Vehículos (PVRP, por sus siglas en inglés) es una variante compleja del problema clásico de rutas de vehículos (VRP, por sus siglas en inglés), que se sitúa en el cruce entre la teoría de la optimización y la logística aplicada. En el contexto del VRP, el objetivo es diseñar rutas eficientes para una flota de vehículos que deben satisfacer la demanda de un conjunto de clientes dispersos geográficamente, minimizando costos como la distancia total recorrida o el tiempo de viaje, y respetando restricciones operativas como la capacidad de los vehículos. El PVRP introduce una nueva dimensión temporal al problema, extendiendo la planificación de las rutas a lo largo de un horizonte de varios días o periodos.

La característica distintiva del PVRP es que las visitas a los clientes no se deben realizar en un solo día, sino que deben distribuirse en un horizonte temporal predefinido. Esto implica que cada cliente tiene una frecuencia de visitas específica que debe cumplirse dentro de este horizonte, y además tiene asociados unos determinados calendarios de visitas, de los que se debe elegir solo uno. Por ejemplo, un cliente podría requerir entregas dos días a la semana, pudiendo esas visitas ser los lunes y martes, o los miércoles y jueves, o los jueves y viernes, mientras que otro podría necesitar visitas tres veces por semana a elegir entre lunes, martes y miércoles, o lunes, miércoles y viernes. Estas características, junto con la periodicidad, añaden un nivel de complejidad adicional al problema, ya que no solo se deben optimizar las rutas diarias respetando la capacidad de los vehículos, sino también hacerlas compatibles con los requisitos de los clientes.

La importancia del PVRP se debe a su amplia utilidad en situaciones del mundo real donde las entregas periódicas son esenciales. Empresas de distribución de bienes de consumo, servicios de correo y mensajería, y la recolección de residuos son algunos ejemplos donde la gestión eficiente de las rutas de vehículos puede tener un impacto significativo en la reducción de costos operativos y la mejora del servicio al cliente. Además, el PVRP también se aplica en contextos industriales y de salud pública, como la distribución de suministros médicos a hospitales y clínicas, donde la regularidad y la puntualidad son cruciales.

Desde el punto de vista teórico, el PVRP es un problema NP-difícil, lo que significa que encontrar una solución óptima es computacionalmente intratable para instancias grandes del problema. Esta complejidad ha llevado al estudio y desarrollo de una gran variedad de métodos y algoritmos para abordarlo, que van desde enfoques exactos, como la programación entera mixta, hasta métodos heurísticos y metaheurísticos. Cada uno de estos métodos ofrece un balance diferente entre la calidad de la solución y el tiempo de cómputo, y la elección del enfoque a menudo depende del tamaño y las características específicas del problema en cuestión. Además, el PVRP ha sido objeto de numerosos estudios en la literatura académica, que han explorado diversas extensiones y variantes del problema. Como por ejemplo, ventanas de tiempo específicas para las visitas a los clientes, o la posibilidad de flexibilizar la frecuencia de visitas y las cantidades entregadas o recogidas en cada cliente en cada visita. Estas extensiones reflejan la necesidad de modelos cada vez más realistas que puedan representar de manera más precisa la complejidad y las restricciones del mundo real.

En resumen, el PVRP, estudiado en esta memoria, es una extensión significativa y relevante del problema clásico de rutas de vehículos, que introduce desafíos adicionales debido a la necesidad de planificar las rutas sobre un horizonte temporal. La memoria se organiza de la siguiente forma. En el Capítulo 1 se hace un repaso de conceptos básicos de teoría de grafos y programación matemática, que nos servirán luego para describir formalmente el problema. El Capítulo 2 pretende poner en contexto al PVRP dentro de los problemas de rutas. El Capítulo 3 está dedicado al PVRP, para el que se presentan dos modelos matemáticos distintos de programación lineal entera mixta, así como algunas familias de restricciones válidas que sirven para reforzar dichos modelos. Estos modelos han sido implementados en Python, y en el Capítulo 4 mostramos los resultados de nuestros experimentos computacionales. Por último, acabamos con algunas conclusiones.

1

Fundamentos teóricos

Antes de abordar el Problema Periódico de Rutas de Vehículos, debemos conocer algunos conceptos previos sobre la teoría de grafos y la programación combinatoria.

1.1. Teoría de grafos

Los orígenes de esta teoría se dan en 1736 con un estudio realizado por el matemático Leonhard Euler, en el que trataba de resolver el problema de los puentes de Konisberg. Este problema consistía en obtener una ruta óptima para cruzar todos los puentes de la ciudad, pasando una única vez por cada uno de ellos. Los resultados de Euler demostraron su improbabilidad, pero fue el punto de partida de numerosos estudios.

La teoría de grafos intenta representar visualmente un conjunto de datos como nodos o vértices, y las uniones entre ellos como artistas. En matemáticas se utiliza en diversas aplicaciones, aunque nosotros nos centraremos en su uso para la solución de problemas de rutas de vehículos.

A continuación, veremos una serie de definiciones básicas.

1.1.1. Conceptos básicos

Un grafo dirigido es un pareja de conjuntos $G = (V, A)$ donde $V = \{1, ..., n\}$ es un conjunto de vértices o nodos y $A \subseteq \{(i, j) : (i, j) \in V, i \neq j\}$ es un conjunto de pares ordenados de vértices denominados arcos, es decir, dado un arco $a = (i, j) \in A$, i es el origen y j el destino del arco. Además, tenemos en cuenta que importa el orden, de forma que $(i, j) \neq (j, i)$, luego cada arco tiene una dirección.

Decimos que $G' = (V', A')$ es un subgrafo del grafo $G = (V, A)$ si $V' \subseteq V$, $A' \subseteq A$ y todos los arcos de A' tienen sus extremos en V'.

Ahora, sea $G = (V, A)$ un grafo dirigido, los conjuntos de arcos que salen y entran de un vértice i se denotan $\delta^+(i)$ y $\delta^-(i)$ respectivamente. Dado un

subconjunto $S \subset V$, definimos como $\delta^+(S) = \{(i,j) \in A : i \in S, j \notin S\}$ al conjunto de arcos que salen desde S, y $\delta^-(S) = \{(i,j) \in A : i \notin S, j \in S\}$ al conjunto de arcos que entran a S. $A(S) = \{(i,j) \in A : i \in S, j \in S\}$ es el conjunto de arcos con ambos extremos en S. El grado de salida de un vértice i es $|\delta^+(i)|$ y el grado de entrada del mismo es $|\delta^-(i)|$.

Definimos un camino como una secuencia de arcos donde el vértice final de un arco coincide con el vértice de inicio del siguiente arco, importando el orden. Decimos que un camino es simple si no repite ningún vértice intermedio y un camino es elemental si no repite ningún arco. Por otra parte, un circuito es un camino donde el primer y el último vértice coinciden. Además, diremos que un camino es hamiltoniano si recorre todos los vértices de un grafo una única vez.

Por otra parte, tenemos que un grafo no dirigido es una pareja de conjuntos $G = (V, E)$ donde $V = \{1, ..., n\}$ es un conjunto de vértices o nodos y $E \subseteq \{\{i,j\} : i,j \in V, i \neq j\}$ es un conjunto de pares no ordenados de vértices denominados ejes o aristas. Los ejes no tienen dirección, es decir $\{i,j\} = \{j,i\}$. Además, dado el eje $e = \{i,j\} = \{j,i\}$, decimos que es incidente en el vértice i (y en j), y que i y j son los vértices extremos del eje.

En un grafo no dirigido $G = (V, E)$, un camino es una secuencia de aristas donde cada par de aristas consecutivas son ambas incidentes en un mismo vértice. Diremos que un camino es simple si sus vértices intermedios son todos diferentes, y es elemental si ninguna arista coincide. Por otro lado, un ciclo es un camino donde el primer y el último vértice coinciden, es decir, sus extremos son un mismo vértice. Además, un camino que visita todos los vértices de un grafo una única vez se denomina camino hamiltoniano.

Por último, un grafo no dirigido se puede transformar en uno dirigido sustituyendo cada eje por dos arcos con direcciones contrarias.

1.2. Programación matemática y optimización

La Investigación Operativa (IO) es una ciencia aplicada que tiene por objeto fundamental asignar, mediante métodos científicos y bajo ciertas condiciones, los recursos o actividades de forma eficaz, en la gestión y organización de sistemas complejos. Sus inicios se remontan en la Segunda Guerra Mundial (1939-1945) ante la necesidad de los ejércitos de convocar a distintos científicos para resolver problemas de distribución y asignación de recursos, uso eficaz de nuevas herramientas y técnicas de defensa, y planificación efectiva de actividades [9].

1.2.1. Optimización

Optimizar se refiere a la capacidad de hacer o resolver algo de la forma más eficiente posible y utilizando la menor cantidad de recursos. Matemática-

mente, es encontrar el valor que deben tomar las variables de decisión, sujetas a restricciones, para que una función objetivo alcance un máximo o un mínimo, dependiendo del problema. Estos conceptos los abordaremos en las siguientes subsecciones.

1.2.2. Programación matemática

La programación matemática forma parte de la Investigación Operativa y de la Optimización. Tiene como objetivo la resolución de problemas mediante la maximización o minimización de una función objetivo sujeta a un conjunto de restricciones. En otras palabras, es un modelo matemático que busca lograr la mejor asignación de los recursos limitados (restricciones) hacia actividades que se encuentran en competencia (variables de decisión), de tal manera que se pueda lograr la optimización (maximización o minimización de la función objetivo).

Modelizar un problema supone trabajar con los siguientes elementos:

- Variables de decisión: representan las posibles decisiones, normalmente son variables cuantitativas y buscan los valores que optimizan el objetivo. Se clasifican en:
  - Variables Enteras: son aquellas a las que se impone que solo tomen valores enteros.
  - Variables Binarias: son aquellas a las que se exige que solo tomen los valores $\{0, 1\}$.
  - Variables Continuas: son aquellas a las que no se exige ninguna condición de integridad. Pueden ser no negativas o libres.
- Restricciones: son las condiciones que garantizan que una solución sea aceptable.
- Función objetivo: es la función que determina las mejores soluciones, es decir, la que se desea maximizar o minimizar, dependiendo del objetivo del problema. Luego, es la que se encarga, a partir de las variables de decisión, de determinar los costos o beneficios.

Las soluciones factibles son aquellas que cumplen todas las restricciones y las óptimas son las mejores soluciones factibles. Puede haber más de una solución óptima si hay soluciones distintas que dan el mismo valor de la función objetivo. Con carácter general, este problema se puede representar de la siguiente manera:

$$min\left\{f(x) : x \in S\right\},$$

donde $S \subseteq \mathbb{R}^n$ es la región factible y $f : S \to \mathbb{R}$ la función objetivo.

Dependiendo de las características de las restricciones, las variables y la función objetivo del modelo, los problemas de progamación matemática se pueden clasificar de las siguientes formas:

- Teniendo en cuenta la linealidad de la función objetivo y de las restricciones. Si estas son lineales, entonces será un problema de programación lineal, en otro caso será un problema de programación no lineal.
- Teniendo en cuenta las características de las variables podemos diferenciar tres casos. Programación continua, cuando las variables solo toman valores reales; programación entera, cuando las variables solo toman valores enteros; y programación mixta, siendo esta una combinación de las otras dos, usando variables enteras y continuas.

