Universidad
de La Laguna

Problema del viajante de comercio con periodicidad

Periodic Traveling Salesman Problem

María Arantxa Peña Rodríguez

Trabajo de Fin de Grado

Departamento de Matemáticas, Estadística e Investigación Operativa

Facultad de Ciencias, Sección Matemáticas

Universidad de La Laguna

La Laguna, 9 de marzo de 2016

Dr. D. **Hipólito Hernández Pérez**, con N.I.F. 45.452.715-T profesor adscrito al Departamento de Matemáticas, Estadística e Investigación Operativa de la Universidad de La Laguna

C E R T I F I C A

Que la presente memoria titulada:

"*Problema del viajante de comercio con periodicidad.*"

ha sido realizada bajo su dirección por Dña. **María Arantxa Peña Rodríguez**, con N.I.F. 42.412.692-W.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 9 de marzo de 2016

Agradecimientos

A mis padres y a mi abuelo.

A Ayoze, por apoyarme siempre.

A Hipólito Hernández, por su ayuda durante el desarrollo de este proyecto.

Resumen

El problema del viajante de comercio es uno de los problemas más importantes y más estudiados de la optimización combinatoria. En este se tiene que visitar un número de ciudades partiendo de un depósito y regresando a él al finalizar la ruta, de forma que se minimice el costo de dicho recorrido.

El problema del viajante de comercio con periodicidad, problema sobre el que se centra esta memoria, es una variante del problema del viajante de comercio. Su objetivo sigue siendo minimizar el costo de la ruta, pero en este caso las ciudades deben ser visitadas un número prefijado de veces, pudiendo haber condiciones sobre los periodos que deben ser visitadas.

Esta variante del problema del viajante de comercio no ha sido tan estudiada en la literatura como otras variantes, sin embargo, existen muchos casos con aplicaciones reales. El objetivo de este trabajo ha sido estudiar la bibliografía, presentar modelos matemáticos y resolver algunos ejemplos de este problema. Para resolver los modelos matemáticos se ha optado por programarlo con dos programas distintos, uno libre y otro comercial.

Palabras clave: Viajante de comercio, periodicidad, rutas de vehículos, optimización combinatoria.

Abstract

The traveling salesman problem is one of the most important and most studied combinatorial optimization problems. A salesman must visit a number of cities starting from a depot and returning to it at the end of the route. The aim is minimize the cost of that journey.

The periodic traveling salesman problem, model on which this report will focus, is a variant of the traveling salesman problem. Its objective remains to minimize the cost of the route, but in this case cities must be visited a predetermined number of times and may be conditions on the periods to be visited.

This variant of the traveling salesman problem has not been studied in the literature as other variants, however, there are many cases with real applications. The aim of this work was to study the literature, present mathematical models and solve some examples of this problem. To solve mathematical models, we have chosen to code it with two different softwares, one free and another commercial.

Keywords: *Traveling salesman, periodic, vehicle routing, combinatorial optimization.*

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

El objetivo de este proyecto es realizar un estudio del problema del viajante de comercio con periodicidad. Para ello analizaremos problemas similares, daremos modelos matemáticos para este problema, resumiremos los trabajos previos y mostraremos algunos resultados computacionales del problema. Estos resultados computacionales los hemos realizado con dos programas diferentes, uno libre y otro comercial.

Este trabajo consta de 5 capítulos. En este primer capítulo realizamos una pequeña introducción hablando de la optimización combinatoria, los problemas de rutas y los problemas con periodicidad. En el segundo capítulo nos centramos en los fundamentos teóricos, explicando el problema del viajante de comercio, conocido en inglés como *Traveling Salesman Problem*(TSP), el cual es uno de los problemas más importantes y más utilizados de la optimización, así como sus variantes y otros problemas de rutas de vehículos para centrarnos finalmente en el problema del viajante de comercio con periodicidad, en inglés *Periodic Traveling Salesman Problem* (PTSP), que como se dijo antes, dicho problema es sobre el que se centra este trabajo. En el tercer capítulo, realizamos una lista resumen de varios trabajos y artículos que están relacionados con el PTSP. Luego, en el cuarto capítulo, hacemos un estudio con resultados computacionales del problema. Y finalmente, mostramos las conclusiones de esta memoria.

1.1. Optimización Combinatoria

La optimización o programación matemática es un proceso de selección del mejor elemento de un conjunto, respecto a un criterio. En general, esta teoría y sus técnicas de formulación abarcan una gran parte de las matemáticas aplicadas, donde el caso más simple consiste en maximizar o minimizar una función real, eligiendo los valores de entrada de manera sistemática y calculando el valor de la función.

La optimización combinatoria es una rama de la programación matemática, relacionada con la teoría de algoritmos y la teoría de la complejidad computacional, así como con

la investigación de operaciones. Estudia problemas en los cuales se busca maximizar o minimizar una función de varias variables que estará definida en un conjunto discreto:

$$\text{mín}\{f(x) : x \in S\}, \text{ siendo } |S| < \infty$$

Los algoritmos que desarrolla la optimización combinatoria están caracterizados por tener un número finito de soluciones factibles.

Si el problema a resolver no es complejo existen algunas técnicas para obtener la solución óptima, algunas son: *Branch and Bound, Branch and Cut*, etc. Pero a medida que aumenta la complejidad, se vuelve más costosa su ejecución, y puede llegar a resultar inviable encontrar una solución óptima.

En este caso existe la posibilidad de encontrar una solución subóptima, en un tiempo razonable, incluso podemos llegar a obtener la solución óptima.

Estas técnicas aproximadas se dividen en dos grandes grupos: Heurísticas y Metaheurísticas.

Heurística

Estas técnicas son algoritmos que encuentran buenas soluciones para problemas combinatorios complejos. Estos son fáciles de implementar encontrando buenas soluciones con esfuerzos computaciones pequeños, pero renuncian a encontrar la solución óptima global del problema. Estos tipos de algoritmos son muy dependientes del problema.

Metaheurística

Estos algoritmos son de propósito general y no depende del problema. Encuentran soluciones muy cercanas a la óptima e incluso la óptima, pero no pueden garantizar que la solución obtenida es la óptima. Se utilizan cuando no existe un algoritmo o heurística específica para resolverlo o bien, son difíciles de implementar.

1.2. Problemas de rutas

En la optimización combinatoria existe un área de investigación que se conoce por *Routing Problems* o Problemas de Rutas.

Estos problemas son unos de los más importantes en el ámbito de la logística del transporte. El objetivo es optimizar un conjunto de rutas de transporte que se realizan por una flota de vehículos que, normalmente, se encuentran en un depósito.

De aquí podemos distinguir dos tipos de problemas, problemas de vértices y problemas de arcos, teniendo en cuenta donde se produce la demanda.

Problemas por vértices

En estos problemas los vehículos deben visitar los clientes, que estarán situados en los vértices de un grafo. En este tipo tendríamos el Problema del viajante de comercio (TSP),

que consiste en pasar por todos los clientes minimizando el recorrido o distancia total, todo esto realizado por un único vehículo.

También tendríamos la generalización del TSP, el problema de Rutas de Vehículos, el cual se conoce en inglés como *Vehicle Routing Problem*(VRP), en este caso para realizar el recorrido necesitaríamos más de un vehículo.

Algunas variantes de este tipo de problemas son:

- Problema con Ventanas de Tiempo, donde los clientes tienen un horario en el que pueden ser visitados. En esta categoría, el problema básico es el VRPTW (*Vehicle Routing Problem with Time Windows*).
- El problema con restricciones de precedencia. En este caso el orden de reparto está limitado por el peso de los paquetes o bultos que tienen que repartir, los más ligeros se reparten antes que los más pesados.
- Límite en el tiempo de transporte de cada mercancía. En este caso tendremos el DARP (*Dial-A-Ride Problem*).
- Recogida y entrega de mercancía. En ocasiones, el mismo vehículo de reparto que debe entregar la mercancía, debe recoger también mercancía de dicho cliente. Estos problemas se denominan *Pickup and Delivery Problems*.

Problemas por arcos

En este caso los vehículos deben recorrer todos o parte de los caminos (arcos o aristas) de un grafo. Aquí, el problema que debe recorrer todos los caminos de un grafo con un único vehículo será el CPP (Problema del Cartero Chino), en este caso deberá recorrer todas las calles de la ciudad minimizando las distancia total.

Este problema generalizado a varios vehículos sería CARP (Capacitated Arc Routing Problem), que al igual que el VRP es difícil de resolver óptimamente y ambos pertenecen a la clase de problemas $\mathcal{NP}$-difíciles.

Existen cuatro tipos de problemas en función de los tipos de caminos, si son arcos, tienen un único sentido y si son aristas, tienen doble sentido.

- Enlaces formados por arcos, que serán enlaces dirigidos que pertenecen al grupo de los problemas dirigidos y cada uno tendrá un coste asociado. Aquí tendríamos una versión del Problema del Cartero Chino (CPP) sobre un grafo dirigido que se llama *Directed Chinese Postman Problem* (DCPP).
- Enlaces formados por aristas con mismo costo en ambas direcciones, serán los enlaces no dirigidos que formarán parte del grupo de los problemas no dirigidos. A este grupo pertenece el Problema del Cartero Chino (CPP).
- Enlaces formados por arcos y aristas, donde las aristas tienen el mismo costo en ambas direcciones, estos reciben el nombre de enlaces mixtos.

- Enlaces formados por arcos y aristas, pero en este caso las aristas pueden tener un costo diferente para cada sentido o dirección. Este tipo de problemas de llaman Enlaces de Viento o *Windy* y son poco estudiados pero tienen alguna aplicación.

1.3. Problemas con periodicidad

Los problemas con periodicidad se caracterizan porque cada localización hay que visitarla varias veces. Muchas veces en estos problemas hay restricciones sobre cuándo se puede volver a visitar una misma localización. Normalmente hay un periodo mínimo entre una visita u otra. El objetivo suele ser minimizar el costo de las rutas pero puede haber otros factores que influyan en la función objetivo del problema (tiempo empleado en realizar las rutas, calidad del servicio realizado, etc.).

También como sucede con los problemas de rutas pueden incorporar otras características como capacidades, que haya un sólo vehículo o haya varios, que haya ventanas de tiempo, vehículos con una capacidad máxima, etc. En algunas variantes existe un depósito de donde el vehículo o vehículos salen y al que deben regresar al final del día o reparto, aunque también puede existir más de un depósito, siempre que haya más de un vehículo, y en tal caso los vehículos pueden partir de uno y al finalizar el reparto ir a otro.

Paletta [13] resuelve el problema del viajante de comercio con periodicidad (PTSP), en el que tenemos una cantidad de días prefijados para visitar las diferentes ciudades en un periodo de tiempo fijado, cuyo objetivo es minimizar la distancia total recorrida sin sobrepasar el periodo establecido. Se utiliza un algoritmo heurístico simple y eficaz para resolverlo.

Garrido y Onaindía [8] resuelven un problema de una agencia de transporte donde se trata de transportar los pedidos con varios vehículos y donde existen varios centros de intermediación, origen y destino. El objetivo es llevar todos los pedidos a sus respectivos destinos con el menor costo. Los vehículos son distintos entre sí (distinto peso y capacidades máximas), con lo cual también se elige el tipo adecuado para cada reparto en función de sus características.

Rodríguez-García [15] y Ruigómez-González [16] resuelven el problema de una empresa canaria que se dedica a tomar muestras de alimentos en los establecimientos de los clientes. Estos clientes tienen unas ventanas de tiempo y unos días de la semana en los que tienen que ser visitados. Cada cliente debe visitarse con distintas periodicidades (desde semanal hasta anual). El objetivo es minimizar el tiempo que se tarda en recoger las muestras.

Francis y Smilowitz [7] resuelve una variante del problema de ruta de vehículos con periodicidad, el Problema de rutas de vehículos con periodicidad con servicio de elección (*Period Vehicle Routing Problem with Service Choice*, PVRP-SC).