Por otra parte, los problemas de progamación matemática también se pueden clasificar según los resultados obtenidos:

- Problema óptimo: se da cuando existe una solución $x^* \in S$ tal que $f(x^*) \leq f(x), \quad \forall x \in S$. A x^* se le denomina solución óptima y se cumple que $min\{f(x) : x \in S\} = f(x^*)$.
- Problema no acotado: se da cuando $\forall n \in \mathbb{R}, \exists x \in S : f(x) < n$, por lo que $min\{f(x) : x \in S\} = -\infty$. En otras palabras, cuando se encuentran soluciones que hacen disminuir la función objetivo de forma infinita.
- Problema no factible: se da cuando $S = \emptyset$, luego no hay soluciones factibles.

Por último, una rama de la programación matemática es la Progamación Combinatoria, que estudia los problemas que tienen un número finito de soluciones factibles, es decir, problemas del tipo $min\{f(x) : x \in S\}$ con $|S| < \infty$.

1.2.3. Algoritmos para la resolución de problemas de Progamación Lineal Entera o Mixta

Los métodos de resolución de los problemas de optimización combinatoria se pueden clasificar de diversas maneras. Por una parte tenemos los métodos heurísticos, que generalmente con poco esfuerzo computacional proporcionan una solución aproximada, pero no necesariamente óptima, del problema. Por otro lado, tenemos los métodos exactos, que proporcionan una solución óptima del problema. A continuación describimos algunos de estos métodos exactos.

Ramificación y acotación

Un algoritmo de ramificación y acotación, en inglés *branch and bound*, es un método de resolución de problemas de programación lineal entera que se basa en la creación de un árbol de búsqueda.

Comienza con la relajación del problema original, para luego resolverlo como un problema de programación lineal, es decir, sin restricciones de integralidad. Si la solución obtenida es entera, entonces es la solución óptima. En caso contrario, creamos dos nuevos subproblemas eligiendo una variable que tiene un valor no entero y añadiendo restricciones sobre el valor que puede tomar

esta (ramificación). Cada nuevo subproblema es un nodo del árbol de búsqueda. A continaución resolvemos cada subproblema como un problema de programación lineal, obteniendo soluciones factibles, en el caso de ser enteras, y en caso contrario, cotas superiores para el valor óptimo de cada subproblema (si es de minimizar) o cotas inferiores para el valor óptimo (si es de maximizar). Si la cota de un subproblema indica que no se puede mejorar la solución entera encontrada hasta ahora, se descarta ese subproblema, es decir, se poda ese nodo del árbol de búsqueda; si un subproblema produce una solución entera mejor que la solución actual, actualizamos la mejor solución encontrada. Repetimos los pasos de ramificación, resolución de problemas lineales y poda hasta que todos los subproblemas sean eliminados o resueltos, por lo que hemos encontrado la mejor solución entera que será la solución óptima del problema original.

En resumen, un algoritmo de ramificación y acotación se basa en el estudio de todas las posibles soluciones de un problema de programación lineal entera a partir de la creación de un árbol de búsqueda y la acotación de las soluciones, eliminando las inviables y reduciendo el espacio de búsqueda.

Hiperplanos de corte

Un algoritmo de hiperplanos de corte, en inglés *cutting planes*, es un método que se basa en la idea de acotar iterativamente el espacio de soluciones posibles, mediante la adición de restricciones (cortes) que eliminan regiones del espacio factible que no contienen soluciones óptimas enteras.

En primer lugar se resuelve la relajación lineal continua del problema, es decir, se relajan las restricciones de integralidad sobre las variables del problema entero, convirtiendo así el problema entero en un problema de programación lineal estándar. Si la solución de la relajación lineal cumple con todas las restricciones de integralidad, entonces esa es la solución óptima del problema entero; en caso contrario la solución no es entera y proseguimos con el algoritmo. Se genera un corte, es decir, una nueva restricción lineal que excluye esta solución no entera, sin excluir al resto de soluciones enteras factibles. Al añadir el corte al conjunto de restricciones del problema, resolvemos nuevamente. Repetimos este proceso hasta que se encuentre una solución entera óptima o veamos que no hay soluciones enteras factibles.

Ramificación y corte

Un algoritmo de ramificación y corte, en inglés *branch and cut*, es un algoritmo para resolver problemas de programación entera y mixta que combina los dos algoritmos anteriores, de forma que es un algoritmo de ramificación y acotación en el cual se generan cortes en cada subproblema del árbol de búsqueda con el objetivo de reducir el número de nodos del árbol (subproblemas).

Se empieza por resolver la relajación lineal del problema entero, generando y añadiendo cortes para hallar la solución de la relajación. Si la solución no es entera, seleccionamos una variable con valor fraccionario y dividimos el problema en dos subproblemas, añadiendo las restricciones sobre el valor que puede tomar esa variable. Se resuelven los subproblemas aplicando la generación de cortes para posteriormente podarlos, eliminando los subproblemas que no mejoran la solución entera encontrada. Repetimos el proceso de ramificación, generación de cortes y poda hasta que todos los subproblemas sean eliminados o resueltos. La mejor solución entera encontrada será la solución óptima del problema original.

Por lo general, la cota producida en cada nodo del árbol es más eficiente que en el algoritmo de ramificación y acotación debido a las nuevas restricciones añadidas, por lo que este algoritmo es más efectivo.

2

Problemas clásicos de rutas

Los problemas clásicos de rutas se introdujeron en los años 30, dando pie al posterior estudio de una gran cantidad de variantes de estos problemas y de algoritmos para resolverlos de forma óptima y eficiente. En este capítulo, introduciremos dos problemas clásicos de rutas, el Problema del Viajante de Comercio y el Problema de Rutas de Vehículos, más directamentes relacionados con el problema objeto de esta memoria. Veremos sus definiciones y varios modelos matemáticos para cada uno.

2.1. El Problema del Viajante de Comercios (TSP)

El Problema del Viajante de Comercio, TSP por sus siglas en inglés (*Travelling Salesman Problem*), es un problema que trata de buscar la mejor ruta posible que recorre una serie de nodos, normalmente ciudades, de los que se conoce la distancia entre cada par de ellos. En dicha ruta, un vehículo debe visitar cada nodo exactamente una vez y regresar al origen, minimizando el costo o longitud total del recorrido.

Los orígenes se remontan a los años 30, cuando el matemático austriaco Karl Menger invitó al resto de la comunidad de investigadores a estudiar desde un punto de vista matemático el siguiente problema:

Un comerciante tiene que visitar exactamente una vez un conjunto de n ciudades y volver al punto de partida. Se conoce el costo de viajar de una ciudad a cualquier otra. Entonces, ¿cuál es el recorrido de menor costo posible que debe hacer el comerciante? [3]

El problema se popularizó en los años 50, cuando se reformuló como el problema del viajante de comercio, gracias a la aparición del artículo *Solution of a Large-Scale Travelling-Salesman Problem* de Dantzig, Fulkerson y Johnson [5], lo que conllevó al estudio de la combinatoria y de problemas de programación lineal.

El problema tiene dos variantes dependiendo del tipo de grafo que definan las distancias o costos entre los nodos: puede ser un grafo simétrico o uno asimétrico. En el primer caso, tenemos un grafo no dirigido, donde la distancia o el costo entre dos ciudades es siempre igual en ambos sentidos. Sin embargo, en el segundo caso, estamos ante un grafo dirigido, donde la distancia o costo entre dos ciudades puede ser diferente dependiendo de la dirección que se tome. Este último es el caso más común y el que nosotros trataremos a continuación.

2.1.1. Planteamiento del problema para el caso asimétrico

Definimos el TSP sobre un grafo dirigido $G = (V, A)$ con el conjunto de vértices $V = \{0\} \cup N$, siendo $\{0\}$ el depósito y $N = \{1, ..., n\}$ el conjunto de nodos que corresponden a las n ciudades. $A \subseteq \{(i, j) : i, j \in V, i \neq j\}$ es el conjunto de arcos del grafo. Cada arco $a = (i, j) \in A$ tiene asignado un costo c_a. Además, como estamos ante el caso asimétrico tenemos que, en general, $c_{ij} \neq c_{ji}, \forall (i, j) \in A$.

Usaremos las siguientes variables binarias de decisión para las distintas formulaciones que existen del problema:

$$x_a = \begin{cases} 1, & \text{si el vehículo para por el arco } a \in A, \\ 0, & \text{en otro caso.} \end{cases}$$

Formulación con restricciones de subciclos

Podemos formular el TSP como:

$$\text{mín} \sum_{a \in A} c_a x_a \tag{2.1}$$

Sujeto a:

$$\sum_{a \in \delta^+(i)} x_a = \sum_{a \in \delta^-(i)} x_a = 1 \qquad \forall i \in V \tag{2.2}$$

$$\sum_{a \in A(S)} x_a \leq |S| - 1 \qquad \forall S \subset V. \tag{2.3}$$

$$0 \leq x_a \leq 1 \qquad \forall a \in A \tag{2.4}$$

$$x_a \in \mathbb{Z} \qquad \forall a \in A \tag{2.5}$$

Esta es la formulación propuesta por Dantzig, Fulkerson y Johnson (1954) para el TSP, y hace uso de las llamadas restricciones de eliminación de subciclos,

que corresponden con las restricciones 2.3, que fuerzan que en cualquier subconjunto de nodos S contenido estrictamente en V, el número de arcos recorridos por el vehículo sea menor que el número de nodos en el subconjunto, obligando que de ese subconjunto salga el vehículo y evitando así los subciclos.

Con las restricciones 2.2 forzamos que entre y salga un único arco de cada nodo. Esto se refuerza con las restricciones 2.4 y 2.5, que hacen que las variables solo tomen valores binarios.

Formulación con variables de potencial

Definimos las variables enteras u_i, que indican la posición de cada nodo i en la ruta del vehículo. Es decir:

$$u_i \equiv \text{posición del nodo } i \in N \text{ en la ruta.}$$

Con estas variables se puede formular el TSP como:

$$\text{mín} \sum_{a \in A} c_a x_a \tag{2.6}$$

Sujeto a:

$$\sum_{a \in \delta^+(i)} x_a = \sum_{a \in \delta^-(i)} x_a = 1 \qquad \forall i \in V. \tag{2.7}$$

$$u_j \geq u_i + x_{ij} - (n-2)(1-x_{ij}) + (n-3)x_{ji} \qquad \forall i, j \in N, i \neq j \tag{2.8}$$

$$0 \leq x_a \leq 1 \qquad \forall a \in A \tag{2.9}$$

$$x_a \in \mathbb{Z} \qquad \forall a \in A \tag{2.10}$$

En esta formulación, vemos que las restricciones 2.7, 2.9 y 2.10 se corresponden con las descritas anteriormente. Luego, respecto a la formulación anterior, cambiamos las restricciones 2.3 por las 2.8, ya que estas fuerzan que, partiendo de una ciudad determinada y llegando a la misma, el orden que se sigue sea coherente, evitando así lo subciclos.

Formulación con variables de flujo

Definimos las variables continuas de flujo $f_a \geq 0$, para cada $a \in A$, que representan un flujo que, al pasar por los nodos, se le resta una unidad.

$$f_a \equiv \text{carga del vehículo cuando pasa por el arco } a \in A.$$

Con ellas podemos formular el TSP de la siguiente manera:

$$\text{mín} \sum_{a \in A} c_a x_a \tag{2.11}$$

Sujeto a:

$$\sum_{a \in \delta^+(i)} x_a = \sum_{a \in \delta^-(i)} x_a = 1 \qquad \forall i \in V \tag{2.12}$$

$$\sum_{a \in \delta^-(i)} f_a - \sum_{a \in \delta^+(i)} f_a = 1 \qquad \forall i \in N \tag{2.13}$$

$$f_{0i} = n x_{0i} \qquad \forall i \in N \tag{2.14}$$

$$f_{i0} = 0 \qquad \forall i \in N \tag{2.15}$$

$$0 \leq f_a \leq n x_a \qquad \forall a \in A \tag{2.16}$$

$$0 \leq x_a \leq 1 \qquad \forall a \in A \tag{2.17}$$

$$x_a \in \mathbb{Z} \qquad \forall a \in A \tag{2.18}$$

En esta formulación, mantenemos las restricciones 2.12, 2.17 y 2.18 descritas anteriormente, y se añaden las restricciones 2.13, conocidas como restricciones de flujo de productos, consiguiendo que se descargue una unidad de flujo cada vez que se pase por un nodo. Con las restricciones 2.16 y 2.14 se garantiza que desde el nodo inicial se salga con un flujo igual al número de ciudades a visitar y que solo haya flujo entre dos ciudades si existe una conexión directa entre ellas. Además, con 2.15 no se permite que el vehículo vuelva al nodo de partida con flujo.

2.2. Problema de Rutas de Vehículos (VRP)

El Problema de Rutas de Vehículos, VRP por sus siglas en inglés *(Vehicle Routing Problem)*, es una variación del TSP que consiste en determinar un conjunto de rutas de costo mínimo que comienzan y terminan en un depósito, cumpliendo que los vehículos, con una determinada capacidad, visiten a los clientes como máximo una vez y satisfagan su demanda. Las restricciones deben modelar la capacidad limitada de los vehículos, de modo que la demanda total de los clientes visitados en una ruta por un vehículo no supere la capacidad del mismo. La minimización del coste total del transporte es uno de los objetivos más frecuentes en estos problemas y es usado diariamente por una gran cantidad de distribuidores y comerciantes por su significativa importancia económica.

El problema fue introducido cerca de los años 50 por Dantzig y Rmaser [6] y desde entonces se ha promovido a grandes rasgos su investigación. Sin embargo, no existe una definición universal del VRP debido a la gran variedad de restricciones y problemas encontradados en la práctica. Muchas de las investigaciones se han centrado en estudiar una versión del problema, llamada VRP clásico, mediante la aplicación de algoritmos heurísticos, diseñados para abordar las situaciones más complejas de la vida cotidiana (ver [7]).

2.2.1. Planteamiento del problema para el caso asimétrico

El VRP clásico está definido sobre un grafo dirigido $G = (V, A)$ con el conjunto de vértices $V = \{0\} \cup N$, siendo $\{0\}$ el depósito y $N = \{1, ..., n\}$ el conjunto de nodos que corresponden a los clientes. En el depósito se encuentran k vehículos iguales con capacidad Q, luego existe un conjunto $K = \{1, ..., k\}$ de vehículos. Cada cliente tiene asociada una demanda no negativa d_i, tal que $d_i \leq Q$. Sea $A \subseteq \{(i, j) : i, j \in V, i \neq j\}$ el conjunto de arcos que une los pares de nodos. Cada arco $a = (i, j) \in A$ tiene asignado un costo no negativo c_a. Además, como estamos ante el caso asimétrico, tenemos en general que $c_{ij} \neq c_{ji}$, para todo $(i, j) \in A$.

Formulación de dos índices

Definimos las variables de decisión x_a :

$$x_a = \begin{cases} 1, & \text{si el vehículo pasa por el arco } a \in A, \\ 0, & \text{en otro caso.} \end{cases}$$

Con estas variables, el VRP se puede formular como:

$$\text{mín} \sum_{a \in A} c_a x_a \tag{2.19}$$

Sujeto a:

$$\sum_{a \in \delta^+(i)} x_a = \sum_{a \in \delta^-(i)} x_a = 1 \qquad \forall i \in N. \tag{2.20}$$

$$\sum_{a \in \delta^+(0)} x_a = \sum_{a \in \delta^-(0)} x_a \leq k \tag{2.21}$$

$$\sum_{a \in \delta^+(S)} x_a \geq \lceil \sum_{i \in S} \frac{d_i}{Q} \rceil \qquad \forall S \subset N \tag{2.22}$$

$$x_a \in \{0, 1\} \tag{2.23}$$

Observamos que con las restricciones 2.20 imponemos que de cada cliente entre y salga un único vehículo. Luego, con las restricciones 2.21 conseguimos que del depósito salgan y regresen los mismos vehículos, teniendo en cuenta que no tienen por qué salir los k vehículos. Por último, con las restricciones 2.22 evitamos la formación de subciclos y que se viole la capacidad de los vehículos. Además, con las restricciones 2.23 indicamos que nuestras variables de decisión son binarias.

Formulación de tres índices

Definimos las siguientes variables de decisión:

$$x_a^k = \begin{cases} 1, & \text{si el vehículo } k \in K \text{ pasa por el arco } a \in A, \\ 0, & \text{en otro caso.} \end{cases}$$

$$y_i^k = \begin{cases} 1, & \text{si el vehículo } k \in K \text{ visita el nodo } i \in V, \\ 0, & \text{en otro caso.} \end{cases}$$

El VRP se puede modelizar como:

$$\text{mín} \sum_{k \in K} \sum_{a \in A} c_a x_a^k \tag{2.24}$$

Sujeto a:

$$\sum_{k \in K} y_i^k = 1 \qquad \forall i \in N \tag{2.25}$$

$$\sum_{a \in \delta^+(i)} x_a^k = \sum_{a \in \delta^-(i)} x_a^k = y_i^k \qquad \forall i \in V, k \in K \tag{2.26}$$

$$\sum_{a \in \delta^+(S)} x_a^k \geq y_j^k \qquad \forall S \subseteq N, j \in S, k \in K \tag{2.27}$$

$$\sum_{i \in N} d_i y_i^k \leq Q y_0^k \qquad \forall k \in K \tag{2.28}$$

$$x_a^k \in \{0, 1\} \qquad \forall a \in A, k \in K \tag{2.29}$$

$$y_i^k \in \{0, 1\} \qquad \forall i \in V, k \in K \tag{2.30}$$

En este modelo, observamos que se especifica que el vehículo recorre cada arco y visita cada vértice. Con las restricciones 2.25 imponemos que cada cliente

sea visitado por un único vehículo. Por otra parte, con las restricciones 2.26, exigimos que si un vehículo llega a un vértice, también debe salir de él. Luego, con las restricciones 2.27 evitamos la formación de subciclos que no pasen por el depósito. Las restricciones 2.28 impiden que se sobrepase la capacidad de los vehículos que se usan para hacer las rutas. Por último, con las restricciones 2.29 y 2.30 indicamos que nuestras variables de decisión son binarias.

Formulación con variables de flujo

Definimos las variables de flujo f_a:

$$f_a \equiv \text{carga del vehículo cuando pasa por el arco } a \in A.$$

Luego, definimos el modelo para el VRP como:

$$\text{mín} \sum_{a \in A} c_a x_a \tag{2.31}$$

Sujeto a:

$$\sum_{a \in \delta^+(i)} x_a = \sum_{a \in \delta^-(i)} x_a = 1 \qquad \forall i \in N \tag{2.32}$$

$$\sum_{a \in \delta^+(0)} x_a = \sum_{a \in \delta^-(0)} x_a \leq k \tag{2.33}$$

$$\sum_{a \in \delta^+(i)} f_a - \sum_{a \in \delta^-(i)} f_a = d_i \qquad \forall i \in N \tag{2.34}$$

$$f_a \leq Q x_a \qquad \forall a \in A \tag{2.35}$$

$$x_a \in \{0, 1\} \qquad \forall a \in A \tag{2.36}$$

En este caso, con las restricciones 2.32 imponemos que un solo arco entre y salga de cada vértice y con 2.33 indicamos que del depósito salen y entran el mismo número de vehículos. Además, pueden no usarse los k vehículos de los que disponemos. Las restricciones de flujo 2.34 imponen que la diferencia entre la carga de un vehículo al llegar a un cliente y al salir de él sea igual a la demanda de dicho cliente. Las restricciones 2.35 obligan a que se respete la capacidad de los vehículos. Por último, como en las otras formulaciones, tomamos variables de decisión binarias con las restricciones 2.36.

3

El Problema Periódico de Rutas de Vehículos (PVRP)

En este capítulo veremos el Problema Periódico de Rutas de Vehículos o PVRP (por sus siglas en inglés *Periodic Vehicle Routing Problem*). Este problema es una extensión del VRP en el que cada cliente tiene que ser visitado una o varias veces a lo largo de un horizonte temporal de varios periodos, y en cada visita demanda la misma cantidad de producto. Los días de visitas a los clientes no están fijados, sino que, por el contrario, cada cliente tiene asociados unos determinados posibles calendarios de visitas y se debe elegir uno. Por ejemplo, si el horizonte temporal son los días de lunes a viernes y un cliente debe ser visitado dos veces, con al menos un día y como mucho dos entre las visitas, entonces los posibles calendarios de visitas para ese cliente son {{lunes, miércoles}, {lunes, jueves}, {martes, jueves}, {martes, viernes}, {miércoles, viernes}}, y de ese conjunto de calendarios hay que escoger uno. El objetivo del problema es elegir un calendario de visita para cada cliente y diseñar las rutas de los vehículos para cada uno de los periodos del horizonte temporal, de manera que se minimice el coste total de las rutas.

3.1. Descripción del problema

El PVRP fue introducido por Beltrami y Bodin en 1974 [2] a causa de la necesidad de planificar rutas a lo largo de un horizonte temporal para satisfaccer la demanda de clientes que requerían múltiples visitas. En particular, estudiaban un modelo de recogida de residuos municipales.

Matemáticamente el PVRP se define sobre un grafo completo no dirigido $G = (V, A)$, con un conjunto de nodos $V = \{0, ..., n\}$, donde el nodo $\{0\}$ representa el depósito y el resto de nodos, $N = \{1, ..., n\}$, corresponden a los clientes, y un conjunto de arcos $A = \{(i, j) : i, j \in V, i \neq j\}$. Cada arco $a \in A$ tiene asociado un costo no negativo c_a. Sea $K = \{1, ..., k\}$ el conjunto de vehículos, cada uno con una capacidad Q, y $T = \{1, ..., t\}$ el conjunto de periodos (días)

que definen el horizonte temporal. Cada cliente $i \in N$ tiene asociado una demanda d_i a satisfacer en cada visita, y un conjunto de posibles calendarios de visitas $P_i = \{1, ..., p\}$. Un calendario $p \in P_i$ consiste en un conjunto de días en los que el cliente i puede ser visitado, es decir, $p \subseteq T$. El objetivo del problema es seleccionar un calendario de visitas para cada cliente y diseñar las rutas para los vehículos en cada periodo, de manera que se minimice el costo total. Una solución factible del PVRP está formada por t conjuntos de rutas que satisfacen las demandas de los clientes sin superar la capacidad de los vehículos.