Capítulo 2

Fundamentos teóricos

En este capítulo se definen y modelan problemas básicos de rutas de vehículos como el TSP. Estos modelos serán generalizados para presentar una formulación matemática para el problema del viajante de comercio con periodicidad. Además también veremos algunas variantes así como algunos métodos para su resolución, tanto para el TSP como para el VRP.

2.1. El Problema del Viajante de Comercio (TSP)

Este problema conocido como problema del viajante de comercio o *Traveling Salesman Problem* es el problema clásico y uno de los más importantes en el campo de la Optimización Combinatoria. Por su sencilla definición y su dificultad a la hora de resolverlo es uno de los más estudiados. Es la base de la mayor parte de los problemas de rutas y equivale a encontrar un ciclo que visite todas las localizaciones de costo mínimo. Es un problema $\mathcal{NP}$-duro, es decir, no existe un algoritmo polinomial en el tamaño del problema (número de ciudades) capaz de resolverlo hasta la optimalidad (salvo que $\mathcal{P} = \mathcal{NP}$).

Este problema trata de:

"*Un viajante quiere visitar n ciudades una y sólo una vez cada una, empezando por una cualquiera de ellas y regresando al mismo lugar del que partió. Supongamos que conoce la distancia entre cualquier par de ciudades. ¿De qué forma debe hacer el recorrido si pretende minimizar la distancia total?.* "(Stockdale [17])

Para este problema existen dos tipos, el TSP simétrico y el TSP asimétrico. En el TSP simétrico, existen caminos en ambas direcciones y la distancia será la misma entre un par de ciudades, por lo que se forma un grafo no dirigido. Este reduce a la mitad el número de soluciones posibles, es decir $n(n-1)/2$. Luego en el TSP asimétrico, que será el caso más general, puede que no existan caminos en ambas direcciones o las distancias sean diferentes, lo que implica que el coste de ir de una localización i a una j no tiene porque ser igual al de ir de j a i, lo que forma un grafo dirigido.

Algunas de las aplicaciones más importantes del TSP son: la fabricación de circuitos integrados, las rutas de vehículos, la recogida (robotizada) de material en almacenes, la instalación de componentes en ordenadores y aparecer como subproblema en otras aplicaciones.

Veamos dos modelos de este problema del tipo asimétrico, uno con restricción de subciclos y otro con variables flujo. Para ambos modelos introducimos la siguiente notación:

Tenemos un grafo dirigido y completo $G = (V, A)$, formado por un conjunto de vértices $V = \{1, ..., n\}$, que serán las ciudades y un conjunto de arcos A, en este caso considerados carreteras, donde cada arco tiene un costo asociado $c_a, a \in A$. A cada arco $a \in A$ también se le denota como el par (i, j) donde i es el origen o punto de partida y j es el destino o punto de llegada.

Modelo matemático con restricción de subciclos

Este modelo es la fórmulación más popular del TSP y fue dada por Dantzig, Fulkerson y Johnson [6], con la cual se consiguió en 1954 la primera de las mayores instancias conocidas para este problema con 42 nodos. Para definir este modelo introducimos la siguiente variable para cada arco (i, j) de A.

$$x_{ij} = \begin{cases} 1, & \text{si se va de } i \text{ a } j \\ 0, & \text{en otro caso} \end{cases}$$

Entonces el modelo sería el siguiente:

$$\text{mín} \sum_{i,j \in V} c_{ij} x_{ij} \tag{2.1}$$

sujeto a:

$$\sum_{i,j \in, j \neq i} x_{ij} = 1 \qquad \forall i \in V \tag{2.2}$$

$$\sum_{i,j \in, j \neq i} x_{ji} = 1, \qquad \forall i \in V \tag{2.3}$$

$$\sum_{i,j \in S, j \neq i} x_{ij} \leq |S| - 1 \qquad \forall S \subset V \tag{2.4}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in V \tag{2.5}$$

La ecuación (2.1) representa la función objetivo, que expresa que el costo total es la suma de los costos de los arcos que se utilizan. Las restricciones (2.2) y (2.3) implican que se debe entrar y salir de cada ciudad exactamente una vez. La restricción (2.4) es la que evita que se formen subciclos, es decir, evitan que la solución final este formada por un grupo de ciclos disjuntos. Finalmente, la restricción (2.5) indica que la variable es binaria.

Modelo matemático con variable flujo

En este modelo añadiremos una variable flujo f_{ij}, la que va restando una unidad cada vez que se visita una ciudad. Esta formulación fue dada por Gavish y Graves [9] en 1978 y la mostramos a continuación. Al igual que antes introducimos una variable de decisión para cada arco, pero además introducimos una variable continua para cada arco. Podemos asumir que esta variable flujo es como la carga del vehículo.

$$x_{ij} = \begin{cases} 1, & \text{si se va de } i \text{ a } j \\ 0, & \text{en otro caso} \end{cases}$$

f_{ij}= Carga que hay en el vehículo cuando va de i a j.

El modelo sería el siguiente:

$$\text{mín} \sum_{i,j \in V} c_{ij} x_{ij} \tag{2.6}$$

sujeto a:

$$\sum_{j \in V, j \neq i} x_{ij} = 1 \qquad \forall i \in V \tag{2.7}$$

$$\sum_{j \in V, j \neq i} x_{ji} = 1 \qquad \forall i \in V \tag{2.8}$$

$$\sum_{j \in V, j \neq i} f_{ji} - \sum_{j \in V, j \neq i} f_{ij} = 1 \qquad \forall i \in V \setminus \{1\} \tag{2.9}$$

$$f_{1i} = (n-1) x_{1i} \qquad \forall i \in V \setminus \{1\} \tag{2.10}$$

$$f_{i1} = 0 \qquad \forall i \in V \setminus \{1\} \tag{2.11}$$

$$f_{ij} \leq (n-1) x_{ij} \qquad \forall i, j \in V \setminus \{1\} \tag{2.12}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in V \tag{2.13}$$

Este modelo con variable flujo tiene la misma función objetivo (2.6) que el modelo con restricción de subciclos. Las dos primeras restricciones, (2.7) y (2.8), indican los mismo que las restricciones (2.2) y (2.3). Las restricciones (2.9), (2.10), (2.11) y (2.12) lo que indican es que cada vez que se pasa por un nodo (ciudad) se descargue una unidad de la variable flujo, lo que obliga a que se debe salir del depósito o nodo de partida con un flujo igual al número de ciudades menos 1, solo habrá flujo entre dos ciudades si hay conexión entre ellas y no se regresará al depósito con flujo, de esta manera se evita que se formen subciclos. Y la última restricción (2.13) implica que la variable es binaria.

El modelo de subciclos tiene un número de restricciones exponencial. Incluso para n relativamente pequeños se hace inviable introducir todas las restricciones (2.4), sin embargo estas restricciones se pueden introducir de forma dinámica sólo cuando son necesarias y por ello tiene mejores resultados aunque su implementación es mucho más compleja.

2.1.1. Métodos de resolución

No existen algoritmos polinomiales para resolver el TSP de forma exacta, pero existen métodos rápidos que permiten resolverlo de forma aproximada. Estos métodos son los heurísticos y metaheurísticos.

Los métodos heurísticos se utilizan para obtener cotas superiores y las relajaciones obtienen cotas inferiores. Ahora detallamos algunos ejemplos.

Métodos heurísticos

"*Un método heurístico es un procedimiento para resolver un problema de optimización bien definido mediante una aproximación intuitiva, en la que la estructura del problema se utiliza de forma inteligente para obtener una buena solución.*"

Estas técnicas heurísticas son algoritmos que encuentran soluciones buenas para problemas complejos, es decir, problemas tipo $\mathcal{NP}$. Estos algoritmos no tienen tanta dificultad a la hora de implementarlos y encontrar una buena solución con esfuerzos computacionales pequeños, pero renuncian a encontrar la solución óptima global, desde el punto de vista teórico. Cuando tenemos problemas de gran tamaño, es muy raro que este tipo de algoritmos encuentren la solución óptima.

- **Heurísticos constructivos**. Son procedimientos iterativos que, en cada iteración, añaden un elemento hasta completar una solución. Normalmente son métodos deterministas y se basan en seleccionar, en cada paso, el mejor elemento, además estos métodos dependen del problema a resolver. Algunos de los métodos más conocidos son: heurísticos del vecino más próximo, que es uno de los más fáciles y trata de construir un ciclo hamiltoniano con mínimo coste que se basa en el vértice más cercano a uno dado, también tenemos los heurísticos de inserción, los heurísticos basados en árboles generadores y heurísticos basados en ahorros.

- **Heurísticos de mejora**. Estos procedimientos, también llamados procedimientos de búsqueda local, se centran en explorar el entorno de una solución. Estos algoritmos dependen del problema que resuelven. Usan una operación básica llamada movimiento que, al aplicarla sobre los diferentes elementos de una solución, facilitan las soluciones de su entorno. Este algoritmo se determina cuando se especifica un entorno y un criterio de selección de una solución dentro del entorno.

- **Heurísticos voraces** (Método *Greedy*).

- **Heurísticos de primales** o heurísticos basados en la resolución de un problema de programación lineal.

- **Heurísticos multiarranque** ("*multistart*").

Métodos metaheurísticos

Los métodos metaheurísticos son un tipo de métodos aproximados, que se crean para resolver problemas con gran dificultad, para los que los métodos heurísticos no han conseguido devolver una solución eficiente. Uno de los métodos de este tipo que se usa para resolver el TSP es la búsqueda tabú, la cual guía un algoritmo heurístico de búsqueda local para explorar el conjunto de soluciones de manera que se eviten los óptimos locales y, de forma iterativa, se desplaza de una solución a otra hasta que encuentra una que cumpla algún criterio de terminación.

Métodos exactos

Este tipo de problemas nos proporcionan una solución óptima del problema. Algunos de estos son los Hiperplanos de corte, Ramificación y acotación, Ramificación y corte, Generación de columnas y Programación dinámica.

Relajaciones

Con este tipo de método se pretende conseguir rápidamente cotas inferiores del valor óptimo y aceleran algoritmos que buscan la solución exacta.

Algunas posibles relajaciones son: relajación por eliminación, subrogada, lineal continua, lagrangiana y por descomposición.

2.2. Variantes del TSP

El problemas del viajante de comercio con ventanas de tiempo

Consiste en encontrar el camino de coste mínimo para un vehículo que ha de partir del depósito y regresar siempre al mismo, visitando cada ciudad dentro de un periodo de tiempo fijado previamente.

El objetivo de este problema es encontrar una ruta de mínimo coste de forma que se respeten las ventanas de tiempo.

Prize-Collecting TSP

El objetivo de este problema es encontrar un tour que visite un subconjunto de ciudades donde el costo sea mínimo pero se debe recaudar al menos una cantidad mínima de premio al visitar las diferentes ciudades, puesto que, en este caso, a parte de los costos asociados a cada carretera (arco), cada ciudad tiene un premio.

Orienteering TSP

El Orienteering TSP trata de diseñar una ruta que sale del depósito donde el coste del circuito no puede superar una cantidad máxima. El objetivo consiste en maximizar los premios obtenidos respetando las restricciones.