En la siguiente sección estudiaremos para el PVRP dos formulaciones MILP, por sus siglas en inglés, *Mixed Integer Linear Problems* (programación lineal entera mixta). En concreto, veremos el denominado modelo de cargas, una formulación basada en variables de flujo propuesta por Archetti et al. [1] en 2017, y también veremos un modelo que usa restricciones de tipo Miller-Tucker-Zemlin (MTZ) propuesto por Basir et al. [4] en 2024.

3.2. Modelos matemáticos

Para formular el PVRP definimos las siguientes variables de decisión para representar, respectivamente, las rutas escogidas, los cliente visitados y los calendarios de visitas elegidos:

$$x_a^t = x_{ij}^t \begin{cases} 1, & \text{si un vehículo pasa por el arco } a \in A \text{ el día } t \in T, \\ 0, & \text{en otro caso.} \end{cases}$$

$$y_i^t = \begin{cases} 1, & \text{si el cliente } i \in N \text{ es visitado el día } t \in T, \\ 0, & \text{en otro caso.} \end{cases}$$

$$s_{ip} = \begin{cases} 1, & \text{si se escoge el calendario } p \in P_i \text{ para visitar al cliente } i \in N, \\ 0, & \text{en otro caso.} \end{cases}$$

Estas variables son comunes en los dos modelos que veremos a continuación.

3.2.1. Modelo MTZ

Definimos las variables continuas de decisión u_i^t de la siguiente forma:

$$u_i^t \equiv \text{carga del vehículo cuando llega al nodo } i \in N \text{ el día } t \in T.$$

Con estas variables y con las escritas anteriormente podemos formular el PVRP como:

$$\text{mín} \sum_{t \in T} \sum_{a \in A} c_a x_a^t \tag{3.1}$$

Sujeto a:

$$\sum_{p \in P_i} s_{ip} = 1 \qquad \forall i \in N \tag{3.2}$$

$$\sum_{p \in P_i : t \in p} s_{ip} = y_i^t \qquad \forall i \in N, t \in T, \tag{3.3}$$

$$x_{ij}^t \leq (y_i^t + y_j^t)/2 \qquad \forall i, j \in N, i \neq j, t \in T \tag{3.4}$$

$$\sum_{a \in \delta^+(0)} x_a^t \leq k \qquad \forall t \in T \tag{3.5}$$

$$\sum_{j \in V \setminus \{i\}} x_{ij}^t = y_i^t \qquad \forall i \in N, t \in T \tag{3.6}$$

$$\sum_{j \in V \setminus \{i\}} x_{ji}^t = y_i^t \qquad \forall i \in N, t \in T \tag{3.7}$$

$$u_j^t \leq u_i^t - d_i y_i^t + Q(1 - x_{ij}^t) \qquad \forall i, j \in N, t \in T \tag{3.8}$$

$$d_i \leq u_i^t \leq Q \qquad \forall i \in N, t \in T \tag{3.9}$$

$$x_a^t \in \{0, 1\} \qquad \forall a \in A, t \in T \tag{3.10}$$

$$y_i^t \in \{0, 1\} \qquad \forall i \in N, t \in T \tag{3.11}$$

$$s_{ip} \in \{0, 1\} \qquad \forall i \in N, p \in P_i \tag{3.12}$$

La función objetivo 3.1 minimiza el costo total de las rutas. Las restricciones 3.2 imponen que se escoja un calendario de visitas para cada cliente. Con las 3.3 imponemos que si se escoge el calendario de visita $p \in P_i$ para el cliente $i \in N$, entonces se visita al cliente en los días $t \in p$, es decir, se relacionan correctamente las variables s_{ip} y y_i^t. Con la restricción 3.4 garantizamos que solo se utilizan los arcos que conectan a las parejas de clientes que se visitan el mismo día. Las restricciones 3.5 fuerzan que salgan del depósito cada día como máximo los k vehículos de los que disponemos. Cualquier cliente que sea servido un determinado día será visitado exactamente una única vez debido a las restricciones 3.6 y 3.7, que juntas garantizan la conservación del flujo de vehículos. Los subtours se evitan gracias a las restricciones 3.8, que imponen que la carga del vehículo disminuya al pasar por cada cliente. Las restricción 3.9

garantizan que se satisfaga las demandas de los clientes respetando la capacidad de los vehículos. Por último, con las restricciones 3.10, 3.11 y 3.12 definimos nuestras variables binarias.

Restricciones válidas para el modelo MTZ

A continuación presentamos algunas inecuaciones válidas, que sirven para reforzar el modelo MTZ e intentar obtener mejores resultados computacionales. Están inspiradas en las restricciones válidas inicialmente desarrolladas para el Problema de Rutas de Vehículos con Capacidades o CVRP (por sus siglas en inglés, *Capacitated Vehicle Routing Problem*), y son descritas en [4].

En primer lugar, tenemos las restricciones I1 (3.13), que refuerzan y sustituyen las 3.8 para prevenir la formación de subtours. La demostración se puede ver en [4].

$$u_j^t - u_i^t + Qx_{ij}^t + (Q - d_j - d_i)x_{ji}^t \leq Q - d_i \qquad \forall i, j \in N, t \in T \tag{3.13}$$

Por otra parte, tenemos las restricciones I2 (3.14) de ruptura de simetría en las rutas cuando los costos de los arcos son simétricos, es decir, cuando cada ruta tiene un mismo costo independientemente de su orientación. Con ellas solo consideramos las rutas con una orientación determinada, ya que la misma ruta con la orientación opuesta tendrá el mismo costo. Entre las dos orientaciones posibles de una ruta, escogemos la que comience por el cliente con menor índice. Para ello imponemos las siguientes restricciones:

$$x_{i0}^t \leq \sum_{r \leq i} x_{0r}^t \qquad \forall i \in N, t \in T \tag{3.14}$$

Es decir, si el arco $(i, 0)$ entra en el depósito en el periodo de tiempo t, entonces debe haber un arco $(0, r)$ con $r \leq i$ que salga del depósito en este periodo.

Por último, las restricciones I3 (3.15) que relacionan las variables x_a^t e y_i^t, afirman que el arco (i, j), que une los clientes i y j, no puede ser usado el día t a menos que ambos clientes sean visitados ese mismo día. Teniendo en cuenta que ningún arco puede ser atravesado en ambas direcciones en el mismo periodo de tiempo, podemos reemplazar estas restricciones por las inecuaciones válidas 3.15.

$$x_{ij}^t + x_{ji}^t \leq (y_i^t + y_j^t)/2 \qquad \forall i, j \in N, i < j, \forall t \in T \tag{3.15}$$

3.2.2. Modelo de cargas

En el modelo de cargas, además de las variables x, y y s definidas anteriormente, utilizamos las siguientes variables continuas:

$l_a^t \equiv$ carga del vehículo cuando atraviesa el arco $a = (i, j) \in A$ el día $t \in T$.

El modelo basado en cargas para el PVRP es:

$$\text{mín} \sum_{t \in T} \sum_{a \in A} c_a x_a^t \tag{3.16}$$

Sujeto a:

$$\sum_{p \in P_i} s_{ip} = 1 \qquad \forall i \in N \tag{3.17}$$

$$\sum_{p \in P_i : t \in p} s_{ip} = y_i^t \qquad \forall i \in N, t \in T \tag{3.18}$$

$$x_{ij}^t \leq (y_i^t + y_j^t)/2 \qquad \forall i, j \in N, i \neq j, t \in T \tag{3.19}$$

$$\sum_{a \in \delta^+(0)} x_a^t \leq k \qquad \forall t \in T \tag{3.20}$$

$$\sum_{j \in V \setminus \{i\}} x_{ij}^t = y_i^t \qquad \forall i \in N, t \in T \tag{3.21}$$

$$\sum_{a \in \delta^+(i)} x_a^t = \sum_{a \in \delta^-(i)} x_a^t \qquad \forall i \in N, t \in T \tag{3.22}$$

$$\sum_{a \in \delta^+(i)} l_a^t - \sum_{a \in \delta^-(i)} l_a^t = \begin{cases} -d_i y_i^t, & i \in N \\ \sum_{j \in N} d_j y_j^t, & i = 0 \end{cases} \qquad \forall t \in T \tag{3.23}$$

$$l_a^t \leq Q x_a^t \qquad \forall a \in A, t \in T \tag{3.24}$$

$$x_a^t \in \{0, 1\} \qquad \forall a \in A, t \in T \tag{3.25}$$

$$y_i^t \in \{0, 1\} \qquad \forall i \in N, t \in T \tag{3.26}$$

$$s_{ip} \in \{0, 1\} \qquad \forall i \in N, p \in P_i \tag{3.27}$$

$$l_a^t \geq 0 \qquad \forall a \in A, t \in T \tag{3.28}$$

De forma análoga al modelo anterior, la función objetivo 3.16 minimiza el costo total de las rutas y mantenemos algunas restricciones descritas anteriormente, como las 3.17 - 3.20, con las que imponemos que se escoja un calendario de visitas adecuado, que en un determinado día solo se utilicen los arcos que conectan a las parejas de clientes visitados ese día, y forzamos a que el número de vehículos utilizados cada día no supere el número de los que disponemos. Las restricciones 3.21 y 3.22 son las de conservación de flujo para los vehículos, obligando así que cada cliente sea visitado una única vez en el mismo día, al igual que las restricciones 3.6 y 3.7 del modelo anterior.

Ahora, añadimos las restricciones 3.23, que son también restricciones de flujo pero para la carga de los vehículos. Estas restricciones imponen que, en cada periodo, la carga de los vehículos que salen del depósito sea suficiente para satisfacer la demanda de todos los clientes visitados ese día, y que a cada cliente se le suministre lo que pide. Además, con las restricciones 3.24 garantizamos que no se sobrepase la capacidad de los vehículos.

Por último, las restricciones 3.25 - 3.28 indican el dominio de cada tipo de variable.

Restricciones válidas para el modelo de cargas

A continuación presentamos algunas restricciones válidas, que ayudan a fortalecer el modelo de cargas y a intentar obtener mejores resultados computacionales.

Particularmente, en este modelo consideramos la restricción I4 (3.29) de suma de las cargas finales, que impone que todos los vehículos vuelvan vacíos al depósito en todos los periodos. Esta restricción no es válida porque existen soluciones factibles que no la satisfacen. Sin embargo, es un corte óptimo, lo que significa que al menos una solución óptima cumple la restricción. Por lo tanto, puede ser utilizada para disminuir el conjunto de las soluciones que se exploran y así reducir el tiempo de resolución del problema. La restricción es:

$$\sum_{t \in T} \sum_{j \in N} l_{j0}^t = 0. \tag{3.29}$$

Por otra parte, las restricciones válidas I2 (3.14) e I3 (3.15) descritas para el modelo MTZ también son válidas para el modelo de cargas.