2.3. Problema de rutas de vehículos

El problema de rutas de vehículos o Vehicle Routing Problem (VRP) es una generalización del problema del viajante de comercio, el cual consiste en determinar un conjunto de rutas de costo mínimo que comiencen y terminen en los depósitos, de forma que los vehículos visiten a los clientes una sola vez donde las demandas son conocidas. Es un problema de optimización combinatoria que se encuentra en la categoría de $\mathcal{NP}$-duro. El interes por resolverlo se encuentra en que no es posible encontrar un algoritmo polinomial para resolverlo

2.3.1. Variantes del VRP

Problemas de rutas de vehículos con capacidad limitada

Este problema es el más general y su objetivo es el mismo que el de rutas de vehículos pero en este tenemos vehículos con capacidad limitada y la suma de las demandas de los clientes que se visitan en cada ruta no puede exceder dicha capacidad. Las características de dicho problema son:

- Existe una flota de m vehículos, de capacidad máxima Q
- Hay un depósito, que denotamos 0.
- Hay un conjunto de clientes $\{1, ..., n\}$ de los que se recoge (entrega) una cantidad de mercancía q_i, $i \in \{1, ..., n\}$.
- Hay un costo c_{ij} de ir de la localización i a la j, con $i, j \in \{0, 1, ..., n\}$.
- Cada vehículo tiene que salir y llegar al depósito.

Problema de rutas de vehículos con ventanas de tiempo

Este problema consiste en encontrar una colección de circuitos de coste mínimo tal que:

- Cada circuito tiene que visitar el depósito.
- Cada cliente debe ser visitado por un único circuito.

- La suma de las demandas de los nodos visitados por cada circuito no supere la capacidad del vehículo.
- Para cada cliente i, el servicio comience dentro de la ventana de tiempo $[a_i, b_i]$.
- El vehículo permanezca en la localización del cliente i tantas unidades de tiempo como tiempo de proceso P_i precise dicho cliente para atender su demanda.

Problema de rutas de vehículos con periodicidad (PVRP)

Este problema, al igual que el VRP, trata de minimizar el costo de la ruta pero en este caso tenemos un tiempo de operación fijado, es decir, un periodo en el que el vehículo debe visitar a todos los clientes una vez.

Problema de rutas de vehículos con multiples depósitos ó *Multi-depot Vehicle Routing Problem* (MDVRP)

Es el problema de rutas de vehículos donde hay varios depósitos y en cada uno de ellos tenemos una cantidad de vehículos independiente.

También existe el problema de rutas de vehículos con multiples depósitos y donde los vehículos tienen una capacidad máxima.

VRP con entrega dividida ó *Split Delivery Vehicle Routing Problem* (SDVRP)

Este tipo de problema decreta que, siempre que el costo total se reduzca, los vehículos pueden atender más de una vez a un cliente, esto suele ocurrir cuando las ciudades demandan mayor cantidad que la capacidad del vehículo.

VRP estocástico ó *Stochastic Vehicle Routing Problem*(SVRP)

En este problema tenemos la opción de que uno o varios componentes tengan carácter aleatorio.

VRP con recogidas y entregas ó *Pick-up and Delivery Vehicle Routing Problem* (VRPPD)

Aquí los clientes tienen la posibilidad de devolver material, es decir, el vehículo puede tener que recoger artículos de los diferentes clientes, lo que hay que tener en cuenta para que no se sobrepase la capacidad del vehículo.

El Problema con Flota Heterogénea ó *Fleet Size and Mix Vehicle Routing Problem*(FSMVRP)

Será el problema de rutas vehículos donde no todos los vehículos son iguales (hay varios tipos de vehículos). En estos problemas los costos y las capacidades de los vehículos varían y existe un conjunto formado por los diferentes tipos de vehículos.

2.3.2. Métodos de resolución

Los métodos más usados para hallar las soluciones del VRP y sus variantes son heurísticos y metaheurísticos pues los métodos exactos no garantizan que se encuentre la solución óptima en un tiempo razonable de computación cuando el número de clientes es grande (más de 50).

Sin embargo, cuando tenemos una cantidad de clientes razonable, algunas formas de resolverlos usando métodos exactos son: se puede resolver una relajación del problema utilizando ramificación y acotamiento con el método de *Branch and Bound*, *Branch and Cut* o *Branch and Price*. También hay algoritmos que se basan en la programación dinámica que aceleran los cálculos.

Métodos heurísticos

Como ya hemos mencionado anteriormente, estas técnicas son procedimeintos simples que dan buenas soluciones en tiempos de cálculo aceptables. Estas soluciones se pueden mejorar utilizando métodos más sofisticados, pero con elevados tiempos de ejecución.

Este tipo de heurísticas también se utilizan para calcular las soluciones de las diferentes variantes del VRP. Algunos métodos heurísticos clásicos para resolver estos problemas son:

El **algoritmo de ahorros** (*saving*), en el que partimos de una solución inicial y realizamos las uniones que nos den mayor ahorro siempre que no se salten las restricciones del problema. Es uno de los algoritmos más nombrados para este tipo de problemas, dentro del cual tenemos el algoritmo de ahorros basado en emparejamiento, en el que decimos la unión que se realiza teniendo en cuenta como afectará a las posibles uniones en las iteraciones siguientes.

Las **heurísticas de inserción** son métodos constructivos en los que, mediante sucesivas inserciones de clientes en las rutas, creamos una solución. En cada una de estas iteraciones obtenemos una solución parcial, donde las rutas visitan un subconjunto de los clientes. Luego seleccionamos un cliente que no haya sido visitado y lo añadimos a la solución.

Métodos metaheurísticos

Los métodos metaheurísticos son procedimientos genéricos que exploran el espacio de soluciones para problemas de optimización y búsqueda. Estos tipos de algoritmos, generalmente, dan mejores soluciones que los algoritmos heurísticos, pero utilizando mayor tiempo para hallar los resultados, aunque siempre será menor que el de los métodos exactos.

Los **algoritmos de hormigas** son procedimientos que usan métodos constructivos aleatorizados y que comparten información entre sí. Los algoritmos de **búsqueda tabú** son métodos de búsqueda local aceptando que, para evitar los óptimos locales, aceptan empeorar las soluciones. Y por último tenemos los **algoritmos genéticos** que, al tratar de cubrir de la mayor parte del espacio de soluciones, intentan mantener un conjunto de soluciones diversas.

2.4. El problema del viajante de comercio con periodicidad (PTSP)

En este problema, como en el TSP, hay n localizaciones ($V = \{1, ..., n\}$) y tenemos un depósito que denotamos por 1. El vehículo, si sale del depósito a realizar visitas, debe volver, además una misma ciudad no se puede visitar más de una vez en el mismo periodo (día), a excepción de la de partida/llegada. Denotamos por p el número de días en el que se puede visitar a los clientes ($D = \{1, ..., p\}$) y por m_i el número de veces que hay que visitar al cliente $i = 2, ..., n$. Existe un costo de ir de i a j, c_{ij}.

El objetivo recae en minimizar el costo cumpliendo las restricciones anteriores.

2.4.1. Modelo matemático con restricción de subciclos

Para la formulación del modelo definimos las siguientes variables:

$$x_{ij}^k = \begin{cases} 1 & \text{si se va de } i \text{ a } j \text{ el dia } k \\ 0 & \text{en otro caso} \end{cases}$$

$$y_i^k = \begin{cases} 1, & \text{si la localizacion } i \text{ se visita el dia } k \\ 0, & \text{en otro caso} \end{cases}$$

Entonces el modelo quedaría:

$$\text{mín} \sum_{k=1}^{p} \sum_{i,j \in V, i \neq j} c_{ij} x_{ij}^k \tag{2.14}$$

sujeto a:

$$\sum_{j \in V, j \neq i} x_{ij}^k = y_i^k \qquad \forall i \in V, k \in D \tag{2.15}$$

$$\sum_{j \in, j \neq i} x_{ji}^k = y_i^k \qquad \forall i \in V, k \in D \tag{2.16}$$

$$\sum_{i,j \in S, j \neq i} x_{ij}^k \leq |S| - 1 \qquad \forall i \in V \setminus \{1\}, k \in D \tag{2.17}$$

$$\sum_{k=1}^{p} y_i^k = m_i \qquad \forall i \in V \tag{2.18}$$

$$y_1^k = 1 \qquad \forall k \in D \tag{2.19}$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j \in V, k \in D \tag{2.20}$$

$$y_i^k \in \{0,1\} \qquad \forall i,j \in V, k \in D \tag{2.21}$$

La ecuación (2.14) representa la función objetivo, que expresa que el costo total es la suma de los costos de los arcos que se utilizan. Las restricciones (2.15) y (2.16) implican que los arcos que entran y salen de la localización en el día k deben ser igual a si la localización se visita o no ese día. La restricción (2.17) es la que evita que se formen subciclos. La restricción (2.18) implica que el cliente i se tiene que visitar exactamente m_i veces. La restricción (2.19) dice que todos los días hay que visitar a algún cliente, es decir, todos los días se sale del depósito. Finalmente, la restricción (2.20) y (2.21) indican que las variables son binarias.

2.4.2. Modelo matemático con flujo

Para este modelo definimos las siguientes variables:

$$x_{ij}^k = \begin{cases} 1, & \text{si se va de } i \text{ a } j \text{ el dia } k \\ 0, & \text{en otro caso} \end{cases}$$

$$y_i^k = \begin{cases} 1, & \text{si la localizacion } i \text{ se visita el dia } k \\ 0, & \text{en otro caso} \end{cases}$$

f_{ij}^k= Carga que hay en el vehículo cuando va de i a j el día k

El modelo matemático será:

$$min \sum_{k=1}^{p} \sum_{i,j \in V, i \neq j} c_{ij} x_{ij}^k \tag{2.22}$$

sujeto a:

$$\sum_{j \in V, j \neq i} x_{ij}^k = y_i^k \qquad \forall i \in V, \forall k \in D \tag{2.23}$$

$$\sum_{j \in, j \neq i} x_{ji}^k = y_i^k \qquad \forall i \in V, \forall k \in D \tag{2.24}$$

$$\sum_{k=1}^{p} y_i^k = m_i \qquad \forall i \in V \tag{2.25}$$

$$y_1^k = 1 \qquad \forall k \in D \tag{2.26}$$

$$\sum_{j \in V, j \neq i} f_{ji}^k - \sum_{j \in V, j \neq i} f_{ij}^k = -y_i^k \qquad \forall i \in V \setminus \{1\}, \forall k \in D \tag{2.27}$$

$$\sum_{i \in V} f_{1i}^k = \sum_{i \in V} y_i^k \qquad \forall k \in D \tag{2.28}$$

$$f_{i1}^k = 0 \qquad \forall i \in V \setminus \{1\}, \forall k \in D \tag{2.29}$$

$$f_{ij}^k \leq (n-1) x_{ij}^k \qquad \forall i, j \in V \setminus \{1\}, \forall k \in D \tag{2.30}$$

$$x_{ij}^k \in \{0, 1\} \qquad \forall i, j \in V, \forall k \in D \tag{2.31}$$

$$y_i^k \in \{0, 1\} \qquad \forall i, j \in V, \forall k \in D \tag{2.32}$$

La función objetivo (2.22) de este modelo es igual a la del modelo de subciclos. Las primeras restricciones (2.23), (2.24), (2.25) y (2.26) tienen el mismo significado que la restricciones (2.15), (2.16), (2.18) y (2.19). Las restricciones (2.27) y (2.30) indican que cada vez que se pasa por una ciudad se descarga una unidad de la variable flujo siempre sea visitada ese día y solo habrá flujo entre dos ciudades si hay conexión entre ellas. La (2.28) significa que la cantidad de flujo con la que se sale del depósito cada día es igual a la cantidad de ciudades que se visitan en dicho día y la restricción (2.29) implica que no se regresa al depósito con flujo. Finalmente las (2.31) y (2.32) nos dicen que las variables son binarias.

2.4.3. Variaciones

- Una ciudad no se puede visitar dos días consecutivos. Se podría con las siguientes inecuaciones.

$$y_i^k + y_i^{k+1} \leq 1, \qquad \forall k = \{1, ..., p-1\}, \forall i \in V \setminus \{1\} \tag{2.33}$$

- Hay un número máximo de ciudades que se visitan cada día. Por ejemplo solo podemos visitar l localizaciones cada día.

$$\sum_{i \in V} y_i^k \leq l, \qquad \forall k = \{1, ..., p\} \tag{2.34}$$

- Hay un beneficio al visitar cada ciudad. Denotamos p_i como el beneficio obtenido al visitar cada ciudad.

  En este caso la restricción (2.25) no existiría o podría estar de otra forma, con $\leq$ existiría un número máximo de visitas por ciudad o con $\geq$ existiría un número mínimo de visitas por ciudad. Para tener en cuenta un beneficio si se visita una ciudad podemos cambiar la función objetivo por:

$$\text{mín} \sum_{k=1}^{p} \sum_{i,j \in V, i \neq j} c_{ij} x_{ij}^k - \sum_{k=1}^{p} \sum_{i \in V} p_i y_i^k \tag{2.35}$$

- Restricción para evitar la simetría. En el caso de que utilicemos el modelo (2.22)-(2.32) se producirían muchas simetrías ya que si tenemos una solución, cualquiera de las $p!$ formas de ordenar los días también serían solución. Esto generalmente hace que un programa que resuelva modelos de programación entera tarde mucho más. Una forma de evitarlo es introduciendo la siguiente familia de restricciones:

$$\sum_{i=2}^{n} 2^{n-i} y_i^k \geq \sum_{i=2}^{n} 2^{n-i} y_i^{k+1} \qquad \forall k = 1, ..., p-1 \tag{2.36}$$

A veces cuando n empieza a ser grande estas restricciones comienzan a hacer que el algoritmo tarde más por lo que se puede coger un término intermedio.

$$\sum_{i=2}^{l} 2^{n-i} y_i^k \geq \sum_{i=2}^{l} 2^{n-i} y_i^{k+1} \qquad \forall k = 1, ..., p-1 \tag{2.37}$$

para $2 \leq l \leq n$.

Capítulo 3

Trabajos anteriores sobre el PTSP

Este capítulo consta de resúmenes de trabajos y artículos sobre problemas de rutas con periodicidad, entre los cuales se encuentra principalmente el TSP con periodicidad.

3.1. Un algoritmo para un problema de periodicidad en rutas de transporte de mercancías

Garrido y Onaindía [8] describen algoritmos cuyo objetivo es minimizar los costes asociados al transporte satisfaciendo una serie de restricciones. Puede tener distintas periodicidades: semanal, quincenal, etc.

3.1.1. Planteamiento del problema

El objetivo de este problema es la determinación y planificación de las rutas de menor coste, seleccionando el modo de transporte más adecuado en función de las características de las expediciones. Dichas características son:

- Plaza de origen y destino.
- Tipo de carga. Está condicionado por las características de la propia carga(peso, volumen, fragilidad, etc).
- Plazo de entrega.
- Periodo de repetición: puede ser semanal, quincenal,etc.
- Trayecto que seguirá la expedición.

La solución óptima alcanzada será aquella en la que se transporten todas las expediciones consideradas como entrada del problema, cumpliendo sus respectivos plazos de entrega.

3.1.2. Formulación del problema

El objetivo del problema es obtener la mejor planificación de rutas y vehículos para transportar las expediciones cumpliendo con todas las restricciones existentes.

La función objetivo que se deberá minimizar consiste en el coste de realizar todas la rutas. Este coste es la suma de las cantidades siguientes: coste de contratación y uso de los vehículos, coste originado por las operaciones de intermediación y coste de paralización de vehículos.

También se deben cumplir las siguientes restricciones:

- Restricciones de demanda de las expediciones.
- Plazos de entrega de las expediciones.
- Restricciones de capacidad de los vehículos.
- Turnos de trabajo de las delegaciones.

No existe restricción de disponibilidad de vehículos, ya que no hay un límite en cuanto al número de vehículos a emplear.

3.1.3. Diseño del algoritmo

El algoritmo está dividido en dos etapas. La primera etapa es el cálculo de la cota pesimista inicial y la segunda etapa es la optimización, mediante un algoritmo admisible que a su vez se divide en dos etapas: solapamiento de carga y reutilización de vehículos.

3.1.4. Características principales (Resultados)

- El algoritmo encuentra la solución óptima al problema en caso de que esta exista. Además, devuelve un conjunto de soluciones alternativas subóptimas.
- El proceso de búsqueda funciona de forma progresiva, devolviendo en cada instante la mejor solución hasta el momento, que se refina sucesivamente.
- El usuario de la herramienta puede detener el proceso de optimización cuando se obtenga una solución que satisfaga las necesidades en cuanto a calidad de la misma.

3.2. Un algoritmo genético para un problema de transporte de pasajeros con periodicidad

En el artículo de Aguado-Aranda y Jiménez-de-Vega [1] el problema consiste en recoger al mayor número de pasajeros posibles con el menor coste, donde el parámetro clave es la

distancia recorrida.

3.2.1. Componentes de la gestión de rutas

Son componentes de la gestión de rutas los pasajeros, las paradas, el transporte y las vías de la ruta.

El trabajo estudia la utilización de varios algoritmos avanzados para su resolución como son el algoritmo de agrupamiento o *Clustering* y los algoritmos genéticos.

Los algoritmos genéticos tienen su campo de aplicación importante en problemas de optimización compleja, donde se tienen diferentes parámetros o conjuntos de variables que deben ser combinadas para su solución.

El algortimo implementado representa la solución del problema, decodifica los cromosoma (para determinar el número real que la cadena binaria de caracteres a representar) y hace una evaluación de cada individuo (nos muestra el valor de aptitud de cada uno de los individuos).

3.2.2. Resolver el problema

El objetivo es elaborar una aplicación de gestión de rutas que permita elaborar, mediante algoritmos de *Clustering* y genéticos un recorrido óptimo de recogida de trabajadores de una empresa.

Se trata de reducir el tiempo y el gasto del traslado de los trabajadores a la empresa (minimizar el espacio y el tiempo).

El algoritmo de *Clustering* solo depende de la distancia entre las paradas mientras que el algoritmo genético está determinado por las distancias y por el tiempo de transporte necesario para recorrerlas.

Como base de datos se utiliza ficheros Excel.

3.2.3. Resultados computacionales

En los resultados obtenidos vemos que los algoritmos de *Clustering* ofrecen resultados óptimos para la agrupación de pasajeros, incluso con altas concentraciones en un perímetro acotado, consigue simplificar la ruta de transporte. Por tanto, el agrupamiento de paradas aplicando *clustering* disminuye mucho el tiempo de la ruta, optimizando el número y la posición de las paradas.

En cambio, el algoritmo genético proporciona resultados óptimos para problemas reducidos pero al aumentar el número de paradas, los resultados empeoran (no se consigue una solución óptima aceptable).

3.3. Técnicas de modelación para problemas de rutas de vehículos con periodicidad

En este trabajo, realizado por Francis y Smilowitz [7], se resuelve un problema de rutas de vehículos con servicio de elección.

3.3.1. Problema de rutas de vehículos con periodicidad con servicio de elección

La frecuencia de visita a los nodos es una decisión del modelo. Las rutas están diseñadas para una flota fija de vehículos capacitados cada día de un periodo de t-días para visitar a los clientes exactamente un número preestablecido de veces.

El problema queda definido de la siguiente manera. Hay un conjunto de nodos con demanda conocida y frecuencia mínima de visita que requiere servicio durante el periodo de planificación, una flota de vehículos capacitados, un conjunto de programas de servicio con intervalos entre servicios y prestaciones y una red con los tiempo de viaje. Se trata de encontrar una asignación de nodos a los horarios de servicio y un conjunto de rutas de vehículos por cada día del periodo de planificación que reduzca al mínimo el costo de enrutamiento neta total incurrido del beneficio de servicio devengados.

En este modelo, los clientes reciben la visita un número predeterminado de veces a lo largo del periodo con un horario que se elige en el menú de opciones de programación.

3.3.2. Métodos de resolución

Se resuelve de dos maneras: con formulación discreta y mediante aproximación continua.

Formulación discreta

Se define para un conjunto de nodos, formado de nodos cliente y un depósito, además de un conjunto de arcos que conectan los nodos.

Este tipo de modelo, con formulación discreta, están limitados a casos de problemas de tamaño moderado, pero es muy adecuado para los problemas operacionales y tácticos cuando las soluciones detalladas son necesarias.

Aproximación continua

En este modelo aproximamos variables discretas y parámetros con funciones continuas.

Las funciones continuas se introducen para describir las asignaciones de servicios y rutas de vehículos. Los nodos no se pueden servir con una frecuencia inferior al mínimo especificado.

Los costos de enrutamiento se dividen en un coste de línea de transporte desde el depósito a la vecindad de los nodos individuales.

Utilizamos la descomposición geográfica para reducir la formulación a un problema más simple que se pueda resolver facilmente.

Se descompone el problema en un conjunto de subregiones donde cada subregión debe contener al menos una ruta.

No hay límites en el número de vehículos o la distancia recorrida por un vehículo, por tanto, una solución óptima se elige para el transporte de cargas del vehículo completo. Dentro de cada subregión las limitaciones de capacidad siempre son vinculantes.

Este es un problema de asignación generalizada con una función objetivo no lineal.

El modelo de aproximación continua produce en menos tiempo los resultados para los problemas grandes en comparación con el modelo discreto.

3.3.3. Resultados

El modelo de aproximación continua se aplica al ejemplo 100b sin opción de servicio para evaluar los resultados de costos de enrutamiento.

El modelo de aproximación continua produce una solución de 2.164 en menos de 1 segundo de CPU que representa más o menos una desviación del 5 % de las soluciones más conocidas. Una de las razones de la discrepancia puede ser que el modelo continuo considera un menor número de opciones de programación. Además, la solución de las tres subregiones de forma independiente en el modelo continuo puede aumentar la función objetivo.

La región donde se realiza el problema se divide en tres subregiones.

Realizamos el ejemplo en tres escenarios distintos: El primero es el PVRP tradicional en la que los nodos deben ser servidos en su nivel de servicio asignado inicialmente, el segundo permite elegir el servicio, pero sólo tiene en cuenta el impacto de esta elección en la eficiencia de enrutamiento; la función objetivo no incluye el término beneficio servicio y el tercero permite la elección de servicios y considera tanto el costo de enrutamiento y servicio en beneficio de la función objetivo.

El nivel de beneficio en la región completa con el primer escenario es 1185, con el segundo es 1340 y considerando el tercer escenario tenemos un beneficio de 3872, lo que implica un mayor beneficio cuando utilizamos el PVRP con servicio de elección.

3.4. Heurísticos constructivos para el PTSP

Este trabajo redactado por Costanzo [5] consiste en que dada una red de ciudades conectadas entre sí, hallar cual es el camino más corto que debería hacer un viajante para

que partiendo de una ciudad determinada visite todas las ciudades una única vez y vuelva al inicial.

Es $\mathcal{NP}$-completo, pero dispone de algoritmos heurísticos que se utilizan para aportar soluciones cercanas a la óptima y presentan estimaciones del error con respecto a la solución óptima.

- *Nearest Neighbor.*
- *Double the Tree.*
- *Tee and Matching.*

Consideramos el problema periódico del viajante donde el viajante dispone de un cierto número de días para visitar n ciudades cada una de ellas un número de veces m_i. Además para cada una de las ciudades se da una lista de las posibles combinaciones de días en que pueden ser visitadas. Cada día se debe volver a la ciudad de partida y no se puede visitar más de una vez el mismo día las misma ciudad con excepción de la de partida/llegada.

Se debe comprobar que no ha quedado ningún día con un itinerario vacío, en cuyo caso se elige las ciudades con menor coste de inserción para incluirlas en los itinerarios vacíos.

3.5. Un nuevo algoritmo heurístico para el PTSP

Este artículo, escrito por Paleta [13], resuelve el problema del viajante de comercio con periodicidad, en el cual existe un depósito, es decir, una ciudad de la que parte y a la que se debe volver al final del recorrido. El objetivo es minimizar el recorrido realizado cada día.