4

Experimentos computacionales

En este capítulo presentamos los resultados obtenidos al implementar los dos modelos para el PVRP propuestos anteriormente, con el objetivo de compararlos. También realizamos experimentos para evaluar la efectividad de las restricciones válidas para los dos modelos y mostramos la representación gráfica de una solución.

4.1. Material y herramientas de trabajo

Como resolvedor hemos utilizado Gurobi, un software comercial especializado en optimización matemática. Actualmente, es de los más utilizados en diversas industrias para resolver distintos tipos de problemas complejos ya que se caracteriza por ser de los más eficientes y avanzados del mercado. Además, ofrece soporte para múltiples lenguajes de programación, incluyendo Python, el lenguaje que hemos utilizado para desarrollar nuestros códigos.

Por otra parte, para ejecutar Gurobi en el ordenador hemos usado el editor de código fuente Visual Studio Code (VS Code) desarrollado por Microsoft. Es conocido por ser ligero, rápido y proporcionar un entorno optimizado y extensible para múltiples lenguajes y plataformas.

Las gráficas realizadas para dibujar los nodos y las rutas se han codificado en Python en Google Colaboratory, puesto que la interfaz gráfica de Visual Studio Code requiere de extensiones. Google Colaboratory, o Google Colab para abreviar, es un entorno gratuito proporcionado por Google que no requiere configuración y con el que se puede escribir y ejecutar códigos en Python en un navegador. Está especialmente diseñado para proyectos de ciencia de datos y aprendizaje automático. Además, permite la colaboración en tiempo real y la integración con Google Drive para la gestión de proyectos.

4.2. Instancias

Las instancias utilizadas para llevar a cabo nuestros experimentos computacionales son instancias del PVRP procedentes de [8]. Estas instancias provienen de la literatura, por lo que han sido utilizadas y documentadas previamente, y tienen las siguientes características. El número de nodos puede ser 11, 21, 31 o 41. El número de vehículos disponibles $|K|$ varía entre 2 y 4, al igual que el número de periodos $|T|$, que corresponde al número de días del horizonte temporal. Para cada combinación de número de nodos, de vehículos y de periodos, tenemos tres instancias, que denotamos con las letras a, b y c.

Las coordenadas de los clientes fueron generadas aleatoriamente en $[0, 100] \times [0, 100]$ y el depósito está situado en el punto de coordenadas $(50, 50)$. Las demandas d_i de los clientes varían entre 1 y 15. Los costos c_{ij} se calcularon como las distancias euclídeas entre i y j redondeadas al entero más cercano por abajo. La capacidad de los vehículos se generó aleatoriamente entre 50 y 100. En las instancias, se indica el número total de posibles calendarios de visitas asociados a cada cliente i y se almacenan todas las posibilidades en una lista. Cada calendario de visitas se codifica como un número decimal equivalente a una cadena de bits, es decir, una cadena formada por dígitos del sistema de numeración binario, representados por 0 y 1. Un 1 indica que ese día se realiza la visita, y un 0 lo contrario. Por ejemplo: si tomamos un horizonte temporal de 5 días, $|T| = 5$, un posible calendario de visitas para el cliente i es [01100], que significa que el cliente debe ser visitado en los días 2 y 3. Este calendario aparecería codificado como 12.

En la gráfica 4.1 mostramos estas características para la instancia 11-2-2-a, una instancia con 11 nodos, $|T| = 2$ y $|K| = 2$. En el centro distinguimos el depósito en color naranja y alrededor los 10 clientes en color azul, distribuidos en un área de 100×100. Identificamos cada cliente con una dupla, en la que la primera componente corresponde al número del cliente y la segunda a la demanda del mismo. En esta instancia, la demanda de los clientes es $d = \{3, 15, 13, 3, 3, 15, 15, 10, 11, 2\}$ y la capacidad de los vehículos es $Q = 51$.

Por otra parte, recalcamos que la complejidad del PVRP se debe no solo a la necesidad de diseñar las rutas para cada periodo, sino a que se debe además elegir un calendario de visita para cada cliente de entre los posibles. En la tabla 4.1 mostramos todos los posibles calendarios de visitas para cada cliente de la instancia 11-2-2-a descrita anteriormente.

Nótese que, en este caso, todos los clientes menos el 3 y el 6 tienen un único calendario de visitas. Los clientes 1, 2, 4, 5, 8 y 9 deben ser visitados en los dos periodos de este ejemplo. El cliente 7 solo puede ser visitado en el segundo periodo, y el cliente 10 solo puede ser visitado en el primer periodo. Los clientes 3 y 6 deben ser visitados solo uno de los dos periodos, pero se debe elegir en cuál de los dos.

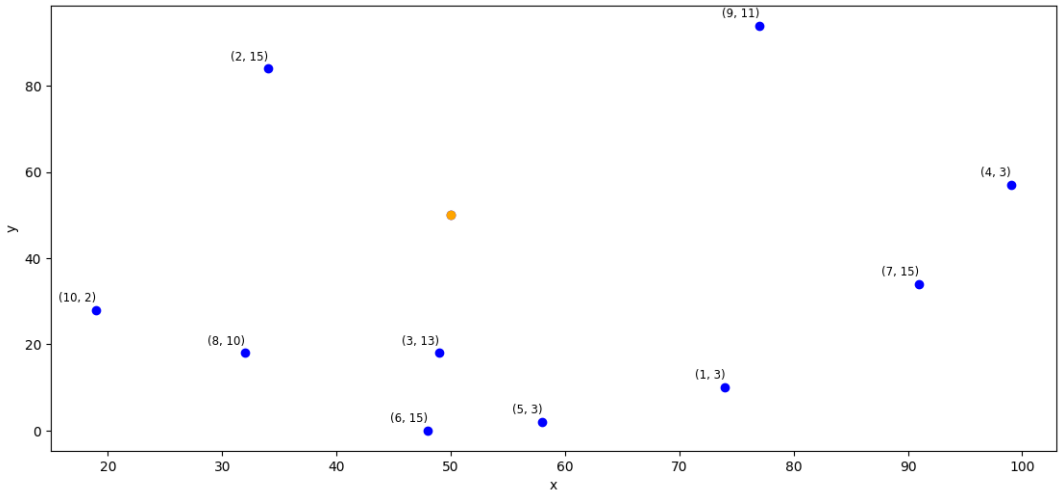

Figura 4.1. Representación instancia 11-2-2-a

Cliente	Calendarios posibles
1	[1, 1]
2	[1, 1]
3	[1, 0], [0, 1]
4	[1, 1]
5	[1, 1]
6	[0, 1], [1, 0]
7	[0, 1]
8	[1, 1]
9	[1, 1]
10	[1, 0]

Tabla 4.1. Calendarios de visitas para los clientes de la instancia 11-2-2-a

4.3. Comparación entre el modelo MTZ y el modelo de cargas

En la tabla 4.2, podemos ver los segundos que tarda cada modelo en hallar una solución para cada una de las instancias de 11 nodos con 2 o 3 periodos ($|T|$) y 2 o 3 vehículos ($|K|$). Para cada combinación de $|T|$ y $|K|$ tenemos 3 instancias denotadas por a, b y c. Para el modelo MTZ ejecutamos en Python el modelo (3.1) sujeto a (3.2)-(3.12), y para el de cargas el modelo (3.16) sujeto a (3.17)-(3.28). Mostramos también el valor óptimo del problema PVRP para estas instancias. En negrita recalcamos el modelo que resuelve el problema en un menor tiempo de ejecución. Observamos que el modelo de cargas es más eficiente que el modelo MTZ pues en 11 de las 12 instancias requiere de un menor tiempo de ejecución.

En la tabla 4.3, mostramos los segundos que tarda cada modelo en obtener la solución para las instancias más sencillas de 21 nodos, con $|T| = 2$ y $|K| = 2$. Mostramos además el valor óptimo del problema PVRP para estas instancias. Cada instancia se ejecutó con un límite de 3600 segundos de tiempo de cómputo. Cuando se alcanza ese límite se indica con la palabra *superado* en la tabla. Vemos que la la instancia 21-2-2-b es no factible. Ambos modelos encuentran la solución óptima de la instancia 21-2-2-a, pero el modelo MTZ tarda unos 1319 segundo, mientras que el modelo de cargas lo hace en menos de un segundo. En cuanto a la instancia 21-2-2-c, el modelo MTZ no es capaz de encontrar la solución óptima en una hora de cómputo, mientras que el modelo de cargas la encuentra en menos de 6 segundo. A la vista de estos resultados, podemos concluir que el modelo de cargas es claramente más eficiente que el modelo MTZ, y será el que usemos para resolver el resto de instancias con 21 o más nodos.

				Segundos para hallar la solución		
$\|V\|$	$\|T\|$	$\|K\|$	Nom.	Mod. MTZ	Mod. Cargas	Valor óptimo
11	2	2	a	2.255	**0.283**	614
11	2	2	b	0.420	**0.222**	547
11	2	2	c	0.432	**0.066**	574
11	2	3	a	2.356	**0.484**	614
11	2	3	b	**0.331**	0.453	547
11	2	3	c	0.399	**0.140**	574
11	3	2	a	388.493	**1.342**	827
11	3	2	b	8.317	**0.532**	923
11	3	2	c	1.371	**0.196**	666
11	3	3	a	381.811	**1.155**	827
11	3	3	b	8.020	**0.631**	923
11	3	3	c	1.085	**0.119**	666

Tabla 4.2.

				Segundos para hallar la solución		
$\|V\|$	$\|T\|$	$\|K\|$	Nom.	Mod. MTZ	Mod. Cargas	Valor óptimo
21	2	2	a	1319.139	**0.738**	818
21	2	2	b			no factible
21	2	2	c	superado	**5.630**	794

Tabla 4.3.

Por otra parte, en la tabla 4.2 se observa que, dada una instancia con un tamaño y un número de periodos determinados, el valor óptimo del problema es el mismo independientemente del número de vehículos disponibles $|K|$ (siempre y cuando la instancia sea factible). Esto es así porque el objetivo del PVRP es minimizar el costo de las rutas, por lo que, como no se impone que salgan todos los vehículos, la solución óptima siempre utiliza el menor número de vehículos posible, aunque se disponga de más.

4.4. Mejoras del modelo MTZ

En la sección anterior observamos que el tiempo de ejecución del modelo MTZ es muy elevado. Por ello, en esta sección evaluamos posibles mejoras en el modelo mediante la incorporación de restricciones válidas. Implementaremos en Python el modelo (3.1) sujeto a (3.2)-(3.12) y las restricciones válidas I1 (3.13), I2 (3.14) e I3 (3.15), descritas en el capítulo 3, y hacemos el experimento usando las instancias con 11 nodos.

Los resultados se presentan en la tabla 4.4. La columna *Base* muestra los tiempos obtenidos resolviendo el modelo (3.1) sujeto a (3.2)-(3.12), mientras que las demás columnas muestran los tiempos obtenidos cuando se amplía el modelo base con una o varias de las familias de restricciones válidas. Para cada instancia se destaca en negrita el menor tiempo de ejecución. Observamos que ninguna configuración es la mejor para la mayoría de instancias, sin embargo, notamos que el modelo base con la familia de restricciones I2 + I3 es la mejor en 3 de las 12 instancias y, en general, para el resto de instancias, obtiene un tiempo similar al mejor tiempo hallado.