Para resolver dicho problema se utiliza un algoritmo heurístico, el cual es un simple procedimiento constructivo con una fase de mejora.

Para obtener los resultados se ha codificado en FORTRAN y se aplica el problema a 33 problemas de la literatura. Se hace una comparación entre los resultados obtenidos por Chao et al. [3], Cordeau et al.[4] y los que se consigue con el nuevo algoritmo utilizando 23 problemas de prueba. Los resultados obtenidos son que el algoritmo encuentra la solución óptima en 18 de los problemas, Cordeau et al. [4] en 10 y Chao et al. [3] en 9. Cuando utilizan 33 problemas para realizar la prueba se comparan los resultados de Cordeau et al. [4] con los del algoritmo nuevo obteniendo 15 y 25 mejores soluciones respectivamente. Por tanto, en ambas pruebas se comprueba que el nuevo algoritmo consigue un mayor número de buenas soluciones.

3.6. Un problema de rutas con periodicidad y ventanas de tiempo en una empresa canaria

Este trabajo, realizado por Rodríguez-García [15] y posteriormente continuado por Ruigómez-González [16] resuelve un problema de logística de la empresa Biocontrol Canarias S.L., relacionado con un problema de rutas de vehículos en el cual se utiliza un solo vehículo.

La periodicidad en este caso es semanal, quincenal, mensual, bimestral, trimestral, semestral o anual, dependiendo de la demanda o el trabajo a realizar en cada empresa.

3.6.1. Características

- Hay 124 clientes y un depósito.
- Un solo vehículo y un solo empleado.
- Cada día se realiza como máximo una ruta.
- Cada cliente necesita un cierto tiempo de proceso, dependiendo del tipo de trabajo que solicite (necesite).
- Existen unas ventanas de tiempo (cada cliente debe ser visitado dentro de un periodo determinado de tiempo)
- Existe restricciones en cuanto al día de visita.
- Cada empresa requiere una periodicidad de visita (dependiendo de la empresa y del trabajo a realizar)
- Las visitas han de realizarse dentro del horario laboral de la empresa Biocontrol Canarias S.L., a excepción de las visitas a otras islas.
- Se conocen las distancias y los tiempos necesarios para los desplazamientos.

El objetivo del problemas es minimizar el tiempo empleado en recoger la muestras utilizando un horizonte semanal.

3.6.2. Como se resuelve el problema

Para la obtención de datos es necesario generar una matriz de distancias y/o tiempos trabajando con las API de Google Maps y para su recopilación se utiliza un hoja de cálculo Excel.

El trabajo realizado por Rodríguez- García [15] plantea dos modelos matemáticos para resolver el problema: modelo matemático fijo y modelo matemático alternativo. En el

primer modelo obliga a elegir entre visitarlo o no sin tener en cuenta el beneficio, es decir, está prefijada la semana en la que se visita cada clientes, en cambio, con el segundo modelo, también usado por Ruigómez-González [16], se tiene en cuenta la posibilidad de que convenga visitar un cliente esa semana aunque no sea necesario, al generar un buen beneficio.

Ruigómez-González [16] introduce un nuevo modelo por caminos o con variables rutas para resolver el problema, motivado por el gran esfuerzo computacional necesario para hallar la solución con el modelo con variables arco (modelo alternativo).

En estos modelos, en la función objetivo se añade un beneficio constante que se obtiene al visitar a los clientes.

Ahora pasamos a escribir los dos modelos usados:

Modelo matemático alternativo o con variables arco

Parámetros

$N = \{1, ..., n, n+1\}$ conjunto de empresas, donde 1 se denomina el depósito (Biocontrol Canarias S.L.).

t_{ij} =tiempo que se tarda de la empresa i a la j.

P_i =tiempo de proceso en la empresa i.

$M_{ij} = \text{máx}\{b_i + P_i + t_{ij} - a_j, 0\}$.

$$d_i^k = \begin{cases} 1, & \text{si la empresa } i \text{ puede ser visitada el dia } k \\ 0, & \text{en otro caso} \end{cases}$$

Variables

$$x_{ij}^k = \begin{cases} 1, & \text{si se va de } i \text{ a } j \text{ el dia } k \\ 0, & \text{en otro caso} \end{cases}$$

T_i^k= tiempo de llegada a la empresa i el día k

$$y_i^k = \begin{cases} 1, & \text{si la empresa } i \text{ es visitada el dia } k \\ 0, & \text{en otro caso} \end{cases}$$

En este modelo utiliza un horizonte semanal otorgando un grado de prioridad a cada cliente y se establecen unos beneficios artificiales o bonus, premiando a aquellas empresas que están lejos del depósito si son visitadas. Es más flexible, aunque es difícil de resolver de forma exacta.

Denotamos la prioridad de cada empresa:

$$h_i = \begin{cases} 0, & \text{si obligatoriamente hay que visitar la empresa } i \text{ a lo largo de la semana} \\ 1, & \text{si puede interesar la visita de la empresa } i \text{ porque aporta un beneficio grande} \\ 2, & \text{si se puede visitar la empresa } i \text{ con un beneficio no tan grande} \\ 3, & \text{si no debe visitarse la empresa } i \text{ en esa semana} \end{cases} \tag{3.1}$$

Se denota el beneficio artificial o bonus con g_i, siendo

$$g_i = \alpha(3 - h_i)(t_{1i} + t_{i1}), \forall i \in N$$

donde α es una constante.

Modelo matemático

$$\text{mín} \sum_{k=1}^{5} \sum_{(i,j)\in A} t_{ij} x_{ij}^k - \sum_{k=1}^{5} \sum_{i \in N} g_i y_i^k \tag{3.2}$$

sujeto a:

$$\sum_{j\in\delta^+(i)} x_{ij}^k = \sum_{j\in\delta^-(i)} x_{ji}^k = y_i^k, \qquad \forall i \in N, k = 1, ..., 5 \tag{3.3}$$

$$a_i \leq T_i^k \leq b_i, \qquad \forall i \in N, k = 1, ..., 5 \tag{3.4}$$

$$T_j^k \geq T_i^k + P_i + t_{ij} - M_{ij}(1 - x_{ij}^k), \qquad \forall j \in N, k = 1, ..., 5 \tag{3.5}$$

$$y_i^k \leq d_i^k, \qquad \forall i \in N, k = 1, ..., 5 \tag{3.6}$$

$$\sum_{k=1}^{5} y_i^k = 1, \qquad \forall i \in N, h_i = 0 \tag{3.7}$$

$$\sum_{k=1}^{5} y_i^k \leq 1, \qquad \forall i \in N, h_i \in \{1, 2\} \tag{3.8}$$

$$y_i^k = 0, \qquad \forall i \in N, h_i = 3, k = 1, ..., 5 \tag{3.9}$$

$$y_1^k \geq y_i^k, \qquad \forall i \in N - \{1, n+1\}, k = 1, ..., 5 \tag{3.10}$$

$$x_{n+1,1}^k = y_1^k = y_{n+1}^k, \qquad k = 1, ..., 5 \tag{3.11}$$

$$x_{ij}^k \in \{0, 1\}, \qquad \forall i, j \in N, k = 1, ..., 5 \tag{3.12}$$

$$y_i^k \in \{0, 1\}, \qquad \forall i \in N, k = 1, ..., 5 \tag{3.13}$$

Si en este modelo tomamos $\alpha = 0$, obtenemos el modelo fijo.

Modelo con variables rutas

En este segundo modelo se estima que la empresa visita máximo a cuatro clientes por ruta que realiza, siendo en este caso el objetivo calcular el número de rutas y el costo asociado a cada una.

Parámetros

$N = 1, ..., n$ conjunto de clientes, $R = \{1, 2, ..., r\}$ conjunto de rutas y $K = \{1, ..., 5\}$ días de la semana.

C_r = costo de la ruta.

$$m_{ri}^k = \begin{cases} 1, & \text{si la ruta } r \text{ pasa por el cliente } i \\ 0, & \text{en otro caso} \end{cases}$$

$$d_r^k = \begin{cases} 1, & \text{si la ruta } r \text{ se puede hacer el dia } k \\ 0, & \text{en otro caso} \end{cases}$$

Generamos un beneficio artificial:

$$g_i = \alpha(3 - h_i)(t_{1,i} + t_{i,1}), \forall i \in N$$

Como en el otro modelo denotamos la prioridad que tiene cada empresa:

$$h_i = \begin{cases} 0, & \text{si hay que visitar la empresa } i \text{ a lo largo de la semana} \\ 1, & \text{si se puede visitar la empresa } i \text{ con un beneficio grande} \\ 2, & \text{si se puede visitar la empresa } i \text{ con un beneficio no tan grande} \\ 3, & \text{no hay que visitar la empresa } i \end{cases}$$

y $\alpha \in (0, 0{,}5)$.

Variables

$$x_r^k = \begin{cases} 1, & \text{si la ruta } r \text{ se hace el dia } k \\ 0, & \text{en otro caso} \end{cases}$$

Modelo matemático

$$\text{mín} \sum_{r \in R} \sum_{k=1}^{5} x_r^k C_r - \sum_{k=1}^{5} \sum_{i \in N} \sum_{r \in R} g_i m_{ri} x_r^k \tag{3.14}$$

sujeto a:

$$\sum_{k=1}^{5} \sum_{r \in R} m_{ri} x_r^k = 1 \qquad \forall i \in N, h_i = 0 \tag{3.15}$$

$$\sum_{k=1}^{5} \sum_{r \in R} m_{ri} x_r^k \leq 1 \qquad \forall i \in N, h_i \in \{1, 2\} \tag{3.16}$$

$$\sum_{k=1}^{5} \sum_{r \in R} m_{ri} x_r^k = 0 \qquad \forall i \in N, h_i = 3 \tag{3.17}$$

$$x_r^k \leq d_r^k \qquad \forall r \in R, k = 1, ..., 5 \tag{3.18}$$

$$\sum_{r \in R} x_r^k \leq 1 \qquad k = 1, ..., 5 \tag{3.19}$$

En estos modelos el número de periodicidades es 5, que correspode a los días de lunes a viernes.

3.6.3. Resultados computacionales

Rodríguez-García [15] compara el modelo alternativo para $\alpha = 0$ y $\alpha = 0,1$ con un ejemplo durante 5 semanas utilizando un paquete de optimización de pago llamado Mosel. Se concluye que el modelo alternativo supone un ahorro del 5,2 % del tiempo de ruta obteniendo para el modelo fijo un total de 6780 minutos repartidos en cinco semanas y para el alternativo un total de 6653 minutos repartidos en cuatro semanas. La solución del modelo alternativo se muestra en un mapa.

En el caso de Ruigómez-González [16], compara el modelo por arcos con el modelo con variables rutas para dos valores de α, $\alpha = 0,1$ y $\alpha = 0,3$. Para generar las variables que corresponden a cada ruta se realiza un preproceso que ha sido programado por Visual Basic para Aplicaciones, componente de Microsoft Excel y luego se utiliza un programa libre que se llama Gusek para resolverlos, el cual lee los datos de la tabla Excel. Con ambos ejemplos comprueba que, pese a que el segundo modelo es más rápido, con el primero se obtienen mejores resultados.

3.7. Otros trabajos

El trabajo de Paletta y Triki [14] trata de buscar un algoritmo heurístico para resolver el problema del viajante de comercio con periodicidad, el cual se basa en una construcción de recorrido seguido por un procedimiento de mejora.

Yu y Yang [18] resuelve el problema de ruta de vehículos con periodicidad y ventanas de tiempo y para ello trata de mejorar el algoritmo de colonia de hormigas. Para evaluar este método se utiliza un conjunto de puntos conocidos.

En el trabajo de Chao et al. [3] se presenta un nuevo algoritmo heurístico para resolver el problema del viajante de comercio con periodicidad. Este se prueba con 23 problemas diferentes de la literatura demostrando que es un heurístico eficiente.