4.5. Mejoras del modelo de cargas

En esta sección evaluamos las posibles mejoras en el modelo de cargas mediante la adición de restricciones válidas. Para ello, implementaremos en Python el modelo (3.16) sujeto a (3.17)-(3.28), y añadimos las restricciones válidas I2 (3.14), I3 (3.15) e I4 (3.29), descritas en el capítulo 3. Realizamos el experimento usando las instancias de 11 nodos.

Podemos ver los resultados en la tabla 4.5, donde la columna *Base* muestra los tiempos de ejecución del modelo (3.16) sujeto a (3.17)-(3.28) y las demás columnas muestran los tiempos obtenidos cuando se amplía el modelo base con una o varias de las familias de restricciones válidas. En cada instancia, se resalta en negrita el menor tiempo de ejecución. Notamos que no hay un modelo que mejore todos los tiempos; algunos de ellos solo mejoran una instancia aislada. No obstante, el modelo base con las restricciones I2 + I3 obtiene el mejor tiempo para 5 de las 12 instancias y obtiene buenos tiempos para todas las instancias pues

			Segundos para hallar la solución							
$\|T\|$	$\|K\|$	Nom.	Base	I1	I2	I3	I1 + I2	I1 +I3	I2 + I3	I1+I2+I3
2	2	a	2.255	1.206	1.353	1.828	1.509	2.632	**1.069**	1.565
2	2	b	0.420	0.674	**0.385**	0.639	0.854	0.428	0.474	0.700
2	2	c	**0.432**	0.491	0.612	0.555	0.514	0.451	0.819	0.553
2	3	a	2.356	1.524	1.343	1.947	1.564	2.311	**1.008**	1.515
2	3	b	**0.331**	0.621	0.711	0.900	0.765	0.624	0.676	0.862
2	3	c	0.399	0.613	0.786	0.802	0.480	0.439	**0.354**	0.502
3	2	a	388.493	425.220	162.581	247.687	**143.131**	640.994	192.456	154.382
3	2	b	8.317	5.802	7.133	11.026	9.544	4.942	4.590	**3.590**
3	2	c	1.371	1.496	1.161	0.920	1.350	**0.595**	0.745	1.531
3	3	a	381.811	468.548	**134.335**	266.682	136.427	404.668	187.342	152.608
3	3	b	8.020	5.990	6.432	10.042	9.324	4.559	4.792	**3.737**
3	3	c	1.085	1.443	1.208	0.869	1.448	**0.604**	0.859	1.240

Tabla 4.4. Resultados para $|V| = 11$ (Modelo MTZ)

todos son inferiores a un segundo. Por lo que, para los siguientes experimentos usaremos este modelo con esta configuración.

			Segundos para hallar la solución							
$\|T\|$	$\|K\|$	Nom.	Base	I2	I3	I4	I2 +I3	I2 + I4	I3 + I4	I2 + I3 + I4
2	2	a	0.283	**0.164**	0.279	0.198	0.108	0.217	0.171	0.205
2	2	b	0.222	0.303	0.088	0.408	0.200	0.441	0.138	**0.060**
2	2	c	0.066	0.145	0.078	0.188	**0.059**	0.082	0.230	0.137
2	3	a	0.484	0.234	0.279	0.192	**0.112**	0.172	0.271	0.389
2	3	b	0.453	0.363	**0.058**	0.321	0.079	0.132	**0.058**	0.102
2	3	c	0.140	0.162	0.064	0.166	**0.059**	0.081	0.065	0.082
3	2	a	1.342	0.843	**0.529**	1.156	0.615	0.983	0.671	0.831
3	2	b	**0.532**	1.525	1.081	1.099	0.906	1.431	0.987	1.341
3	2	c	0.196	0.341	0.081	0.105	**0.073**	0.148	0.091	0.080
3	3	a	1.155	0.596	1.173	0.815	**0.487**	0.760	0.923	1.186
3	3	b	0.631	1.153	**0.485**	0.788	0.714	0.865	1.262	0.863
3	3	c	0.119	0.258	0.143	0.096	0.137	0.172	**0.081**	0.108

Tabla 4.5. Resultados para $|V| = 11$ (Modelo de cargas)

4.6. Resultados completos

En esta sección resolvemos el PVRP en el resto de instancias con 21, 31 y 41 nodos usando el modelo de cargas reforzado con las restricciones válidas I2 e I3, ya que esta es la configuración que resulta más ventajosa según lo visto anteriormente. Los resultados se muestran en las tablas 4.6, 4.7 y 4.8. Cada instancia se procesó con un tiempo límite de 10 minutos, y mostramos el tiempo de computación requerido y el valor de la solución encontrada. Cuando se alcanza el tiempo límite se indica con la palabra *superado* en la tabla.

En la tabla 4.6 se ve que se consigue resolver de forma óptima todas las instancias factibles con 21 nodos, 2 o 3 periodos, y 2, 3 o 4 vehículos. Se observa que, para una flota de igual tamaño, el problema es más difícil si pasamos de 2 a 3 periodos. Por otra parte, en estas instancias no se observa una tendencia clara en la evolución de los tiempos de cómputo cuando permanece fijo el número de periodos y se aumenta el número de vehículos.

Las tablas 4.7 y 4.8 muestran los resultados para las instancias con 31 y 41 nodos, respectivamente. Vemos que en la mayoría de los casos el problema es no factible si el número de vehículos es $|K| = 2$, esto quiere decir que la capacidad de los vehículos es insuficiente para cumplir con la demanda de tantos clientes. Cuando el número de vehículos es 3 o 4, en la mayoría de las instancias se llega al tiempo máximo de cómputo. Cuando esto ocurre, señalamos con un asterisco (*) la mejor solución hallada en el tiempo establecido, pero no podemos asegurar que la solución sea óptima. Podemos observar que, como cabe esperar, el problema se hace más difícil conforme aumentan el número de clientes, el número de periodos y el número de vehículos.

Tiempo y valor óptimo de cada problema

		$\|K\| = 2$		$\|K\| = 3$		$\|K\| = 4$	
$\|T\|$	Nom.	seg.	sol	seg.	sol	seg.	sol
2	a	0.470	818	0.581	818	0.572	818
2	b		no factible	27.41	784	23.759	784
2	c	2.830	794	8.382	793	8.423	793
3	a	2.654	1073	2.577	1073	3.024	1073
3	b		no factible	75.024	1193	77.603	1193
3	c	22.047	949	14.430	949	35.368	949

Tabla 4.6. Resultados para $|V| = 21$

Tiempo y valor óptimo de cada problema

		$\|K\| = 2$		$\|K\| = 3$		$\|K\| = 4$	
$\|T\|$	Nom.	seg.	sol.	seg.	sol.	seg.	sol.
2	a		no factible	superado	904*	superado	893*
2	b		no factible	24.568	1001	56.190	1001
2	c		no factible	66.457	898	287.560	898
3	a		no factible	superado	1261*	superado	1261*
3	b	56.061	1277	110.282	1277	112.630	1277
3	c		no factible	superado	1296*	superado	1288*
4	a		no factible	superado	1729*	superado	1729*
4	b	superado	1493*	superado	1501*	superado	1493*
4	c		no factible	superado	1768*	superado	1797*

Tabla 4.7. Resultados para $|V| = 31$

Tiempo y valor óptimo de cada problema

		$\|K\| = 2$		$\|K\| = 3$		$\|K\| = 4$	
$\|T\|$	Nom.	seg.	sol.	seg.	sol.	seg.	sol.
2	a		no factible		no factible	superado	1069*
2	b		no factible	448.208	1099	superado	1100*
2	c		no factible	superado	1092*	superado	1088*
3	a		no factible		no factible	superado	1451*
3	b		no factible	444.837	1542	577.272	1542
3	c		no factible	superado	1598*	superado	1598*
4	a		no factible	superado	1748*	superado	1732*
4	b		no factible	superado	1916*	superado	1943*
4	c		no factible	superado	2054*	superado	1970*

Tabla 4.8. Resultados para $|V| = 41$

4.7. Representación gráfica de una solución

Para acabar, mostramos en esta sección la representación gráfica de la solución óptima de la instancias 11-2-2-a. Se trata de la instancia con diez clientes, dos periodos y dos vehículos con capacidad 51, descrita en la sección 4.2. La solución óptima de esta instancia vale 614. En las figuras 4.2 y 4.3 se muestran las rutas de los vehículos en los periodos 1 y 2, respectivamente. Al lado de cada cliente i aparece la dupla (i, d_i), siendo d_i la demanda de ese cliente. Observamos que en el primer periodo se necesita un solo vehículo y en el segundo periodo se

requieren los dos (las rutas de los dos vehículos se representan con distinto tipo de línea). La suma de las demandas de los clientes visitados en cada periodo por cada vehículo no supera la capacidad máxima del mismo, y cada cliente se visita siguiendo uno de sus posibles calendarios de visita. Así, los clientes 1, 2, 4, 5, 8 y 9 son visitados en los dos periodos, el cliente 7 es visitado en el segundo periodo y el cliente 10 es visitado en el primer periodo. Respecto a los clientes 3 y 6, deben ser visitados en solo uno de los dos periodos, y se ha escogido visitarlos a ambos en el segundo periodo.

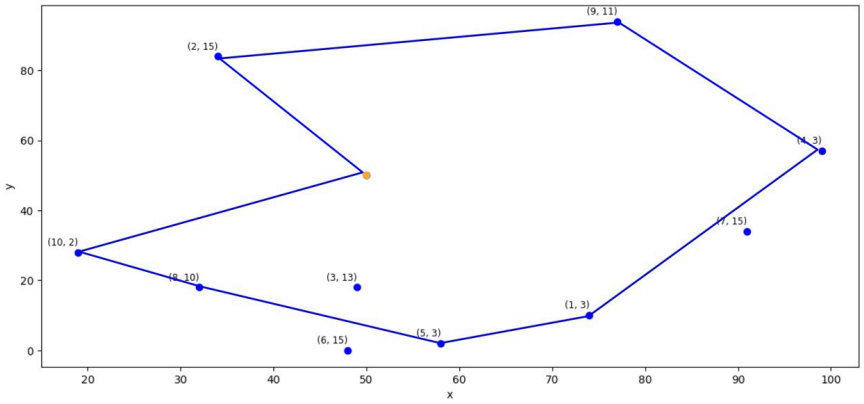

Figura 4.2. t=1

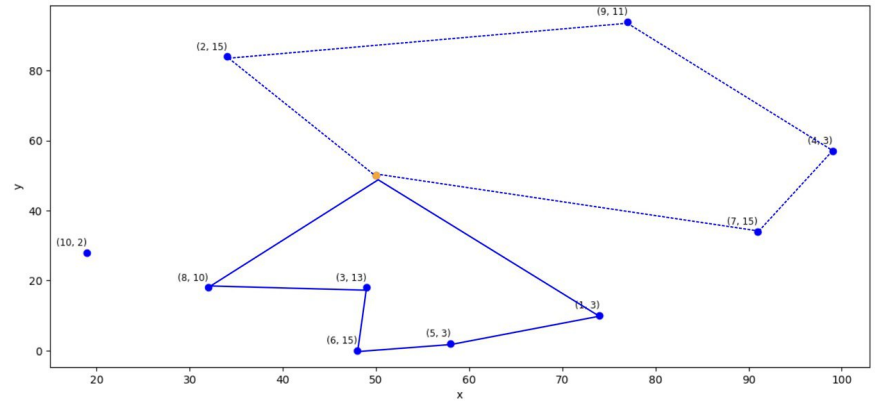

Figura 4.3. t=2

Conclusiones

En este trabajo hemos abordado el Problema Periódico de Rutas de Vehículos, una variante del clásico Problema de Rutas de Vehículos en el que se deben diseñar las rutas a seguir por una flota de vehículos, no para uno sino para varios periodos a lo largo de un horizonte temporal. Esto es así porque los clientes requieren varias visitas a lo largo del tiempo; además, estas visitas deben realizarse siguiendo uno de los posibles calendarios de visita que cada cliente admite.