El trabajo de Cordeau et al. [4] relaciona tres problemas, el problema de ruta de vehículos con periodicidad, el problema del viajante de comercio con periodicidad y el problema de rutas de vehículos con multiples depósitos. Para resolverlos propone un algoritmo heurístico de búsqueda tabú utilizando parámetros generados aleatoriamente.

Cacchiani et al. [2] trata del problema de rutas de vehículos con periodicidad y se propone un algoritmo heurístico basado en la relajación de la programación lineal de un modelo. Se prueba en casos conocidos de relevancia y en casos realistas.

Capítulo 4

Resultados computacionales para PTSP

El objetivo de este trabajo es resolver un modelo para el TSP con periodicidad definido en la Sección 2.4, con resolutores lineales y en este capítulo se recogen los resultados computacionales de resolverlo con dos programas diferentes, Gusek y Xpress-Mosel, los cuales resuelven problemas de programación lineal entera. Hacemos una breve introducción sobre cada uno y finalmente nos centramos en las soluciones.

4.1. GUSEK

Es un software libre. El programa Gusek es la versión para windows del paquete GLPK (*GNU Linear Programming Kit*) que se destina a resolver problemas de programación lineal, programación entera mixta y otros problemas relacionados. GLPK soporta el lenguaje de modelado GNU MathProg, que es un subconjunto del lenguaje AMPL. Este se puede usar tanto como un solucionador de problemas independiente o como una biblioteca.

El paquete GLPK incluye los siguientes componentes principales:

- Métodos simplex primal y dual (primal and dual simplex methods)
- Método de punto interior primal-dual (primal-dual interior-point method)
- Método de la rama y corte (*branch-and-cut method*)
- Traductor para GNU MathProg
- Interfaz de programación de aplicación
- Stand-alone LP / solucionador de MIP

4.2. XPRESS-MOSEL

Es un programa de pago que proporciona un lenguaje que es un modelado y un lenguaje de programación a la vez. Es un entorno para los problemas de modelado y de resolución. Este, al igual que Gusek, resuelve problemas de programación lineal y programación entera mixta. La originalidad de este lenguaje es la sinergia entre la declaración del modelo (declarar una variable de decisión, expresar una restricción) y un procedimiento que realmente lo resuelva (llamada a un comando de optimización).

Por otro lado, al tener cada tipo de problema sus propias clases de variables y restricciones, utilizar un único tipo de optimizador para todas las posibles combinaciones resulta poco eficiente. Este sistema, teniendo esto en cuenta, no integra ningún optimizador de forma predeterminada, aunque si ofrece una interfaz dinámica para resolver externos proporcionados en forma de módulos. Estos módulos vienen con su propio conjunto de procedimientos y funciones que extienden directamente el vocabulario y las capacidades del lenguaje Mosel.

Para este programa existe una versión completa para proyectos de fin de carrera a la cual no se puede acceder desde las empresas o particulares y una versión de estudiantes, que es más simple.

4.3. Resultados computacionales

Con ambos programas ejecutamos los mismos ejemplos para ver el resultado y el tiempo que tarda en obtenerlo cada programa, comparando así que programa es más eficiente. Como Gusek tarda demasiado solo se utiliza $n = 8$ para comparar ambos programas, para el resto de n mayores se resolverán con Mosel.

El ordenador usado para hallar las soluciones tiene las siguientes características: Intel Core 17-2600 CPU a 3.40 GHz con 16 GB de memoria RAM, aunque se utilizó muy poca memoria RAM y un solo hilo de ejecución.

Con estos programas queremos calcular el costo mínimo de visitar las diferentes ciudades en un cierto número de días. Tenemos una cantidad de ciudades a visitar cuyo número lo denotamos por n, tenemos un número días en los que se pueden visitar dichas ciudades denotado por p, existe un número de veces para visitar cada una que se escribe con m_i, donde i significa la ciudad que será visitada, además existe un depósito, tomado como la localización 1, tal que, si el vehículo sale del depósito, debe regresar a él todos los días. También tenemos un valor máximo para los m_i, que denotamos por M y una semilla con la que se ha generado cada ejemplo.

Los datos que usaremos para conseguir los resultados con ambos programas se obtienen de forma aleatoria, para cada ejemplo de tamaño n se generan n pares de puntos en un cuadrado de $[0, 100] \times [0, 100]$, es decir, las coordenadas de estos puntos se generan con una uniforme entre 0 y 100 para cada cliente y luego se calcula el costo usando la distancia

euclídea. Los m_i se generan también aleatoriamente como una uniforme discreta entre 1 y M.

Se han generado un grupo de 5 ejemplos aleatorios diferentes (con semillas 1, 2, 3, 4 y 5), para los diferentes valores de n y para $M \in \{3, 5\}$

Utilizamos el modelo (2.22)(2.32) del PTSP con variables flujo y añadiendo la restricción (2.37), cuyo fin es evitar la simetría en las rutas.

En esta primera tabla haremos una comparación entre los programas con los mismos datos, esta constará, además de los valores antes descritos, de los tiempos de ejecución de cada programa, tomados en segundos, y con estos veremos que programa es más eficiente. Solo usaremos, como indicamos al principio de esta sección, $n = 8$ para hacer esta comparación, poniendo un tiempo límite a ambos programas de 7200 segundos.

n	**p**	M	seed	$\sum m_i$	Costo Mosel/Gusek	Tiempo Mosel	Tiempo Gusek
8	5	3	1	14	584	0.2	11.2
8	5	3	2	17	841	0.3	6.2
8	5	3	3	9	693	0.2	0.8
8	5	3	4	17	729	0.7	1132.7
8	5	3	5	15	739	0.7	79.9
8	5	5	1	22	875	0.4	4810.4
8	5	5	2	27	1301	0.1	112.8
8	5	5	3	11	823	0.5	2.4
8	5	5	4	26	1154 /1156	0.7	7200.0
8	5	5	5	21	943	1.1	162.4

Cuadro 4.1: Comparación de tiempos con Gusek y Mosel

Lo primero que vemos en esta tabla con más relevancia es la gran diferencia entre los tiempos de ejecución de ambos programas, mientras que Mosel tiene un tiempo pequeño de a lo sumo 1,1 segundos, el Gusek necesita en la mayoría de los casos un tiempo mucho mayor, hasta llegar en un caso a no conseguir la solución óptima antes de llegar al tiempo límite.

En esta tabla se muestra solo un costo pues ambos modelos devuelven el mismo resultado, salvo para $n = 8$, $p = 5$, $M = 5$ con semilla 4, donde el Gusek llega al tiempo límite antes de conseguir la solución óptima y devuelve como costo 1156 en lugar de 1154, que sería el resultado.

Como se puede ver, los valores de $\sum m_i$ son mayores cuando $M = 5$ y esto es así porque hay que visitar más ciudades cada día. También vemos que en este caso los costos son mayores.

Ahora tenemos una tabla resumen, creada haciendo la media de $\sum m_i$, del tiempo empleado en Gusek y el de Mosel, así como el costo medio, cada cinco líneas de la tabla anterior.

n	**p**	M	$\sum m_i$	Costo	Tiempo mosel	Tiempo gusek
8	5	3	14.4	717.2	0.4	246.2
8	5	5	21.4	1019.2	0.6	2457.6

Cuadro 4.2: Resumen de los resultados con Gusek y Mosel

En esta tabla, como en la general, lo más significativo es la gran diferencia que existe entre los tiempos de ejecución, se ve claramente como los tiempos de Mosel en media no llegan al segundo y los de Gusek son mucho mayores.

Veamos ahora, para valores más grandes de n, los resultados que se obtienen con Mosel.

n	**p**	M	seed	$\sum m_i$	Costo	Tiempo
10	5	3	1	18	730	1,0
10	5	3	2	18	809	1,1
10	5	3	3	15	843	1,2
10	5	3	4	19	678	2,5
10	5	3	5	18	669	1,2
10	5	5	1	28	1109	1,0
10	5	5	2	28	1300	0,6
10	5	5	3	22	1098	2,8
10	5	5	4	29	1030	2,5
10	5	5	5	26	999	1,7
12	5	3	1	24	752	3,1
12	5	3	2	23	895	3,2
12	5	3	3	26	1021	2,8
12	5	3	4	16	625	24,9
12	5	3	5	23	821	4,5
12	5	5	1	35	1096	8,9
12	5	5	2	33	1339	2,3
12	5	5	3	38	1476	46,8
12	5	5	4	22	773	10,9
12	5	5	5	35	1242	7,2

Cuadro 4.3: Resultados computacionales con Mosel para $n = 10$ y $n = 12$.

14	5	3	1	29	889	6,3
14	5	3	2	23	841	41,1
14	5	3	3	31	1017	10,0
14	5	3	4	18	618	41,0
14	5	3	5	28	826	9,7
14	5	5	1	42	1415	46,3
14	5	5	2	31	1152	2828,7
14	5	5	3	46	1493	3,5
14	5	5	4	22	829	350,8
14	5	5	5	40	1234	52,3
16	5	3	1	29	856	11,1
16	5	3	2	28	942	1311,9
16	5	3	3	33	1034	11,7
16	5	3	4	28	885	270,7
16	5	3	5	31	871	987,2
16	5	5	1	44	1371	785,7
16	5	5	2	41	1406	7200,0
16	5	5	3	51	1581	11,4
16	5	5	4	41	1326	7200,0
16	5	5	5	44	1182	1421,9

Cuadro 4.4: Resultados computacionales con Mosel para $n = 14$ y $n = 16$.

En estas dos tablas con los resultados de Mosel, si miramos el $\sum m_i$, vemos que es más pequeña cuando $M = 3$ que cuando es igual a 5 y también se puede comprobar que a medida que aumenta el tamaño de n, estos valores aumentan. Algo muy parecido ocurre con los costos, también son mayores para $M = 5$.

Ahora si nos fijamos en el tiempo se puede ver que también aumenta cuando aumenta el valor de n, y lo más curioso es que hay dos casos donde no se consigue el valor óptimo pues se llega al tiempo límite antes de llegar a este, estos casos son para $n = 16$, $p = 5$ y $M = 5$, con semillas 2 y 4.

Tabla resumen de los valores de Mosel.

n	**p**	M	$\sum m_i$	Costo	Tiempo
8	5	3	14.4	717.2	0.4
8	5	5	21.4	1019.2	0.6
10	5	3	17.6	745.8	1.4
10	5	5	26.6	1107.2	1.7
12	5	3	22.4	822.8	7.7
12	5	5	32.6	1185.2	15.2
14	5	3	25.8	838.2	21.6
14	5	5	36.2	1224.6	656.3
16	5	3	29.8	1191.8	518.5
16	5	5	44.2	1373.2	3323.8

Cuadro 4.5: Resumen de los resultados de Mosel

En esta tabla resumen se ve como los tiempos aumentan, sobretodo para $n = 16$ y $M = 5$, el tiempo es bastante mayor motivado por los dos ejemplos donde se llega al tiempo máximo.

Fijándonos en los costos vemos que, para cualquier n, siempre es menor para $M = 3$. Hallando el porcentaje tenemos que para todos los valores de n, el costo para $M = 3$ es mejor con un porcentaje muy próximo al 50 %, salvo para n=16, que no hay mucha diferencia, pues el porcentaje de mejora de $M = 3$ frente a $M = 5$, es de un 15 %

Con este modelo podemos resolver el TSP cuando todos los m_i toman el valor 1 y $p = 1$, aunque esta no es la finalidad de este trabajo.

Veamos las gráficas para dos ejemplos con n=10, p=5 y seed=1 mostradas por día, para los dos valores de $M = 3$ y $M = 5$. Como p es el número de días en los que hay que visitar las diferentes y lo tomamos con valor 5, veremos las rutas con un horizonte semanal.

Primeramente veremos las rutas para $M = 3$.

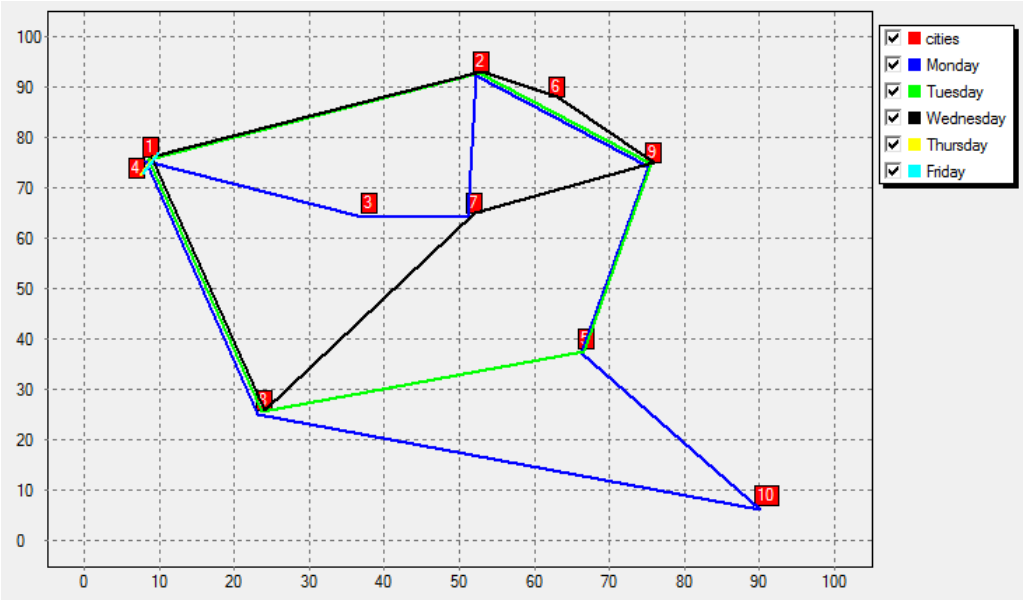


Figura 4.1: Representación gráfica para $n = 10$, $p = 5$ y $M = 3$

En este ejemplo el primer día se visitan 7 ciudades y estas serán: 8, 10, 5, 9, 2, 7 y 3, siendo la ruta en ese orden. El segundo día se visitan solo 4 ciudades: 8, 5, 9 y 2. El tercer día se visitan 5 ciudades: 8, 7, 9, 6 y 2. Y por último, en los dos días restantes se visita solamente la ciudad 4. En todas estas rutas se parte del depósito y se regresa a él al finalizar.

Ahora veamos las rutas con M=5.

Figura 4.2: Representación gráfica para $n = 10$, $p = 5$ y $M = 5$

En este nuevo ejemplo el primer día se visitan 6 ciudades: 2, 6, 9, 7, 3 y 8. El segundo día se visitan 5 ciudades y estas son 8, 3, 7, 9 y 2. El tercer día se visitan las ciudades 4, 8, 10, 5, 9, 6 y 2, en ese orden. El cuarto día se visitan las ciudades 2, 9, 5, 8 y 4. Y el último día se visitan 5 ciudades: 2, 9, 7, 8, 4. En todas estas rutas también se sale y se regresa al depósito al finalizar el día.

Las ciudades en el primer ejemplo, como se deduce del valor de M, se visitan como máximo 3 días a la semana y podemos ver que en las tres primeras rutas se visita a la mayoría de ellas, quedando para los dos últimos días solo la ciudad 4, repitiéndose el recorrido en ambos. Sin embargo, en el segundo ejemplo, al ser el valor de M mayor, en este caso igual al de p, se deben visitar las ciudades más días a la semana.

Luego, como se puede deducir fácilmente, en el segundo ejemplo, al ser el valor de M mayor que en el primero, las rutas de los distintos días son más largas, es decir, cada día se visitan más ciudades. En este segundo caso no se llega al caso de hacer una sola ruta uno o dos días a la semana como ocurre en el primero.

Capítulo 5

Conclusiones

En la presente memoria se han estudiado numerosos trabajos de la literatura sobre el PTSP. Se ha investigado y se han encontrado varios artículos relacionados con este problema aplicados a casos reales. Se han seleccionado y extraído las ideas principales. En estos se han visto varios métodos para resolver el PTSP, donde han destacado los métodos heurísticos, como por ejemplo los algoritmos genéticos o de agrupamiento.

Se proponen dos modelos matemáticos diferentes para este problema, uno con restricciones de subciclos y otro con variables flujo. Se decide usar el segundo modelo pues el primero, al tener un número de restricciones exponencial, resulta más difícil de implementar.

Una vez elegido el modelo, se procedió ha programarlo con dos programas distintos (Gusek y Mosel) y realizar los mismos ejemplos con los dos para poder comparar la eficacia de cada uno, es decir, el tiempo de ejecución que necesitaban. Los datos se generaron aleatoriamente con uno de ellos de forma que el otro programa también pudiera leer dichos datos.

Se probaron varios ejemplos con ambos programas y se comprobó que Gusek era mucho más lento. Para ejemplos más grandes (en los que el número de ciudades era mayor) se utilizó sólo Mosel.

Luego se comprobó que con ese modelo se producían muchas simetrías y se buscó una buena restricción para evitarlas, pues esto provocaba que el modelo tardara mucho más. Se probaron varias familias de restricciones que evitaban simetrías y se opto por una mejoraba bastante los tiempos de ejecución.

Finalmente se elaboraron las tablas con los datos obtenidos, tomando sólo ejemplos pequeños para Gusek y el resto para Mosel.

Apéndice A

Códigos

A.1. Código del modelo con Gusek

```
##############################################################################
# ptsp.mod
##############################################################################
#
# Mª Arantxa Peña Rodríguez
#
# Periodic Traveling Salesman Problem, PTSP
#
##############################################################################

/*Parametros*/

param n, integer, >= 3;
/* number of nodes */

param p, integer;


set V := 1..n;
/* set of nodes */

set C :=2..n;
/*clientes*/

set E, within V cross V, default V cross V;
/* set of arcs */

set D := 1..p;
/*número de días que se puede visitar a los clientes*/


param coord_x{i in V}, default Uniform(0, 100);
param coord_y{i in V}, default Uniform(0, 100);
```

```
param m{i in C} ,  default floor (Uniform(1,3)) ;
/*número de veces que hay que visitar al cliente i*/


param c{(i,j) in E}, default round( sqrt((coord_x[i]-coord_x[j])^2 + (coord_y[i]-coord_y[j])^2) );
/* costo de ir de i a j */


/*Variables*/

var x{(i,j) in E, k in D}, binary;
/* x[i,j]  means that the salesman goes from node i to node j */

var y{i in V, k in D}, binary;


/*Función Objetivo*/

minimize total: sum{k in D, (i,j) in E} c[i,j] * x[i,j,k];
/* the objective is to make the path length as small as possible */


/*Restricciones*/

s.t. leave{i in V, k in D}: sum{j in V: j!=i} x[i,j,k] = y[i,k];
/* the salesman leaves each node i exactly once */

s.t. enter{i in V, k in D}: sum{j in V: j!=i} x[j,i,k] = y[i,k];
/* the salesman enters each node j exactly once */



s.t. visit{i in C}: sum{k in D} y[i,k] = m[i];
/*el cliente i lo tenemos que visitar exactamente m[i] veces*/

s.t. uno{k in D}: y[1,k] = 1;
/*Todos los días hay que visitar a algún cliente*/




var f{(i,j) in E, k in D}, >= 0;
/* y[i,j] is the number of cars, which the salesman has after leaving
   node i and before entering node j; in terms of the network analysis,
   y[i,j] is a flow through arc (i,j) */


s.t. cap{(i,j) in E, k in D}: f[i,j,k] <= (n-1) * x[i,j,k];
/* if arc (i,j) does not belong to the salesman's tour, its capacity
   must be zero; it is obvious that on leaving a node, it is sufficient
   to have not more than n-1 cars */
```

```
s.t. node{i in C, k in D}:sum{j in V: j!=i} f[i,j,k] - sum{j in V: j!=i} f[j,i,k] = -y[i,k];
      /* summary flow into node i through all ingoing arcs */

s.t. flow_arrive_depot{i in C, k in D}: f[i,1,k]=0;

/*Evita que se formen subciclos*/

s.t. flow_leave_depot{k in D}: sum{i in C} f[1,i,k]= sum{i in C} y[i,k];

s.t. sim{k in 1..(p-1)}: sum{i in 2..7} 2^(7-i)*y[i,k] >= sum{i in 2..7} 2^(7-i)*y[i,k+1];
/* Añadimos una restricción para evitar simetrías */



param stime:= gmtime();
/*Calcula el tiempo utilizado antes de resolver*/

printf"gmtime: %f\n", stime;


solve;


param etime := gmtime();
/*Calcula el tiempo utilizado después de resolver*/

printf"gmtime: %f\n", etime;




printf"Tiempo total: %f\n", etime - stime;

printf  "Suma de los m{i}=  %g\n", sum{i in C} m[i];

printf "Optimal tour has length %d\n",  sum{(i,j) in E, k in D} c[i,j] * x[i,j,k];


# printf " n     p     sum_mi  cost    time\n" >> 'sol_gmpl.txt';
printf  "%2d", n >> 'sol_gmpl.txt';
printf  "    %2d", p >> 'sol_gmpl.txt';
printf  "    %5d", sum{i in C} m[i] >> 'sol_gmpl.txt';
printf  " %7.2f", sum{(i,j) in E, k in D} c[i,j] * x[i,j,k] >> 'sol_gmpl.txt';
printf  " %7.1f", gmtime() - stime >> 'sol_gmpl.txt';
printf  "\n" >> 'sol_gmpl.txt';


end;
```