Nuestro objetivo ha sido centrarnos en la modelización y posterior resolución exacta del problema usando un software comercial. Hemos encontrado en la literatura dos formulaciones MILP, denominadas formulación MTZ y formulación basada en cargas. En ambas formulaciones se utilizan variables continuas auxiliares para definir las restricciones que impiden la formación de subtours, y ambas son formulaciones compactas, es decir, no implican un número exponencial de restricciones. Dado que las variables continuas definidas en la formulación MTZ tienen una dimensión menos que las utilizadas en la formulación basada en cargas, la formulación MTZ tiene un menor número de variables en comparación con la de cargas. También, presentamos en el trabajo algunas desigualdades válidas y cortes de optimalidad para reforzar los modelos.

Para realizar los experimentos computacionales hemos optado por implementar los modelos usando Python y el software de optimización matemática Gurobi. Como batería de pruebas hemos tomado un conjunto de instancias procedentes de la literatura. El primer hecho que pudimos constatar es que el modelo basado en cargas mejora desde un punto computacional al modelo MTZ. Por otro lado, la introducción de las desigualdades válidas y cortes de optimalidad contribuyen a mejorar la eficiencia de ambos modelos. Por último, con el modelo de cargas reforzado con dos de esas familias de desigualdades conseguimos resolver problemas con hasta 41 nodos, 4 periodos y 4 vehículos (aunque no siempre de forma óptima debido al límite de tiempo que impusimos en nuestras pruebas).

A

Apéndice

A.1. Código Modelo MTZ

```
import os
import math
import random
import numpy as np
import time
import gurobipy as gp


## VARIABLES GLOBALES

EPS = 0.001

class Customer:
    def __init__(self, i, x, y, d, q, f, a, visit_combinations):
        self.i = i     #cliente
        self.x = x     #coordenada x
        self.y = y     #coordenada y
        self.d = d     #nada
        self.q = q     #demanda
        self.f = f     #frecuencia de visitas
        self.a = a     #nº esquemas de visitas posibles
        self.visit_combinations = visit_combinations    #matriz de
        esquemas de visitas


## FUNCIONES AUXILIARES
```

```
#Función lectura de fichero
def read_file(file_path):
    with open(file_path, 'r') as file:
        lines = file.readlines()

        # Analizar la primera línea
        line1 = lines[0].strip().split()
        file_type, m, n, t = map(int, line1)

        # Analizar la información de los días o periodos
        day_info = []
        for i in range(1, t+1):
            d, Q = map(int, lines[i].strip().split())
            day_info.append((d, Q))

        # Analizar la información de los clientes
        customers = []
        for i in range(t+1, len(lines)):
            line = lines[i].strip().split()
            i, x, y, d, q, f, a = map(int, line[:7])
            visit_combinations = list(map(int, line[7:]))

            # Pasar a cadenas binarias las combinaciones de visitas codificadas
            binary_combinations = [format(combination, f'0{t}b') for
            combination in visit_combinations]

            # Convertir las cadenas binarias en arrays de 0s y 1s
            binary_arrays = [list(map(int, binary)) for binary in
            binary_combinations]

            customer = Customer(i, x, y, d, q, f, a, binary_arrays)
            customers.append(customer)

    return file_type, m, n, t, day_info, customers



## PROGRAMA PRINCIPAL

# Lectura de datos de un fichero:
file_path = 'C:/Users/patri/Documents/ejemplos/test11-p3-m3-c.dat'
#Reemplazar por la ruta a tu fichero
```

```
file_type, nVeh, nC, nDays, day_info, customers = read_file(file_path)

Q = day_info[0][1]  # capacidad de los vehículos
nV = nC+1           # número de nodos, incluyendo al depot

print("File Type:", file_type)
print("Number of Vehicles (m):", nVeh)
print("Capaciy of the Vehicles (Q):", Q)
print("Number of Customers (n):", nC)
print("Number of Days (t):", nDays)
print("Day Information:", day_info)
print("Customers:")
for customer in customers:
    print(f"Customer {customer.i}:")
    print(f"  x: {customer.x}, y: {customer.y}")
    print(f"  Service Duration: {customer.d}, Demand: {customer.q}")
    print(f"  Frequency: {customer.f}, Number of
    Possible Visit Combinations: {customer.a}")
    print(f"  Visit Combinations (Binary Arrays):
    {customer.visit_combinations}")


V = range(0, nC+1) #va de 0 a n
N = range(1, nC+1)  # va de 1 a n
T = range(nDays)  # va de 0 a |T|-1


#Matriz de costos
c = np.zeros((nV, nV))

for i in V:
    for j in V:
        if j != i:
            dx = customers[i].x - customers[j].x
            dy = customers[i].y - customers[j].y
            c[i, j] = math.floor(math.sqrt(dx*dx + dy*dy))
            #Costo igual a distancia euclídea redondeada
```

```
# Modelo
model = gp.Model('PVRP')

# Variables
x = {(i, j, t): model.addVar(vtype=gp.GRB.BINARY, name='x[%i,%i,%i]'
% (i, j, t)) for i in V for j in V for t in T if i != j}
y = {(i, t): model.addVar(vtype=gp.GRB.BINARY, name='y[%i,%i]' % (i, t))
for i in N for t in T}
z = {(i, p): model.addVar(vtype=gp.GRB.BINARY, name='z[%i,%i]' % (i, p))
for i in N for p in range(customers[i].a)}
u = {(i, t): model.addVar(lb=0.0, ub=gp.GRB.INFINITY, name='u[%i,%i]' % (i, t))
for i in N for t in T} #variable continua

# Función objetivo
model.setObjective(gp.quicksum(c[i, j]*x[i, j, t] for i in V
for j in V for t in T if i != j), gp.GRB.MINIMIZE)

#Restricciones
[model.addConstr(gp.quicksum(z[i, p] for p in range(customers[i].a)) == 1)
for i in N]
[model.addConstr(gp.quicksum(customers[i].visit_combinations[p][t] * z[i, p]
for p in range(customers[i].a)) == y[i, t]) for i in N for t in T]
[model.addConstr(x[i, j, t] <= (y[i, t] + y[j,t])/2 ) for i in N for j in N
for t in T if i != j]
[model.addConstr(gp.quicksum(x[i, j, t] for j in V if i != j) <= nVeh)
for i in N for t in T]
[model.addConstr(gp.quicksum(x[i, j, t] for j in V if i != j) == y[i, t])
for i in N for t in T]
[model.addConstr(gp.quicksum(x[j, i, t] for j in V if i != j) == y[i, t])
for i in N for t in T]
[model.addConstr(u[j,t] >= u[i,t] + customers[j].q * y[j,t] - Q*(1-x[i,j,t]))
for i in N for j in N for t in T if i != j]
[model.addConstr(u[i,t] >= customers[i].q * y[i,t] ) for i in N for t in T]
[model.addConstr(u[i,t] <= Q* y[i,t] ) for i in N for t in T]

#Restricciones válidas
#I1
#[model.addConstr((u[j,t] - u[i,t] + Q*x[i,j,t] + (Q - customers[j].q * y[j,t]
- customers[i].q * y[i,t])*x[j,i,t]) <= (Q - customers[i].q* y[i,t]))
for i in N for j in N for t in T if i != j]
#I2: ruptura de simetría en las rutas
```

```
#[model.addConstr(x[i,0,t] <= gp.quicksum(x[0,r,t] for r in N if r <= i))
for i in N for t in T]
#I3
#[model.addConstr( (x[i, j, t] + x[j, i, t]) <= (( y[i, t] + y[j,t])/2) )
for i in N for j in N for t in T if i < j]


# Resolvemos

# Registro de tiempo de inicio
start_time = time.time()

# Establecer límite de tiempo en segundos
model.setParam('TimeLimit', 1800)

# Actualizar el modelo
model.update()

# Optimizar el modelo
model.optimize()

# Registro de tiempo de finalización
end_time = time.time()

if model.status == gp.GRB.OPTIMAL:
    print('Valor óptimo =', model.objVal)
    print("Tiempo:", end_time - start_time, "segundos")
    print( '\n')
    print("Solución óptima:")
    for t in T:
        for i in V:
            for j in V:
                if i != j and  x[i, j, t].x > EPS:
                    print(f"{x[i,j,t].VarName} = {x[i,j,t].x}")
elif model.status == gp.GRB.TIME_LIMIT:
    print('Se ha superado el límite de tiempo.')
else:
    print('El problema no tiene solución óptima.')


# Eliminar el modelo para liberar recursos
model.dispose()
```