A.2. Código del modelo con Mosel

```
/############################################################################################
 # PTSP.mos
 ############################################################################################
 #
 # Mª Arantxa Peña Rodríguez
 #
 # TSP periódico
 #
 ############################################################################################

 model "PTSP"
 uses "mmxprs","mmive","mmsystem"; !gain access to the Xpress-Optimizer solver

 !optional parameters section
 parameters
  n = 8
  p = 5
  seed = 1
  M = 3 !valor máximo de los m(i)
 end-parameters

 forward procedure write_data
 forward procedure write_results
 forward procedure draw

 !sample declarations section

 declarations
   NCITIES = n
   NDIAS = p

   max_lex_sym = 7

   CITIES = 1..NCITIES ! Ciudades
   DIAS = 1..NDIAS !numero de días
   C = 2..NCITIES

  x: array(CITIES) of integer
  y: array(CITIES) of integer

   COST: array(CITIES,CITIES) of integer !Costo de ir de un ciudad a otra
   fly: array(CITIES,CITIES,DIAS) of mpvar ! 1 si se va de i a j
   select: array(CITIES,DIAS) of mpvar  ! 1 si la ciudad esta en la ruta
   NEXTC: array(CITIES) of integer ! Next city after i in the solution
   mips,points,roads:integer
   flow: array(CITIES,CITIES,DIAS) of mpvar
   !Objective:linctr

 end-declarations

 writeln("Solving a PTSP with n=", NCITIES," cities and ", NDIAS, " days.")
```

```
setrandseed(seed)
forall(i in 1..NCITIES) do
  x(i):=round(random*100)
  y(i):=round(random*100)
end-do

forall(i in 1..NCITIES) do
 m(i):=round(random*M+0.5)
end-do


! We write data in a file for GUSEK
data_file_name := "ptsp.dat"
fopen(data_file_name, F_OUTPUT)
write_data
fclose(F_OUTPUT)


!We calulate the cost-matrix
forall(i,j in CITIES | i<j) COST(i,j):=round(sqrt((x(i)-x(j))*(x(i)-x(j))+(y(i)-y(j))*(y(i)-y(j))))
forall(i,j in CITIES | i<j) COST(j,i):=COST(i,j)

!MODEL

!Objetive function
TotalCost:= sum(i,j in CITIES, k in DIAS| i<>j) COST(i,j)*fly(i,j,k)

!Constraints
forall(i in CITIES, k in DIAS) sum(j in CITIES | i<>j) fly(i,j,k) = select(i,k)
forall(i in CITIES, k in DIAS) sum(j in CITIES | i<>j) fly(j,i,k) = select(i,k)

forall(i,j in CITIES, k in DIAS | i<>j) fly(i,j,k) is_binary
forall(i in CITIES, k in DIAS) select(i,k) is_binary

forall(k in DIAS) select(1,k)= 1

forall(i in C) sum(k in DIAS) select(i,k) = m(i)

forall(i,j in CITIES, k in DIAS) flow(i,j,k) <= (NCITIES-1)*fly(i,j,k)

forall(i in C, k in DIAS) sum(j in CITIES | i<>j) flow(i,j,k) - sum(j in CITIES | i<>j) flow(j,i,k) = -sele

forall(i in C, k in DIAS) flow(i,1,k) = 0

forall(k in DIAS) sum(i in CITIES) flow(1,i,k) = sum(i in CITIES) select(i,k)


! Avoid symetries

forall(k in 1..(NDIAS-1)) sum(i in 2..max_lex_sym) 2^(max_lex_sym-i) * select(i,k)  >=  sum(i in 2..max_lex


!We set a time limit
```

```
setparam("XPRS_MAXTIME", 7200)

now1 := datetime(SYS_NOW)
!Solve the problem
minimize(TotalCost)
now2 := datetime(SYS_NOW)

!We write the computational results on screen
write_results


!We write the computational results in a file
fopen("sol_mosel.txt", F_OUTPUT + F_APPEND)
write_results
fclose(F_OUTPUT)


!We draw the solution (only with mosel interphase)
draw


procedure write_data
 writeln("data;");
 writeln;
 writeln("param n:= ", NCITIES);
 writeln(";");
 writeln("param p:= ", NDIAS);
 writeln(";");
 writeln("param coord_x:= ");
 forall(i in 1..NCITIES) do
 writeln(i, "   ", x(i));
 end-do
 writeln(";");
 writeln("param coord_y:= ");
 forall(i in 1..NCITIES) do
    writeln(i, "   ", y(i));
 end-do
 writeln(";");
 writeln("param m:= ");
 forall(i in C) do
    writeln(i, "   ", m(i));
 end-do
 writeln(";");
 writeln("end;");
end-procedure


procedure write_results
 !writeln(" n   p  seed   M  sum_mi    cost    time")
 writeln(strfmt(n,2), strfmt(p,4), strfmt(seed,6), strfmt(M,4), strfmt(sum(i in C) m(i),8),
 strfmt(getobjval,8,2), strfmt(now2-now1,8,2))
 !writeln("Solution:")
 !forall(k in DIAS) do
 ! write("day ",k, ", arcs: ")
```

```
 ! forall(i,j in 1..NCITIES | i<>j) do
    ! if getsol(fly(i,j,k)) >= 0.99 then
    ! write("(", i, ",", j,")")
    ! end-if
 ! end-do
 ! writeln("")
 !end-do
end-procedure


procedure draw
 IVEerase
    IVEzoom(-5,-5,105,105)
    points:=IVEaddplot("cities",IVE_RED)
    roads1:=IVEaddplot("Monday",IVE_BLUE)
    roads2:=IVEaddplot("Tuesday",IVE_GREEN)
    roads3:=IVEaddplot("Wednesday",IVE_BLACK)
    roads4:=IVEaddplot("Thursday",IVE_YELLOW)
    roads5:=IVEaddplot("Friday",IVE_CYAN)

 forall(i in 1..NCITIES) do
 IVEdrawlabel(points,x(i),y(i),""+i)
 end-do
 !draw links

 forall(i,j in 1..NCITIES) do
     if getsol(fly(i,j,1)) = 1 then
       IVEdrawline(roads1,x(i),y(i),x(j),y(j))
     end-if
 end-do
 forall(i,j in 1..NCITIES) do
     if getsol(fly(i,j,2)) = 1 then
       IVEdrawline(roads2,x(i),y(i),x(j),y(j))
     end-if
     end-do
 forall(i,j in 1..NCITIES) do
     if getsol(fly(i,j,3)) = 1 then
       IVEdrawline(roads3,x(i),y(i),x(j),y(j))
     end-if
     end-do
 forall(i,j in 1..NCITIES) do
     if getsol(fly(i,j,4)) = 1 then
       IVEdrawline(roads4,x(i),y(i),x(j),y(j))
     end-if
     end-do
 forall(i,j in 1..NCITIES) do
     if getsol(fly(i,j,5)) = 1 then
       IVEdrawline(roads5,x(i),y(i),x(j),y(j))
     end-if
     end-do
end-procedure

end-model
```

Bibliografía

[1] A. Aguado Aranda y J. Jiménez de Vega. Optimización de rutas de transporte. Technical report, Universidad Complutense de Madrid, 2013.

[2] V. Cacchiani, V.C. Hemmelmayr y F. Tricoire. A set-covering based heuristic algorithm for the periodic vehicle routing problem. *Discrete Applied Mathematics*, 163:53–64, 2014.

[3] I. M. Chao, B. L. Golden y E. A. Wasil. A new heuristic for the period traveling salesman problem. *Computers & Operations Research*, 22:553–565, 1995.

[4] J. F. Cordeau, M. Gendreau y G. Laporte. A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Network*, 30:105–119, 1997.

[5] A. Costanzo. On the periodic traveling salesman problem. Technical report, Universitat Politècnica de València, 2008.

[6] G. Dantzig, D. Fulkerson y S. Johnson. Solution of a large–scale traveling salesman problem. *Operation Research*, 2:393—410, 1954.

[7] P. Francis y K. Smilowitz. Modeling techniques for periodic vehicle routing problems. *Transportation Research Part B: Methodological*, 40:872–884.

[8] A. Garrido y E. Onaindía. Un algoritmo para la optimización de rutas de transporte. Technical report, Universidad Politecnica de valencia.

[9] B. Gavish y S. C. Graves. Traveling salesman problem and related problems. Technical report, Massachusetts Institute of Technology, 1978.

[10] D Hidalgo-Campos, M. Olvia-Páez, P. Ortiz-Núñez y F. Pino-Canales. Heuristic algorithms, applications and instances of vehicle routing problem. 2013.

[11] Salazar-González J. J. *Programación Matemática*. Díaz de Santos. Madrid, 2001.

[12] A. Olivera. Heurísticas para problemas de rutas de vehículos. Technical report, Universidad de la República, 2004.

[13] G. Paletta. The period traveling salesman problem: a new heuristic algorithm. *Computers & Operations Research*, 29:1343–1352.

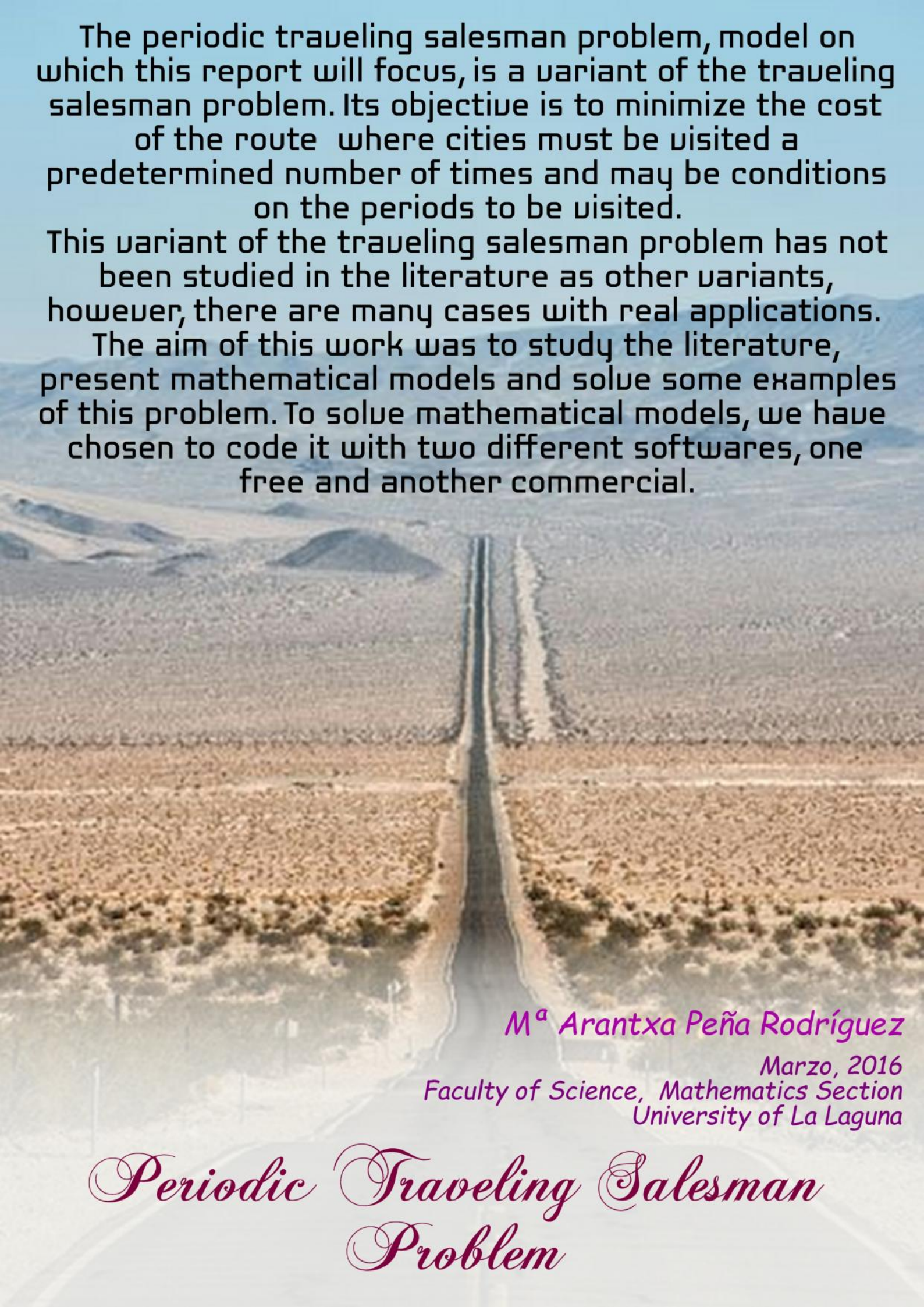
[14] G. Paletta y C. Triki. Solving the asymmetric traveling salesman problem with periodic constraints. *Network*, 44:31–37, 2004.

[15] I. Rodríguez García. Un problema de rutas con periodicidad y ventanas de tiempo. Technical report, Universidad de La Laguna, 2014.

[16] A. Ruigómez González. Resolución de un problema de rutas con ventanas de tiempo y periodicidad. Technical report.

[17] M. L. Stockdale. *El problema del viajante: un algoritmo heurístico y una aplicación.* PhD thesis, 2011.

[18] B. Yu y Z. Z. Yang. An ant colony optimization model: The period vehicle routing problem with time windows. *Transportation Research Part E: Logistics*, 47:166–181.



The periodic traveling salesman problem, model on which this report will focus, is a variant of the traveling salesman problem. Its objective is to minimize the cost of the route where cities must be visited a predetermined number of times and may be conditions on the periods to be visited.
This variant of the traveling salesman problem has not been studied in the literature as other variants, however, there are many cases with real applications.
The aim of this work was to study the literature, present mathematical models and solve some examples of this problem. To solve mathematical models, we have chosen to code it with two different softwares, one free and another commercial.

Mª Arantxa Peña Rodríguez

Marzo, 2016
Faculty of Science, Mathematics Section
University of La Laguna

Periodic Traveling Salesman Problem