A.2. Código Modelo de Cargas

```
import os
import math
import random
import numpy as np
import time
import gurobipy as gp


## VARIABLES GLOBALES

EPS = 0.001

class Customer:
    def __init__(self, i, x, y, d, q, f, a, visit_combinations):
        self.i = i  #cliente
        self.x = x  #coordenada x
        self.y = y  #coordenada y
        self.d = d  #nada
        self.q = q  #demanda
        self.f = f  #frecuencia de visitas
        self.a = a  #nº esquemas de visitas posibles
        self.visit_combinations = visit_combinations   #matriz de esquemas de
        visitas posibles a_pt


## FUNCIONES AUXILIARES

#función lectura de fichero
def read_file(file_path):
    with open(file_path, 'r') as file:
        lines = file.readlines()

        # Analizar la primera línea
        line1 = lines[0].strip().split()
        file_type, m, n, t = map(int, line1)

        # Analizar la información de los días o periodos
        day_info = []
```

```
        for i in range(1, t+1):
            d, Q = map(int, lines[i].strip().split())
            day_info.append((d, Q))

        # Analizar la información de los clientes
        customers = []
        for i in range(t+1, len(lines)):
            line = lines[i].strip().split()
            i, x, y, d, q, f, a = map(int, line[:7])
            visit_combinations = list(map(int, line[7:]))

            # Pasar a cadenas binarias las combinaciones de visitas codific
            binary_combinations = [format(combination, f'0{t}b') for
            combination in visit_combinations]

            # Convertir las cadenas binarias en arrays de 0s y 1s
            binary_arrays = [list(map(int, binary)) for binary in
            binary_combinations]

            customer = Customer(i, x, y, d, q, f, a, binary_arrays)
            customers.append(customer)

    return file_type, m, n, t, day_info, customers



## PROGRAMA PRINCIPAL

# Lectura de datos de un fichero:
file_path = 'C:/Users/patri/Documents/ejemplos/test31-p4-m4-b.dat'
#Reemplazar por la ruta a tu fichero

file_type, nVeh, nC, nDays, day_info, customers = read_file(file_path)

Q = day_info[0][1]  # capacidad de los vehículos
nV = nC+1           # numero de nodos, incluyendo al depot

print("File Type:", file_type)
print("Number of Vehicles (m):", nVeh)
print("Capaciy of the Vehicles (Q):", Q)
print("Number of Customers (n):", nC)
print("Number of Days (t):", nDays)
```

```
print("Day Information:", day_info)
print("Customers:")
for customer in customers:
    print(f"Customer {customer.i}:")
    print(f"  x: {customer.x}, y: {customer.y}")
    print(f"  Service Duration: {customer.d}, Demand: {customer.q}")
    print(f"  Frequency: {customer.f}, Number of Possible Visit Combinations:
    {customer.a}")
    print(f"  Visit Combinations (Binary Arrays):
    {customer.visit_combinations}")


V = range(0, nC+1) #va de 0 a n
N = range(1, nC+1)  # va de 1 a n
T = range(nDays)  # va de 0 a |T|-1


# Matriz de costos
c = np.zeros((nV, nV))
# Matriz de costos
for i in V:
    for j in V:
        if j != i:
            dx = customers[i].x - customers[j].x
            dy = customers[i].y - customers[j].y
            c[i, j] = math.floor(math.sqrt(dx*dx + dy*dy)) # Costo igual a dist



# Modelo
model = gp.Model('PVRP')

# Variables
x = {(i, j, t): model.addVar(vtype=gp.GRB.BINARY, name='x[%i,%i,%i]'
% (i, j, t)) for i in V for j in V for t in T if i != j}
y = {(i, t): model.addVar(vtype=gp.GRB.BINARY, name='y[%i,%i]' % (i, t))
for i in N for t in T}
z = {(i, p): model.addVar(vtype=gp.GRB.BINARY, name='z[%i,%i]' % (i, p))
for i in N for p in range(customers[i].a)}
l = {(i, j, t): model.addVar(lb=0.0, ub=gp.GRB.INFINITY, name='l[%i,%i,%i]'
% (i, j, t)) for i in V for j in V for t in T if i != j}
```

```
# Función objetivo
model.setObjective(gp.quicksum(c[i, j]*x[i, j, t] for i in V for j in V
for t in T if i != j), gp.GRB.MINIMIZE)

#Restricciones
[model.addConstr(gp.quicksum(z[i, p] for p in range(customers[i].a)) == 1)

[model.addConstr(gp.quicksum(customers[i].visit_combinations[p][t] *
z[i, p] for p in range(customers[i].a)) == y[i, t]) for i in N for t in T]

[model.addConstr(x[i, j, t] <= (y[i, t] + y[j,t])/2 ) for i in N
for j in N for t in T if i != j]

[model.addConstr(gp.quicksum(l[i, j, t] - l[j, i, t] for j in V if j != i)
== -customers[i].q * y[i, t]) for i in N for t in T]

[model.addConstr(gp.quicksum(l[0, j, t] - l[j, 0, t] for j in V if j != 0)
== gp.quicksum(customers[i].q * y[i, t] for i in N)) for t in T]

[model.addConstr(gp.quicksum(x[i, j, t] for j in V if i != j) == y[i, t])
for i in N for t in T]

[model.addConstr(gp.quicksum(x[i, j, t] for j in V if i != j) ==
gp.quicksum(x[j, i, t] for j in V if i != j)) for i in V for t in T]

[model.addConstr(l[i, j, t] <= Q*x[i, j, t]) for i in V for j in V
for t in T if i != j]

[model.addConstr(gp.quicksum(x[0, j, t] for j in N) <= nVeh) for t in T]

#restricciones válidas
# I2: ruptura de simetría en las rutas
[model.addConstr(x[i,0,t] <= gp.quicksum(x[0,r,t] for r in N if r <= i))
for i in N for t in T]
#I3
#[model.addConstr( (x[i, j, t] + x[j, i, t]) <= (( y[i, t] + y[j,t])/2) )
for i in N for j in N for t in T if i < j]
#I4: suma de las cargas finales
#[model.addConstr(gp.quicksum(gp.quicksum(l[j,0,t] for j in N) for t in T)
== 0)]
```

```
# Resolvemos

# Registro de tiempo de inicio
start_time = time.time()

# Establecer límite de tiempo en segundos
model.setParam('TimeLimit', 600)

# Actualizar el modelo
model.update()

# Optimizar el modelo
model.optimize()

# Registro de tiempo de finalización
end_time = time.time()


if model.status == gp.GRB.OPTIMAL:
    print('Valor óptimo =', model.objVal)
    print("Tiempo:", end_time - start_time, "segundos")
    print( '\n')
    print("Solución óptima:")
    for t in T:
        for i in V:
            for j in V:
                if i != j and  x[i, j, t].x > EPS:
                    print(f"{x[i,j,t].VarName} = {x[i,j,t].x}")
elif model.status == gp.GRB.TIME_LIMIT:
    print('Se ha superado el límite de tiempo.')
else:
    print('El problema no tiene solución óptima.')



# Eliminar el modelo para liberar recursos
model.dispose()
```

Bibliografía

[1] Archetti, C., Férnandez, E., Huerta-Muñoz, D.L., *The flexible periodic vehicle routing problem.* Computers and Operations Research, vol 85, 58-70, 2017.

[2] Beltrami, E.J., Bodin, L.D. Networks and vehicle routing for municipal waste collection. *Networks* vol. 4, 65-94, 1974.

[3] Bonyadi, M.R., Azghadi, M.R., and Hosseini, H.S. Population-Based Optimization Algorithms for Solving the Travelling Salesman Problem. En Federico Greco, *Traveling Salesman Problem* pp. 1–2. InTech, 2008.

[4] Basir, A.S., Sahin, G., Ozbaygin, G. *A Comparative Study of Alternative Formulations for the Periodic Vehicle Routing Problem*, Computers and Operations Research, vol 165, 2024.

[5] Dantzig, G., Fulkerson, R., Johnson, S. *Solution of a large-scale traveling salesman problem.* Journal of the Operations Research Society of America, vol. 2, 393–410, 1954.

[6] Dantzig, G., Ramser, J. H . *The truck dispatching problem.* Management Science, vol. 6, 80-91, 1959.

[7] Laporte, G. *What You Should Know about the Vehicle Routing Problem*, Wiley Periodicals, Inc. Naval Research Logistics, vol 54, 811-819, 2007.

[8] Rodríguez-Martín, I., Salazar-González, J.J., Yaman, H. *The periodic vehicle routing problem with driver consistency*, European Journal of Operation Research, vol 273, 575-584, 2019.

[9] Salazar Gónzalez, J.J. *Progamación matemática*. Madrid : Diaz de Santos, 2001.

The Periodic Vehicle Routing Problem

Patricia Sánchez Vega
Facultad de Ciencias • Sección de Matemáticas
Universidad de La Laguna
alu0101253015@ull.edu.es

Abstract

THE PERIODIC VEHICLE ROUTING PROBLEM (PVRP) *is a generalization of the classic VRP. There is a fleet of vehicles, located at a depot. Each customer has a demand and requires one or several visits over a time horizon of several periods. The dates of the visits are not set in advance and each customer has associated a list of allowable visit schedules, from which one must be chosen. The problem therefore consists of choosing a visit schedule for each client, and designing the vehicle routes for each period, so that the total cost of the routes is minimal. In this work we present two different mathematical models for the PVRP, along with some families of valid inequalities that serve to strengthen them. Furthermore, we code both models in Python and present computational results.*

1. Introduction

FIRST OF ALL, we will enumerate several key points that have been addressed in this work:

- Mathematical programming: Mathematical models used to solve problems such as decision problems.
- Combinatorial optimization: Finding the best solution within a finite but large set of options.
- Optimal solution: Feasible solution where the objective function reaches its maximum (or minimum) value.

2. Formulations

WE investigate two alternative formulations. First, a mixed integer programming formulation, called *MTZ-based formulation*, involving flow variables for the customers and a polynomial number of Miller-Tucker-Zemlin type SECs. Secondly, a commodity flow formulation referred to as *the load-based formulation*, involving commodity flow variables for the arcs, and a polynomial number of SECs. The objective is minimize the total cost, so the objective function is:

$$\min \sum_{t\in T}\sum_{a\in A} c_a x_a^t \tag{1}$$

These are the SECs inequalities that prevent subtours in the MTZ model:

$$u_j^t \le u_i^t - d_i y_i^t + Q(1 - x_{ij}^t) \qquad \forall i,j \in N, t \in T. \tag{2}$$

These are the constraints from the load-based model that guarantee the commodity flow conservation:

$$\sum_{a\in\delta^+(i)} l_a^t - \sum_{a\in\delta^-(i)} l_a^t = \begin{cases} -d_i y_i^t, & i \in N \\ \sum_{j\in N} d_j y_j^t, & i = 0 \end{cases} \qquad \forall t \in T. \tag{3}$$

We also present some valid inequalities adapted from the literature to strengthen the proposed formulations. These are the most efficient constraints for both models:

$$x_{i0}^t \le \sum_{r\le i} x_{0r}^t \qquad \forall i \in N, t \in T \tag{4}$$

$$x_{ij}^t + x_{ji}^t \le (y_i^t + y_j^t)/2 \qquad \forall i,j \in N, i < j, \forall t \in T \tag{5}$$

3. Computational results

TO CONCLUDE, we compared the computational results from both models, in presence and absence of the valid inequalities. We saw that the load-based model is better than the MTZ-model because it takes less computation time. Finally, we can observe the representation of an optimal PVRP solution for a particular instance with 10 customers, two vehicles and a planning horizon of two periods, so-called 11-2-2-a.

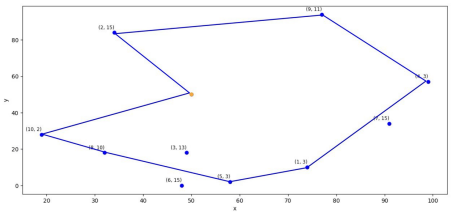

Figure 1: *First period (t=1)*

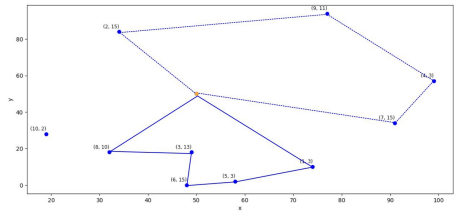

Figure 2: *Second period (t=2)*

References

[1] Archetti, C., Férnandez, E., Huerta-Muñoz, D.L., *The flexible periodic vehicle routing problem.* Computers and Operations Research, vol 85, 58-70, 2017.

[2] Basir, A.S., Sahin, G., Ozbaygin, G. *A Comparative Study of Alternative Formulations for the Periodic Vehicle Routing Problem*, Computers and Operations Research, vol 165, 2024.